TECHNICAL REFERENCE SUITE ADDRESSING CHALLENGES OF PROVIDING ASSURANCE FOR FAULT MANAGEMENT ARCHITECTURAL DESIGN

Rhonda Fitz
MPL Corporation, rhonda.s.fitz@ivv.nasa.gov

Gerek Whitman
TASC, an Engility Company, gerek.whitman@engilitycorp.com

ABSTRACT

Research into complexities of software systems Fault Management (FM) and how architectural design decisions affect safety, preservation of assets, and maintenance of desired system functionality has coalesced into a technical reference (TR) suite that advances the provision of safety and mission assurance. The NASA Independent Verification and Validation (IV&V) Program, with Software Assurance Research Program support, extracted FM architectures across the IV&V portfolio to evaluate robustness, assess visibility for validation and test, and define software assurance methods applied to the architectures and designs. This investigation spanned IV&V projects with seven different primary developers, a wide range of sizes and complexities, and encompassed Deep Space Robotic, Human Spaceflight, and Earth Orbiter mission FM architectures. The initiative continues with an expansion of the TR suite to include Launch Vehicles, adding the benefit of investigating differences intrinsic to model-based FM architectures and insight into complexities of FM within an Agile software development environment, in order to improve awareness of how nontraditional processes affect FM architectural design and system health management.

The identification of particular FM architectures, visibility, and associated IV&V techniques provides a TR suite that enables greater assurance that critical software systems will adequately protect against faults and respond to adverse conditions. Additionally, the role FM has with regard to strengthened security requirements, with potential to advance overall asset protection of flight software systems, is being addressed with the development of an adverse conditions database encompassing flight software vulnerabilities. Capitalizing on the established framework, this TR suite provides assurance capability for a variety of FM architectures and varied development approaches. Research results are being disseminated across NASA, other agencies, and the software community. This paper discusses the findings and TR suite informing the FM domain in best practices for FM architectural design, visibility observations, and methods employed for IV&V and mission assurance.

INTRODUCTION

Fault Management (FM) software is becoming increasingly important for space mission safety and success in support of critical functionality and mitigating operational risk. As NASA spaceflight systems increase in capability and complexity, establishing effective FM architectures to predict, detect, diagnose, and prevent deviations from mission objectives is crucial. Varied approaches to the design and implementation of FM necessitate an adaptive attitude toward the provision of software safety, reliability, and mission assurance. This assurance challenge prompted this research initiative to investigate the complexities of FM as a systems engineering discipline, and to explore how architectural design decisions affect system capabilities and reduction of risk. Even with similar challenges between missions, commonalities are difficult to discern or communicate due to differences in perspectives and terminology across the domain. In order to facilitate efficient and effective software assurance (SA), a technical reference (TR) suite was created to evaluate robustness, assess visibility for validation and test, and define SA methods most successfully applied to FM architectures.

This paper reflects results and is a product of an encore to the NASA Software Assurance Research Program (SARP) Fault Management Architectures (FMA) investigation begun in October 2014 which looked into the varied approaches to FM across eight missions in the NASA Independent Verification and Validation (IV&V) Program portfolio. SARP FMA performed this research, looking across projects from Deep Space Robotic, Human Spaceflight, and Earth Orbiter missions, and has begun inquiry into Launch Vehicles in the follow-on effort. The NASA IV&V Fault Management Community of Interest (FM COI) was leveraged to perform this initiative and continues to be involved in the collection of diverse FM architectural insights and peer review activities. Given limited resources of the two initiatives and the large number of permutations across missions, it is recognized that not all FM architectures are represented in this study; just a few of the most commonly used architectural styles have been described. Much of the information is broadly applicable across other projects, especially within the same mission domain. SA methods and IV&V techniques that have been applied to the various architectures and designs were compiled as well, forming a TR suite that is utilized to communicate lessons learned and improve SA techniques employed at IV&V, across NASA, as well as within the larger space community. The FMA Encore (FMAE) initiative may be described in terms of three growth capacities: first, to improve and expand the TR suite for a more comprehensive coverage of NASA missions; second, to further refine a NASA IV&V Program asset, the prototype adverse condition (AC) database, for increased assurance expedience and broader utilization; and third, to socialize findings and products within the SA community to exchange knowledge for the advancement of FM assurance across the Agency.

The approach and preliminary findings from early research were imparted in a Tech Track paper and presentation at the 31st Space Symposium entitled “Fault Management Architectures and the Challenges of Providing Software Assurance”[1]. Keeping the redundancy to a minimum, the reader is advised to refer to that publication for more in-depth process details. Additional findings and the description of the continuation of this effort are presented in this paper. Beginning with a brief introduction to the NASA IV&V Program and then a description of IV&V methodology, context is given for applicability of the TR suite. A description of the scope of the new FMAE initiative, and then an arranged, logical view of the TR suite, including the FM Architecture Matrix TR, FM Visibility Matrix TR, and FM Assurance Strategy TR, is presented in the subsequent sections, along with discussion of analysis and findings. The data and results provided in this paper have been condensed in order to avoid including any developer-specific or regulation-controlled information. The FM lexicon used is in agreement with that established in the NASA Fault Management Handbook, which is considered the central authority on FM terminology for the Agency.[2]

NASA Independent Verification and Validation Program

The NASA IV&V Program was founded in 1993 under the NASA Office of Safety and Mission Assurance (OSMA) as a direct result of recommendations made by the National Research Council (NRC) and the Report of the Presidential Commission on the Space Shuttle Challenger Accident.[3] The NASA IV&V Program was established as part of an Agency-wide strategy to provide assurance that NASA safety and mission-critical software will operate reliably and safely and to advance systems and software engineering disciplines.

The NASA IV&V Program has a primary business focus to support NASA missions. The Program takes a systems engineering approach to enable the highest achievable levels of safety and cost-effective IV&V services through the use of broad-based expertise using adaptive engineering best practices and tools. NASA IV&V performs independent testing and analysis throughout the software development lifecycle resulting in objective evidence that provides a level of assurance that system software has been developed in accordance with quality standards, will operate reliably, safely, and securely and that sufficient risk mitigation has been applied to the software that controls and monitors critical NASA systems.

NASA IV&V Methodology

NASA IV&V methodology has three guiding principles in planning and performing analysis. They are: Criticality, the Three Questions (3Qs), and Assurance Strategy.

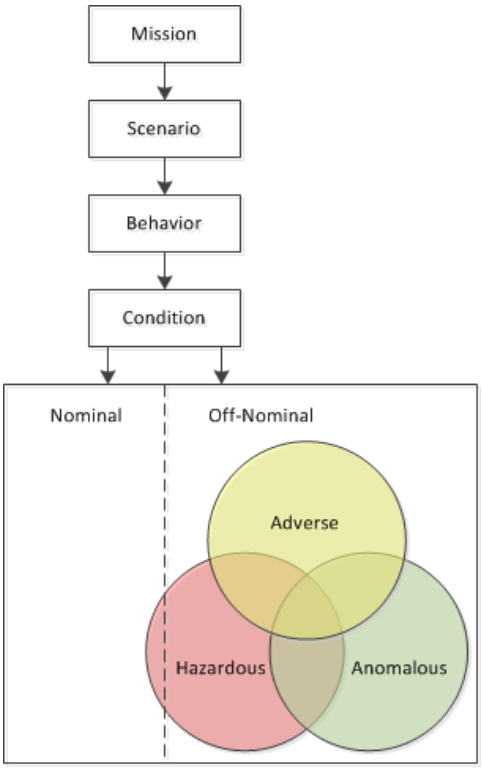
Criticality

NASA IV&V performs a criticality analysis to build an understanding of the software being analyzed. This criticality analysis is a risk-based methodology which begins with a system decomposition of the primary behaviors and an architectural understanding of the software entities. This risk-based assessment feeds into the planning process described in the Assurance Strategy. Because FM is associated with asset and human safety, this discipline typically is highly ranked in all NASA IV&V criticality analyses.

Three Questions

IV&V analysis is performed with the following perspectives described in IVV 09-1, the IV&V Technical Framework, known as the 3Qs[4], to support the Assurance Strategy:

Q1: Will the system's software do what it is supposed to do?
Q2: Will the system's software not do what it is not supposed to do?
Q3: Will the system's software respond as expected under adverse conditions?



Examining Q2 and Q3 are major challenges of FM software. To clarify, an *adverse condition* is considered a subset of an off-nominal state that prevents a return to nominal operations and compromises mission success unless an effective response to the causal fault is employed. How a system is architected to handle faults and adverse conditions is crucial for the satisfaction of functional and performance requirements for mission success.

Exhibit 1: Adverse conditions illustrated by the decomposition of an off-nominal state associated with a specific behavior within a scenario of a particular mission

Assurance Strategy

IV&V techniques involve defining an Assurance Strategy to support IV&V planning and execution. Illustrated in Exhibit 2, this strategy has three distinct phases. The first phase involves understanding the risk posture of the

software and the criticality of the software elements in performing desired behaviors in order to define Assurance Objectives to assure the quality of the flight software for critical behaviors and entities in scope. In the second phase, the IV&V plan is developed to support assurance with objective evidence and documented assumptions and then executed, using documented IV&V methods which result in issues or risks against the Assurance Objectives. The results of the IV&V effort allow Assurance Conclusions to be made in the third phase, communicating the assessment of the software, qualified based on the scope of the analysis, underlying assumptions, and so on. This process is iterated throughout IV&V lifecycle phases.

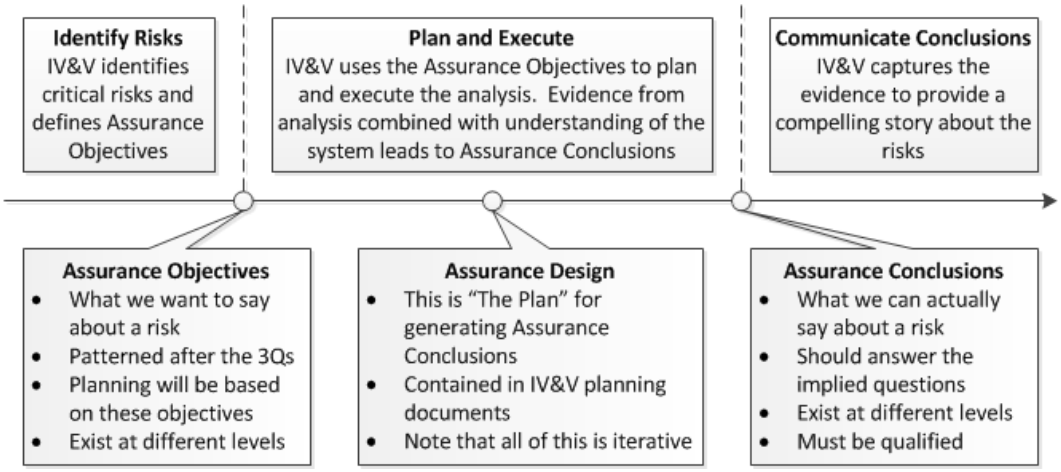


Exhibit 2: IV&V Assurance Strategy

It is common practice for NASA IV&V to adapt to projects in order to be responsive to their development methods. The IV&V Technical Framework and methods for providing SA were developed based on traditional software development lifecycle models. Current IV&V planning and scoping are based upon receiving traditional requirements, design, implementation, test, and traceability artifacts which are used to thoroughly analyze capabilities in-phase and assess and mitigate risk. As innovative development frameworks such as Agile or model-based systems engineering are utilized more frequently for NASA software, NASA IV&V is challenged to modify its approach to SA. FM appears to be leading some of the more dramatic changes, with some of the most complex missions turning to new approaches that will supersede the limitations of monitor-response architectures. NASA IV&V attempts to maintain rigor yet remain in-phase with the developer, adapt and continually improve its capability to fulfill SA needs for the ever-changing environment of NASA projects.

SCOPE OF FAULT MANAGEMENT ARCHITECTURES ENCORE INITIATIVE

The FMAE initiative may be described in terms of three growth capacities that continue the efforts and expand the results from the original FMA research initiative. First, FM for an additional NASA mission, Space Launch System (SLS), is being investigated to improve and expand the FM Architecture Matrix TR. The addition of this NASA IV&V project from the Launch Vehicle domain will provide for more comprehensive coverage of NASA mission types. Delving into FM for SLS will give insight into differences intrinsic to model-based FM architecture as well as how the complexities of FM are handled within an Agile software development environment. Capitalizing on the established framework of the TR, observations will be integrated and the value of usage promoted.

Second, the prototype adverse condition (AC) database promises to be a repository that will exponentially increase the usefulness of fault, failure, and hazard data, potentially becoming an important SA asset. NASA OSMA has promoted the need for identification and test coverage of off-nominal conditions. How a system responds to

these conditions, including what software controls or mitigations are integrated into the system design, is an important aspect of hazard analysis and FM. Understanding what ACs the mission may face, and ensuring they are prevented or addressed is the responsibility of the assurance team, which necessarily should have insight into ACs beyond those defined by the project itself. Early efforts defined terminology, categorized data fields, and designed a baseline repository that centralizes and compiles a comprehensive listing of ACs and correlated data relevant to NASA missions. In order to perform more effective analysis (IVV-09 Q3 in particular) and provide greater test coverage for critical missions, this prototype tool helps projects improve analysis by informing the creation of a comprehensive AC list, tracking ACs, and allowing traceability and queries based on project, mission type, domain/component, causal fault, and other key characteristics. The repository has a firm structure, initial collection of data, and a rudimentary interface established for informational queries. Further refinements are underway with the addition of data from multiple sources including IV&V projects, SA assessments, security threats and vulnerabilities, and hazard analyses, and with input from user workshops that define typical analyst and other stakeholder workflow scenarios.

Third, innovative strategies for improving SA methods and tools have been gleaned from the FMA initiative and will be disseminated. The NASA IV&V FM COI has not only contributed to this effort, but has benefited from the exchange of information and ideas of how to optimize SA techniques through this technical outreach forum, established in relation to FY15 goals. Broadening outreach to socialize outcomes at select NASA centers and through publication of research findings and results will continue, with benefit to both the wider FM community as well as SA practitioners. The arrangement of roundtable discussions on FM architectures and assurance approaches with OSMA personnel and FM subject matter experts (SMEs) will provide an environment of technical knowledge exchange and form connections that will improve the state of FM assurance practice. The SARP FMAE initiative is summarized in Exhibit 3.

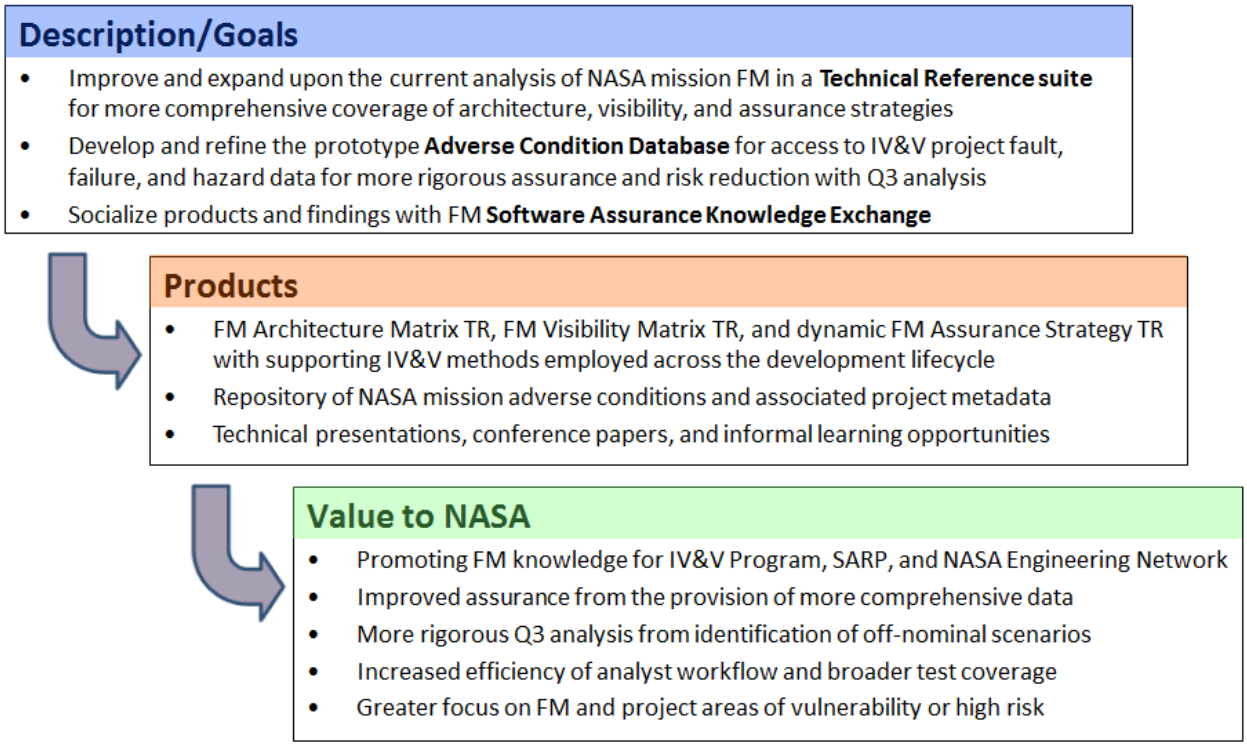


Exhibit 3: SARP Fault Management Architectures Encore Initiative

Technical reference excerpts, analysis, and key takeaways are described in the following sections. The open-ended and qualitative nature of the data collected presents a challenge with regards to condensing and organizing it into a comprehensible form. Eventually, three separate products were developed: first, a FM Architecture Matrix TR based upon FM architectural survey questions which provides summary observations by mission domain; second, a FM Visibility Matrix TR organized by development phases and artifacts which spawned the collection of FM-related IV&V methods that are currently utilized for FM assessment; and third, a FM Assurance Strategy TR, an interactive table of assurance statements organized by phase of the project software development lifecycle and mapped to relevant 3Qs. It is important to note that the versions of these products presented in this paper do not comprise the full dataset; in many cases details indicative of a specific project have been excluded or generalized to protect proprietary concerns, but the full TR suite is available from the author by request.

FM ARCHITECTURE MATRIX TECHNICAL REFERENCE

The goal of this initiative was to examine a diverse subset of FM architectures, and this drove the selection of projects belonging to the following broad mission types: Earth Orbiter, Deep Space Robotic, and Human Spaceflight, with the addition of Launch Vehicles just underway. This categorical convention was found to be suitable in terms of organizing the data. Exhibit 4 lists the nine projects chosen for the initiative.

Project Name	***Mission Type***
Mars Science Laboratory (MSL)	Deep Space Robotic
International Space Station (ISS)	Human Spaceflight
James Webb Space Telescope (JWST)	Deep Space Robotic
Multi-Purpose Crew Vehicle (MPCV)*	Human Spaceflight
Joint Polar Satellite System (JPSS)	Earth Orbiter
Magnetospheric Multiscale (MMS)	Earth Orbiter
Geostationary Operational Environmental Satellite R-Series (GOES-R)	Earth Orbiter
Solar Probe Plus (SPP)	Deep Space Robotic
Space Launch System (SLS)	Launch Vehicle

Exhibit 4: Overview of focus projects chosen for the Fault Management survey

The IV&V Program hosts the FM COI to share best practices and lessons learned across the Program. SMEs from the FM COI agreed to participate in research supporting this initiative, and have contributed to the data analysis and peer review process. The FM COI additionally serves as a platform through which findings and recommended practices are disseminated. FM survey questions for SMEs were generated and revised as further considerations emerged during interviews. The final set consisted of thirty-five FM architecture and nine IV&V methodology questions to address the characteristics of the FM architectures in order to differentiate and delineate FM systems. Questions were grouped into four categories: *Structure* (to obtain a high level view of each architecture and provide insight into size, complexity, and scale), *Concept* (to address the design process and major design ideas and themes of the FM architecture), *Implementation* (to acquire technical detail about how the FM architecture was built and how it works), and *Other Architecture-Related Questions* (to assess heritage and other mission parameters in order to provide additional context). Together, these questions and the additional insight

* The data collected on MPCV includes architecture and analysis from both Exploration Flight Test 1 (EFT-1) and Exploration Mission 1 (EM-1). Though neither of these are crewed missions, the system is being designed and built to support future crewed capability, which is why it has been characterized under the Human Spaceflight domain.

gained from discussions held during interviews provide preliminary understanding of each FM system and its architecture.

The full FM Architecture Matrix TR is available from the author by individual request. In order to present some conclusions to the interested reader, an abbreviated version of the TR with all of the actual survey questions and the generalized responses is included as Exhibit 5. This matrix is based off of the spreadsheet used to collect survey responses. The projects were grouped by the mission domains described previously: Deep Space, Earth Orbiter, Human Spaceflight, and Launch Vehicle (although the Launch Vehicle dataset has not been analyzed and incorporated into the summary as of yet). After examining a few different ways of categorizing projects, it was determined that this convention fit the emerging data patterns the closest. Survey responses were summarized and generalized along mission domains, and then an overall summary was generated to address cross-domain trends or additional observations. In this way, high-level patterns of different styles of architectures used in each mission domain became more apparent. It is this summarization that has been included in this matrix for the sake of information manageability. These generalizations have been reviewed by the FM COI for accuracy.

This matrix only contains data gathered from the architecture portion of the survey. Survey data from the IV&V section was omitted because it is addressed in greater detail in the FM Visibility Matrix and FM Assurance Strategy Technical Reference sections. The FM Architecture Matrix TR became the basis for developing an understanding of the FM architectures in use by developers, and the similarities and differences in styles and structures. This analysis and its results are described further in the next section, FM Architecture Types.

Fault Management Architecture Matrix	
Survey Question	**Cross-Mission Observations**
Structure - How is it structured/organized?	
Is the FM architecture fully local? System? Hybrid? Some other organization?	A tradeoff exists between the simplicity of a centralized system level approach, and the robustness of a hybrid, tiered approach. The lower the level at which the fault can be handled, the less impact it has on the system. Earth Orbiters tend to be more centralized, Human-rated vehicles more distributed, and Deep Space falling anywhere along the scale depending on the mission parameters and developer.
How many tiers/layers are there in the FM architecture? Do these tiers/layers overlap?	Tiers are used to organize systems that are not centralized, but even the most centralized examples here still have hardware layer FM. Often there are two tiers: local and system. Sometimes FM is just primarily system level (with some additional hardware layer FP), and sometimes one or more intermediate tiers are used in between local and system, depending on the complexity of the spacecraft architecture. Usually these tiers have to overlap the same faults to allow them to be handed up from a lower tier to a higher one, but this is always done in a systematic, logical way.
How is the FM architecture implemented within the FSW architecture?	Local FM is usually distributed among various software modules; system FM is usually contained within one or two modules that are not necessarily fully dedicated to FM (responses may be the domain of a module that is responsible for all autonomous command sequencing, for example). A table-driven or similar approach is common to all these missions in some way, usually at the system level. System level FM is sometimes considered its own subsystem (or at least a part of the system-level design), with its own engine or software structures, but local FM is usually not treated this way.
What parts of the FM system (software and hardware architecture) are redundant?	Redundancy in FM monitors/responses is usually implemented for greater failure tolerances. If a mission does not require more than one failure tolerance, it is usually just dual-sided.
Are there some parts of the FM system that rely on other parts in order to function?	When an FM system is tiered, higher tiers depend on lower tiers to pass faults up. Local and centralized, and sometimes higher-tier FM relies on telemetry data from other hardware and software.
To what degree is the FM system modifiable? Can monitor logic and fault responses be easily changed, or was this capability avoided/not considered during development?	Modifiability is a highly-desired trait. Table-driven or similar approaches often have high modifiability, because FM logic is stored as a memory object rather than hard-coded, so a rebuild is not required to make changes. This is easier to achieve on a centralized system, where all the fault logic is handled at a single level.
How are the fault monitors organized, both in terms of HW and SW?	Monitors with hardware components are naturally kept close to the fault locations. Coding patterns of some kind are typically used to make all monitor software uniform and clear. In multi-tier systems, faults are handled by the subsystem to which they are local.
Concept - What are the big design ideas?	
What fault analysis was used in the development of the system?	FMEA and fault trees are common.
What was the process used to develop the FM architecture and system?	Developers tend to fall back on what they know and have experience in - heritage programs, prior life cycle processes, even ones that are of different mission domains. Human-rated missions require a slightly different approach, however, and may require a more unique process.
Is the system fully automated? Does it allow for human intervention? Is it designed with humans in the loop?	Timing often requires high autonomy, either because human reaction time is too slow, or because of communication delays. Most Earth-Orbiting and Deep Space missions are not designed around having human controllers constantly watching, and some don't even dictate regular contact, but ground ops is always given the capability to perform FM procedures. Degree of autonomy appears to correlate loosely with distance from operators (onboard or on the ground).
If humans have a role in the FM system, how important is their role? How frequently do they intervene?	Deep Space missions are often designed to have infrequent ground contact as compared to other mission types, and commands are given to span a much longer time than is expected before next contact. Usually ground's most important role in FM for robotic vehicles is commanding out of safe mode so that operations can continue.

Does the mission have multiple phases (e.g. Launch, Operations, Orbit, Cruise, EDL, or perhaps some more mission-specific phases during which the spacecraft is operating in distinctly different ways)? Does the FM system act the same in each phase?	The FM system is not redesigned for various phases, even if the environment or hardware being controlled changes. Usually if a mission phase requires the system to act differently, new FM logic is uplinked or inhibit commands are used. The largest difference between mission phases is the level of autonomy required for the spacecraft. Transitional phases (launch, burns, etc.) can be problematic with respect to monitor-response functions. Because the state of the spacecraft during these times is such that it would normally trigger faults, these faults are usually just inhibited for these brief time periods.
How does the system know when there is a fault? How are faults defined to the system? e.g. rules-based, monitor-response, state & goal-based, model-based, filters, events, etc.	Rules-based and monitor/response style architectures are still the norm.
Can the system respond to a fault that was unforeseen & unplanned for?	Rule-based systems don't have this express capability, because they won't be monitoring for an unplanned fault, but if its effects propagate, it will trigger a response of some kind, like safe mode, or human controllers will be able to issue commands to respond.
Are HW and SW faults managed in different ways by the system?	On the hardware layer FM, a hardware fault may not even trigger any FSW involvement, just something internal to the device. At higher tiers, hardware and software faults are handled by the same systems and treated the same way - only the logic in the responses differentiates between the two. Differences in the way faults are handled, if any, are usually based on priority or other logic, not the type of fault.
How is redundancy in HW/SW treated by the FM system? Does it utilize hot, cold, or warm redundancy? What types of redundancy are used?	This depends mostly on how much fault tolerance is required, and the criticality of failures. Critical hardware items, like processors, are often hot or warm backups, whereas other things like instruments, sensors, and the like are cold. Where more failure tolerance is required, more functional redundancy is used in addition to hardware-identical redundancy.
What does the FSW do if it can't correct a fault on the first/second/etc. attempt? Does it have multiple methods of correction? Does it wait for human intervention?	Usually FM logic dictates response repetition, either until it is clear nothing is happening (in which case ground ops may inhibit the response if the spacecraft is not capable of doing so), or the fault propagates into something more severe. Occasionally tiered responses are used, which require more complex logic. Human intervention into FM on robotic vehicles usually only happens when safe mode is entered due to an unrecoverable fault.
Implementation - How was it built, how does it work?	
At what stage of the mission life cycle was the FM system designed and built?	More and more, FM design is happening sooner, more in phase with the rest of the spacecraft systems, guided by heritage and previously-developed standardized architectures, but it still has the potential to lag behind, especially to adapt to changes in other subsystems.
Was the FM system built from a top-down perspective, bottom-up, or a combination of the two?	Usually both. Component level FM requires a bottom-up understanding of the hardware functionality, but architecting system level FM must be done from the top down to minimize complexity.
How many fault monitors and unique responses does the system have?	The more requirements the FM system has for preserving functionality when something goes wrong, the more monitors and response logic it is going to need to do its job. Generally a system will have more monitors than responses, since different monitors or faults will trigger the same response.
How many lines of code does the FSW have?	LOC is perhaps not a good measure of complexity. Source code was out of the research scope, but was considered for a time as a metric for evaluation.
Are there multiple monitors for any particular faults?	This sometimes indicates criticality of fault, or simply that one monitor is not capable of covering the entire fault area.
Are there faults that trigger more than one response?	Depends on how monitors-responses are mapped. Sometimes a response is just simply multiple commands to carry out, but sometimes responses are able to call other responses in a nested fashion, perhaps even with some additional logic or parameters attached.
Does the system try to predict faults before they happen?	The faults for which this is possible are those that involve a more control loop-like procedure (e.g. thermal or attitude control). Could indicate that a control loop approach to FM in general may allow for more prognostics to be done in other areas.
Can the system determine if a fault is detected erroneously? To what degree is persistence utilized?	Persistence is a staple of all FM systems, and keeps fault responses from triggering when they should not. Above this, any sort of intelligent reasoning about fault triggers by the spacecraft is not implemented.
Can the system keep one fault from triggering others? How?	Inhibits, redundant switching, and other isolation techniques are common across all systems.

What happens if more than one fault is detected? Are they managed in parallel or series? Do faults/responses have priorities? Is response collision a concern?	A lot of variance in this area. Usually faults at a local/component level in different subsystems can be handled simultaneously, but at higher levels, it depends on the design. Priority is not always used, and when it is, responses may or may not be able to interrupt each other.
Do the subsystems communicate with each other about faults? How?	Usually they only communicate up to the system level, which then directs other subsystems.
Other Architecture-Related Questions	
Are there any other key design features or descriptive qualities that are important for understanding the FM architecture?	In centralized architectures, the schemes used to link monitors and responses in memory constructs are key to understand. In more distributed architectures, interfaces between local and system level FM, and the way in which information is exchanged between the different tiers and partitions are important.
Is this FM architecture inherited from another mission or based on a previously-developed standardized architecture?	All projects have some degree of inheritance, in the actual architecture and design or development process. Developers often draw from their accumulated knowledge of what does and does not work in FM architecture development.
Are there any design features that set this FM architecture apart from its predecessors? Were any new approaches to FM used?	None of the projects examined used cutting-edge approaches to FM such as model-based development or state & goal based architecture.
How did the mission domain and parameters influence the design of the FM architecture?	Critical mission events and other significant mission parameters like autonomy, onboard crew, and failure tolerance are often the largest drivers for structural and functional FM architecture design.
Did the FM architecture change in design as mission parameters changed?	When architected early with a top-down view and kept as simple as possible, architectural changes later on are less likely.

Exhibit 5: FM Architecture Matrix

FM ARCHITECTURE TYPES

After survey information was collected, analysis was performed in order to find trends and patterns. Though the sample size of this study was not statistically significant to the FM flight software domain as a whole, there are interesting observations that have been drawn from the dataset, discussed in this section. With more detailed data collection, the TR will increasingly serve to clarify abstractions of consideration for FM architecture classification.

There are many differing viewpoints on what "architecture" means when applied to FM systems and software systems in general.[†5] Common to most is the concept of two distinct architectural views: functional and structural[‡]. Chapter three of R. Schmidt's *Software Engineering: Architecture-Driven Software Development* contains descriptions of these concepts most similar to those used in this paper.[6] Other texts contain similar ideas, including chapter four of S. Albin's *The Art of Software Architecture: Design Methods and Techniques*, which adds an additional property, fabrication, that describes the architecture's quality[7]; section 1.3 of I. Gorton's *Essential Software Architecture*, which takes the further step to include a development view that describes the organization of the architecture within a development environment[8]; and section 3.1 of R. Taylor, N. Medvidovic, and E. Dashofy's *Software Architecture: Foundations, Theory, and Practice*, which contains numerous definitions of software architecture that reference the ideas of structure and function[9].

Concepts of structure and function used in this paper generally follow the definitions within literature; functional architecture describes how a system operates from start to finish, what tasks and transactions are needed, and how they relate to one another. The structural architecture describes how those functional segments are organized and implemented in hardware components and code. Typically in software development, the functional architecture is defined first, and then the structural architecture is developed so that it supports and logically follows from the functional architecture, but structural constraints, particularly those related to hardware, can also influence functional design.

The information gathered on NASA spacecraft FM architectures within the FM Architecture Matrix TR supports this notion of two separate architectural views. The following diagrams and explanations were created to represent the data in a way that shows similarities and differences in architectural styles, and to provide a simplistic reference for common FM architectural themes across projects. Each of these is an example of a monitor-response or rules-based FM architecture; all projects investigated fell into this general category. The rules-based approach to FM is defined and described in detail in J. Rustick, M. Wisniewski, and D. Termohlen's *Fault Management for Complex Space Systems: Assessing the Use of Rule-Based Approaches*.[10]

Centralized versus Hybrid

In analyzing the data, it became apparent that the general architectural configurations fit into one of two broad categories: centralized and hybrid[§]. This distinction is not unprecedented; centralized/distributed terminology is also used in Rustick, Wisniewski, and Termohlen.[9] Centralized FM architectures have a single FM

[†] For a somewhat different perspective on the description and breakdown of FM architecture than the one in this paper, see K. Barltrop's appendix to the *NASA Study on Flight Software Complexity* from 2009. (Numeric bibliographic reference is provided in-text next to footnote.)

[‡] Some references use the term 'physical' rather than 'structural,' including Schmidt and Gorton, probably to avoid confusion of terminology due to the fact that a functional architecture has itself a structure of some kind. However, FM systems are made up of both software and hardware components, so 'physical' may be misconstrued as referring only to the hardware aspects of the FM architecture in this context.

[§] "Hybrid" here refers to a hybrid between a centralized and distributed approach. A truly distributed architecture is not used in flight software; there is always a master control entity at the top level that can command any other entity.

system at the top level that controls all software-based FM functionality. Hybrid FM architectures, as the name suggests, have FM functionality distributed to various local software partitions, usually on a subsystem basis, in addition to the top-level system FM module.

The simple centralized architectures are more often used in smaller, less complex missions, like Earth Orbiters, while the more robust hybrid styles are usually favored in deep space and human-rated missions where failure tolerance and autonomy requirements are more extensive. The following sections describe the basic functional and structural aspects of each configuration which are shared by most of the systems that were examined, using the associated figures as a reference.

Centralized Functional Architecture

The functional process for any FM system can be described in five main tasks: data collection, detection, response, isolation, and recovery. Exhibit 6 decomposes the first three of these further into component functions that are reflective of typical centralized FM structures and relationships. Isolation, in this context meaning the analysis of a fault effect to determine its cause, and recovery are both typically performed by ground operators and are not included in this discussion of flight software architecture.

Data Collection

Before it can make any evaluations, the system FM module, or engine, must collect all the necessary data. Data may be input from sensors within hardware components, sensors that are external to hardware components, or from other subsystem software modules that encapsulate or store data in a more meaningful way than raw sensor output. This often is done in a repetitive, polling fashion. Error checking schemes and other verification or validation procedures are usually engaged prior to the FM software making any responsive decisions in order to prevent propagation of errors or false positives. False negatives are mitigated by requiring more than one sensor to fail and by adding in logical patterns to detect permanence of “out of boundary” conditions.

Detection

Once the system FM module has valid data, it makes a comparison against defined FM logic to find out if there are any discrepancies. In these centralized systems, this logic is typically stored as tables or other objects in memory so it can be easily modified without changing and rebuilding any software. This logic consists of threshold values, persistency counts, and similar information.

Response

If the system FM module finds that there is a valid variance, it then checks for and initiates the associated response command sequence. In centralized systems, these sequences are also usually stored in tables in memory, and the FM logic table typically just points to an entry in the response command table. These command sequences will either order some action by effectors within hardware components or separate entities, or cause some kind of mode change or reconfiguration, which usually has consequences for both hardware and software. These architectures are usually designed to execute a single response sequence at a time, and so a prioritizing scheme is sometimes employed in order to accommodate interruption when a more critical response is necessary to intervene to restore core functionality. Once a response has been completed, the FM engine references a cleanup table and resets or clears persistence values from all monitors that are associated with that particular response, bringing the monitors back to a baseline detection state.

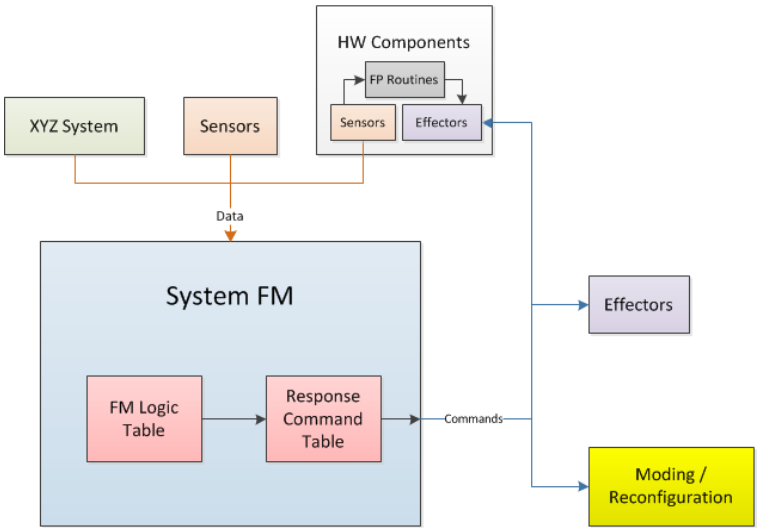


Exhibit 6: Centralized Functional Architecture

These three tasks are also done at the hardware layer within individual hardware components that have built-in sensors, Fault Protection (FP)** routines, and effectors, without any influence from the software layer. This could be something simple, such as a circuit breaker protecting a component from current overload, or something more complex, like a movable component halting because its firmware recognizes that it has been commanded past its bounds of freedom. The software layer is made aware of hardware FP incidents, and may have further FM rules and responses that relate to them, but is not involved in the execution of hardware FP.

It should be apparent that the centralized functional diagram and hybrid functional diagram below both describe an open-loop system, which is typical of rule-based architectures. Sensors do not directly check the result of a performed response, but only watch for violations of their rules, and monitors only designate a pre-determined number of response sequences when a fault is detected, as opposed to continuous commanded adjustment. Certain faults, particularly those related to attitude or thermal control, may instead have closed-loop algorithms, but these are rare. Also of note, during the timeframe when the spacecraft is going through a configuration or mode change, allowable FM engine behavior should be clearly defined so as to not interfere with the overall system change during the transition.

Centralized Structural Architecture

All FM structural architectures consist of two distinct layers: software and hardware. IV&V scoping focuses on the software layer and addresses the hardware layer when necessary to complement analysis of software. Exhibit 7 shows the typical structural components and relationships of a centralized architecture.

** This difference in terminology is not arbitrary. Hardware responses are based around the idea of preventing faults and avoiding damage caused by faults, so is thus referred to here as Fault Protection. Software has the additional task of coordinating responses to and preserving system functionality in spite of both hardware and software faults, and so is termed Fault Management.

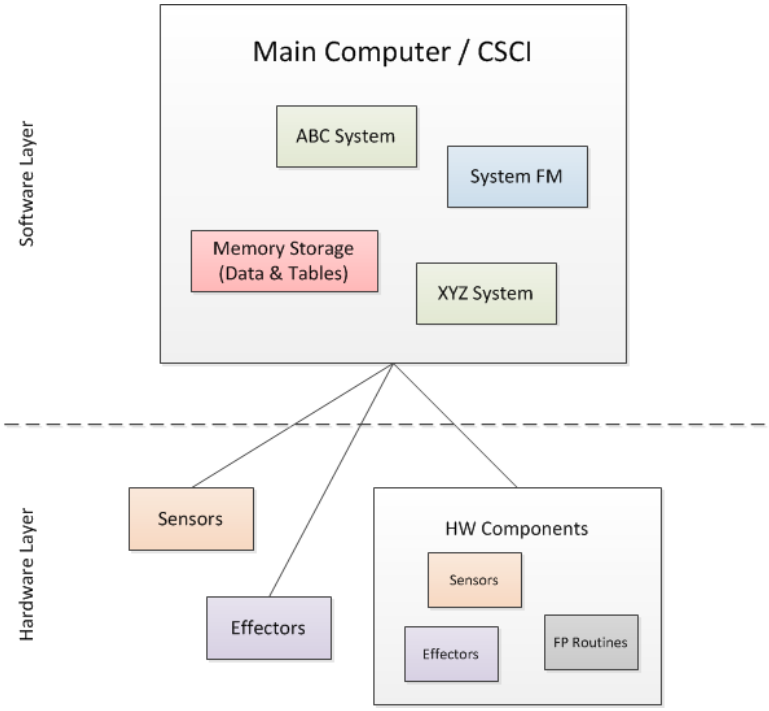


Exhibit 7: Centralized Structural Architecture

Software Layer[††]

In a centralized structural FM architecture, there is a single physical entity (such as a computer or computer system), and sometimes also a single virtual entity (such as a computer software configuration item (CSCI)), that encapsulates all the flight software and data. Within this entity are the various subsystem software partitions, which are often structured in some kind of hierarchy. What makes a structural FM architecture centralized is that it has only one FM system at this top level, with no other subsystems having their own FM software functionality; all the FM software tasks are handled by this single software system. Without a single physical entity to encapsulate the entire flight software, the FM system necessarily cannot be structurally centralized.

Hardware Layer

The single entity in the software layer controls all the actions of the hardware layer, which consists of sensors, effectors, and hardware components that may have built-in sensors, effectors, and simple FP routines that can act without influence from the software layer. This structure allows the single FM system to have control over the hardware it needs to respond to any fault detected at the software layer.

Hybrid Functional Architecture

A hybrid functional architecture can be described using the same three FM tasks that were listed above for the centralized functional architecture. Exhibit 8 shows an architecture with two tiers, system and local. Note, however, that hybrid architectures can have more than two tiers, though any more than three are rare.

[††] Although it is termed the 'software' layer, note that this layer also contains the data entities the flight software reads from and writes to as part of its operations, such as FM logic and command tables stored in memory, which are not typically considered to be part of the software code.

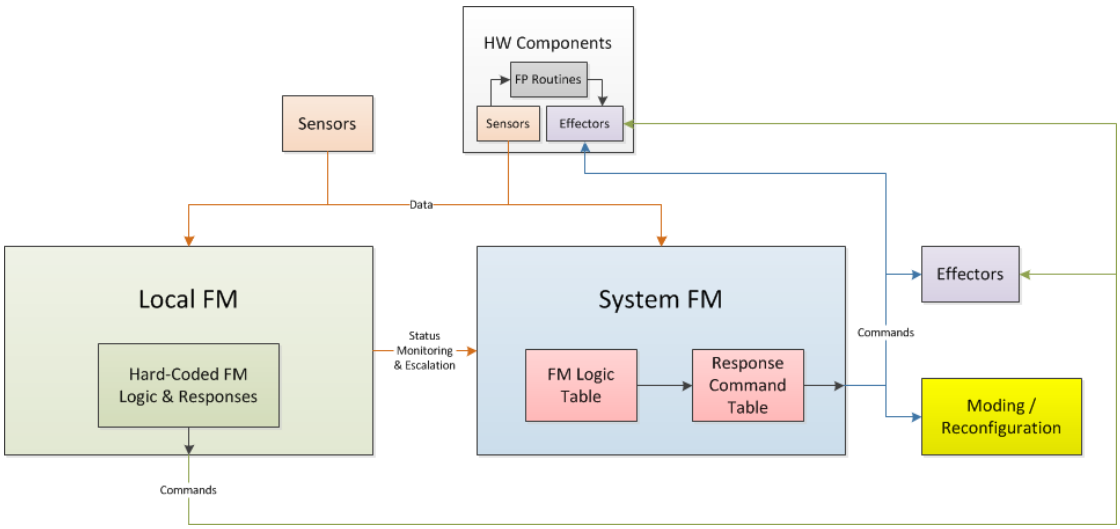


Exhibit 8: Hybrid Functional Architecture

Data Collection

Data collection in hybrid architecture happens in much the same way as in a centralized architecture, but with more than one end location gathering data. Both system and local FM collect data from sensors, but the local FM segments only collect data relevant to their operations, and pass to the system FM any relevant information for status monitoring and potential escalation purposes. System FM does not collect all the data that each local FM segment does, because it does not monitor for the same faults.

Detection

At the local level, FM logic is usually hard-coded into the software, so the software will check directly to see if any of the data it receives indicates a fault. Rarely does local FM have a table-driven logic/response implementation. System FM logic, on the other hand, usually is table-driven, similar to the centralized style, and thus tends to be more modifiable than local FM logic. Because the system FM may not have direct access to all the information that local FM does, it often relies on local FM to pass on data in order to make its evaluations.

Response

After detecting a fault, the local FM will execute its hard-coded response on one or more effectors that it has access to. Likewise, the system FM will execute a command sequence, but from its table stored in memory, not directly in the software. Because of the distributed architecture, system FM may not have direct access to all the effectors, so it may have to relay commands through subsystem software. Only system-level FM has the capability to make system mode or reconfiguration changes.

It should also be apparent from this diagram that concurrent responses are much more likely in hybrid architectures. Two local responses in different subsystems due to independent faults are not likely to be problematic, but response interaction is an area of concern when local and system FM are both responding to faults. Concurrent responses and interactions may be intentional, for added robustness, but when they are not, they may have serious and unexpected effects on the health of the spacecraft. As in a centralized architecture, also, the hardware layer can still perform detection and response on an individual component basis.

Hybrid Structural Architecture

As discussed above, the FM structural architecture consists of a software and hardware layer. This diagram details a structural view of architecture with three software tiers, as opposed to the two-tiered functional architecture in the previous diagram.

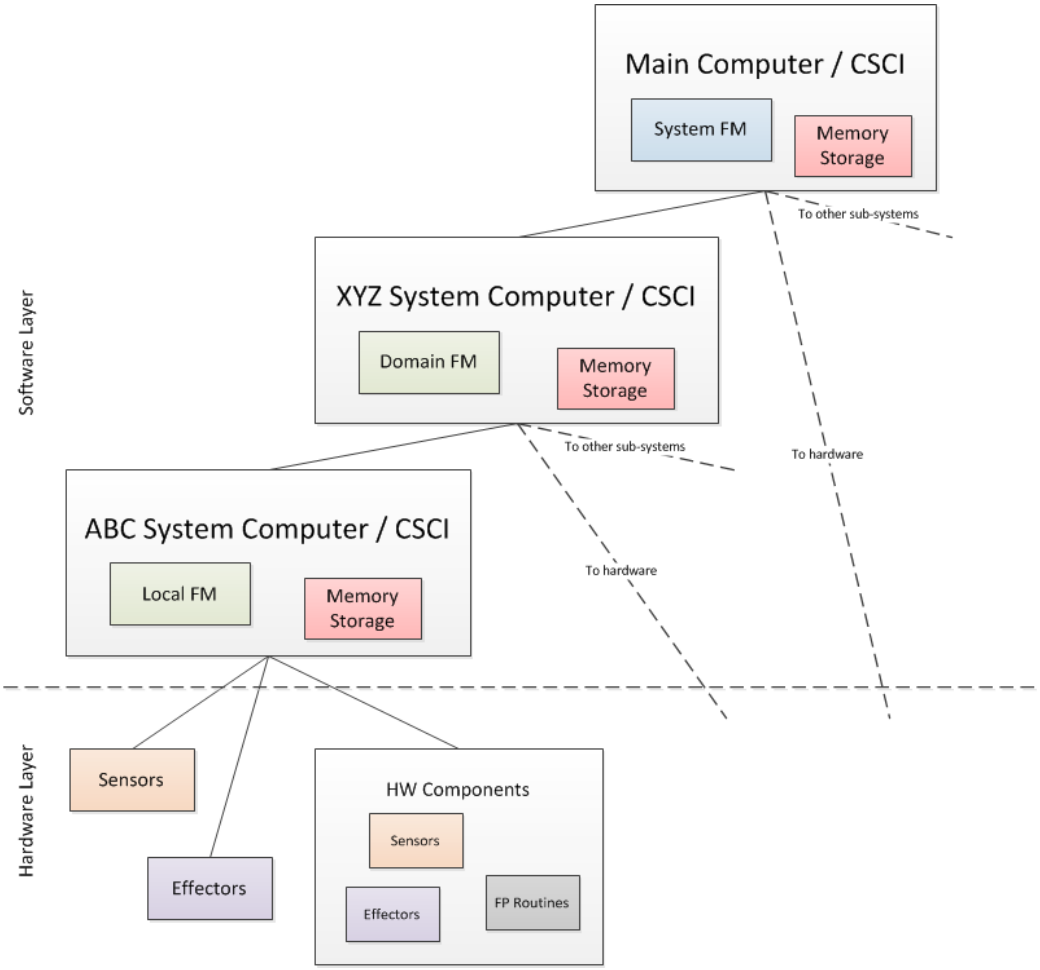


Exhibit 9: Hybrid Structural Architecture

Software Layer

The key difference in structural architecture between the centralized and hybrid styles is in the FM structure in the software layer. Hybrid architecture has FM embedded in multiple tiers as part of other subsystems, in addition to a top-level system FM. This distribution of FM can be either physical or virtual; a system with multiple computers will necessitate distributed FM, but a single computer or computer system can have subsystem software partitioned such that individual partitions have their own embedded FM. In addition, computers in physically distributed systems likely all have distinctly separate memory storage, whereas virtually distributed systems can have shared memory.

Hardware Layer

The hardware layer in a hybrid structural system, shown in Exhibit 9, looks very much the same as one in a centralized structural system, with the key difference being that control of hardware components is distributed among subsystem partitions in the software layer. The lowest software tier is not the only tier that can have hardware control; any software tier can directly control hardware, even the top tier. Hardware control is typically

delegated to the software that is most 'local' to a particular hardware element, which could be physical locality (e.g. each separate computer controls all the hardware in its module), or functional locality (e.g. the guidance and control subsystem controls all attitude determination and control hardware).

Architectural Development Observations

Within the discussion of architectural styles, it is worth reporting that the data and discussions from SMEs have indicated that, in some cases, the developer of a project has a larger impact on the FM architecture than the mission domain and parameters. This should not be surprising, as many have remarked upon the developer's influence on software architecture, dating all the way back to M. Conway in 1968[11], but it remains a noteworthy one. Developers, for good reason, tend to fall back on what they know both in terms of products and processes, especially for complex system-level capabilities like FM, even when the heritage mission is completely different. For example, one developer of a complex, high criticality deep space project that was studied also built an Earth satellite with very similar hybrid FM architecture, only scaled down in size. The reverse is also the case; another developer is using a simple centralized architecture from an Earth orbiting mission as heritage for a deep space project with more required autonomy and FM logic rules. This kind of reuse is not inherently a bad thing; in fact, the NASA Office of Chief Engineer's *NASA Study on Flight Software Complexity* states that "there is no such thing as an inherently good or bad architecture."[12] If a FM system or architecture works in accomplishing a mission, then theoretically the goal has been met. However, poorly designed architectures may result in extensive rework when system modifications arise. It is crucial for both developers and independent analysts to understand both the heritage system, and the implications of reusing that system on a project for which it was not designed.

One encouraging trend regarding reuse of architectures is the emergence of software architectures, including FM systems, that are designed and built separately of any mission with the intent of being usable, or at the very least a firm starting point, for a variety of missions. An example of one reusable software framework that is platform and project independent is Goddard Space Flight Center's Core Flight System (cFS)[13]. A number of the developers in this study use some form of architectural heritage, or reuse. This enables familiarity among both developers and analysts, and minimizes some of the effort required at the start of analysis to learn and understand the system. In addition, this practice allows for FM architecture to be set in place early on, which often leads to a more well-developed and easier to understand FM system. This was also one of the findings of P. Schmidt, D. Brueck, M. Rokey, and J. Pope's *Independent Assessment of Two NASA Fault Management Software Architectures*. In examining two different architectures repeatedly used over time in different NASA missions, they found that as familiarity with the architectures grew, improvements in multiple mission phases, including planning and testing, were apparent.[14]

FM architecture designed early and with intention forms an excellent resource for enabling communication with stakeholders as well as managing system complexity. Many of the quality concerns for the system as a whole are wrapped up in the deliberate design of a safe, reliable, affordable, and sustainable system in the face of potential hazards. Addressing not only the nominal functions early in the software development effort but also reasoning about how the system will handle the off-nominal states is the conscientious approach to FM design. Beginning with a solid FM architecture, specific to the operational environment and with minimal interfaces or data handoffs, will contribute to early detection of potential flaws or defects introduced prior to implementation and testing. FM requirements need to be included in the earliest project efforts, regardless of the choice of Agile, waterfall, hybrid, or any other development framework, and should be concurrent with, and preferably embedded within, the software system as a whole. Separating out local or system responses to faults as add-on features or after-the-fact design is considered an unacceptable practice for safety-critical missions, and poses a significant risk.

Independent analysis of a system's FM architecture by a team of SMEs assists developers in tackling the complexities inherent in FM. Projects should provide visibility into FM capabilities beyond just monitoring for hardware faults; relaying overall system health concerns, detecting resource limitations, raising flags for parameters outside of safe zones, coordinating multiple, potentially conflicting responses, performing goal-oriented behaviors, recognizing security vulnerabilities, and in general, mitigating software-related hazards, are only some of the incredibly numerous aspects that are encompassed within the FM architecture. Assessing architectural patterns, sometimes with the use of Architectural Analysis and Design Language (AADL) or other model-based approaches, or constructing capability based (goal/claim) assurance cases are ways the IV&V Program might have greater system-wide insight into complex FM principles. Ways in which projects might simplify FM architectures to improve system robustness need further exploration. The expansion of the TR suite in development as part of the FMAE research initiative will inform this effort and begin to provide a resource for improved IV&V and SA. The following sections will briefly touch on the other two components of the TR suite: the FM Visibility Matrix TR, and the FM Assurance Strategy TR.

FM VISIBILITY MATRIX TECHNICAL REFERENCE

The full FM Visibility Matrix TR is available from the author by individual request. "Visibility" is a term that was chosen specifically for one of the goals of this initiative and does not have a precedent in architectural literature, so a definition was generated and adhered to in order to provide cohesion of data. Visibility is defined as a qualitative measure of the difficulty in viewing, understanding, and making evaluations about a system or system architecture. Querying IV&V SMEs about IV&V analysis of FM architectures through survey and interviews provided the benefit of being able to gain knowledge related to FM architectural attributes, SMEs' grasp of the overall FM system, and insights into the IV&V analysis process specific to FM. The purpose was to investigate potential visibility issues inherent to specific FM architectures or to particular methods employed for assurance. For example, if a SME was unable to provide an answer to a question on the survey, additional discussion was warranted for clarification during the interview. Was the question unclear? Did it not apply or was it impossible to answer for this particular project? Was the SME lacking some information about the architecture due to unavailable artifacts, an incomplete assessment, the topic being out of scope, or some other reason? Each of these explanations is valuable in a different way, potentially to refine the survey, advance understanding of the project's parameters, or indicate a lack of visibility. Time and effort savings were realized by conducting interviews rather than merely combing through design documents for FM architectural information. This enabled insight into both the complexity of FM systems and the challenges in developing and executing an appropriate IV&V Assurance Strategy. The nine survey questions associated with IV&V analysis are shown in Exhibit 10.

Survey: IV&V Analysis Questions
• What were the key drivers to IV&V on this project?
• What were the critical errors that IV&V was focused on assuring against?
• What other Assurance Strategies were involved in the IV&V of this project?
• What kinds of artifacts did you get from the developer to use in the analysis, and how did the types of artifacts you received affect your analysis?
• Were there types of artifacts you did not receive or the developer did not generate that would have made analysis easier/faster/more complete?
• What kinds of technical reference(s) did you generate during your analysis?
• If the FM system was inherited or standardized, how did this influence your analysis?
• What language was used to write the FSW? How did this choice in language make analysis easier/more difficult?
• What was the highest benefit analysis? In retrospect, were there things you or the IV&V team would or should have done differently?

Exhibit 10: Survey Overview: IV&V Analysis Questions

The first version of the FM Visibility Matrix TR was developed as an extension to the FM Architecture Matrix TR, but it soon became a separate TR entity since it made more sense to organize it by phase and artifact, in line with a traditional IV&V approach, rather than by question. The FM Visibility Matrix TR outlines a list of development artifacts, some specific, others more notional, descriptions of the artifacts, and related visibility notes and associated IV&V methods or techniques. The FMAE research initiative began primarily interested in visibility from the point of view of an analyst working independent of the development team, most often at a remote site. However, there are two somewhat different aspects to visibility that are segregated in the FM Visibility Matrix TR: "analyst visibility", as described above, and "architectural visibility". Architectural visibility is a property of the system itself, driven mainly by complexity. It changes only when the system architecture or design changes. For example, a system with many interfaces may have a lower visibility than one with few interfaces, because the latter is typically easier to comprehend. Analyst visibility is a separate property that specifically addresses the independent analyst's insight into the system and architecture. It is driven by a number of factors, including availability and quality of artifacts, interactions with developers, and independent investigation. Analyst visibility changes over time as an analyst learns more about a system or provided documentation improves. An excerpt from the FM Visibility Matrix TR is provided in Exhibit 11.

These two aspects of visibility often play off of one another; for example, when architectural visibility is low, analyst visibility likely needs to increase through additional or more rigorous investigation to achieve the desired level of assurance. Likewise, when architectural visibility is already high, further examination may not be warranted as cost-effective unless barriers to analyst visibility exist. A low degree of visibility of either kind should be a driver for applying SA techniques beyond standard analysis regimen. Comments and notes on visibility challenges within the FM Visibility Matrix TR were developed out of survey and interview responses, continued discussion with SMEs, and general understanding of existing and potential problems. The IV&V methods employed by NASA IV&V for FM analysis were gathered from the NASA IV&V Catalog of Methods Process Asset Selection Service (COMPASS), and from FM COI SMEs, who also provided an evaluation of their effectiveness. A table of these IV&V methods and brief descriptions of each is provided in Exhibit 12.

The FM COI reviewed the accuracy and completeness of the FM Visibility Matrix TR with respect to their projects. Not only is this TR useful as a knowledge-sharing tool across IV&V projects and for other SA groups, but it can also indicate a number of things about FM development and assurance in general, such as what kind of techniques are useful for a particular type of architecture, or which development artifacts typically present the biggest challenge to analysts. Findings related to this analysis are discussed further in the following section.

Fault Management Visibility Matrix			
Lifecycle Phase	**Development Artifact**	**Architectural Visibility**	**Analyst Visibility**
Any	New, innovative, or unique features or development processes in FM	New processes or approaches to FM are automatically a challenge to visibility.	These new features or processes can emerge during any lifecycle phase, so analysts need to be adaptive, particularly if the developer deviates from traditional development processes. As an analyst works with a system, they should naturally begin to pick out what aspects and qualities of the system are most important and most valuable for targeted analysis. Lack of visibility due to a new or unique feature should lead to further investigation.
Concept	System and FM architecture representations	Overall system architectural structure is typically straightforward and understandable. Some visibility problems may emerge when FM architecture is distributed and FM functionality exists in many FSW modules. The more tiers there are to the FM architecture, the greater the potential impact to visibility due to complexity, but when tiers are well-defined, it may actually be easier to learn and understand the levels of severity and response defined within the architecture. The earlier the architecture is introduced, the better the visibility both for the developers and for independent analysts.	Lack of comprehensive architectural representation can prompt work for analysts to create one. A frequently updated architecture representation is important to keep up with changing functional responsibilities throughout the lifecycle. Good documentation with clear traces can offset problems with distributed FM architecture visibility.
	Fault analysis results (e.g. FMEA, FMECA, fault trees, hazard analysis, etc.)	N/A	The type and availability of fault analysis performed by the developer can have a significant impact on visibility. When fault analysis products are not provided, it can be difficult to assure that all reasonable faults have been identified. Almost always, developer's fault analysis is incomplete with respect to software faults. The goal is to have a comprehensive list of faults, failures, and potential combinations to supplement later analysis. Security scenarios should also be considered if they are not already captured in this list.
	Fault Management Plan	When planned for and designed early in the lifecycle, FM architectures are generally more well-developed and documented, and therefore more visible, as opposed to architectures designed more as an afterthought.	Comprehensive knowledge of the development process provides context for architectural decisions and thereby increase visibility. A top-down design approach may lead to higher visibility, simply due to the flow of designs and documentation.
	ConOps: Mission, System, and FM High-level Requirements Documents (L1/L2)	Mission parameters in the ConOps lead to architectural decisions, and are therefore important to understand. Ground-based, autonomous, or other representations of system performance found in the ConOps provide additional visibility into the architecture. Variability in ConOps representations, such as varying levels of abstraction or detail, differing points of view, and few (tens) of concept requirements traced to many (hundreds) of next-level requirements can impact visibility.	Visibility with respect to most features found in the ConOps depends mostly on detail of documentation, which can vary between documents. The various representations of system performance in the ConOps can also provide further understanding of FM layers/tiers. Use cases and nominal scenarios are described at this high level and are used for capability-based assessment.

Lifecycle Phase	Development Artifact	Architectural Visibility	Analyst Visibility
Concept	Heritage studies	Choosing to re-use an architecture/design/software segment has no different effect on architectural visibility than creating a new one of equal complexity, other than the chance that some developers/analysts may already have familiarity with it.	If documentation for re-used architectures/designs/software is provided (either from its heritage use or re-written versions), this will likely not pose a visibility problem. If not, then heritage segments can become black boxes that analysts have little to no information about and must dig further to understand.
Requirements	Functional Requirements Specifications (L5/L6), Interface Control Documents, Interface Requirements Specs	Requirements specify how software limits are employed to detect and guard against failure and recover from anomalous events and conditions. Missing and low-quality requirements or lack of traceability impede visibility.	Requirements for Fault Management Detection, Correction and Responsive behaviors are not always explicitly indicated. Requirements decomposition leads to multiple levels of abstractions. Establishing appropriate Fault Management details for each level is necessary. Requirement specifications serve as a further description of the architecture and hierarchy of the FM system, and how it is intended to operate.
Design	Physical and Functional FM Diagrams	Monitors are usually arranged and organized logically in the physical and functional system structures. A large number of dependencies can increase complexity and decrease visibility. If a system handles certain faults in different ways, it adds an additional layer of complexity that can challenge understanding.	The relationships and interfaces of FM systems are usually well-documented and understood by analysts. Instances where faults are handled differently are primarily hardware faults that trigger basic responses like redundant component swaps. These cases are usually documented in subsystem documents, but may be left out of system-level monitor/response lists because they happen on a low level, perhaps not even extending out of the hardware layer. An understanding of the physical components of the system is necessary in order to gauge whether appropriate monitors are defined.
	Monitor/Response lists	More monitors and responses generally increase the complexity of the system. Tiered responses can add a degree of complexity. Otherwise, low-level decisions about monitor and response logic do not themselves have much impact on visibility, but the way in which monitors/responses and their logic are documented can.	More monitors, responses, and code also increase the effort required for analysts to have the required visibility into the system, especially when monitor/response documentation is distributed. One-to-many relationships can be a challenge to visibility if not documented well. Monitors that are not labeled or otherwise made easily noticeable in documentation can impede visibility. When spread out among many documents in various formats, accumulating a full system list can be difficult. Fault responses that may cause damage to a healthy system are usually examined to ensure that the risk of false detection is low. Tiered responses necessitate analysis to determine that the correct response is being used at any given time, and to ensure that responses reset properly. Concurrent responses present additional work to investigate interactions in order to gain more visibility.
	Redundancy Diagrams and Documentation	Redundancy is the primary way in which fault tolerance requirements are implemented. Functional redundancy is more complex than hardware-identical redundancy, but is often used where more robust fault tolerance is needed.	Documentation of FM redundancy in particular is not typically a problem, since FM redundancy is largely driven by the need to avoid catastrophic hazards. Clear diagrams detailing cross-strapping of redundant hardware components are also very common. Functional redundancy requires more attention, to not only understand what the layers of redundancy are, but how they may interact if used concurrently. Processor swaps and resets are usually examined closely by analysts, since they can be detrimental if they occur at the wrong time.

Lifecycle Phase		Architectural Visibility	Analyst Visibility
Design	FM Design Documentation	These kind of low-level decisions do not themselves have much impact on visibility, but the way in which they are documented can.	The more complex the algorithms are for FM procedures, such as persistence, response decision-making, and prognostics, the more detailed the documentation behind them needs to be. Clear documentation of inhibit mechanisms and isolation zones is required for full system comprehension. Typically, this information is not absent from documentation, but may be distributed among various documents or subsystems.
	FM PDR/CDR Materials RFAs/RIDs	Often contain diagrams or other representations that did not exist prior. Design synthesis, design definition overview. RFAs/RIDs identify potential limitations to architecture and design.	May contain material that helps bridge the gap between concept/requirements and design. Generally a good place to start for overview of FM system. However, limited by quality of review, and is only a snapshot in time.
	C&DH Documentation FSW Interface Documentation	Defines system data layer that will drive software data layer.	The communication layer presents its own visibility challenges. Understanding how and what nodes communicate is important to fully understanding FM functions. The relationship between the FM/Autonomy engine and monitors is also important to understand.
Implementation	Source Code	More source code generally increases the complexity of the system. Factors affecting complexity include multitasking, inter-process communication, amount of auto-coding (and source of auto-coding), reuse, COTS.	Solid designs and well-written requirements enable code visibility. Language and code structures used can also impact the understandability of the software without clear supporting documentation. Complexity with items like multitasking or complex inter-module communications complicate code visibility, even when strong requirements and designs exist.
	Fault Response Sequences & FM Logic (within or separate from source code, potentially table-driven)	Typically, using a highly modifiable, centralized, table-driven implementation leads to higher visibility, because information is necessarily aggregated and kept in one place. Architecture visibility for response sequences may be difficult if the sequences are not part of the FSW and are uploaded during flight.	Use of tables is often tied to patterns in implementation. Once patterns are understood, visibility into the code is simplified, using identification of the pattern in the code. This is contrasted with distributed architectures which have both a lesser degree of modifiability as well as variations in code implementation and associated requirements or design documentation. This can cause visibility problems for analysts who need to search through documentation to form a complete understanding of all the monitors and responses. In-flight changes may be out of scope for IV&V and SA.
Test	Test Plan	N/A	Review of test plans affords analysts additional insight into the types of testing that may be expected once test products are released. Single-tier testing vs. multi-tier testing, testing with simulations vs. testing with real hardware, etc.
	Test Cases	N/A	Despite its criticality, FM can be difficult to test thoroughly, since introducing errors may cause damage to a system if actual flight hardware is used in the test environment. Scenarios that the developer does not test due to this, or time constraints, are candidates for independent simulation. Additionally, scenarios involving combinations of faults or failures are usually not tested, and comprise a far larger state space than nominal or single-fault cases. Additionally, independent testing can give analysts dynamic experience with the software that increases their knowledge of how it works, which can benefit many other analysis activities.
	Test Results	N/A	Results files offer the actual evidence that the FM capabilities are working as intended.

Exhibit 11: FM Visibility Matrix

NASA IV&V Methods
The latest versions of all NASA IV&V Methods are available through COMPASS (http://compass.ivv.nasa.gov/) or by request.
M-2 Validate Requirements by Inspecting Against Quality Criteria and System/Software Background Artifacts
Method for tool-supported manual inspection of a set of requirements to assess and document the degree to which they individually and collectively exhibit desired quality attributes (Unambiguous, Verifiable, Consistent, Correct, Complete, Design Independent, and Feasible). Use documents that inform the validation target to ensure requirements are complete and correct.
M-3 Validate Requirements by Inspecting Bidirectional Traces
Method for tool-supported manual inspection of a set of requirements to assess and document the degree to which they adequately specify a logical decomposition of the parent requirements, and any functional allocations identified by the developer. This method addresses the integrity of the requirements structure, and identifies faults in correctness, completeness, consistency, and bi-directional tracing of parent to child requirements.
M-4 Verify Requirement Implementation in Source Code/Script and Design by Inspecting Traces
Method uses manual bi-directional tracing of in-scope requirements to developer-supplied software artifacts to detect defects in requirements, design, and/or code/script artifacts.
M-9 Verify Software Code Quality using Static Analysis Tools
This method applies one or more static code analysis tools to ensure the source code is free of impacting code defects, syntax errors and other code deficiencies, including (but not restricted to): - buffer overflows - security vulnerabilities - null pointer dereferences - infinite loops - unused code - coding standard violations The static code analysis tool(s), configured to exclude undesired warning types, generate candidate issues from a build of code, which are then to be manually reviewed to determine the final set of reportable flaws in the build.
M-14 Verify Software Behavior for Off-Nominal Conditions using Independent Testing
This method provides an approach for testing software behavior for IV&V Q2 (software will not do what it is not supposed to do) and Q3 (software behaves adequately under adverse conditions). Test scripts are independently created and executed within the IV&V Test environment.
M-17 Validate Software Architecture by Inspecting Traces to Essential Properties
Method supports a manual evaluation of software architecture by developing a set of essential properties against which elements of the software architecture may be compared. Method detects defects in the software architecture (inadequate coverage of essential properties, missing capabilities, inessential capabilities), and supports regression testing as developer-provided software architecture artifacts mature.
M-37 End-to-End Fault Management Verification through Database Development and Analysis
Method applies to the development of a System FM Database that captures relationships and behaviors to aid in the analysis of FM Systems in large distributed systems. A series of incremental databases are built, maintained, and integrated to derive a total system perspective. The database is an extension of the SMART/AWB [Analyst Work Bench] traceability database where FM Requirements are identified along with their drivers (i.e. parent requirements, Failure Mode and Effect Analysis, and Fault Tree Analysis). In addition, the database is further developed to include an "Event Network" that provides a quasi-dynamic (i.e. executable) abstraction of the system. Queries are used to produce scenarios where interactions are more complex, and potential resource conflicts are likely. Manual analysis is then used to determine the validity of such scenarios and uncover defects. For scenarios that are too complex or time-dependent to analyze manually, test procedures are developed for execution by either the developer or IV&V. Primarily, the method is intended to reduce the set of large or infinite scenarios for analysis/test to a reduced set that either has errors identified, or has a higher likelihood of error.
M-39 Verify Software Design by Inspecting Traces to Requirements and Software Architecture
Method supports manual evaluation of the integrity of the software design to ensure that all requirements are represented in the appropriate elements of the design and that the design does not introduce capability that is not required, and to identify defects in its satisfaction of the software architecture and validated software requirements. Software design documentation is also evaluated to ensure that the design provides the required capability (meeting software architecture and software requirements), is able to reliably meet user needs, and is sufficiently stable to proceed with implementation, and to identify defects in consistency, ambiguity, correctness, completeness, and testability.

M-57 Verify System Software Safety by Comparing Concept Documentation, Requirements, Testing, Design and Code with Hazard Analysis Documentation to Establish a Safety Case, Across the Software Development Life Cycle
A safety-case is a structured argument which provides evidence sufficient to make a conclusion about the "safety of a system" or about the focus of the safety case. The intent of this analysis is to provide a well-reasoned evaluation of the factors that contribute to the safety of the system as it relates to software. This method provides a systematic approach to analyze (and document) the software safety of a system. This analysis can be done on the system as a whole, or on a targeted (focused) part of the system (as determined by risk/criticality assessment). A Safety Case may also be performed on a specific behavior or function.
M-59 Verify Interface Implementation in Software by Simulated Dynamic Testing to Demonstrate Successful Software Component Integration
Method supports analysis of interface implementation by exercising interface components in a test environment engineered to verify/validate that software components integrate properly with hardware elements (physical or simulated) of the system under study. Defects discovered by this Method include code flaws, mismatches between expected hardware function and software implementation of the function, and timing or other performance constraints.
M-60 Verify Software Capabilities through Independent Testing of Operational Scenarios
Having test environments available aid in determining the operational readiness of software. If the test environment has the proper fidelity, operational day-in-the-life scenarios can be executed which can increase confidence in the operational readiness.
M-64 Verify Performance Requirements Implementation via Simulated Dynamic Testing to Stress Software Boundaries and Limitations
This method uses performance requirements to generate test cases (aka scenarios) that will stress the system and its interfaces under test by exercising the boundaries and limits described by the performance requirements. Performance requirements are any requirements which are described in terms of quantity, quality, coverage, timeliness or readiness. These requirements are not limited to those listed under "Performance Requirements" in requirements specifications for the system under test. Additionally, stress tests should exercise the system by creating "stressful" conditions by exceeding operating constraints and design margins. Stress testing should demonstrate the robustness and recoverability of the system under test. The execution of stress tests should take place upon or near the final release of software, after IV&V has been performed on all phases of the system. This will ensure that the tests are being executed on the most mature, complete and error free revision of the software. The goal of stress testing is to identify errors in software that can remain hidden from standard or traditional unit and acceptance testing, including any fault conditions that can cause hazards. It is not until the system is subjected to out-of-the-ordinary conditions that errors arise or system performance degrades below operational levels. By analyzing the results of stress testing the IV&V team will be able to identify the root cause in these undesired software system behaviors.
M-68 Validate Key Capabilities via Dynamic Testing against High Risk Scenarios to Reduce Risk of Operations
Develop high risk scenarios, especially those that include off-nominal and fault scenarios and those that require cooperation of multiple CSCIs, CSCs, and otherwise cross integration boundaries. Provide assurance that capabilities needed to correctly operate as expected are complete and correct by executing the scenarios in a validated test bed.
M-103 Verify and Validate Requirement Implementation using Flow Diagrams to Uncover Missing, Conflicting, or Unnecessary Behavior
Method uses Flow Diagrams to analyze software implementation of requirements to ensure the correct and complete implementation of requirements on a system level as well as an atomic level. (Level is dependent upon the abstraction in the modeling chosen by the analyst as well as the available level of artifacts being targeted). Further, the method is applied to the source code that is not specified by requirements or not specified directly.
M-108 Validate Software Requirements meet dependability criteria and control identified hazards by inspection of traces to identified dependability attributes and safety hazards
Method uses manual tracing of in-scope dependability and safety requirements to dependability attributes and safety hazards. Utilizes a multi-level traceability strategy to link software related system requirements, PBRA mission capabilities, dependability properties, and software requirements to determine that the software requirements are complete for software contributing or controlling factors as related to the dependability attributes for each mission capability and safety hazards. Each dependability attribute describes the software factors that relate to dependability properties for a given capability. The intent of this analysis is to provide a well-reasoned evaluation of the factors that contribute to the dependability of the system as it relates to software. The dependability attributes are identified from developer system level requirements that are correlated to mission defined capabilities and dependability properties. Analysis can be done on the system as a whole, or on a targeted part of the system as determined by risk/criticality assessment.
M-109 Verify System Software Safety Documentation Identifies all known software based hazard causes and controls by inspection of Adverse Conditions/Failure Modes
This method provides a systematic approach to analyze (and document) the software safety of a system by identifying safety hazards where software is a cause or control of a safety hazard. This analysis can be done on the system as a whole, or on a targeted (focused) part of the system (as determined by risk/criticality assessment). Ensure all known hazards that threaten the safety of the system (as they relate to software) are properly understood and well-documented.

Exhibit 12: IV&V Methods for FM Analysis

FM VISIBILITY TRENDS

Though the FM Visibility Matrix TR is a valuable stand-alone product, additional analysis was performed on the dataset to look for significant trends in analyst or architectural visibility. First, in populating the matrix with visibility data, it became obvious that availability and quality of development artifacts have a large impact on independent analysis. An independent analyst's understanding of the system is highly dependent upon these artifacts, since the only other way for an analyst to learn the system is by interaction with the development team. If an artifact is not complete or accessible, further investigation is necessary. Analysts sometimes develop an informal representation, or model, of the artifact, and have it verified by the development project, in order to use as a reference in later analysis. Many times, however, the artifacts exist, but the desired system details are not organized in a way that is conducive to analysis. In this case, techniques are used to mine the artifacts for the relevant data and organize the information into a comprehensive system view. This is particularly difficult with larger or more distributed FM architectures, often described over multiple subsystem design documents, and is compounded with many developers or large development teams. These projects may have all the required FM documentation with high quality information, but no standardization on how information should be organized within documents, which presents additional work for analysts in order to gain a system-wide perspective.

These artifact challenges are of particular concern considering the rise in use of Agile software development methods, in which documentation is often a lower priority and created later in development than typically done in a traditional waterfall development process. Additionally, artifacts often exist in a more fluid state, allowing for flexibility and modifications to the baseline, but also creating challenges with configuration management and with system-wide comprehension and communication of architectural decisions and rationale for changes. IV&V and other SA practitioners need to develop new analysis strategies in order to provide the same level of assurance for such projects, especially those with safety-critical software components.

Model-based FM may help to alleviate difficulties encountered from complex, scattered FM information by imposing centralization of FM architectural design in a single source of authority. Likewise, manipulation of views, scalability, customization of interfaces, and more automated integration with other enterprise models bring about significant benefits. Other vulnerabilities, however, may be introduced with issues of misrepresentation, model complexity, difficulty in validation of the model, and with keeping the model current in the face of changes. Visibility for some stakeholders may in fact be decreased, depending on the technical expertise and culture within the organization. Innovative approaches to meeting visibility challenges warrant investigation, however, within acceptable levels of risk.

In addition, while occasionally only a small portion of concept documentation undergoes IV&V analysis, all of it is nonetheless very important for developing an understanding of the system and a reference for analysis in later phases. Most techniques that involve concept phase artifacts, like ConOps and planning documents, do not generate issues with those artifacts, but instead populate a TR that is maintained and consistently used to supplement analysis for the remainder of the project. Because this TR is carried through the project lifecycle, this validation is an important step in the analysis process; a strong TR is helpful in avoiding challenges to visibility as the project progresses. It can be challenging to quantify the value of establishing a TR, particularly when it does not produce direct results like issues or risks, but the collected details of results and effectiveness of techniques indicates the proven value of these activities, both in furthering understanding to benefit future analysis, as well as increasing confidence that the software and system is designed and built correctly to meet requirements.

Finally, it was noted that in general, larger, more complex missions tend to be the ones that require new, adaptive or additional techniques or approaches both for analyst visibility and for architectural visibility. This can

be due to a number of factors, such as increased complexity lowering architectural visibility, multiple developers or large development teams leading to varied approaches and less-standardized documentation, high criticality meriting the use of multiple assurance methods, or a broad and unfamiliar state space requiring a new avenue of attack. Regardless of the reason, analysts given the responsibility for providing evidence-based assurance on a large or complex project, particularly one that is unlike anything attempted before, are encouraged to be innovative and adaptive with the techniques they use to provide assurance. Some strategies for assurance provision are discussed in the next section.

FM ASSURANCE STRATEGY TECHNICAL REFERENCE

The full FM Assurance Strategy TR is available from the author by individual request. Information on IV&V assurance techniques was compiled initially by conducting keyword searches in COMPASS, and later expanded through surveys and interviews founded on the IV&V analysis questions (previously outlined in Exhibit 10). Direct requests and conversations with SMEs from the FM COI were held on both formal and informal bases. The goal was to understand the methodologies for providing assurance for several of NASA's most critical, high-profile projects within the IV&V portfolio.

The collection of FM Assurance Objectives and Assurance Conclusions across multiple projects was the main tactic to achieve a view of and appreciate the overall Assurance Strategy for this high-risk area of software. Associated data, such as limitations that were encountered and submitted to the project via issues or risks, and assumptions made in the analysis process were captured along with Assurance Conclusions. The intent was to use these assurance products to gain further insight into the purpose, techniques, and effectiveness of IV&V analysis being performed. The statements identify which aspects of a FM system are most critical for analysis, and the avenues of scrutiny that were taken during each phase of the software lifecycle to provide assurance that the system's software will do what it is supposed to do, will not do what it is not supposed to do, and will respond as expected under adverse conditions, in accordance with the 3Qs. The aggregate set of Assurance Objectives and Conclusions became an interactive TR for future IV&V activities associated with FM, as well as a model for other special domain areas of interest across the Program (for example, the Guidance, Navigation, and Control COI followed suit with a similar procedure).

The FM Assurance Strategy TR was presented in a previous publication[1], so will not be repeated here. The full version of this interactive compilation is hosted on an IV&V Confluence page, where it can be viewed by all analysts and also actively sorted and filtered with a variety of parameters. The statements were organized by lifecycle phase and mapped to the IV&V Technical Framework objectives and 3Qs in order to allow for a more comprehensive picture of how FM analysis fits into the larger context of full IV&V analysis.[4] Off-nominal cases are of particular importance to FM analysis, so it is not surprising to see almost as many Q2- and Q3-related statements as Q1. Additionally, patterns or gaps in Technical Framework use can indicate which framework objectives are most applicable to FM analysis, and which may fall out of scope for this particular subsystem.

The lack of Assurance Statements from Earth Orbiter missions was evident from a brief examination of the FM Assurance Strategy TR. This was determined to be due to the fact that all three Earth Orbiter projects were more limited in scope of IV&V analysis due to being reimbursable and having defined mission objectives that were of a lower criticality by nature. The most recent sets of assurance statements were collected from projects, with the expectation that this TR will continue to grow in the future. This will provide greater representation of all mission domains and software development phases.

CONCLUSION

Research into FM architectures that serve to physically and functionally structure safety- and mission-critical software within the NASA IV&V portfolio has enhanced a TR suite for improved assurance capability. With FM systems ranked high in risk-based assessment of criticality within flight software, the importance of establishing highly competent domain expertise to provide assurance for NASA projects has been emphasized, especially as spaceflight systems continue to increase in complexity. Insight into specific characteristics of FM architectures seen embedded within the overall software systems at NASA IV&V and low-level features summarized by domain-focused surveys has been given within the FM Architectural TR. Earth Orbiters, Human Spaceflight, Deep Space Robotic, and now Launch vehicles are mission types addressed with the SARP FMAE initiative. General architecture descriptions in the Architecture Types section of this paper are a solid, albeit high-level reference for new FM practitioners or those branching into a different domain of FM, and serve as a starting point for coalescing FM architectural styles. Many descriptions, along with benefits and limitations, of current IV&V methods have been outlined within the context of addressing FM Assurance Strategies based on the 3Qs of IV&V analysis. These strategies encapsulate the identification of FM risk, the planning and execution of analysis to determine the role of software in mitigation of risk, and evidence-based conclusions of quality, safety, reliability, and security of flight software and associated FM artifacts. Augmentation of FM mission assurance across the developmental lifecycle, regardless of the method employed, is supported with focus on architectural and analyst visibility, techniques, and drivers in the FM Visibility TR. Lastly, rounding out the TR suite is an interactive aggregation of Assurance Objectives and Conclusions from current and former IV&V projects, collectively bringing value to the Agency in the realization of labor savings as analysts capitalize on deeper understanding of FM architectures, Assurance Strategies, and methods.

Benefits are aimed beyond the IV&V community to those that seek ways to efficiently and effectively provide SA to reduce the FM risk posture of NASA and other space missions. Currently underway is the SARP FMAE effort to extend research in three growth capacities. SLS, the first IV&V project addition in the Launch Vehicle domain, delivers broader coverage of FM architecture domains. SLS utilizes model-based FM design within an Agile-type approach to software development providing an opportunity to investigate how nontraditional processes affect FM architectures and system health management design. Second, a NASA IV&V Program asset, the prototype AC database, is being refined to provide for: increased assurance expedience from the provision of more comprehensive data, more rigorous Q3 analysis from identification of off-nominal scenarios, increased efficiency of analyst workflow and broader test coverage, and greater focus on FM and project areas of vulnerability or high risk. Lastly, collaboration with OSMA and FM experts across the Agency is planned with the goal of knowledge exchange for advancement of FM assurance. The IV&V FM COI has been an invaluable resource throughout the research effort. The members provide data and peer review, and the meetings are a valuable platform to share and discuss FM architectural designs and methods with SMEs with extensive experience across many domains. Utilizing the FM COI in this way enabled a robust dataset and confidence in the resultant TR suite.

As with any software system, there is no one-size-fits-all analysis regimen. Every mission has unique characteristics and objectives, and the Assurance Strategy must reflect the mission. Even the most widely used methods are not always applicable to every project. Analysts need to bring their collective knowledge and experience to the table to decide how best to build and execute an Assurance Strategy. Part of this is recognizing when an architecture or design is similar to a previous mission and planning accordingly; methods used in the past are applicable to the new project, and knowledge gathered in the TR will be carried over. Planning is not always enough, however; analysts must also be prepared to adapt to visibility challenges as they appear. Barriers to visibility can appear in any lifecycle phase, and the analysis that was planned may no longer be as effective as it

could be if the plan is not revisited when these barriers arise. Adaptive assurance is especially important alongside an Agile development project, in which changes and iterations occur relatively quickly.

From research and data analysis described above, architectures investigated were found to fit into one of two broad categories: centralized, found more often among Earth Orbiter missions, and hybrid, more often used in Deep Space Robotic and Human Spaceflight missions. These two styles represent the tradeoff between a single-tier, easy to understand architecture, and a robust, multi-tier system with extended fault tolerance. Developers' previous mission experience can also have a large effect on the architecture used, sometimes independent of the mission domain. Availability and quality of documentation were found to have significant impact on visibility, and, in general, the larger, more complex missions often have more drivers for new or adaptive assurance techniques. The products generated from this initiative build a strong foundation to fill the existing gaps in the FM knowledge domain and are useful across the Agency and beyond.

Communication and knowledge-sharing across assurance projects is extremely important. Despite encountering developers that use different FM ontologies or terms, which threaten to inhibit understanding, analysts perform better with more knowledge about FM architectural design approaches from multiple projects. The scope of FM analysis may vary from project to project, with differences in risk assessments, level of rigor required, or prioritization of Assurance Objectives, but often have similar complexity issues that are reflected in the visibility of the FM architecture, with a design that is distributed among numerous artifacts. With mission heritage and developers' experience as key drivers for FM architectural decisions, building a culture or community, like the FM COI, that values cross-project communication for continual improvement should be a priority. Additional platforms, like enterprise-compatible data repositories or web-based indices of assurance methods, tools, and templates also contribute to this goal. Within this family is where the FM TR suite now resides, promoting the sharing of information both within and between Agency centers, increasing value for SA. Results from the FMAE initiative will be incorporated into the NASA Fault Management Handbook, providing dissemination across NASA and the entire spaceflight software community. In conclusion, this FMAE research is already showing successful progress in the accomplishment of goals and in providing significant benefit to the FM and SA community.

---

[1] Savarino, S., Fitz, R., Fesq, L., & Whitman, G. (2015, Apr. 17). *Fault Management Architectures and the Challenges of Providing Software Assurance.* Proceedings of 31st Space Symposium. Retrieved from http://ntrs.nasa.gov/search.jsp?R=20150005781.

[2] *Fault Management Handbook*. (2012, Apr. 2). Draft 2. National Aeronautics and Space Administration. Retrieved from http://www.nasa.gov/pdf/636372main_NASA-HDBK-1002_Draft.pdf.

[3] Asbury, Michael. (2016, Mar. 21) *NASA IV&V Facility.* National Aeronautics and Space Administration. Retrieved from http://www.nasa.gov/centers/ivv/home/index.html.

[4] *Independent Verification and Validation Technical Framework.* Version P. (2012, June 6). National Aeronautics and Space Administration. Retrieved from http://www.nasa.gov/sites/default/files/atoms/files/ivv_09-1_-_ver_p.pdf.

[5] Barltrop, K. (2009, Mar. 5). Managing Fault Protection Software Complexity for Deep Space Missions. *NASA Study on Flight Software Complexity* (Appendix F). Retrieved from https://www.nasa.gov/pdf/418878main_FSWC_Final_Report.pdf.

[6] Schmidt, R. F. (2013). *Software Engineering: Architecture-Driven Software Development*. Morgan Kaufmann Publishers. (Books 24x7 version).

[7] Albin, S. T. (2003). *The Art of Software Architecture: Design Methods and Techniques*. John Wiley & Sons. (Books 24x7 version).

[8] Gorton, I. (2011). *Essential Software Architecture*. Second Edition. Springer. (Books 24x7 version).

[9] Taylor, R. N., Medvidovic, N., & Dashofy, E. (2010). *Software Architecture: Foundations, Theory, and Practice*. John Wiley & Sons. (Books 24x7 version).

[10] Rustick, J. P., Wisniewski, M. J., & Termohlen, D. (2009). *Fault Management for Complex Space Systems: Assessing the Use of Rule-Based Approaches*. Retrieved from enu.kz/repository/2009/AIAA-2009-2030.pdf.

[11] Conway, M. E. (1968). *How Do Committees Invent?* Retrieved from http://www.melconway.com/Home/Committees_Paper.html.

[12] Dvorak, D. L. (2009, Mar. 5). *NASA Study on Flight Software Complexity*, 45. Retrieved from https://www.nasa.gov/pdf/418878main_FSWC_Final_Report.pdf.

[13] Strege, S. (2015, Jan. 28). *Core Flight System: A paradigm shift in flight software development*. Retrieved from https://cfs.gsfc.nasa.gov.

[14] Schmidt, P., Brueck, D., Rokey, M., & Pope, J. (2012). *Independent Assessment of Two NASA Fault Management Software Architectures*. Retrieved from www.nasa.gov/pdf/637607main_day_1-phillip_schmidt.pdf.