SOFTWARE CONFIGURATION MANAGEMENT FOR SAFETY-RELATED APPLICATIONS IN SPACE SYSTEMS: EXTENDING THE APPLICATION OF THE USAF 8-STEP METHOD

C.W. Johnson

Department of Computing Science, University of Glasgow, Scotland.
http://www.dcs.gla.ac.uk/~johnson, Email: Johnson@dcs.gla.ac.uk
+44 (0)141 330 6053 (Tel.), +44 41 330 4913 (Fax).

ABSTRACT

Configuration management ensures that the requirements and constraints, identified in previous stages of development, are preserved throughout the design, implementation and operation of complex systems. Space-related, software systems pose particular problems because, for instance, it can be hard to determine what code is actually running on a platform as successive updates are performed over many months of remote operation. It is, therefore, important we learn as much as possible from previous mishaps that have involved configuration management; given that software continues to play a critical role in the safety of many space missions. The following pages extend the US Air Force's 8-Step Method to identify lessons learned from space related incidents. This approach builds on Boyd's OODA (Observe, Orient, Decide and Act) Loop and provides a common framework for the analysis of these complex incidents. It is important to stress that the application of an existing general approach to problem solving, rather than the development of a specific approach for configuration management, is intended to reduce training costs and to increase the value added from existing investments in the use of the 8-Step Method. Many specialised software engineering techniques are not used because they cannot easily be applied within the financial limits and deadlines that constrain most space programmes. The closing sections of this paper identify areas for further work; in particular, we stress the importance of links with recent European Space Agency problem solving techniques that support the early-stage development of long duration space missions.

1. Introduction

Configuration management ensures that requirements and constraints, identified in previous stages of development, are preserved through subsequent modifications. Within this general description there are a range of more specific concerns - for example, one aspect of configuration management focuses on the maintenance of well defined interfaces between system components (Johnson et al, 2009). More broadly, configuration management consists of processes that are intended to ensure the consistency of a product with both functional and non-functional requirements throughout the development, operational and decommissioning lifecycles.

This paper focuses on configuration management for complex space-related, software systems. Software poses particular issues because, for instance, it can be far hard to determine what code is actually running on a remote platform. This might seem trivial; however, the general problems can be illustrated by the Solar and Helioscopic (SOHO) observatory. The company that developed the satellite had a mission simulator. NASA ran ground control and maintained a second simulator. ESA coordinated development and ran a third. The diagnosis and response to the subsequent mission interruption was complicated because there were inconsistencies between the code of the simulators and the software that was installed on the satellite.

The importance of software configuration management for the success and safety of future space missions should not be underestimated. Financial constraints imply the need for multi-agency missions where different consortia must pool resources during the development and operation of complex platforms NASA/ESA, 2009). This creates complexity; it can be hard to identify the agency that is responsible for controlling each aspect of the shared platform during all phases of the mission. Given the increasing importance of software as an enabling technology, it is critical that we ensure the integrity of our code in order to support the success of future missions. This paper, therefore, describes how the US Air Force's 8-Step Method can support software configuration management (Fletcher et al, 2009). The 8 Step Problem Solving Model is a standard process based on Boyd's OODA (Observe, Orient, Decide and Act) Loop:

1. Clarify and Validate the Problem;
2. Break Down the Problem and Identify Performance Gaps;
3. Set Improvement Target;
4. Determine Causes and Contributory Factors;
5. Develop Countermeasures;

6. See Countermeasures Through;
7. Confirm Results and Process;
8. Standardize Successful Processes.

Previous joint work, between NASA, the USAF and Glasgow University, has demonstrated that this approach can be used to identify configuration management problems in a number of previous space missions (Johnson et al, 2009, Fletcher et al, 2009). This previous work looked at systems safety issues and did not focus directly on software engineering. The application of an existing general approach to problem solving, rather than the development of a specific approach for configuration management, is intended to reduce training costs and to increase the value added from existing investments in the use of the 8-Step Method. Many specialised software engineering techniques are not used because they cannot easily be applied within the financial limits and deadlines that constrain most space programmes.

A number of significant problems remain in refining the high-level activities of the 8-Step Method into the detailed configuration management processes that might support complex software engineering tasks in space missions. For example, there are many difficulties in 'seeing countermeasures through' when different organisations may be using a range of different development practices and even different tool sets to support the design and operation of complex code. Similarly, it can be hard to determine the specific role played by configuration management activities in phases 7 and 8 of the process when a host of other factors also contribute to successful outcomes.

2. The SPIRIT Case Study

The following pages use a mishap involving the Spirit Mars Exploration Rover (MER) to illustrate the application of the USAF process to software configuration management for space applications (NASA, 2004, Reeves et al, 2004). Spirit landed on Mars at 04:35 Ground UTC on 4th January 2004. This was three weeks before its twin, Opportunity, completed Entry, Descent and Landing on the other side of the planet. Spirit's original mission was scheduled to last around 90-solar days on Mars. This paper focuses on a mission interruption that started on Sol 18 (21st January 2004). Spirit lost the ability to execute any task that requested memory from its flight computer. The rover operated in a degraded mode until Sol 33 (6th February), when normal operations were restored.

The Sol 18 interruption started at around 09:00 (local solar time), when a direct to Earth, High-Gain Antenna communications session started as planned. By 09:11 uplink errors were detected and the signal was unexpectedly lost around 09:16. This was some 14 minutes before the scheduled end of the transmission. By 11:20 it was decided to command a priority communication session using the high-gain antenna. No response was detected from the MER.

Problems continued into Sol 19 (22nd January). The UHF communication session between Spirit and the Mars Global Surveyor satellite did not begin at the scheduled time nor did it last for the anticipated duration. Instead, a 'PsuedoNoise' code was received for a little more than two minutes. By 04:00 no signal or data had been received during the scheduled UHF session nor was contact established during the 09:00 direct to Earth, High-Gain Antenna communications session. This should have triggered a system response on the rover that would have scheduled further direct to earth communications using the low gain antenna but, as before, no signal was observed by 11:00. Efforts to restore communication continued and eventually by 14:40 a 'minimal communication' beep was seen. However, no further data or signals were received during the scheduled UHF communication with the Odyssey spacecraft. No transmissions were detected during a commanded low gain direct to earth communication.

The ground teams concluded that a system level fault had occurred on or before Sol 19. This had degraded the MER's communications system and had impaired some of their ability to command the vehicle. The ground teams were concerned that this was caused by a hardware failure that could potentially end the mission. However, there was considerable uncertainty and commands were issued to trigger the transmission of diagnostic data from the MER. By Sol 20, several short low-bit rate messages were received via an X band link to the orbiting Mars Odyssey. Data suggested that the rover was continuing to process information rather than entering into a sleep mode. Both commanded and autonomous shutdowns were failing and the vehicle probably had not closed down in a while. This created a concern that Spirit would exhaust its finite battery power or risk overheating. The MER continued to ignore requests to shut down.

Attention began to focus on the possibility that the rover had entered a reboot loop. On start-up, the MER was designed to execute a number of initial commands that helped to create the operating environment in which it was possible to run subsequent programs from Random Access Memory (RAM). Before these initial commands could be completed, it was hypothesized, that a fault caused Spirit to begin another reboot operation. This cycle would then continue so that the rover was never able to execute the code in RAM. It was argued that the problem could have been caused by a hardware fault or it might be due to faulty code from the boot sectors

stored in Electronically Erasable Programmable Read Only Memory (EEPROM) flash memory.

The designers of the MER had anticipated such a contingency and created a mechanism whereby commands could be sent to the rover so that it would complete the reboot cycle without attempting to access the EEPROM. By Sol 21 (January 24), the ground team was confident that the problem centered on the rover's flash memory system. The amended reboot command that avoided the EEPROM references appeared to have succeeded.

More detailed hypotheses formed around the file management software. Incorrect configuration parameters were set for two VxWorks operating system software modules that controlled the storage of files in the heap area of system memory. The initial reboot was triggered by the creation of a large number of files in flash memory. These were created when the rover began to calibrate its instrumentation. The calibration files were in addition to the large amount of data that had been stored during the cruise phase of the mission. However, much of the information gathered before EDL was no longer needed.

The reboot was triggered because a parameter in the dosFsLib module could temporarily assign 'overflow' data to system memory. This storage was usually allocated to the heap but was itself relatively limited and so it too quickly became exhausted. These problems were compounded by another parameter, this time in the memPartLib module, which was incorrectly set to suspend any task using memory when no additional memory was available. This task suspension forced the reset of the flight computer. The associated NASA 'lessons learned' entry notes that this was 'never supposed to occur' (NASA, 2004). Other side-effects included memory corruption, inability to turn the vehicle off as a result of task deadlock and the repeating system resets described above.

The total size of the file system structure was determined not by the number of current files but by the maximum number of files that has ever existed at any point in the mission. Rebooting the system only deleted data in system memory. It did not create space in the non-volatile EEPROM nor did it reset the maximum file structure parameters, mentioned above. This set up the cycle that was observed by the ground teams as each successive reboot failed to address the cause of the problem. The effects of overburdened flash and system memory were not recognized nor tested during system level ground testing.

The Mission Operations teams identified a number of potential solutions. These were assessed in terms of their potential risk as well as their ability to address the causes, mentioned above. It was decided that operations could best be restored by manually reallocating system memory. Steps were also taken to delete unnecessary directories and files. Over time, it was possible to create a new file system. However, a plan to rewrite elements of the dosFsLib and memPartLib modules was rejected. Major revisions to the flight software were considered too risky. Instead, it was decided that changes in operation would be introduced to conduct periodic checks on the build-up of files within the MER flash storage.

The following sections use the 8-Step method to provide a framework for the detailed analysis of the configuration management issues that contributed to this mission interruption. Before doing this, however, it is important to identify the causes that were identified in the NASA (2004) 'lessons learned' review. The underlying causes of the interruption were associated with tight deadlines across the MER software development schedule. There had also been a continuous reprioritization of activities where attention was focused and refocused on a small number of high priority concerns. This shift of attention between a few major issues helped to obscure some of the apparently more minor concerns, including configuration management for the file structures. In addition, the review identified a number of generic recommendations from this incident. These can be summarized as follows:

1. 'Enforce the project-specific design guidelines for COTS software, as well as for NASA-developed software. Assure that the flight software development team reviews the basic logic and functions of commercial off-the-shelf (COTS) software, with briefings and participation by the vendor.

2. Verify assumptions regarding the expected behavior of software modules. Do not use a module without detailed peer review, and assure that all design and test issues are addressed.

3. Where the software development schedule forestalls completion of lower priority action items, maintain a list of incomplete items that require resolution before final configuration of the flight software.

4. Place high priority on completing tests to verify the execution of flight software internal functions.

5. Early in the software development process, create a comprehensive suite of tests and automated analysis tools. Ensure that reporting flight computer related resource usage is included.

6. Ensure that the flight software downlinks data on system resources (such as the free system memory) so that the actual and expected behavior of the system can be compared.

7. For future missions, implement a more robust version of the dosFsLib module, and/or use a different type of file system and a less complex directory structure'.

3. Applying the USAF 8-Step Model

This section applies the US Air Force's 8 Step Problem Solving Model to identify software configuration management lessons from the MER interruption. The intention is not to derive new insights; this would be difficult given that NASA conducted an extensive review after the events of Sol 18-21. Instead, the intention is to demonstrate that the 8-Step framework can be used to guide future investigations of software configuration mishaps and also to determine whether the application of this approach might alter the emphasis of the official recommendations.

The first process in the 8-step model is to ***'Clarify and Validate the Problem'.*** The ground teams' initial investigations helped to identify the symptoms of the interruption. Procedural 'work arounds' were introduced when it was decided too risky to update the underlying file management software. In addition, longer-term investigations began identified the underlying development problems that contributed to the mission interruption. It was argued that NASA independent verification and validation (IV&V) activities could have identified the configuration management issues during development (Costello, 2004). Many aspects of the interruption stemmed from the ways in which development teams had set up and used a 'Commercial off the Shelf' (COTS) operating system. This created particular problems for IV&V. Independent analysts must focus on the interface between bespoke software and the COTS applications because they, typically, do not have access to the commercial source code.

The second stage of the 8-step model ***'Breaks down the problem and identifies performance gaps'.*** In order to understand the detailed causes of the MER mission interruption, it is important to understand the role of NASA's IV&V support within the Safety and Mission Assurance Office. As the name suggests, a core competency is to provide independent input to the processes that establish mission assurance. The work of the group is, therefore, particularly important in terms of software configuration management. They help to ensure that constraints and requirements are satisfied from design through development to operations. The process of commissioning NASA's IV&V involvement in software development can be summarised as follows:

1. NASA Headquarters publishes a prioritised list of missions that are to receive IV&V resources for the next fiscal year.

2. The IV&V Facility prepares an initial cost estimate for their involvement in the missions from 1 to assist in drafting the overall budget.

3. The IV&V Facility initiates a planning and scoping effort for each new mission to throughout the lifecycle based on software criticality and risk assessment following NASA NPD 8730.4A.

4. The IV&V Facility develops a breakdown of the work associated with each project and this is then subjected to an internal review.

5. The project's Governing Program Management Council (GPMC) is responsible for overseeing the project's IV&V approach.

6. Differences between the project's approach and the IV&V Facility recommendation are resolved by the project's GPMC.

7. The IV&V Facility determines the distribution of resources between the project's development sites and the IV&V Facility.

8. IV&V covers all phases of the software development lifecycle.

9. IV&V status is reported to Project Offices, Office of Safety and Mission Assurance, and GPMCs.

As we shall see, stages 6 and 7 of this programme are not as straightforward as they might seem. For now it is sufficient to observe that a NASA press release issued several weeks after the interruption described how:

> "Based on this high level of risk, the IV&V Facility provided full software development lifecycle (SDLC) support to the MER program. This support included requirements analysis, interface, design evaluations, behavioral architecture analysis, code analysis and test analysis of the MER flight software (includes the spacecraft and the rovers). The goal of the effort was to provide the MER project with a detailed technical analysis of the software being developed for the mission that would allow them to make timely and informed decisions about issues and risks in the software… The IV&V Team

analyzed the flight software in concert with the development phase. That is, when the MER development team was doing requirements development the IV&V Team was doing requirements analysis, when the MER development team was doing design, the IV&V Team was doing design analysis et cetera. The largest focus of the IV&V effort, however, was performing code analysis. This effort focused on three primary tasks, performing a LINT analysis, determining code complexities and assessing code changes for impact to functionality… The work of the IV&V Facility improved the chances of mission success for the MER Project. Upon completion of the IV&V effort, the IV&V Team had identified 1018 issues of which 347 were mission critical or mission catastrophic and 6 possible risks to the project that could result in a mission failure". (Costello and Ozburn, 2004)

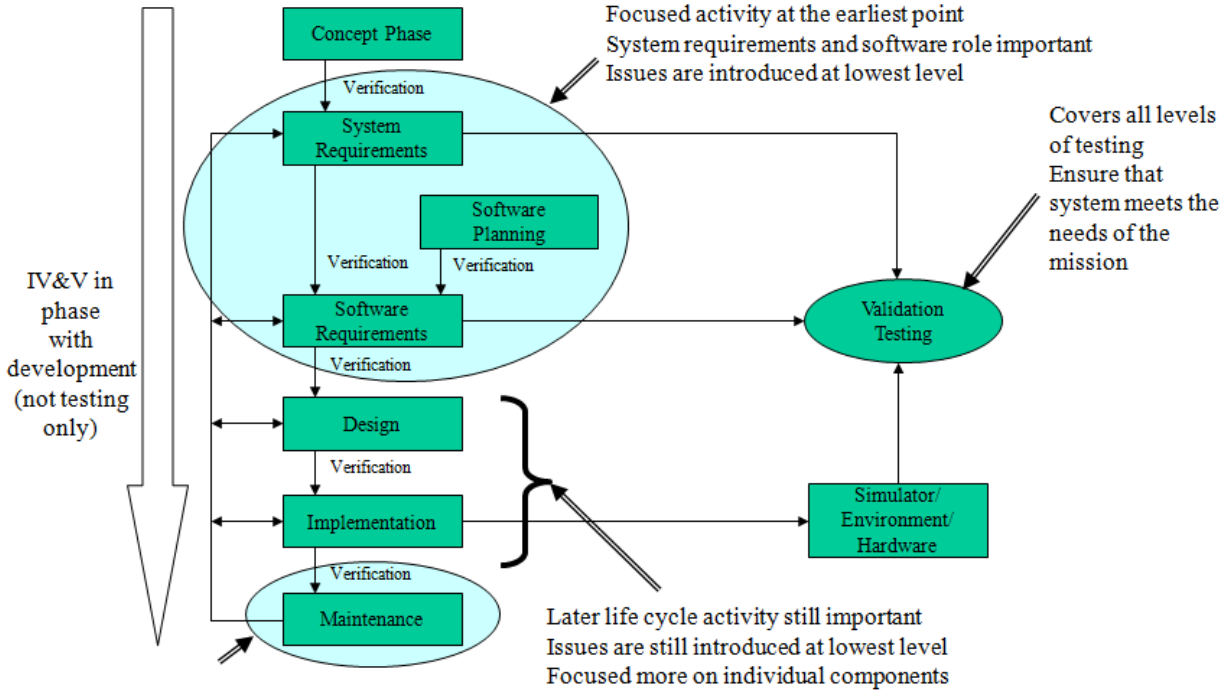

Figure 1: NASA IV&V Support across the Project Lifecycle (Acknowledgement: NASA IV&V Office)

It is difficult to underestimate the IV&V challenges for the MER programme. There were more than 500,000 lines of source code that integrated both bespoke as well as 'Commercial off the Shelf' (COTS) applications. Figure 1 illustrates the overall approach to verification and validation that is promoted within NASA projects. As can be seen, there is a deliberate intention to expand the scope of the work beyond the traditional testing phase. There are many justifications for this. A key limitation with late verification is that bugs tend to be discovered too late in the lifecycle when there is limited time and resources to put the problem right. A further justification for early stage validation is that it can be easier to identify critical requirements for subsequent testing if IV&V teams are also involved in the initial stages of development. The approach advocated in Figure 1 is entirely consistent with the view of configuration management as a process that must be supported from the earliest phases of development.

The initial work to develop an IV&V plan for the MERs began in June 2001. A risk based approach identified that the file system routines were a critical part of the overall system architecture; the consequences of a bug in this code could jeopardize the mission. The initial risk assessments also determined that some elements of the operating system file handling code were highly complex. However, the relative maturity of the systems being used led the IV&V teams to conclude that the likelihood of bugs was comparatively low. The overall criticality assessment associated a low priority with these routines. This analysis corresponds with the third stage in the 8-step method '***Set Improvement Target***' – in this case the mishap suggests that improvements can be made to the processes that were used in identifying IV&V requirements for software configuration management. In order to do this, it is first important to understand the risk assessment processes that were used to analyse file handling routines.

As mentioned previously, the fourth stage in the USAF problem solving method is to '***Determine Causes and Contributory Factors***'. This phase helps to identify the reasons why analysts failed to associate a higher level of criticality with the operating system routines that were implicated in the Sol 18 mishap. One explanation is that project management had to make tough decisions in order to optimise the allocation of scarce resources. The IV&V effort in the MER development process was initially estimated to require around ten fulltime staff. However, less than five people were allocated to these activities within the overall project budget.

The reduction in resources forced the IV&V team to further prioritize their activities; hence it arguably reduced the likelihood that they would have sufficient time to question the original risk assessments that failed to accurately identify the criticality of the file handling code. The goal was now to "cover the MER Flight Software to a reasonable depth so that the IV&V Team could feel comfortable supporting launch and operational readiness reviews for the project" (Costello, 2004). The teams focused on establishing conformance at the level of major systems rather than examining the behavior of individual software components. The lack of resources also limited the range of requirements that were tested during the IV&V lifecycle illustrated in Figure 1. Rather than develop a full range of test suites to demonstrate compliance with a set of agreed requirements, much of the activity focused on specific scenarios without considering whether these cases provided sufficient coverage of the range of operational demands that might be placed on the MER. There was also a lack of transparency when some of the teams could not determine how particular scenarios were identified. The scope of these scenarios remained a critical issue because they drove the analysis of MER configuration management.

The limitations of the IV&V activities were recognized because requirement and test completeness had been identified as the highest risk associated with the project. In other words, the IV&V team was not confident that all software requirements had been identified. In such a situation, it was unlikely that sufficient tests would be conducted to ensure that the code would meet all of the demands this might be placed on it during an eventual mission. The project team asserted that testing was completed. However, it was recognized that portions of the file system code were very complex and that some of these modules were difficult to maintain. The IV&V teams had developed Test Improvement Models (TIMs) for further unit testing on the file system code that accessed system memory. These were still 'open' issues when the code was uploaded for the mission. Further concern focused on the stability of the code; changes continued to be made up until the time at which it was uploaded; around 10% of all code was affected by the final update, including this proportion of the file handling routines (Costello, 2004). These factors all increased the problems associated with configuration management; ensuring that requirements were maintained across the development cycle from design to operation.

It was initially planned to upload the software on the 2nd December. However, some development teams recommended a delay at a readiness review on the 25th November 2003 to allow for further testing, By the 5th December, the IV&V teams continued to report 'significant concerns' for the 'Final Requirements Risk' even though the software had by this time been uploaded. They continued testing to the extent that 'this was possible' and to advocate a thorough review of all existing test results. 'Significant concerns' represented the mid-point in the internal criticality index; "IV&V has a less optimistic view of the requirements discovery than does the project" (Costello, 2004). Specific concerns included system memory usage: risk tracking, issue tracking, code analysis, requirements analysis, test analysis, code complexity and code stability. However, lack of resources, problems with communication between development and verification teams, access to lifecycle artifacts all combined to prevent more sustained analysis of these issues before the Sol-18 interruption. A further review on the 5th December argued that inadequate IV&V potentially threatened the project's scientific returns.

The remaining elements of the 8-step process focus on the organisational response to configuration management problems on the MER. The fifth stage is to '***Develop Countermeasures'.*** In the aftermath of the mishap, it was determined that IV&V should receive sufficient resources to implement the risk based approach advocated in previous sections. This implied sufficient funds for testing to be conducted not just at the systems level but on the individual components associated with 'high risk' operations, in accordance with NASA NPD 8730.4A.

Further countermeasures included the identification of critical artefacts for the software development lifecycle. In previous missions, there had been no obligation for teams to follow a specific software development lifecycle. One consequence was a perceived lack of adequate MER requirements documentation. This had the knock-on effect of undermining the IV&V activities that were intended to establish conformance with those requirements. The IV&V team argued that the development groups had been more focused on addressing each successive project risk that emerged as a result of poorly defined requirements than they were

in looking at the need to develop better requirements for the project software as a whole. The development teams were satisfied that testing mitigated the individual risks created by the dynamic and ill-defined requirements rather than accept the need to clarify those requirements. This created a situation where any gaps in the testing might exposure the MER to risks that stemmed from partial and changing requirements. These problems were compounded by ambitious test schedules so that some studies had not been completed by the time that the code was uploaded. IV&V teams began to commission research into alternate testing techniques that could support projects which did not generate sufficient requirements documentation to support existing approaches. This search for new techniques addressed the symptoms of a deeper problem; it did not focus on the need to develop sufficient requirements for a standardized software development lifecycle based on existing skills within the IV&V facility.

The sixth phase of the 8-Step method focuses on '***Seeing the Countermeasures Through***'. A range of actions were taken in response to the analysis of the Sol-18 interruption. In particular, both development and IV&V teams began to work together more closely in the early stages of the software lifecycle to agree on the overall testing philosophy. The intention was to avoid disagreements, for instance about the importance of requirements documents, that would have implications for all subsequent software development activities. The more pro-active role of the validation and verification groups was intended to encourage greater acceptance of the need for formal software development processes and software engineering practices. However, initiatives to introduce these countermeasures had to be balanced against the dynamic nature of space systems development. For example, it can be hard to draft detailed software requirements when there is still uncertainty over the engineering of hardware platforms.

Further 'countermeasures' were proposed to embed configuration management more directly into the development teams. This relationship was complicated during the early phases of the MER lifecycle because the IV&V teams were unable to work directly with the subcontractor. They could not access the development database that recorded critical information about the test suites that had already been constructed. These issues are not as straightforward as they might appear. There is a difficult balance to be made between engagement with development teams and the need to maintain independence during testing and validation.

The seventh element of the 8-step method is to '***Confirm Results and Process***' and the last stage helps to '***Standardize Successful Processes***'. As a result of the findings described in previous paragraphs, the IV&V teams created an action plan to implement the lessons that were learned from the MER interruption. Many of these lessons reinforced existing policy, such as the doctrine embedded in NPD 8730.4, which required all NASA programs and projects that contain mission or safety critical software to document decisions concerning the use of IV&V. A news release issued in the days after the interruption argued that; "While missions to Mars are high risk endeavors and always will be, NASA is gaining a better understanding of how to control and mitigate those risks with each successful (and unsuccessful) mission. The application of IV&V to mission critical software is just one step on the ladder to mission success. The IV&V Facility is NASA's dedicated organization for ensuring software success throughout the agency. As NASA moves forward into an era of greater human participation in space exploration, the need for IV&V should only increase" (Costello and Ozburn, 2004).

4. Linking ESA TRIZ and the USAF Approach

The USAF 8-Step process is not the only generic problem solving technique that might be used to guide the analysis of complex, software configuration mishaps. The Theory of inventive problem solving (TRIZ) also provides a method for identifying engineering issues and recommending potential solutions (Altshuller, 1999). European Space Agency Contract 21584/08/NL/HE recently completed initial work to extend the application of this approach to support the planning for complex, long-duration missions. This project shared many similar objectives to the USAF approach described in this paper. TRIZ builds on idea that there is a contradiction at the heart of most problems. If we can identify and clarify this conflict of ideas then we will be better placed to suggest appropriate solutions. The underlying philosophy behind TRIZ corresponds closely to stage 1 of the USAF approach 'Clarify and Validate the Problem' and stage 2 'Break down the Problem and Identify Performance Gaps'.

TRIZ categorises each conflict as physical, technical or administrative. They are represented using the cells in a table where each row represents a critical parameter for the success of a mission. Each column then denotes a hazard that might undermine the mission. For instance, at one level of abstraction the rows could refer to critical development activities, such as IV&V. The columns might refer to hazards such as lack of time, insufficient staff, and inadequate integration between teams and so on. Each intersection represents a potential problem scenario. During the later stages of analysis, these cells are annotated with ways of preventing or mitigating a hazard from affecting the corresponding mission parameter. For instance, close adherence to the

provisions of NPD 8730.4A might be proposed as a means of mitigating several of the problems identified in the aftermath of the MER interruption. These stages of the TRIZ method resemble phases 5 'Develop Countermeasures' and 6 'See Countermeasures Through' in the 8-stage process.

The TRIZ approach has recently been developed to generate scenarios that identify the engineering and human factors challenges that future crews might encounter during missions to the Moon and Mars. Future work is required to determine whether we can build on the similarities between this TRIZ approach and the USAF 8-Step problem solving method to develop a unified technique for software configuration management (Whitely et al 2008, 2008a). If this cannot be done then there is a danger that we will develop incompatible approaches to address common problems that undermine the safety of future joint space missions.

5. Conclusions

Configuration management ensures that the requirements and constraints, identified in previous stages of development, are preserved throughout the design, implementation and operation of complex systems. Space-related, software systems pose particular problems because, for instance, it can be hard to determine what code is actually running on a platform as successive updates are performed over many months of remote operation. It is, therefore, important that we learn as much as possible from previous mishaps that have involved configuration management problems; given that software continues to play a critical role in the safety of many space missions. This paper has extended the US Air Force's 8-Step Method to identify lessons learned from previous space related incidents. This approach builds on Boyd's OODA (Observe, Orient, Decide and Act) Loop and provides a common framework for the analysis of these complex incidents. The closing sections of this paper have identified areas for further work; in particular, we have stressed links with recent European Space Agency techniques that support the early-stage development of long duration space missions.

6. References

G. Altshuller, The Innovation Algorithm, TRIZ, Systematic Innovation And Technical Creativity. Technical Innovation Center Inc, Worcester, Massachusetts, USA, 1999.

K. Costello, IV&V Lessons Learned: Mars Exploration Rovers and the Spirit SOL-18 Anomaly: NASA IV&V Involvement, NASA IV&V Facility, Fairmont Virginia, USA, S111/MAPLD, 2004

K. Costello and D. Ozburn, Mars Exploration Rover and Independent Verification and Validation, NASA Independent Verification and Validation Facility, 20th February 2004. Available on: http://www.nasa.gov/centers/ivv/news/news_mars_prt.htm

L.L. Fletcher, J.M. Kaiser, C.W. Johnson and C. Shea, Configuration Management: A Critical Analysis of Applications Using the 8-Step Problem Solving Method. In J.M. Livingston, R. Barnes, D. Swallom and W. Pottraz (eds), Proceedings of the 27th International Conference on Systems Safety, Huntsville, Alabama, USA 2009, International Systems Safety Society, Unionville, VA, USA, 2807-2817, 2009.

C.W. Johnson, The Natural History of Bugs: Using Formal Methods to Analyze Software Related Failures in Space Missions, Lecture Notes in Computer Science, Springer Berlin / Heidelberg, Volume 3582/2005, 2005. Pages 9-25

C.W. Johnson, L.L. Fletcher, C.M. Holloway and C. Shea, Configuration Management as a Common Factor in Space Related Mishaps. In J.M. Livingston, R. Barnes, D. Swallom and W. Pottraz (eds), Proceedings of the 27th International Conference on Systems Safety, Huntsville, Alabama, USA 2009, International Systems Safety Society, Unionville, VA, USA, 3047-3057, 2009.

NASA, Overview of IV&V, NASA Independent Verification and Validation Facility, 2009.

NASA Lesson Number: 1483, MER Spirit Flash Memory Anomaly, Submitted by: M. Boyles and D. Oberhettinger, JPL, 23rd August 2004. Available on: http://www.nasa.gov/offices/oce/llis/1483.html

NASA and ESA Mars Joint Exploration Initiative, NASA Headquarters, Washington, (9/11/09). Accessed March 2010: http://www.nasa.gov/mission_pages/mars/news/mars-20090708.html

G. Reeves, T. Neilson and T. Litwin, Mars Exploration Rover Spirit Vehicle Anomaly Report, Jet Propulsion Laboratory Document No. D-22919, May 12, 2004.

I. Whiteley, O. Bogatyreva, C.W. Johnson, M. Wolff and M. Townend, Human Missions to Mars: Designing decision-support tools for a safety critical environment. In Proceedings of the 3rd International Association for the Advancement of Space Safety (IAASS) Conference, 'Building a safer space together', International Association for the Advancement of Space Safety (IAASS), Rome, Italy, 21-23 October, 2008.

I. Whiteley, O. Bogatyreva, C.W. Johnson, M. Wolff and M. Townend, A Structured Approach to Scenario Generation for the Design of Crew Decision Support Tools. In Proceedings of the 3rd International Association for the Advancement of Space Safety (IAASS) Conference, 'Building a safer space together', International Association for the Advancement of Space Safety (IAASS), Rome, Italy, 21-23 October, 2008.