Multi-Mission End-to-End OBCP Configuration Control

Daniel T. Lakey[1] and Matthias Eiblmaier[2]
SCISYS Deutschland, Darmstadt, Germany

Michel Denis[3] and Bruno Teixeira de Sousa[4]
ESA, Darmstadt, Germany

Roberto Porta [5] and Martin Shaw[6]
VEGA Space, Darmstadt, Germany

and

Tiago Francisco[7]
Telespazio Deutschland, Darmstadt, Germany

ESA's interplanetary missions Rosetta, Mars Express and Venus Express have common on-board software, which includes an On-board Control Procedure (OBCP) execution system. These OBCPs are managed on-board as binary files stored in the Mass Memory and in the central computer RAM, possibly in several copies for redundancy purposes. On ground, the source files and compiled output have historically been stored in a variety of places, from version control tools to shared network drives. Jointly the missions identified a need to consolidate into a unified system, capable of supporting the different approaches to OBCPs as adopted by each mission. In parallel to the configuration management aspects, the development of new OBCPs has migrated to a procedure-based methodology which allows OBCP creation in a manner more consistent with the procedure development interface more familiar to ground controllers . The intention is for spacecraft operations engineers to be able to generate new OBCPs without any particular programming skill. Although developable without software background, the OBCPs represent modifications to spacecraft on-board autonomy with major function and potentially critical impact. Thus the new configuration management system needs to integrate with the procedure generation tool and enable rapid end-to-end development while maintaining strict version control as required by onboard software. This paper presents the requirements derived for such a multi-mission system, the implemented solution common to the ESA planetary missions, and lessons learned.

I. Introduction

As deep space missions, Rosetta, Mars Express and Venus Express experience long one-way light times, making real-time operations impossible and requiring a high level of onboard autonomy to deal with spacecraft anomalies in a timely manner. Restrictions on uplink data rates and on the total number of commands that can be loaded in the on board Mission Time Line (MTL) place constraints on the volume of commands that can be sent to the spacecraft. The onboard software offers solutions to these problems by providing a system of autonomous

[1] Spacecraft Operation Engineer (MEX), HSO-OPM, ESOC, Darmstadt, Germany.
[2] Spacecraft Operation Engineer (VEX), HSO-OPV, ESOC, Darmstadt, Germany.
[3] Spacecraft Operation Manager (MEX), HSO-OPM, ESOC, Darmstadt, Germany.
[4] Spacecraft Operation Engineer (VEX), HSO-OPV, ESOC, Darmstadt, Germany.
[5] Spacecraft Operation Engineer (Rosetta), HSO-OPR, ESOC, Darmstadt, Germany.
[6] Spacecraft Operation Engineer (MEX), HSO-OPM, ESOC, Darmstadt, Germany.
[7] Spacecraft Operation Engineer (VEX), HSO-OPV, ESOC, Darmstadt, Germany.

protections and offering a single high-level command which then manages commanding of lower-level components such as the Solid State Mass Memory unit.

The onboard software can also run onboard control procedures, software routines outside of the core flight program running in a managed, isolated environment allowing them to use commands and manipulate telemetry from the spacecraft database “as if” they were procedures run by an operator, but in an autonomous fashion. With the ability to read the spacecraft telemetry data pool, issue commands and generate events, OBCPs can supplement existing onboard software functions or implement new ones.
As they execute in a “sandbox” protected software environment, OBCPs limit the potential to degrade or otherwise compromise the performance by the On-board Software (OBSW) of its other functions and applications [ECSS-E-ST-70-01C*]. Since the OBCP language is similar to a high level scripting language, and a complete preparation environment with syntax- & consistency-checking can be used, OBCPs are simple to implement and test compared to changes in the OBSW itself.

While they can range from simple sequences of commands to execute, to complex interactions with the data handling system, OBCPs offer the spacecraft operator an opportunity to enhance overall behavior of the operational capabilities of the onboard software with less risk than the traditional OBSM patching techniques, since the original flight software remains untouched. The onboard software “ecosystem” of core Onboard Software, Application Program software and OBCPs should not be changed without due care and attention, which leads to the question of how to best manage the configuration, and how to manage changes within it.

A. Overview of OBCP Status Per Mission

1. OBCP Overview : Rosetta

Rosetta’s OBCP set is at the moment rather static, with no major changes envisioned for the comet phases, when the spacecraft emerges from hibernation to go into orbit around the comet 67-P/Churyumov-Gerasimenko in early 2014. A thorough system review of the Safe Mode process, run before entering hibernation phase, has led eventually to design a new OBCP which is aimed to start spacecraft configuration operations autonomously on board in order to facilitate the recovery from ground. Some other pre-hibernation developments include a thermal cycling OBCP to be used to monitor and maintain the benign thermal environment of the subsystems during the hibernation and another OBCP used to control the Reaction Wheel baseplate heaters during the re-lubrication process run to mitigate the friction torque problem observed on two different wheels.

One possible modification planned for the Comet Phase, concerns the requirements received from Rosetta Science Ground Segment to be able to assign downlink priority levels to science packet stores. In this case the Start Dump OBCP shall be modified to be able to manage the priority levels.

Rosetta has always made extensive use of OBCPs for both instrument and platform operations, with over 100 defined. Onboard memory dumps for a recent EEPROM refreshing exercise were all managed through the use of OBCPs, for example. Such a large set of OBCPs has required careful configuration control from the start, originally using an ESOC-wide solution until it was found not to meet requirements.

2. OBCP Overview: Mars Express

Mars Express has historically used OBCPs very sparingly, as the interpreter was disabled due to another software patch utilizing some of the memory space. Since 2008 the OBCP manager has been available for use, and originally three OBCPs were developed for routine use. These were to offer a simple way of setting the onboard telemetry mode, and to re-route the onboard storage of packets on a daily basis[1]. There was enough margin in the uplink opportunities and the onboard mission timeline to not require further reductions of the command volume via additional OBCPs. The spacecraft remained in a very stable state and while onboard autonomy enhancements were proposed and some had been implemented and tested, no others had yet been loaded onboard.

This changed dramatically in August 2011 when the solid state mass memory (SSMM[2]) suffered a series of severe anomalies, that ultimately resulted in a suspension of science operations. An alternative operations concept was quickly devised that worked around the continuing SSMM anomalies and quickly returned the mission to 100% science, but at the cost of no longer being able to use the 3000-command long MTL. A “short” MTL of 117 TCs remained available and is used to run the mission, but nominal operations required too many commands to be able to fit in such a reduced space. This lead to a dramatic increase in the number of OBCPs, which ranged from transmitter operations to automation of observations. Operations that required 50 command are now achieved through one

* European Cooperation for Space Standardization – On-board Control Procedures, available from http://www.ecss.nl/

command to start the relevant OBCP. To deal with the continuing SSMM anomalies, further OBCPs are planned to enhance the error handling, to reduce interruptions from potentially days to a few seconds[3] .

This veritable explosion of OBCPs (greater than tenfold increase) fortuitously occurred after the configuration management question had been dealt with, in the manner explained in this paper

3. OBCP Overview: Venus Express

VEX uses OBCPs on a daily basis for routine operations. The operations are dealing with the TT&C, Data Handling System as well as for some payloads (the science instruments SPICAV and VERA).

The following operations are handled via OBCPs:

- The activation/deactivation of the transmitters before the beginning and the end of a pass
- Switching between the A and B transmitters
- Managing the activation of the Ultra Stable Oscillator (USO) on VERA during radio science passes
- Stopping and starting of the SSMM dump and Enabling/Disabling routing of housekeeping data to the real-time downlink channel
- Swapping storage of platform data between packet stores on alternating days
- Controlling the Shutter of the SPICAV ultraviolet instrument

All the above mentioned OBCPs were introduced in the early phase of the mission and since they are working as expected there are no changes planned in the near future. However in the future there may be a need to enhance autonomous FDIR protections and work-around any persistent spacecraft anomalies (as with Mars Express). During the Aero-braking phase at Venus, proposed for 2015, OBCPs may also be very useful to manage the transition into and out of breaking mode or to speed up the recovery configuration after a safe mode.

B. What is End-to-End Configuration Management?

End-to-End Configuration Management describes the techniques and tools used to keep track of the configuration of OBCPs on ground and track their process from the definition stage, through implementation and testing, to the final uplink to the spacecraft.

The stages can be summarized as:

1. Specification : *what to do*
2. Design : *how to do it*
3. Implementation : *how it's actually done*
4. Unit Testing : *does it meet the specification?*
5. System Validation : *does it affect anything else?*
6. Uplink : *send to spacecraft*
7. Post-uplink configuration : *what is onboard?*

Each stage results in the output of a configuration item, which becomes input to the next stage. Specifically:

1. User Requirements Document
2. Design Document[†]
3. Source Code
4. Compiled output
5. Unit Test scripts & report
6. System validation report
7. Spacecraft configuration document update

End-to-End Configuration Management is the process to manage the transition from stage to stage, without losing or duplicating information at each step. At its simplest, a configuration management scheme can simply be a spreadsheet with a list of OBCPs and additional fields for meta-data such as date of uplink. This is not true end-to-end configuration management, as it doesn't provide a capability to deal with the biggest challenge: change.

Configurations change. They can change because an OBCP is added or removed or because there is a change to an OBCP itself, and that change needs to be documented in a practical manner. There is a balance to be found between

[†] The User Requirements Document and the Design Document are be combined into one document, with pseudo-code and a flowchart detailing the desired functionality of the OBCP.

the complexity imposed by a very safe system and the feasibility of operations improvements within the boundaries of reasonable delays and effort.

"Safe" change means controlled change, controlled such that changes are properly justified, reviewed, tested and are traceable. Spacecraft operators are risk (and hence, change) averse for good reasons, and a good configuration management scheme should reduce risks inherent to change.

Finally, configuration management means being able to have a definitive answer to "what is the configuration at this time?". Time is the key element, and this is where the naive spreadsheet approach falls down. It is unable to track the history of change across time. More advanced tools are needed to track how the configuration changes overall, but also at the level of each OBCP. The "end-to-end" aspect here is achieved by having said tool available at each stage in the process of change, from definition to uplink, such that nothing can "get lost" in the system.

C. Why OBCPs need to be placed under such a scheme

As OBCPs play a crucial role in the way the spacecraft is operated, and ought to be considered as an addition to the onboard software operational capabilities, it is clear why their configuration should be carefully managed. The ease with which they can be changed leads to a temptation to attempt more change than might be necessary. Furthermore, should there be a spacecraft anomaly that causes a loss of the on-board OBCPs stored in the SSMM, it is critical that the "correct" set of OBCPs be available for reloading quickly, and without any ambiguity. The ideal configuration management scheme should address these concerns.

II. Requirements for a Multi-Mission End-to-End CM System

The tool chosen is a critical part of configuration management. Too complex a system will be a barrier to its acceptance and use. Too simple a system may not offer sufficient control over the configuration to be worth the change from spreadsheets. The requirements and their explanation are listed below. The system needs to be:

A. Flexible

Getting the correct balance between allowing change and maintaining a stable configuration is not an easy task, and will be different for each mission. The balance can also change throughout the lifetime of a mission.

During periods of routine operations, when there is no strong driver to alter the way in which the spacecraft is operated, the balance naturally falls on the side of tighter regulation to avoid making unnecessary changes that may affect the stability. Conversely, there are periods when change needs to be more rapid, such as to deal with a spacecraft anomaly or a change in the mission profile[4].

A multi-mission configuration management system needs to be flexible enough to cope with both scenarios, while maintaining consistency.

B. Capable

Different missions have different ways of operating, and a multi-mission configuration management scheme must reflect this. It should not enforce a particular workflow, or procedures unless necessary for good configuration practice. This is especially the case if a scheme is introduced to already flying missions, as in this case, where there is a legacy of tools, spreadsheets, procedures and working habits.

The configuration items related to OBCPs consist of their source code and compiled output, along with associated test scripts and supporting data. The compiled output is in the form of a binary file. The chosen system should be as capable of dealing with binary files as it is with plain text. Furthermore, ultimately all documentation associated with OBCPs should be within the same CM system, so smooth handling of common office file types is a strong requirement.

C. Reliable

If a tool is used as a key element in the scheme, it needs to be available when needed, so that it can be depended upon to deliver the correct results all the time. A central point of entry into the system is crucial here - a spreadsheet is reliable in itself but it is all too easy to create extra copies, and then none can be relied upon to be the "correct" version. Methods to restrict modification of mission critical documents are also relevant here.

D. Available

Availability is not the same as reliability. The primary reason for abandoning the previous tool[‡] was that the system was not always available due to hardware or software problems, that relied on the willingness of a single person to maintain the system. The architecture therefore must lend itself to being highly available (server-class hardware and software) and supported by the ESOC backup infrastructure to ensure availability of data should there be a server problem.

E. Multi-User

It is clear from the name "multi-mission" that a single user system will not suffice. The majority of modern configuration management tools all provide concurrent editing capabilities and offer powerful merging functions to resolve differences should conflicts arise. It is not the intention for multiple parties to be modifying the same OBCP at the same time, but as the ability exists it should be retained.

F. Secure

This security requirement includes both the protocols used, and a user-level access control to determine who can read and write to each controlled item.

"End-to-end" implies that the data held within system is available at each stage of the process. For security reasons the operational commanding machines are isolated from the development and office networks. It would not be an "end-to-end" system if the OBCPs could not be easily yet securely transferred from one LAN to another. FTP has been the traditional solution to transferring between LANs, which is neither secure nor good configuration management, as one tends to end up with numerous copies of files with similar names in a variety of locations. The chosen solution should offer a secure method of data transfer.

G. Accessible

The OBCP development system (Solaris[§] 6), used for writing and compiling the source code, is of a different architecture to the PCs (Windows[**] 7) used by the database analyst responsible for producing the spacecraft database abstraction required for compilation, which is again a different system to the operational commanding machines (Solaris 8, soon to be Solaris 10). The chosen solution must therefore be accessible on all systems in use at each stage of the OBCP development process.

H. Maintainable

As multiple spacecraft will be relying on the system to maintain the configuration of their onboard software, the chosen solution should be well supported. An obscure proprietary system provided by a single vendor does not offer sufficient protection no matter how good their current technical support may be. This directs one towards popular open source solutions, which have a large community following and are widely used by industry. Choosing a solution which is akin to an industry standard gives further confidence in the quality and robustness of the system. The maintenance overhead of the system should be low, so as not to use engineering time for basic system administration tasks.

I. Mission-Specific Requirements

While no explicit mission-specific requirements are identified, and indeed are discouraged, each mission has its own history of OBCPs and their configuration management items. This inevitably leads to "hybrid" systems while information is transferred from legacy tools to the common one, and procedures take time to catch up. This is not a drawback as long as the documentation is sufficient to manage the changeover.

III. Presentation of Solution

After comparing the available solutions on offer, "subversion[††]", often known as its abbreviated command name "svn" was chosen. It met all of the requirements as stated above, as well as being available for "free": free in terms of both cost and open source. Based upon a client-server architecture, its decentralized nature allows clients on different hardware architectures to communicate seamlessly, and its widespread industry support means it is unlikely to leave an unsupported legacy system requiring a costly porting exercise later.

[‡] IBM's Rational ClearCase

[§] Trademark of the Oracle Corporation.

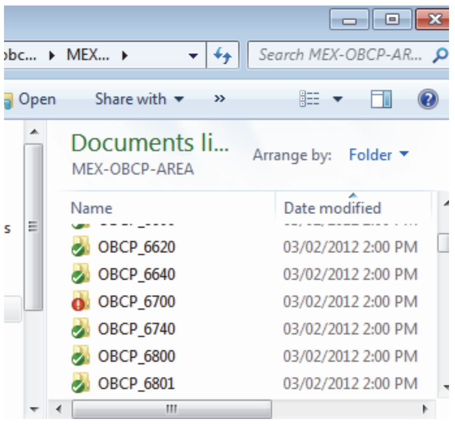
[**] Trademark of the Microsoft group of companies.

[††] Trademark of the Apache Software Foundation

Of particular importance to OBCPs is the versioning of binary files and the branching/merging functionality. This allows keeping track of the compiled output and to maintain a main "trunk" representing the latest version, while also allowing version controlled development in a side branch. Such a branching system is of particular use when making significant changes to an existing OBCP, such that the current version can be maintained in parallel.

Subversion can be configured to automatically inform a pre-defined set of recipients whenever change was committed in the repository via Email, thus simplifying the information exchange between the users and missions.

Subversion works on a client-server principle, and the wide availability of clients was a particular draw. The provided command-line interface clients are powerful but not intuitive, so more user-friendly clients were investigated. For Windows clients, TortoiseSVN[‡‡] was chosen as it provides close integration with the Windows shell, offering simple "right-click" access to subversion functions such as commit, update and repository browsing (Figure 1).



Name	Date modified
OBCP_6620	03/02/2012 2:00 PM
OBCP_6640	03/02/2012 2:00 PM
OBCP_6700	03/02/2012 2:00 PM
OBCP_6740	03/02/2012 2:00 PM
OBCP_6800	03/02/2012 2:00 PM
OBCP_6801	03/02/2012 2:00 PM

Figure 1. Differences to Repository
Clients such as TortoiseSVN integrate with the Windows shell to provide real-time information on changes to version controlled files.

The various Solaris clients in use on the Operational LAN offered an additional restriction, in that the installation of any software must be approved by a series of configuration change review boards, and the core subversion client has a long list of dependencies. A set of approved software items exists however, and this includes Mozilla[§§] 1.6, an elderly yet functional web browser. A free web-based client, Polarion Software's "WebClient for SVN"[***], was identified as being compatible with the browsers available on Solaris. It offers basic commit/update and repository browsing capabilities, along with text-based file differencing, all within the browser. This was the key element to the success of the End-to-End system: OBCPs can be defined on Windows, written on the archaic Solaris 6 box, transferred to the Operational LAN for system validation, and finally uplinked to the spacecraft.

A. Solution as Implemented

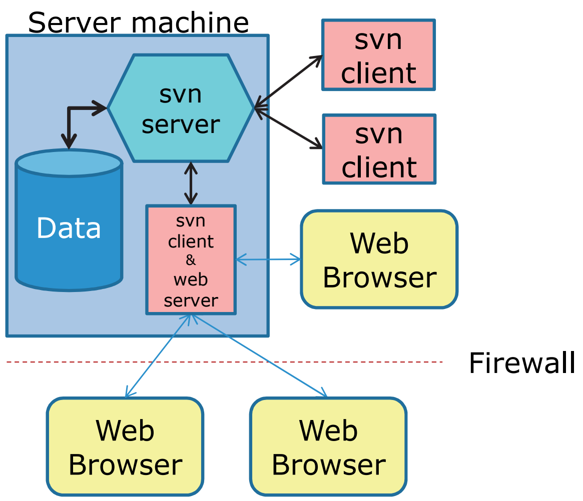


Figure 2. Solution as Implemented
The client-server nature is apparent, with clients connecting to the svn server. A web server with embedded client provides access to the repository via an intuitive browser interface.

The solution as implemented is illustrated in Figure 2, simplified to show the client-server configuration. The subversion server accepts connections on an HTTP port, to which various clients connect. For Windows or other machines using native clients, this is the extent of the complexity: they simply enter the server address and connect.

Also running on the server machine is a web server (Apache Tomcat), in which the WebClient server software runs. It contains a fully-functional subversion client, which is configured to connect to the svn server as with the other native clients. Via the web server, it interprets the output of the svn client as html for display in a standard browser, and also translates http requests into svn commands back to the subversion server. Thus, it acts as a "bridge" between the web browser and the svn client.

The Tomcat server offers connections over the HTTPS protocol which is routed through the operational LAN firewall on a designated port. This ensures the data to and from the server is protected against interception and alteration.

Originally it was intended that only the Solaris

[‡‡] Available from http://tortoisesvn.net/
[§§] Trademark of the Mozilla Foundation
[***] Available from http://www.polarion.com/products/svn/svn_webclient.php

workstations would use the web browser interface. However, it has proven sufficiently simple and usable for general use that it has become the preferred method of svn interaction for some members of the flight control team. The advantage of not having to use a dedicated client is especially apparent to casual users who only occasionally need access to a file.

The actual data is stored on the server machine in a directory managed by the svn server. This directory is included in the normal ESOC nightly backup policy. By default, svn works on a differential storage scheme whereby only the differences to the previous version are stored. This makes differential backups very efficient as each set of changes are represented as discrete set of new files. Each mission has its own independent repository, "ROS", "MEX", and "VEX" for the three missions respectively. A shared "HSO-OP" repository is intended for common tools used by the database analysts for all missions within the division.

Subversion offers various tools for data import / export for migration and backup purposes. At present these have not been used.

B. End-to-End OBCP Workflow

Using the implemented solution, in end-to-end OBCP configuration management terms, the process works as below (also see Figure 3):

1. [Windows] Engineer commits approved Requirements and Design documents to server
2. [Windows] Database abstraction committed to server from database analyst
3. [Windows] OBCP designed and exported from procedure development tool and committed to server[†††]
4. [Solaris 6] OBCP created or altered on development machine, committed to server
5. [Windows] OBCP checked out to engineer's local workstation for review
6. [Solaris 6] OBCP checked out to development machine for unit level test, report committed to server
7. [Solaris 8] OBCP checked out to test system, send to spacecraft simulator for validation
8. [Solaris 8] OBCP checked out to operational commanding system for uplink to spacecraft
9. [Windows] Engineer updates spacecraft configuration documents

Of course, the flow is not fixed, and one may return to a previous stage if errors are found or improvements are required. At each point a 'commit' is made it is immediately becomes the "latest version", available immediately to the downstream stages.. Each 'commit' creates a unique revision number, which can be used as a global token to uniquely identify a particular version. Documentation can now refer uniquely to the source code, test scripts and compiled output of each OBCP, avoiding any ambiguity in the configuration.

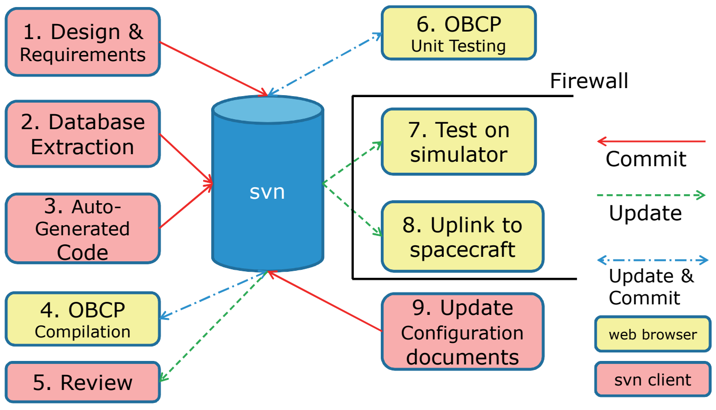


Figure 3. End-to-End Workflow
The subversion repository sits at the center of the workflow, accessed at each stage by a variety of clients.

The act of committing a new revision to the server does not force others to immediately update their working copy, allowing a managed resolution of conflicts between versions. Many svn clients (including TortoiseSVN) will alert the user to a change in the repository of a file in their working area. This is of particular importance if different people are developing and testing the OBCPS as it provides direct, real-time feedback of changes and removes the chance of a change going un-noticed (see Figure 1).

1. Integration with Procedure Generation Tool

As described in Ref. 1, the move towards automated code generation from within the ESOC standard procedure generation tool (MOIS) introduces an additional step to the workflow, along with an additional CM tool (Microsoft VisualSourceSafe) which is used by the procedure generation tool itself. The tool generates OBCP source code which can be exported to a Windows workstation. At this point it can be committed to subversion, where it is then

[†††] Extra step if using the automated code generation features of the procedure development tool "MOIS"

immediately available on the OBCP development machine for compilation. MOIS retains responsibility for the high-level procedure view, and subversion retains the source code, test scripts and compiled products.

This approach has the advantage that all steps of the chain remain under configuration control (VisualSourceSafe and Subversion). The disadvantages inherent in having multiple CM systems (increased complexity, possible divergence between them) are mitigated by having a clear separation between the high level procedures, and the low level code. Because the code is auto-generated by the tool, modifications must be made within the tool, keeping the flow of changes one-way. This simplifies the interface between the two systems.

The OBCP documentation, which itself should kept under configuration control in subversion, is used as the configuration link between the two systems by the listing applicable version revision identifiers from both.

2. *Testing and Uplink*

Having a central repository simplifies the testing and uplink significantly by offering a secure mechanism of transferring files to the operational systems. It removes the requirement to keep local copies on individual machines, eliminating the risk of using an incorrect version, by allowing easy retrieval of a known revision number whenever required. With the rapid OBCP development phase that Mars Express is currently experiencing, this has proven to be a valuable asset.

C. Changes applied to procedures

In general, no changes have been required to spacecraft procedures. The MCS cannot load OBCPs directly from subversion, so an intermediate step is still required to get the relevant revisions of the binary files out from svn and placed into a local directory for the MCS to load. This step of “what to get” and “how to get it” from svn is the area in which procedures are starting to be changed. Rosetta, for example, have modified the procedure to restore SSMM configuration in terms of system / payload packets and OBCPs, in addition to the section of the Flight Operations Procedures relating to all OBSM procedures and OBCPs management.

Rosetta are beginning to use subversion also for Spacecraft Configuration Control e.g. Thermal Tables, DMS-AOCMS patch chain in used, and OBSW source code and binary images. The procedures and documents relating to these also will require updates in the same vein as those for OBCPs.

Overall, a mission wide strategy to update, add and prepare files must be developed which makes it clear and easy for the FCT to utilize the system in the most efficient and secure way. This includes introducing the use of the File locking for unmergeable files ("reserved checkouts") to prevent concurrent editing for files where this is relevant.

D. Changes applied to MCS

The advantage of using a web-based client is that no changes are necessary to the existing software systems. Almost all architectures have a web browser available that is capable enough to display the web pages generated by the server. The only change required to the MCS was to have an “updated” web browser installed[‡‡‡] after which the user could interact with subversion as easily as on any other platform.

IV. Conclusions and Lessons Learned

A. Conclusion

In summary, the solution as presented works, and is currently in use by three missions. The end-to-end nature has been demonstrated on an almost weekly basis for MEX as the number of OBCPs continues to rise to deal with the SSMM anomaly experienced in 2011. It has been the enabling factor in allowing different teams (power, thermal, instruments, data handling) to work in parallel on developing and testing OBCPs.

There is still work to be done, however. With nearly a decade in space, each mission has its own way of operating and there is some reliance on the traditional spreadsheet on a shared drive, which holds meta-data, such as the date OBCPs were uplinked and their reference revision numbers. Ultimately the aim is to move all the spreadsheets and other supporting documentation into subversion so that they can benefit in the same ways the OBCPs themselves have.

[‡‡‡] The browsers were updated to Mozilla 1.6 (2003) from Netscape 4.76 (2000) on the Solaris 8 workstations, and Firefox 1.0.6 (2005) from Netscape 4.61 (1999) on the Solaris 6 OBCP development machine, which didn’t fall under the tighter software policy restrictions of operational workstations.

One interesting observation is that all three missions have independently started using subversion for managing not only OBCPs, but:

- MPS configuration files
- Spacecraft thermal configuration files
- Onboard software
- FCT-developed software and scripts
- Many other spreadsheet-type configuration files

The most important conclusion from this is that a mission has many configuration items and the need for a centralized, easy-to-use but powerful configuration management tool is greater than anticipated. The ease of use offered by a well-supported tool with web access and integrated Windows shell clients enables non-technical users to better address the need that has always been there.

It is interesting to note that the scheme to do away with configuration spreadsheets has instead reinforced their utility, by enforcing a strict versioning of them and making it clear which version is the "reference version". The TortoiseSVN client is excellent at performing differences between Word and Excel files based upon different svn revisions, which is an extremely useful functionality.

B. Lessons Learned

1. Training

Training is key, both in the principles of good configuration management practice and in the usage of the tools themselves. Migrating from simplistic version numbering of file names has been a challenge. While everyone accepts in principle that configuration management using a tool such as Subversion is a valuable, some are reluctant to change. This is purely an issue of training, to demonstrate the ease with which the tools can be used and the immediate benefits they impart (history tracking, central location, etc.). Due to high workload caused by the MEX SSMM anomaly, the anticipated training exercises and documents have not yet been completed.

2. Operationalization

The current installation and setup of the server software was implemented as a proof-of-concept by Mars Express. While very successful to the point where it has become the operational system, the lessons learned from the setup process have not had a chance to be put into practice. These include password management, user / group setup, SSL certificate configuration. The lesson here is that the transition from the proof-of-concept to the operational system should be actively managed. It is instructive to note that the main reason for the swift operationalization of the system was driven by the extremely high demand for the system from Rosetta and Venus Express.

One particular benefit of Subversion is the well documented data export functionality. Once a "clean" operational installation is established, migrating the data and meta-data should be a very smooth process, transparent to the users save for the change in server location in their local clients.

3. Maintenance

Related to both previous points is that of maintenance. Subversion is a low-maintenance system and is extremely stable. However, the question of ultimate responsibility for the maintenance of the system is outstanding, with FCT engineers being assigned to work on it on a "best effort" basis when other duties allow. As it becomes more tightly integrated into operations and procedures, the importance of having the maintenance responsibilities formally allocated increases. Ideally this will ultimately become a tool provided as part of the generic ESOC infrastructure.

C. Future Work

OBCPs are ultimately software, and we should follow the best practices of the software development industry which has been using such configuration management techniques for years. Current trends of "continuous integration" whereby changes are quickly applied to operational systems do not map well with spacecraft operations. That said, automated generation and running of unit tests and generation of validation reports is an area that would release engineering time for less tedious tasks.

A first step could be automated generation of email every time a change is made, such that concerned parties will be notified of a change. This would alleviate the perennial problem of communication between concerned parties. The Mars Express data handlers could be notified of updates to related Venus Express OBCPs, for example. Care must be taken with automated emails to avoid "spamming".

Another simple step would be automated checking of 'commit' comments to ensure that a sensible comment had been entered. Such comments can be invaluable when attempting to understand the evolution of an OBCP or other complex item.

Appendix A
Acronym List

AOCMS	Attitude and Orbit Control Management System
CM	Configuration Management
DMS	Data Management System
ESA	European Space Agency
ESOC	European Space Operations Centre
FCT	Flight Control Team
FDIR	Fault Detection, Isolation and Recovery
FTP	File Transfer Protocol
HSO-OP	Human Spaceflight and Operations – Operations, Planetary division
HTTP	Hyper Text Transfer Protocol
HTTPS	HTTP Secure
LAN	Local Area Network
MCS	Mission Control System
MEX	Mars Express
MPS	Mission Planning System
MTL	Mission Time Line
OBCP	Onboard Control Procedure
OBSM	Onboard Software Maintenance
OBSW	Onboard Software
ROS	Rosetta
SSL	Secure Socket Layer
SSMM	Solid State Mass Memory
SVN	Subversion
TC	TeleCommand
TM	Telemetry
TT&C	Telemetry, Tracking and Commanding
VEX	Venus Express

Appendix B
Glossary

Aero-braking	Use of a planet's atmosphere to provide a braking force through friction
Solid State Mass Memory	Large (10 Gigabit-class) onboard memory unit, used to store science, telemetry and commanding data.
Sandbox	A protected environment in which software can run, which prevents software errors such as divide-by-zero and infinite loops from affecting the host system.

Acknowledgments
We are grateful to the Flight Control Teams of Rosetta, Mars Express and Venus Express for their cooperation and their enthusiasm for finding new uses for the subversion tool.

References
[1]P. Choukroun, M. Denis, P. Schmitz, M. Shaw, "Evolving ESA Mars Express Mission Capability with On-Board Control Procedures", *AIAA SpaceOps Conference Proceedings 2010,* Huntsville.

[2]M. Shaw, A. Moorhouse, M. Denis, R. Porta, Z. Mounzer, "File transfer, Mass Memory and Mission Time Line – providing spacecraft remote commanding at Mars", *AIAA SpaceOps Conference Proceedings 2006,* Rome.

[3]D. Lakey et al., "FAST: A new MEX Operations concept, quickly!", *AIAA SpaceOps Conference Proceedings 2012,* Stockholm.

[4]M. Shaw et al., "Mission automation and autonomy: In-flight experience derived from more than 8 years of science operations in orbit about Mars", *AIAA SpaceOps Conference Proceedings 2012,* Stockholm.