Deploying an Enterprise-Class Software Lifecycle Management Solution for Test Program Sets

Timothy W. Davis and Gary S. Kane
TPS Acquisition and SSA Branch
FRCSE/ISSC, Code 4.8.4.3
Jacksonville, FL, USA

Abstract—Information Technology tools for Software Lifecycle Management have advanced in recent years to enable organizations to implement a more centralized, enterprise-level lifecycle management solution. Despite these tools having become commonly used in the software development world, the Test Program Set (TPS) community remained behind the times. This paper presents the history and implementation of TPS Lifecycle Management as deployed by the TPS community for the Consolidated Automated Support System (CASS) Family of Testers (FoT). Prior to any efforts to enact centralized TPS lifecycle (i.e. configuration) management, TPS owners controlled their software in a variety of ways. The results of their efforts could be characterized as barely effective and often problematic. The initial attempt at solving the TPS lifecycle management problem was to mandate the use of a commercially available product which was not well supported by the developer of the application. That, coupled with both political issues and technical problems, relegated this system to becoming simply a repository for data. After addressing many of the problems with the initial solution and moving to a more widely-used product, the next iteration of TPS lifecycle management has been much more successful. It has become a legitimate enterprise solution providing both version control and lifecycle management has been much more successful. It has become a legitimate enterprise solution providing both version control and implementing change management principles to include traceability between software change requests and released code. Many of the specific design details are discussed in the paper and include: a flexible workflow, a defined code branching strategy, an ownership protection scheme and a process for release. The paper concludes with a detailed example of how TPS lifecycle management has benefitted the CASS team and, in particular, those who provide support of TPSs. It also shows how those benefits can be attained by other TPS support teams looking to effectively manage their test programs on other types of Automatic Test Equipment.

Keywords—tpss; management; configuration; lifecycle; cass; software; support

I. INTRODUCTION

About ten years ago, the groups charged with managing baselines for their Consolidated Automated Support System (CASS) Test Program Sets (TPSs) did so with varying levels of success. At best, they usually provided their customers with the correct software versions. At worst, they lost track of changes and, in at least one case, lost entire sets of files. To compound the problem, the numerous Fleet Support Teams (FSTs) were spread amongst several sites around the country and shared few common processes.

The problem became evident during the development of the Reconfigurable Transportable CASS (RTCASS), sponsored by the Naval Air System Command's (NAVAIR) PMA-260. One of the primary goals of RTCASS was to be able to run existing CASS TPSs without having to change the underlying TPS source code. To be able to design the system to meet that goal, the RTCASS developer, the Boeing Company, required access the complete set of over 400 CASS TPSs that would be have to run on RTCASS. At PMA-260's request, the FSTs delivered what was supposed to have represented the latest released code, but the delivered code was often problematic; in some cases it couldn’t even be compiled. There was clearly a general breakdown in the management of the TPS baselines.

At this point, PMA-260 saw an opportunity to help the FSTs get a better handle on managing their TPSs while benefitting from the collection of TPS analysis tools born as a result of the RTCASS program. PMA-260 commissioned the CASS Software Support Activity (SSA), which had over 15 years’ experience in managing the CASS software baseline, to establish a software configuration management system. The thinking was that the FSTs would use the tool to actively manage individual changes to their TPSs and be able to provide more timely and accurate products to their customers. Additionally, PMA-260 would have access to a large cache of TPS source code it could use to establish a TPS capability baseline to evaluate future changes to its growing family of Automatic Test Equipment (ATE). In around 2006 the SSA stood up the Automatic Test System Source Data Repository (ASDR).

Unfortunately, the initial deployment of ASDR was not as successful as had been hoped. The software configuration tool that PMA-260 selected had its own technical problems, especially coupled with the environment in which it was deployed. The web interface was clunky and often slow to respond and customization was complicated and limited. More significantly, political problems drove a wedge between PMA-260 and the FSTs. There were questions about data ownership, server location, and responsibilities. The result was a well-populated but underused data repository with almost no traceability to any configured software.
The outlook was improving, however, as some of the FSTs better understood the benefit of using a software tool to manage their TPS baselines. In fact, in some cases they had begun using their own tools internally to compensate for the barely usable ASDR. At this point, PMA-260 decided to address the technical and political issues associated with ASDR by first commissioning a study of various software configuration management tools. The conclusions of the study indicated that Microsoft Team Foundation Server (TFS) would be a technically acceptable choice for managing TPSs. To address the political issues, PMA-260 decided to take a less prominent role and allowed both the FSTs and the SSA to jointly revamp ASDR and come to an acceptable common solution for Software Lifecycle Management (SLM). In 2012, the CASS SSA rolled out what became known as ASDR 3.0.

II. IMPLEMENTATION

A. Useful features of TFS

The objectives of Software Lifecycle (or Configuration) Management include [1]:

- Identify, define, and control all relevant items of the project
- Control modifications of the items
- Record and report the status of items and modification requests
- Ensure the completeness of the items
- Control storage, handling, release, and delivery of the items

Many of the features of Team Foundation Server make it an excellent choice for performing Software Lifecycle Management of Test Program Sets. These include:

- Team Project Management – Allows separate Team Projects for each aircraft platform FST, using a common, standardized folder structure.
- Source/Version control – Maintains complete version history of each TPS Source code item and permits branching and merging to protect the baseline during development.
- Work Item Tracking – Provides traceability of every software defect or enhancement back to its source code. It aids project management by tracking tasks and permits users to customize work items dependent on project specifics.
- Data Collection and Reporting – gathers reporting information relative to individual Team Project Work Items by querying the back-end SQL Server database.

For the most part, these are the main attributes that make up the current state of TPS Software Lifecycle Management as implemented in ASDR 3.0. There are many other features of TFS that have not currently been put to use, such as enhanced reporting using the SQL Server Analysis and Reporting Services and the Project Portal which taps into the SharePoint services capability to integrate TFS with SharePoint for improved project management functions. The decision not to take advantage of these features has primarily been due to local networking and computing policies.

B. Team Project Management

When first organizing and implementing ASDR 3.0, the design and deployment personnel identified each major aircraft platform FST as a separate data owner, and created distinct Team Projects for them within ASDR 3.0. The security posture chosen for the data enabled each FST to upload and manage their TPS source code independently from the rest while still allowing the entire CASS community to have read-only access. This access benefits all the teams by providing hundreds of thousands of lines of known-good sample TPS code that can be used during new TPS development. The ASDR 3.0 development team wanted to standardize the structure of ASDR 3.0 to the greatest extent possible. Therefore, careful consideration was given to the proper configuration of the standardized folder structure and the branching strategy to be utilized within ASDR 3.0. Generally speaking, TFS is much more than a simple file manager used as a repository for files, documents and source code. TFS represents a robust, commercially-available, enterprise-level product for managing software projects (e.g. TPSs) throughout their entire lifecycle. Additionally, ASDR 3.0 consists of all of the structures, processes and data built on top of TFS.

The folder and branch structure designed into ASDR 3.0 is straightforward, having a separate top-level folder for each Operational Test Program Set (OTPS), and allowing branching at the next level below. Branching and Merging are implemented based on commercial best practices using the Development and Release branches to achieve isolation and is shown in Fig. 1.

![Fig. 1. Branching structure employed in ASDR 3.0 for each OTPS](image_url)

- Main Branch – This branch always contains the Master Source Code library and represents stable code.
- Development Branch – This branch is to be used by FST personnel (e.g. TPS Engineers) to check out code to work on and to check back in upon completion and is not necessarily stable. It initially gets created as a copy of Main, but diverges as a result of new check-ins.
• Release Branch – Each Branch in the Release folder represents a deployed version of TPS software and contains all associated source code and executables.

• Migration Branch – This branch supports a major effort to migrate source code from legacy CASS platforms to the latest CASS family member in eCASS. Each FST Configuration Manager (CM) creates the Migration branch from the Development branch and permits Migration Team members to perform Checkouts and Check-ins and then for the FST to merge completed code back into the Development branch. All of this is accomplished in a traceable manner allowing all parties to understand the nature of the changes.

C. Source/Version Control

Source control is essential for managing TPS development and sustainment efforts. TPSs consist of many files that can have many different release versions. Being able to support different versions of the TPS by using the designed branching strategy and linking code changes to Work Items is what sets apart TFS [2]. In order to properly execute TPS source control, it is vital to use TFS Workspaces. The Workspace is essentially a mapping of files on the TFS server to an area on a local file system. Once a Workspace is mapped, TFS must be explicitly made to download the latest files to the Workspace through the use of the “Get Latest” command.

TFS maintains a mapping between the server repository and each user’s local file system as depicted in Fig. 2. TFS keeps track of changes when the local source is committed to the server. This happens when local changes are checked in.

![Fig. 2. TFS Workspaces](image)

Check Out – FST starts working on a TPS file and possibly control access to the file based on settings. Check In – FST commits the changes made to the TPS back to the server [4].

It is with these source control mechanisms (check-in, checkout), along with many other features that makes TFS a robust, enterprise-level source control system and gives the FST’s TPS sustainment group the ability to manage different versions of a TPS in ASDR 3.0.

Benefits of TFS Workspaces:

• By keeping track of which versions of files contained in the local file system and only updating the modified files, ASDR 3.0 is able to optimize network traffic (i.e. minimizing the amount of information passed between server and clients).

• If a file gets renamed or deleted on the server, the next time a user performs a Get Latest, it will be renamed or deleted in the local workspace.

Consequences of TFS Workspaces:

• To minimize data movement, TFS only acts (e.g. to download the latest version of a file) when explicitly commanded.

• Users must always tell the server when to do something to a controlled file or folder (e.g. edits, renames, or moves).

The single best practice to be adopted by ASDR 3.0 users is to never modify files outside of a TFS client. Other best practices adopted by ASDR 3.0 users include:

• Typically a user would want to be working with the latest version of a file.

• Perform a Get Latest prior to checking out.

• Checking in “stale” files creates conflicts and extra work to merge.

• Check out only the files that are needed.

• Check files back in promptly (as soon editing and testing are complete).

• Ensure other FST team members have access to the latest changes.

• Write good check-in comments. An example of a good check-in comment might state, “Modified limits on T1330 for RCN:RXXXX-20-XXXX” whereas a bad one might simply say, “Updated files” [5].

• Associate all changes with one or more related Work Items.

D. Work Item Tracking

Work Items are one of the most useful features of TFS employed in the ASDR 3.0 system. The Work Item system has benefited the FST’s TPS development and sustainment processes through the incorporation of change control management, which becomes a virtual necessity due to the large number of TPSs supported. Work Items can exist in many forms, depending on the needs of the organization. By default, TFS comes with many different Work Item types, the most useful being: Bugs, Requirements, Issues, and Tasks. Custom Work Items can also be easily created to meet the needs of a team’s unique processes. Work Items are stored in TFS in a simple eXtensible Markup Language (XML) structure, so it easy to extend and modify already existing work items with additional field types and rules according to the team’s requirements. One example of a custom Work Item type created in this manner is the CASS Change Request
(CCR). The SSA modified the existing Change Request (CR) Work Item provided by TFS to include additional data fields and workflow rules. As an example, to support the legacy method of reporting problems with CASS Family software, the SSA mimicked the previous legacy change management system, and created a reporting form and a state-flow for how the change mechanism item (e.g., CCR) was processed. The CCR state flow, depicted in Fig. 3, starts in the Proposed State and works its way through the workflow until the software change eventually gets released.

![Fig. 3. CASS Change Request (CCR) Workflow](image)

Each individual state within the process is owned by one of the development team members. These roles have been established to identify the responsibility as it pertains to the Work Item’s lifecycle. The roles consist of a CCR Lead, Technical Lead, Software Engineer, Test Engineer, Software Integrator, and Quality Assurance (QA) Representative. Basically, the CCR Lead is assigned each new proposed CCR for review. It is the CCR Lead’s role to determine whether the CCR should be accepted for analysis or whether it is determined to be out-of-scope and rejected. If the CCR is accepted, it is then assigned to a software engineer for the Analysis phase. The software engineer then proceeds to analyze the CCR for possible impacts to TPSs, Architecture, User Experience, Design/Development, and Technical Publications. This analysis is required and is an integral part of determining the nature and scope of the proposed change and to help in planning the time and resources needed to resolve the problem. Once the CCR has been analyzed and all the necessary impacts are identified, this data is then undergoes a review process, which includes both a preliminary review and a final design review. Upon final acceptance during the design review, the CCR and all generated data then enters the Implementation (i.e. Coding and Unit Testing) phase. Following successful implementation, the CCR is then determined to have been Resolved and subsequently gets sent to QA for quality check. The QA process includes:

- Code has been compiled and the CCR linked to TFS Changesets.
- Detailed design documentation has been completed.
- Software code headers (i.e. change logs) have been updated.
- Changes summarized in difference listings in the Attachments section.
- Comments are clear and concise.

- CCR Metadata (e.g. CCR Title, Points of Contact, Priority Level, Description of the final solution) has been completed.

If any of the QA checks fail, the CCR state-flow is set to “Resolved” with “QA Fail” as the reason for the state change and returned back to the software engineer to be resolved. Once the issue is resolved, state-flow then returns to QA for a follow-up check and continued processing. The QA checklist becomes part of the objective evidence of the CCR process and gets uploaded into the attachments section of the CCR for future reference or analysis.

Another example where the SSA employed TFS Work Items in ASDR 3.0 was to develop a custom Work Item type needed by the eCASS TPS Migration Team in support of the enormous task of migrating existing legacy TPSs to the new eCASS station. The Migration Team required a mechanism to track changes to each TPS being modified and a way to manage the progress of each TPS as it was reviewed by the different FST members. A new Work Item type was created, based on the out-of-the-box “Issue” Work Item and named “Migration Problem Identification Report” (MPIR). In addition to the fields on the MPIR form, a workflow was created that identified the different roles and states needed to support the Migration Team’s process. The roles identified included the migration team Project Engineer, Migration Team Member, FST, and Test Team. The state-flow that was developed is shown in Fig. 4.

![Fig. 4. Migration Problem Information Report (MPIR) Workflow](image)

The original “Issue” workflow consisted of only four states (Proposed, Active, Resolved, and Closed), but was a fairly straightforward effort to extend the capability of the “Issue” Work Item type to the MPIR Work Item by adding five new states (Reviewed, Acknowledged, Regression Tested, Tested, and Merged). Also designed into the process was a mechanism to capture the date on which each state transition occurred, beginning with the Resolved state. This provided a way for the Project Engineer to capture applicable metrics relating to the amount of time the MPIRs took to be processed. If a failure occurred during the process and had to revert back to a previous state, the associated date value was reset. Even without this customization, TFS automatically captures all of the date values, along with a lot of other metadata, through the History feature which keeps a log of the details of every
transaction. Everything that is performed in TFS, from Work Item processing, to code check-out and check-in, to creating branches, etc., is all captured in the TFS history data records. Thus, all of the history of each individual MPIR is available forevermore in ASDR 3.0. Every bit of data and metadata is available to the users and is easily accessed through the query and reporting features of TFS.

E. Data Collection and Reporting
TFS utilizes SQL Server as the database backend, which enables the user to generate reports based on the query engine that exists with the database. The bulk of the reporting required by most users of ASDR 3.0, consists of the use of Work Item Queries. TFS provides a user-friendly interface for building queries, without the need for intricate knowledge of the SQL language. Fig. 5 shows the simple query interface provided by TFS.

![Team Foundation Server Query Interface](image)

The interface provides the FST with a few default clauses, starting with automatically selecting the current Team Project in use. This is done using the “@Project” keyword within the clause. Additional clauses can then be added to the query. These include: the Work Item type, the State of the Work Item, the Assignee, etc. The use of these queries is limited only by the data fields that are available within the Work Item and to the user’s imagination in adding selection logic. Fig. 6 shows a typical example of a user query that returns a list of Work Items that they are currently assigned.

![Sample Query showing Work Items currently assigned](image)

As shown, the query starts with the selection of the current “Team Project” field, where the clause uses the TFS query keyword “@Project”. Next the “Work Item Type” field is added (e.g. CASS Change Request, MPIR, or Task). In this example, the user grouped the query in order for it to be treated as a single clause and to correctly show results for multiple Work Item Types. The next clause looks at the “State” field and demonstrates the use of the not equal operator (<> ) to include only those Work Items that are NOT closed. Finally, the last clause looks at the “Assigned To” field to match only those Work Items that are specifically assigned to the current user by using the “@Me” query keyword.

Another nice feature when building Work Item queries is the capability to select additional columns of data fields to be included in the report. Fig. 7 shows the dialog box that appears when selecting the “Column Options” button while building the query. This gives the query builder the capability to add or remove data items from the report.

![Adding data columns to a report](image)

As queries are created, they can be saved for future use in the “My Queries” area to be used only by the creator, or they can be saved in the “Team Queries” area to be shared amongst the team.

III. OTHER APPLICATIONS
The success of ASDR 3.0 is in part due to the flexibility in configuring Microsoft Team Foundation Server such that users are limited to the data they’re able to read and write. Because of this, performing Software Lifecycle Management on Test Program Sets should have a much broader appeal well beyond the CASS FST community. There are three main pieces that make managing a user’s access to data in TFS simple and straightforward: User authentication through Windows Active Directory, Client access through Visual Studio, and Managing access at the different structural levels of TFS.

User control and authentication in TFS is provided by the Active Directory infrastructure built into Windows server-class machines. Because of this, administrators can take advantage of more secure authentication schemes such as Public Key Infrastructure (PKI) at the server level and know that access to TFS is subject to the same rigorousness. Because of this, it may be easier to comply with Information Assurance policies should other organizations seek to employ TFS.

Access to data within TFS can be achieved through a standard web browser; however, the TFS client most users will choose will be either Microsoft Visual Studio or Microsoft Team Explorer. Team Explorer is essentially a stripped-down version of Visual Studio that doesn’t contain any compilers and is available as a free download from Microsoft.
within either tool, a user with appropriate privileges can add users to TFS Groups, the three primary ones being: Readers, Contributors, and Project Administrators. Project Administrators are then able to set access restrictions on specific files, folders and branches most easily by setting permissions for a specific group or individual user. It is important to understand the way data is structured within TFS to be able to design a data structure (See Figs. 8 and 9) that makes sense from a security standpoint and best meets the needs of the organization. At the topmost level of the data hierarchy is the TFS Instance that represents a single installation of TFS on a server. For an organization wanting to perform Software Lifecycle Management of Test Program Sets, one instance should be sufficient in all but the most extreme cases.

The next levels down in the hierarchy are the Team Project Collections and the Team Projects. Each TFS Instance may contain one or more Team Project Collections while each Team Project Collection may contain one or more Team Projects. The precise configuration of the Team Project Collections and the Team Projects depends on the needs of the organization since together their structure represents a logical grouping. The important point is that many TFS settings, including security, get implemented at one of these two points in the hierarchy. As an example, one organization could deploy TFS using a single Team Project Collection for the entire company and use Team Projects to represent different Divisions within the company. Alternatively, the company could instead choose a structure where each Team Project Collection represents the different Divisions within the company with Team Projects for each Team within.

Given the large number of Test Program Sets managed by the different Services, which spans dozens of aircraft and vehicle platforms, the need to employ a sophisticated TPS Software Lifecycle Management system is without question. The immediate benefit of employing one is that the management activity gains instant version control of their source files. The long-term benefit, though perhaps less obvious, is that the activity gains a better grasp on their compete set of lifecycle processes and improves the quality of their product over time. In part, due to the flexibility of Microsoft Team Foundation Server, the CASS TPS FSTs have achieved a level of control over their software that far exceeds what they had just ten years ago.

ACKNOWLEDGMENT
T.W.D. thanks Barbara May for a career dedicated to Software Lifecycle Management (or “CM” as they called it in those days), and especially for her many years on the CASS program having to train countless engineers in the Art of CM. The fact that the basic premises she established are still in use today are a testament to her success.

REFERENCES