

# TransVERSE: A case study of re-engineering a mission critical planning tool for HST operations

Mark Giuliano

Space Telescope Science Institute  
3700 San Martin Drive, Baltimore, MD 21218  
[giuliano@stsci.edu](mailto:giuliano@stsci.edu)

## Abstract

The task of infusing new technology into mission critical systems is often filled with many pitfalls and roadblocks. System stakeholders are often wary of new technologies and do not want to change the way things are done. This paper describes the technical and human processes required to make the TransVERSE project to re-engineer a critical HST planning system a success.

## Introduction

The task of infusing new technology into mission critical systems is often filled with many pitfalls and roadblocks. System stakeholders are often wary of new technologies and do not want to change the way things are done.

The TRANS software system plays a critical role in the Hubble Space Telescope (HST) ground system. Part planner and part scheduler, TRANS embeds substantial HST engineering knowledge and is used to provide input to other mission critical portions of the ground system including the interface to the end user astronomer, planning and scheduling, and telescope commanding. TRANS was originally coded in the mid 1980s using an expert system shell (Rosenthal 1986). In the late 1980s, TRANS was rewritten using a homegrown scripting language built on top of common LISP (Gerb 1991a). In 1998 we started the TransVERSE project (Curtis et al 1998) to re-engineer TRANS using new technologies including constraint reasoning and search based optimization. The project was controversial at first due to the use of new technologies and to the accepted idea that critical parts of the ground system should be written once and never changed.

Although the project had a rocky start, it ended as a major success. The project has lowered the cost of adding new capabilities to the system, lowered the need for system maintenance, and supports the needs of all of its customers. This paper describes the technical and human processes required to make the TransVERSE project a success. The

remainder of the paper reads as follows. First, the functionality of the TRANS system is described. Next, the TransVERSE project is described including the motivation for rewriting the system and the new system architecture. The main body of the paper describes aspects of TransVERSE development giving lessons learned during the process. Finally, lessons learned are summarized.

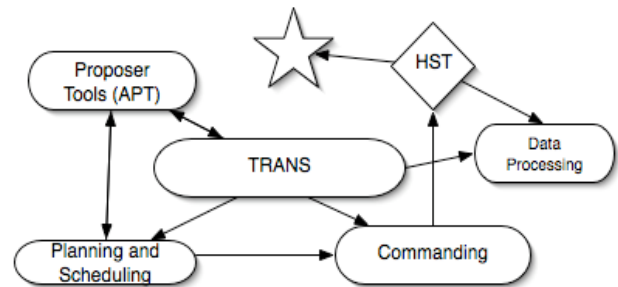


Figure 1: TRANS Interfaces

## TRANS

Figure 1 shows how TRANS plays a central role in the HST ground system. TRANS maps from a specification of an individual observation given by an end user astronomer to a description of the observation that can be used by the rest of the ground system to plan, schedule, and command the telescope. TRANS interfaces with the Astronomer's Proposal Tool (APT) (Krueger 2004) to allow end users to plan their science activities; TRANS provides input to the planning and scheduling systems providing a hierarchical set of activities to schedule; TRANS provides exposure information to the commanding system; TRANS provides exposure grouping information to the back-end system to support data calibration and analysis. The input that TRANS provides to the commanding system is mission critical. Incorrect output from TRANS can not only prevent an observation from executing but also can harm the telescope.

Specific functions in TRANS include:

- Expanding pre-defined patterns of exposures to support sky mosaics or multiple copies of exposures.
- Modeling the behavior of HST scientific instruments to determine overheads and transition times between activities.
- Modeling the pointing between activities to determine slew overheads.
- The creation of support activities including automatic calibrations and data dumps.
- Creating a hierarchy of scheduling activities called the *scheduling unit hierarchy* to be used by components of the ground system to schedule and command the telescope. The lowest level in the hierarchy contains exposures and support activities. The next level contains groupings of exposures with a single spacecraft pointing. As lower level activities in the hierarchy are non-interruptible, this selection impacts schedulability.
- Expanding or shrinking exposure durations by up to 20% to better utilize scarce orbit time.

HST is in low earth orbit and circles the earth every 96 minutes. For a typical target, the earth occults visibility for about 45 minutes during each orbit. Time is allocated to HST users in integral orbits, and APT is used to plan how the science fits in orbits. Astronomers group related exposures into visits. Visits are executed in one or more contiguous orbits each with one or more exposures. Figure 3 gives a visit specification with two exposures. The first exposure has 3 copies each 500 seconds long. The second exposure has a different target and consists of a single copy with duration 300. Figure 2 shows how the exposures fit into an abstract HST orbit, as determined by TRANS and displayed by APT. The display shows a pointing maneuver between exposures 1 and 2 that accounts for the use of different targets. The black bars beneath the exposures represent data dumps inserted by TRANS. Note that a little over two minutes of unused visibility occur at the end of the orbit.

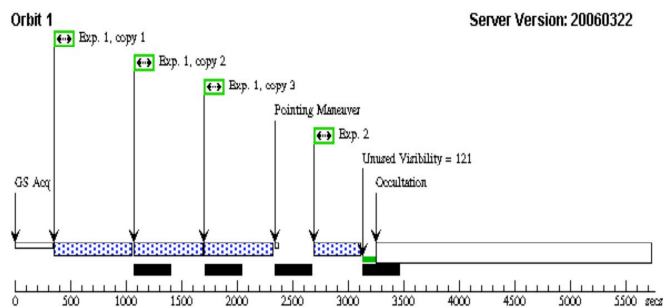


Figure 2: Observation orbit structure as rendered by APT. Exposures are shown in dotted boxes. Black rectangles represent activities dumping data from instruments to the solid state recorder.

```

Exposure_Number: 1
  Target_Name: NGC3079
    Config: ACS/WFC
    Opmode: ACCUM
    Aperture: WFC1
    Sp_Element: F435W
    Wavelength:
    Optional_Parameters: CR-SPLIT=NO
    Number_of_Iterations: 3
    Time_Per_Exposure: 500 S
Exposure_Number: 2
  Target_Name: NGC3079-OFFSET
    Config: ACS/WFC.
    Opmode: ACCUM
    Aperture: WFC1
    Sp_Element: F435W
    Time_Per_Exposure: 300 s
  
```

Figure 3: Exposure specification for Figure 2 layout.

While most of the features computed by TRANS involve hard and fast rules (e.g. the overhead between two exposures), there are many features where TRANS needs to select the best option out of many feasible choices. For example, TRANS can expand or shrink exposure durations by up to 20% to better utilize orbit resources. There are many ways to adjust exposure durations that stay within the +/- 20% boundary. Some will result in better orbit packing than others. These features include grouping exposures into a hierarchy, insertion of calibration and dumps, and adjusting exposure durations. The job of TRANS is to select a good enough assignment for these features. In this way, TRANS can be viewed as an optimization problem.

### Motivation for TransVERSE

The primary motivation for the TransVERSE project was to reduce the on-going cost of development and maintenance of TRANS. Shuttle servicing missions update or add new scientific instruments to HST. Each additional scientific instrument requires significant TRANS coding. At the start of the project, three new HST instruments were planned for development. In addition, the instrument models in TRANS change as the on-board capabilities of HST systems degrade with time. As such, it is highly desirable for TRANS to be easy to modify. This will be especially true in the final years of the HST mission where maintenance dollars will be scarce.

By 1998, the existing homegrown scripting implementation of TRANS was becoming hard to maintain. While the scripting language had a simple notion of objects, it did not support object-oriented hierarchies of information. Code sharing between instruments was minimal even though instruments often share common features. The TRANS code consisted of a set of scripts to

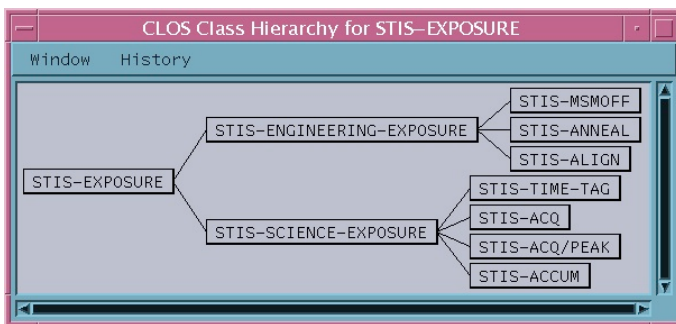
be executed in sequence. Each script set a small number of related object properties. The scripted nature of the system made it very hard to track and code dependencies. A change in one feature often requires re-computing another feature. As more inter-dependent features were added to TRANS the code became tangled. The scripted approach did not promote modularization. Although the requirements were defined by HST instrument, the scripts were written for general properties. How each script impacted an instrument was not clear. Code for instruments no longer in operations was not removed from the system. The final consequence was that the code was hard to understand, as it was not clear which portions were actually in use.

At the start of the HST mission, TRANS had no optimization capabilities. TRANS was a simple mapping from input parameters to output structures. The optimization features in TRANS were added in a piecemeal manner independent of other code features. As a result there was no infrastructure to support the examination of multiple possible solutions to find the best.

Another motivation for TransVERSE was the need to integrate with up-to-date tools for creating astronomical proposals. At the time of project inception, astronomers used a first generation GUI based tool to prepare observations for scheduling. This system was essentially a GUI front-end to run TRANS and other scheduling tools in batch. Each TRANS run required minutes to finish. Astronomers used this tool iteratively to try and fill each of their allocated orbits as full as possible. Combining a long run-time with multiple iterations resulted in frustrated end users. The initial design of the next generation proposal preparation tool was done as the TransVERSE project was starting. This tool as eventually embodied in APT required iteration times on the order of tens of seconds. The existing scripting approach to TRANS was inadequate to achieve these run-times.

### TransVERSE Design

The TransVERSE architecture contains multiple aspects that were considered new technology at the time of project inception including object-oriented programming, constraint sequencing, and the use of search to find solutions.



The TransVERSE architecture defines an object oriented design pattern for modeling HST activities. Figure 4 shows the portion of the exposure class hierarchy pertaining to the Space Telescope Imaging Spectrograph (STIS) instrument. Activities in the hierarchy share common data and behavior across different instruments and modes. The design pattern includes rules for expanding exposure patterns, modeling instrument mechanisms such as filters, modeling the instrument data buffers, and determining how the activities can be placed into the scheduling unit hierarchy. Using the design pattern, adding a new instrument or instrument mode to TRANS requires extending a class definition and instantiating a well defined set of protocol methods.

A key problem tackled by the TransVERSE architecture is handling code dependencies. For example, suppose that the overhead for an exposure depends on whether or not the filter for this exposure is the same as the filter for the previous exposure. If the filters are the same, the overhead is zero and if the filters are different, the overhead depends on a computed filter move time. Typically, developers are responsible for inserting update statements in the code. In the example the developer could insert explicit code to update the overhead for a subsequent exposure when assigning the filter for the current exposure. The sequencing of requirements in TRANS is complex and varies depending on instrument and mode. For example, in one mode the filter could determine the duration of an exposure and in another mode the duration determines the filter. To solve this problem, a general purpose Constraint Sequencing Infrastructure (COSI) was designed that generalizes methods as constraints (Curtis et al 1998). The infrastructure simplifies application development by providing a declarative syntax that separates the application logic from the control of execution. The infrastructure is an example of the equation Algorithm = Logic + Control (Kowalski 1979) where the developer only codes the logic and the control is handled by the constraint sequencer. A useful analogy is that COSI provides a spreadsheet update ability to object properties. Developers code methods in a slightly modified version of Common LISP (constraints are defined by defconstraint instead of defmethod). Using extensions coded in the Meta-Object Protocol (Kiczales 1991), the system tracks dependencies during execution of constraint methods and re-execute methods if dependencies are triggered.

Given an input to TRANS there are multiple legal outputs that can be used by the rest of the ground system to execute the observation on HST. TRANS can have multiple options in grouping exposures into the scheduling-unit hierarchy, insertion of calibrations and dumps, and adjusting exposure durations. The job of TRANS is to find a feasible structure that makes effective use of telescope resources. The scripting approach to TRANS examined the set of possible structures in an ad-hoc manner. Each feature was optimized separately. A major problem with

this approach was that there was no separation of the code that generates potential solutions from the code that tests if a potential solution is feasible. An explicit search mechanism is supported in TransVERSE. Constraint sequencing determines the implications of search decisions. Once a decision has been made constraint propagation determines the feasibility of the decision. This allows the search mechanism to iterate through multiple potential structures to find one that is good enough.

## **Project Implementation and Lessons Learned**

The remainder of the paper reviews different aspects of the implementation of TransVERSE. Each aspect is described and the lessons learned during implementation are presented. While many of the lessons learned may seem trivial, the combination of these aspects is as critical to the project success as the advanced technology utilized.

### **Project Planning**

The development team made the common and costly mistake of not involving key stakeholders during the project inception stage. The development team had already defined the system architecture before the first presentation to stakeholders was made. The initial reaction of stakeholders was to put on the brakes. There was no agreement that we needed to overhaul the existing system and that if we did whether we should use untested technologies such as object-oriented programming, constraint sequencing, and search. A multi-month phase B study was commissioned to investigate each of these issues.

While Object-Oriented Programming (OOP) had been used in GUIs and in parts of the scheduling system, many stakeholders were unfamiliar with OOP and were wary of its applicability to mission critical systems. Of particular concern was the idea that the different instrument models could share code. It was thought that this would lead to the instrument models incorrectly inheriting the behavior of other instruments.

The use of search technology raised multiple concerns. First, system stakeholders were not used to looking at ground system components as optimization systems. Second, they were concerned about ensuring the quality of the output. How can we test a system where the requirements do not explicitly define the desired system output? The use of constraint sequencing brought out two stakeholder concerns. Would the code be efficient enough to execute? Would the constraint sequencer introduce infinite loops?

Several tactics were used to address the stakeholders' concerns. A multi-disciplinary team was put together to address technology as well as project development issues. We also developed a prototype on a vastly simplified set of

requirements. A former TRANS developer who was not vested in the project was drafted to define a set of simplified requirements. The requirements were coded using the high level TransVERSE design. The prototype used search and was able to use the core of the constraint sequencing code as it was already coded. The code was demonstrated and reviewed. Prototyping had positive results. It demonstrated that the technologies could perform in the desired domain. Search could find good orbit packings in reasonable run-times. Constraint sequencing was efficient and did not result in infinite loops. Mistakes made in the prototype were noted and were actively avoided when actually developing TransVERSE.

The phase B study culminated in an external project review. Domain experts were brought in from other organizations and from other parts of the Space Telescope Science Institute. The full day review covered the design of the system as well as the project development strategy. Although some concerns were raised, the external committee validated the use of advanced technology.

### **Project Development Strategy**

One major controversy in the phase B study was deciding on a strategy for implementing the project. Several options were discussed including:

- Implementing the whole system and having one big release at the end of the project (i.e. the Big-Bang approach).
- Implementing the system a science instrument at a time. There are typically 4-5 instruments operational in TRANS at any one time. After the initial release there would be two versions of TRANS. One with the old code and one with the new instruments. Subsequent releases would add instruments to the new code.
- Implementing the system in a series of releases where each release replaces specified functionality in the existing code with the new TransVERSE code.

The argument for the first two approaches is that there would be no cost for integrating with the existing baseline system as the new and old systems are independent. Also, it was claimed that these approaches would lessen testing costs as we would only have to test once. Most project stakeholders wanted one of these two approaches and the development team the last approach. Ultimately the decision to proceed with the project and the phasing approach used was determined by a management review team including representatives of operations, development, and engineering. The first two approaches were not selected for the following reasons. First, they provided no risk mitigation. Until the big release or the first instrument release there would be no feedback on whether or not the approach would work. Second, these approaches did not provide any integration with on-going

TRANS development and maintenance. Significant development for new instruments was planned during the TransVERSE time-frame. The phased approach did not have either of these problems. In addition, it was not clear that a Big-Bang delivery approach would lower testing costs. When making small changes to the code, it is fairly easy to know how changes in the code impact behavior. When changing everything at once, changes in different parts of the code mask and interfere with each other.

The project was divided into 11 3-6 month phases. Each phase would integrate new functionality with the existing TRANS infrastructure. The initial project phases were grouped as follows:

- Phases 1-3 Exposure expansion and pointing
- Phases 4-6 Science Instrument Modeling
- Phases 7-8 Input and Output Modules
- Phases 9-11 Search and orbit layout capabilities.

Each phase was divided into requirements, design, code, test, and install steps.

Project planning also considered the use of revision control tools and procedures. TRANS uses CVS to handle revision control. All TransVERSE coding was done on a CVS branch separate from the main development branch. On completion of each phase the TransVERSE branch was merged back into the main branch and given a final test. A file containing tag and merge history was maintained and proved important when trying to understand how the code evolved.

### Requirements

Development proceeded with a requirements effort for each project phase. New requirements were based on the existing outdated requirements document, examining the existing code, and stakeholder reviews. During requirements development we sometimes explicitly changed some code outputs. However, we expected the new code to produce mostly the same output as the baseline system. In this matter we were unpleasantly surprised during the testing of the first few project phases. Although the new code could be verified against the new requirements, it often produced different results than the baseline version of TRANS being used in HST operations. As the operational version had been used for many years we did not expect to see major differences. This led to the realization that instead of a simple relationship between code and requirements we had a triangle between the TransVERSE code, the existing baseline TRANS, and the new requirements (see Figure 5). When differences were found we needed to determine which portion of the requirements triangle was at fault:

- Does the baseline code contain implicit engineering requirements not documented in the current or old requirements?

- Does the new code have a bug?
- Do the new requirements have a bug.?

A major part of making the project successful was understanding how the implications of the triangle needed to be incorporated into the entire development process.

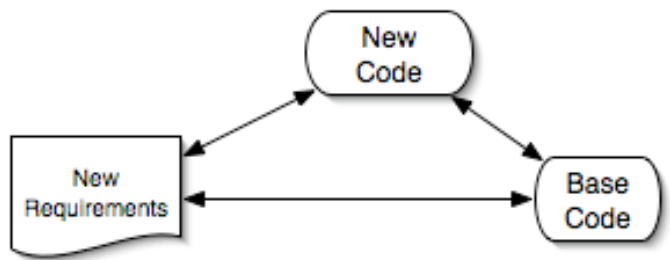


Figure 5: Requirements Triangle

### Testing

Learning how to test TransVERSE resulted in significant improvements in our internal test process and the way that the development and the test teams interact within our organization.

Prior to the TransVERSE, TRANS used an expert system tool called the looker (Gerb 1991b) for automated testing. The tool allows developers to easily encode expected test cases in a manner that is robust to code changes. For example, tests could be coded on the difference in duration of two exposures with different filters. Even if the absolute duration of the exposures changed, the relative difference should be the same given the duration of the filter move. Over eight years the system was used to encode thousands of tests. It was assumed that this system would be sufficient to ensure that the TransVERSE system produced correct results. Although the looker tool found many problems, it was not sufficient to support TransVERSE development. The test team ran massive regression suites comparing TRANS and TransVERSE on almost every observation ever executed or planned for HST. These tests found many issues that were not found by the looker test suite and were responsible for us recognizing the requirements triangle discussed in the previous section. As the project proceeded, we depended more on regression testing to find issues. To automate the process we developed tools to support difference testing. We developed and categorized large test suites of HST observations. Scientific instrument as well as other features such as target type and observation mode categorized the suites. Running large test suites required the utilization of system tools to distribute processes across multiple machines and strategies for handling disc storage. We also developed simple scripts based on the UNIX diff utility that support intelligent differencing of output files. The tools filter known differences and examine certain portions of the output products. The tools automate much

of the error-prone and tedious portions of testing. Our current focus on test tools concern how and when to baseline output products in the test suite.

In addition to changing the test tool, TransVERSE changed the way the development and test team interact. Prior to TransVERSE, there was a high barrier between the test and development groups. Our process caused part of the barrier where the development team did not involve the test team until the code was ready to test. The developer was supposed to have completely tested the system such that testers were not supposed to find any new problems. The developer designed, coded, and tested with little to no interaction with the test team until a formal ready-to-test notification was sent. In short, the test team was not considered to be part of the development team and was only involved when the code was thrown over the wall. During the TransVERSE project the relationship between the code and test teams changed. Through the shared responsibility of having to get a big project done, testers became integrated with the development team. Instead of waiting until the code was finished, testers worked on systems as soon as they were compiled and smoke tested. The test team would iterate with the development team using lightweight informal processes to identify issues and to determine which portion of the requirements triangle was impacted. Regression driven test efforts also encouraged developer and tester interaction. If the main test tool is a massive CPU intensive regression suite, it does not make sense to duplicate this effort in multiple groups. It is more efficient for testers to find the last bugs in a system than for the development team to spend their time. As the test and development teams became more integrated, communication became more frequent and less formal. Although testing hand-off procedures were still used for tracking, they became more of a formality and were no longer the focal point for team interaction.

Another important factor for testing is the code compilation and build strategy. TRANS uses a concurrent development model. At the start of a development effort the coder checks out a complete source tree called a *parallel world* from the code repository. Each parallel world can be compiled into a working TRANS system. After checkout the developer modifies and recompiles the world as needed. A TransVERSE phase might be supported by several developers each working in their own parallel world on a separate portion of functionality. This approach has several advantages over the method where testing does not occur until all the developer code is integrated. First, each portion of the functionality is tested as a separate entity. This reduces the interference and bug masking that occurs when testing multiple changes at one time. Second, each parallel world can be tested when it is ready, allowing scarce test resources to be used more flexibly than when trying to plan a single large release.

## Reviews

Several reviews were held at each development phase to investigate project issues. Design, code, and test reviews were used to identify and if possible resolve issues. A wide range of project stakeholders including coders, testers, system engineers, instrument engineers, and operators attended reviews. Inclusion of multiple stakeholders allowed us to identify which portion of the requirements triangle needs to be fixed to resolve an issue. These reviews worked best when they were attended by peers and concentrated on finding issues and not distributing blame. When possible, issues were resolved in the meeting. Possible resolutions include deciding an issue is a non-issue, deciding that a particular group needs to fix the problem, or deciding that the problem can be a lien on the release. Some issues could not be resolved in the group of peers.

Management reviews addressed larger issues that could not be agreed upon by the greater development team. For example, nearly every project phase had a release review to certify that the code was ready for operations. There were also multiple occasions where a formal review was held to determine if the project should continue.

## Project Phasing Again

The project phases were changed multiple times in response to our understanding of the project and external factors.

After completion of the first three phases the project was behind schedule. Several factors contributed to the schedule overruns. First, the quality standards that the TransVERSE code was being held to were higher than expected. The new code had to not only re-implement the existing functionality but also to remove any existing bugs. Second, many of the stakeholders had not fully signed on to see the project through completion. Both of these factors map directly to parameters that can linearly impact the time for a work item as estimated using common tools such as Cost Xpert. Based on the overruns in phases 1-3 the development schedule was rebuilt. The remaining phases were completed on schedule with a few minor exceptions.

Starting with phase 4, the project moved towards breaking down the phases into more manageable releases. Phases 1-3 each resulted in a single operational release. The size of these releases contributed to schedule issues. This was especially true in phase 3 where we added significant new functionality. Phases 4-6 were each subdivided into separate sub-releases for each scientific instrument. Each sub-release was designed, coded, tested, and released to operational as a separate entity. The first sub-release for a phase would build infrastructure to be used in all of the

sub-releases. Development for the remaining science instruments could be done in parallel after the infrastructure was put in place. This approach greatly simplified testing and reviews as it removed situations where one issue masks another. This approach also simplified development by allowing for an easy way to divide the work among developers.

After completing phases 4-6, the project phases were shuffled again. This time the motivation was to fit in with the APT development schedule. Supporting APT required completing the search and orbit layout phases of the project (i.e. 9-11). APT requires that TRANS produce output in ~10 seconds for a typical case. The existing code required minutes. Phases 7 and 8 were not needed for APT and represented a small risk to the project, as they were similar in scope to phases 4-6 which were already completed. In contrast, the search phases required the use of new technology and thus represented the greatest risk for project failure. The project was reorganized so that phases 9-11 were collapsed into two phases and moved forward to support APT. The TransVERSE project was declared finished after delivering operational support for APT in 2003. The remaining work in phases 7 and 8 were broken down into small releases and were done as part of routine TRANS maintenance. The maintenance work was completed in 2004.

The TransVERSE project phases can be summarized as follows. In phases 1-3 the team made mistakes. In phases 4-6 we learned from the mistakes but still did not have complete project buy in. In phases 9-11 we had a clear goal of supporting APT and was able to make the project a success. A key lesson here is to be flexible with respect to project phases. Build the ability to realign your project goals into your plans and schedules.

### **Search Capabilities**

The baseline version of TRANS had two competing factors when structuring an observation. On one hand, TRANS determined the number of orbits an observation requires. On the other hand, TRANS can shrink or expand the duration of exposures by up to 20% to fit observations into orbits. There was no explicit preference to use more or less orbits. What should the new code do? Should it shrink exposure durations to fill the least number of orbits or should it expand exposure durations to fill the most number of orbits. Fortunately at this point the decision to support the APT interactive proposal tool changed the success criteria of TransVERSE. In an interactive tool TRANS could move from a decision making tool to a mixed initiative decision support tool. Astronomers using the tool need to be able to specify the number of orbits an observation can take. To support this in APT, TRANS provides the following capabilities. TRANS can be told which exposures belong in which orbit. If TRANS is not told the orbit mapping (either on the first run or after a

reset) the system is free to put exposures into the orbits it chooses. After an initial run the orbit numbering is fed back into APT. From here, the user can adjust where the orbit breaks occur. Once exposures are fixed in orbits the astronomer typically wants to ensure that the science utilizes as much of the orbit visibility as possible. To support this capability TRANS provides the ability to auto-adjust the exposure durations within a single orbit. The system includes the ability to give the maximum adjustment for each exposure. When given the command TRANS determines whether the orbit is under or over filled and attempts to adjust the exposure durations evenly to exactly fill the orbit. The auto-fit capability gives the right tools to APT users. They can specify their orbits and can fill up the orbits without having to iterate multiple times selecting different exposure durations. This approach also simplified TRANS development by removing a free variable in the orbit packing equation.

### **Testing Search**

Even with the reduced search space, the requirements do not deterministically dictate a single output. TRANS still had to select the best output out of many possible outputs. This presented serious challenges in validating the system. In previous project phases simple difference testing could identify changed behavior and it was relatively straight forward to hold reviews to determine whether or not the new behavior was desirable. In the search phase we expected more differences than could be reasonably handled by a formal review process. The final decision as to whether or not an output was satisfactory was often a human tester examining the graphic output of the system. Does the output look right? Does the output efficiently utilize scarce HST resources? Tools were developed to help the test team to triage the output. The tools would compare the baseline and the new output to determine:

- If the output is the same or similar enough to the baseline to not care about.
- If exposures have moved between orbits.
- If the merging of exposures into the scheduling-hierarchy had changed.

Even with these tools the key enabler for this test effort was a high caliber test team. The test team was familiar with not only how to use the proposal preparation tools but also with the operation of HST instruments. The test team was quickly able to weed out non-issues and to concentrate on real problems with the new code. Working in tight coordination with the development team, issues could be fixed right away or be prioritized for later work based on a project review.

Two other factors helped the test effort. First, the auto-fill capability was tested as a separate operational release. Second, during later phases of the test effort the larger in-house HST user community was brought in for beta testing.

## Conclusions

In the end, the TransVERSE project successfully updated TRANS to use new technology. The main portion of the project completed in 2003 with some clean-up finishing in 2004. The TRANS maintenance level has decreased to less than a full developer from a high of four developers before TransVERSE. The system is stable and is easy to modify. The process of getting there was not easy. The major lessons learned in the project are:

- Team early and often with project stakeholders. Integrate your teams for all project development efforts including requirements, design, coding, testing and reviews.
- Work on the system one small step at a time. Each step should result in a working system. Avoid the Big-Bang release. Use development methodologies and tools that support incremental changes.
- Plan for project change. The goals for a multi-year project will change along the way. Embrace the change and let it make a better system.

Hopefully hearing the struggles and victories of this project will be useful the next time you bring new technology into mission critical portions of a space mission.

## Acknowledgements

The author would like to thank the reviewers as well as Reiko Rager, Christine Ritchie, Wayne Bagget, and Gary Curtis for their useful comments. I would also like to thank all TRANS stakeholders past and present for their efforts.

## References

Curtis G., Donaldson T., Douglas R., Gerb A., Giuliano M., Nigro N., 1998. "TransVERSE an Architecture for Configuring Astronomical Observations," In *Proceedings of Astronomical Data Analysis Software and Systems VIII*, 73-76, Urbana Illinois.

Curtis G., Giuliano M., 1999. "COSI: Adding Constraints to the Object-Oriented Paradigm," In the *Proceedings of the LISP User Group Meeting*. San Francisco, California.

Gerb A. 1991a. "Transformation Reborn: A New Generation Expert System for Planning HST Operation," In *Proceedings of the 1991 Goddard Conference on Space Applications Of Artificial Intelligence*, ed. J.L. Rash, NASA Conference Publication 3110 (Greenbelt NASA), pp. 45-58, reprinted in *Telematics and Informatics*, 8, 283-295.

Gerb A. 1991b. "The Looker: Using an Expert System to Test an Expert System," In *Proceedings of the 1991 World Congress on Expert Systems*, pp 1005-1012, Goddard Space Flight Center, Greenbelt MD.

Johnston M. and Miller G. 1994. "Spike: Intelligent Scheduling of Hubble Space Telescope Observations," In Zweben M. and Fox M. eds. *Intelligent Scheduling*, 391-422. Morgan-Kaufmann.

Kiczales, G. et al. 1991 *The Art of the Metaobject Protocol*. The MIT Press. ISBN 0-262-61074-4.

Kowalski, R. 1979 *Logic for Problem Solving*. North Holland, Elsevier Science Publishing Co., Inc.

Krueger, T. 2004. "APT: Hubble Space Telescope's Proposal Preparation Tool," In *Astronomical Data Analysis Software and Systems XIV*.

Rosenthal D., Monger P., Miller G., Johnston M. 1986 "An Expert System for the Ground Support of Hubble Space Telescope," In *Goddard Conference on Space Applications of Artificial Intelligence*, Goddard Space Flight Center, Greenbelt, MD.