BIO
PRESENTATION
PAPER

T3

11/17/2005 10:00 AM

A Flight of Fancy - The Evolution of a Test Process for Spacecraft Software

Brenda Clyde
Johns Hopkins University

International Conference On
Software Testing Analysis & Review
November 14-18, 2005
Anaheim, CA USA

Brenda Clyde

Brenda A Clyde joined the Johns Hopkins University Applied Physics Lab in 1984 after earning a BS in computer science from Mary Washington College. Early in her career Ms. Clyde was responsible for the implementation of software to qualify, reduce, store and display telemetry data from the Trident II missiles and re-entry bodies. In 1990 Ms. Clyde earned a MS in Computer Science from Johns Hopkins University. Ms. Clyde was promoted to Senior Professional Staff in 1991 and had the responsibility for managing the maintenance of existing software and the development of additional software to be used in reducing and qualifying submarine data. In 1994, Ms. Clyde began work on the Commercial Vehicle Information Systems and Networks (CVISN) project as a systems analyst responsible for planning, implementing, and testing various prototype CVISN systems.   In 2002, Ms. Clyde transferred to the Space department and joined the test team for the MESSENGER spacecraft.   Ms. Clyde is currently leading the independent software acceptance testing for the New Horizons spacecraft.

A Flight of Fancy – The Evaluation of a Test Process for Spacecraft Software

Brenda A. Clyde
The Johns Hopkins University Applied Physics Laboratory
11100 Johns Hopkins Road, Laurel, MD USA 20723-6099
E-mail: Brenda.Clyde@jhuapl.edu

Session Objectives

- Need for Study
- Mission background
- Test Process Description
- Test Process Evolution
- Test Process Evaluation
- Lessons Learned
- Recommendation
- Conclusion

The Need for A Study

- The test process began as a guideline and is continuing to evolve into a defined and effective process.
- This evolution took place over the course of five years and the development of flight software for four spacecraft.
- The approach to testing needed to change with each mission, as resources were over-extended and schedules were compressed.
- Several changes to the process were attempted to achieve better cost effectiveness with little or no improvement.
- More formal and objective techniques were used in this study to identify weaknesses and recommendations for change.
- This study is captured in the paper "*The Evolution of a Test Process for Spacecraft Software*", D. A. Clancy, B. A. Clyde and M. A. Mirantes.

A Journey to New Frontiers

- **COmet Nucleus TOUR (CONTOUR)** - CONTOUR’s objective was to increase our knowledge of key characteristics of comet nuclei and to assess their diversity by investigating two short period comets. CONTOUR was launched July 3, 2002.
- **MErcury Surface, Space ENvironment, GEochemistry and Ranging (MESSENGER)** - MESSENGER’s mission is to orbit Mercury and perform a focused scientific investigation to answer key questions regarding this planet’s characteristics and environment. MESSENGER was launched August 3, 2004.

A Journey to New Frontiers

- **Solar-TErestrial RElations Observatory (STEREO)** - STEREO’s mission is aimed at studying and characterizing solar Coronal Mass Ejection (CME) disturbances from their origin through their propagation in interplanetary space and their effects on the Earth. STEREO is scheduled to be launched in 2006.
- **New Horizons** would seek to answer key scientific questions regarding the surfaces, atmospheres, interiors, and space environments of Pluto, Charon and the Kuiper Belt Objects. NASA proposed to launch New Horizons in 2006.

A Look at Mission Differences & Complexities

Mission	Number of flight software requirements	Number of external interfaces	Number of science instruments	Lines of Code	% Software Reuse
CONTOUR	690	12	4	37893	30%
MESSENGER	1035	19	7	143121	30%
STEREO	1422	15	4	126054	15%
New Horizons	1074	12	7	145618	35%

Evolving the Test Process

2000 2001 2002 2003 2004 2005 2006

Feb 2000 – Sep 2002

CONTOUR

Ad-hoc planning
Limited Influence Prior to Code/Unit Test
Manual Testing
Formal review for Complete Deliverables
Dedicated Resources
Requirements based testing

Oct 2001 – Jul 2004

MESSENGER

MS Project & Excel
Limited Influence Prior to Code/Unit Test
Automated Testing
Formal review for Complete Deliverables
Dedicated Resources
Requirements based testing *& Test Like You Fly*

Jan 2002 – Dec 2005

STEREO

MS Project & Excel
Limited Influence Prior to Code/Unit Test
Automated Testing
Formal review for *Partial* Deliverables
Dedicated Resources
Requirements based testing & Test Like You Fly
Prioritized Test Cases

Nov 2002 – Dec 2005

New Horizons

MS Project & Excel
Limited Influence Prior to Code/Unit Test
Automated Testing
***Informal* review for Partial Deliverables**
Dedicated Resources
Requirements based testing & Test Like You Fly *Concurrently*
Prioritized Requirements

Using a 10 Step Process For Testing

Step 1 **-** Planning the Testing Effort

Step 2 - Evaluation of the Requirements

Step 3 - Create the Test Plan Outline

Step 4 - Define the Test Cases

Step 5 - Review Pieces of the Test Plan

Step 6 - Implementation of the Test Scripts

Step 7 - Execution of the Test Cases

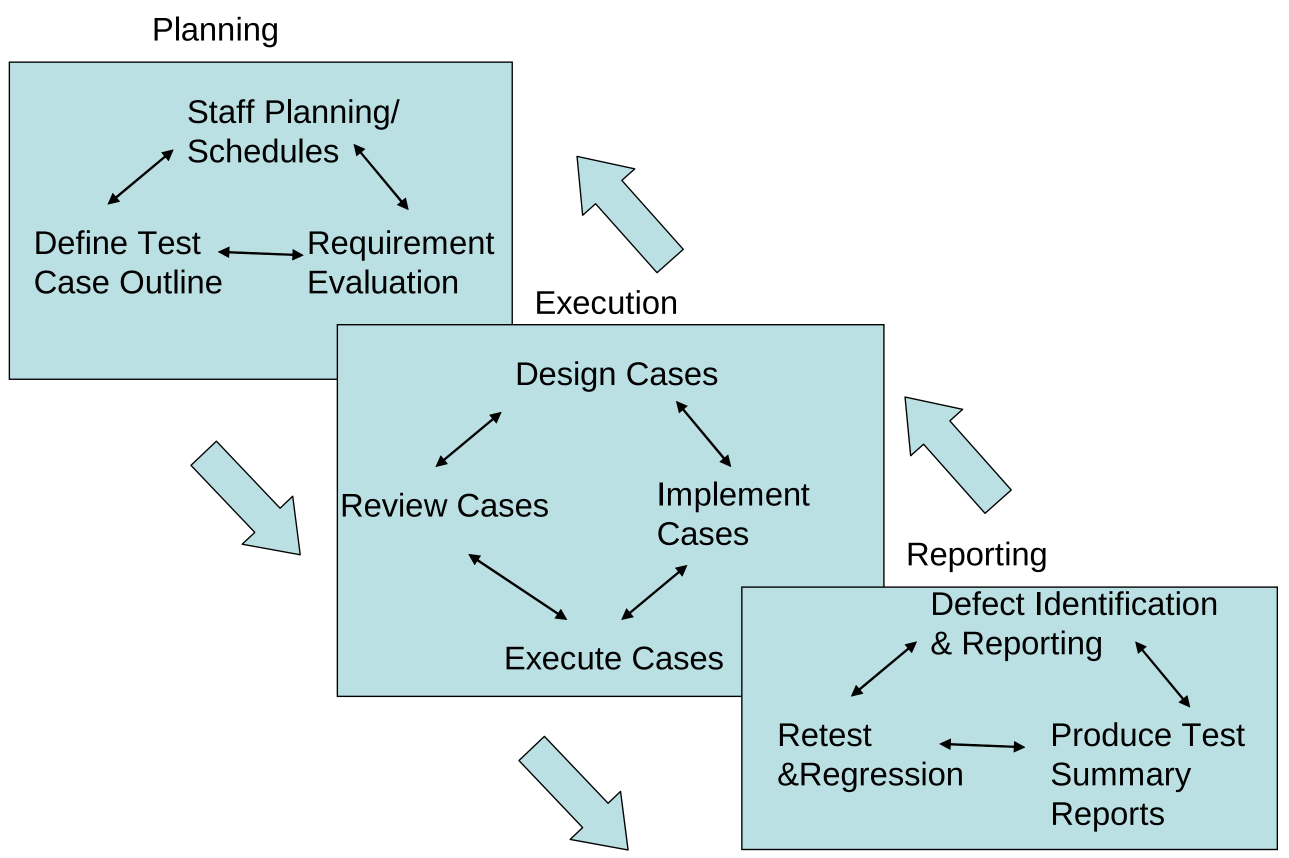
Step 8 - Defect Creation & Tracking

Step 9 - Maintain and Report Test Status

Step 10 - Create Test Report Summary

An Iterative and Incremental Test Process

Current Process



Identifying Test Process Weaknesses

- Test process perceived as NOT cost effective!
- Working groups were established in July 2003 & January 2004
- Some recommendations were implemented but the expected improvement in cost and schedule didn't materialize.

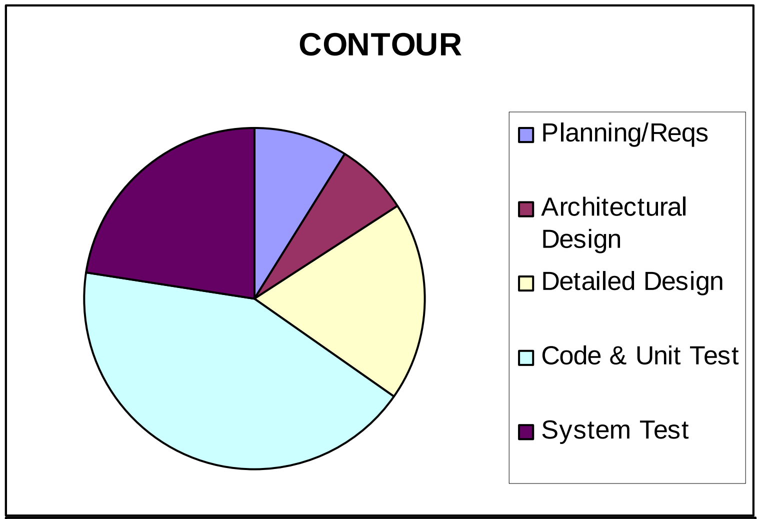
Sample Working Group Findings

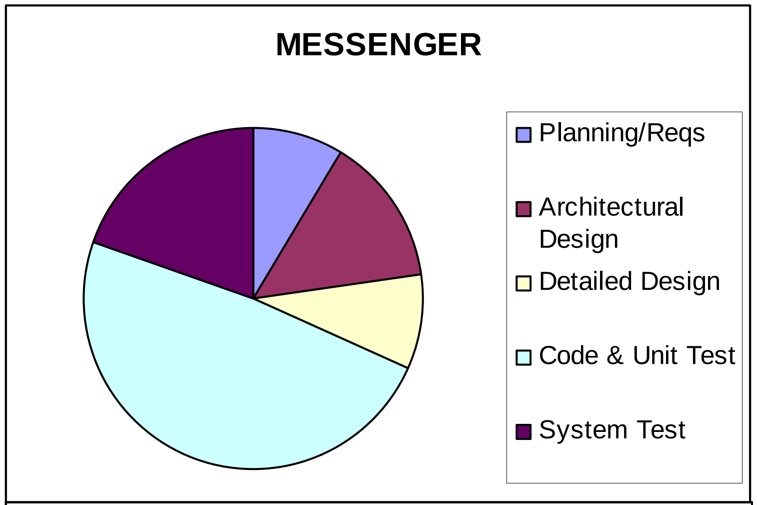
#	Factor	Mitigation
1	Contents of flight software builds often change unexpectedly which can affect the test team	•Keep test team informed of changes •Assess impact of change to test team •Focus testing on functional areas rather than builds •Delay testing of functional areas that are not complete
2	Requirements are often difficult to test using “black box” methods	•Consider alternative test methods (e.g. , inspections, analysis, inference and others) •Consider having development team verify these requirements during unit and integration testing •Assure that requirements do not contain design material
3	Requirements contain design information or contain too much detail	•Change requirements to remove design detail •Assure that requirements review panel contains test representative
4	Flight software and testbed documentation not available when needed	•Prioritize the needs of the test team. •Put milestones in schedule for required documentation •Don’t deliver build without required documentation
5	Late changes to requirements can affect test team	•Keep test team informed of changes (use Change Request system) •Assess impact of change to test team before approving it

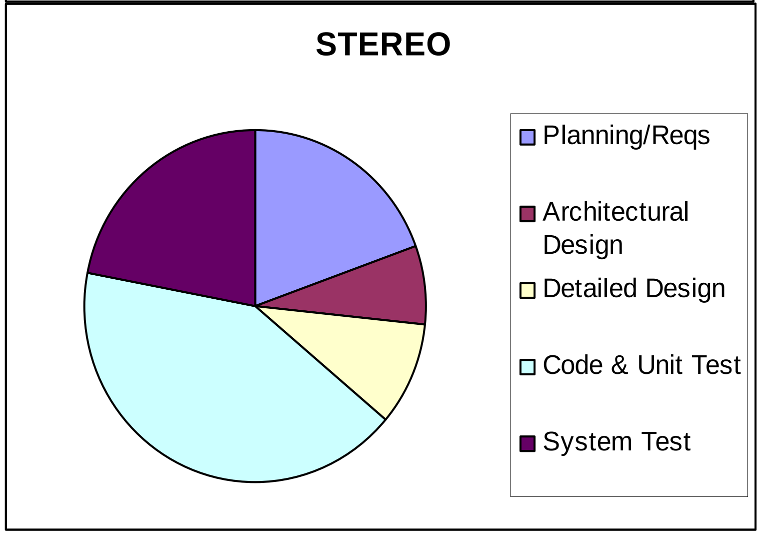
Using Metrics to Evaluate the Process

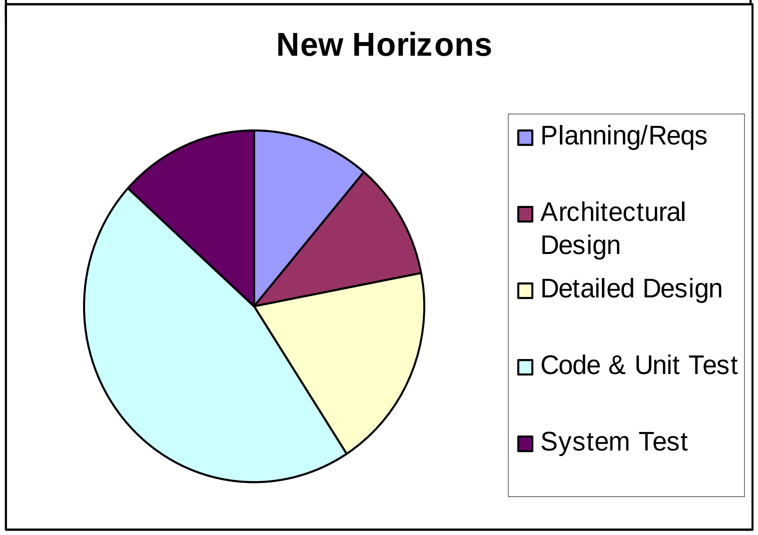
- Metrics used to evaluate the process include:
  - Percentage of overall development life cycle
  - Planned vs. actual staff months
  - Staff experience
- Metrics for Defect Removal Efficiency could not be computed due to weaknesses in data gathering for when the defects were found as well as by whom the defects were discovered.

Percentage of the Development Life Cycle







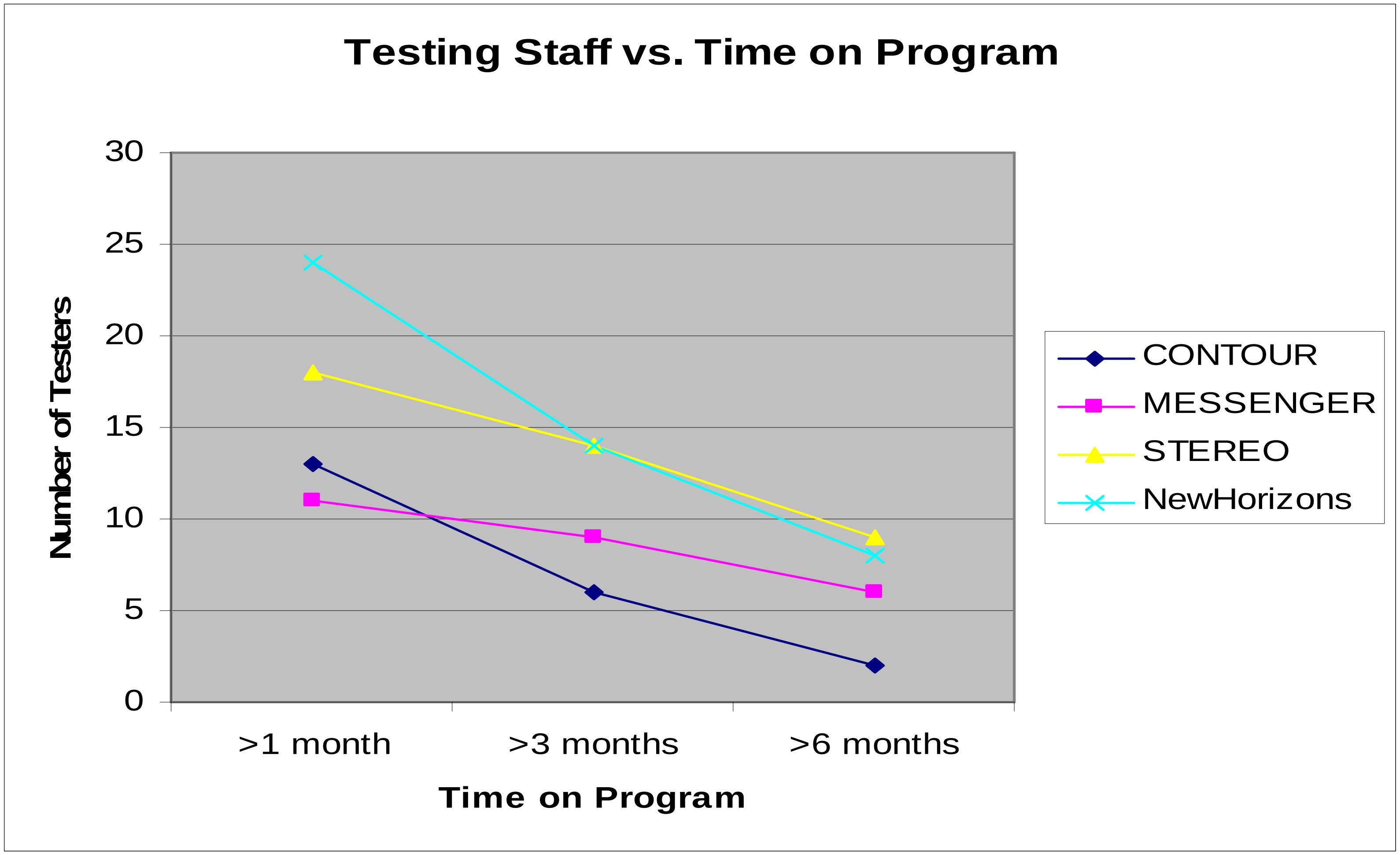


Percentage Relative to Estimate of Costs For Software Testing

Mission	Percentage Relative to Estimate: (Actual – Planned)/Planned $\times$ 100
CONTOUR	62%
MESSENGER	93%
STEREO	35% *
New Horizons	–18% *

* Based on effort to date.

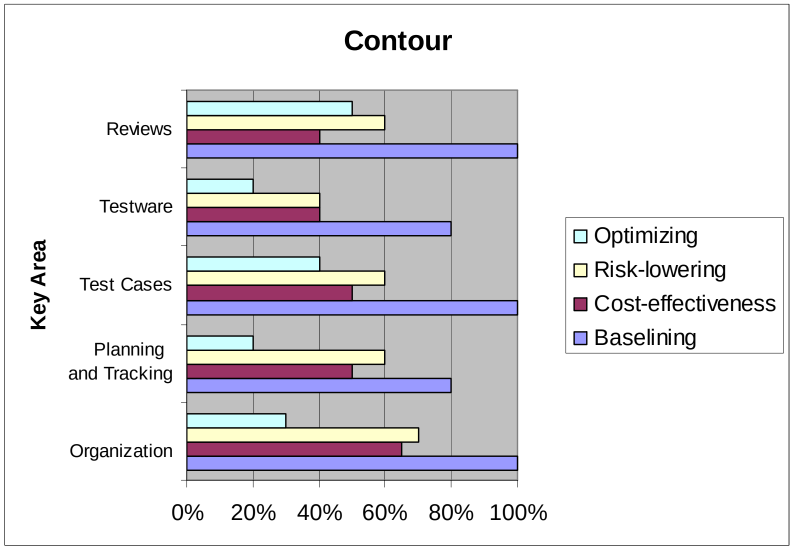
Staff Experience

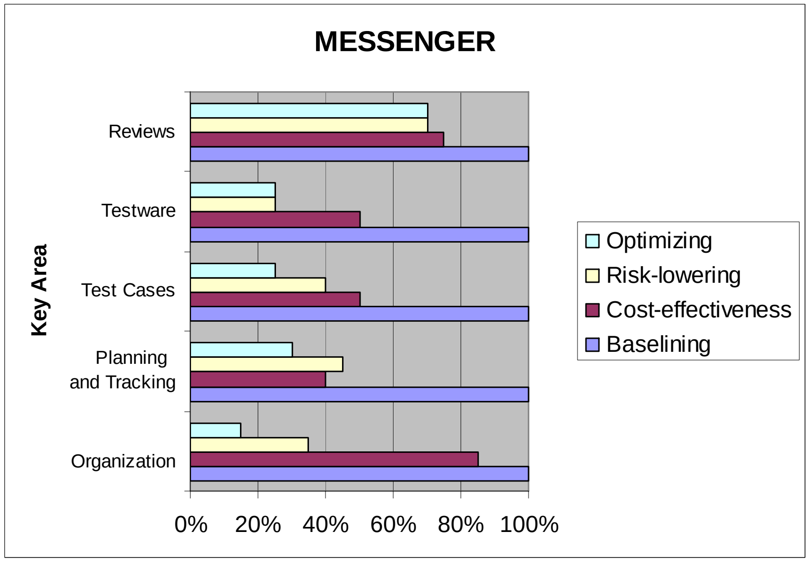


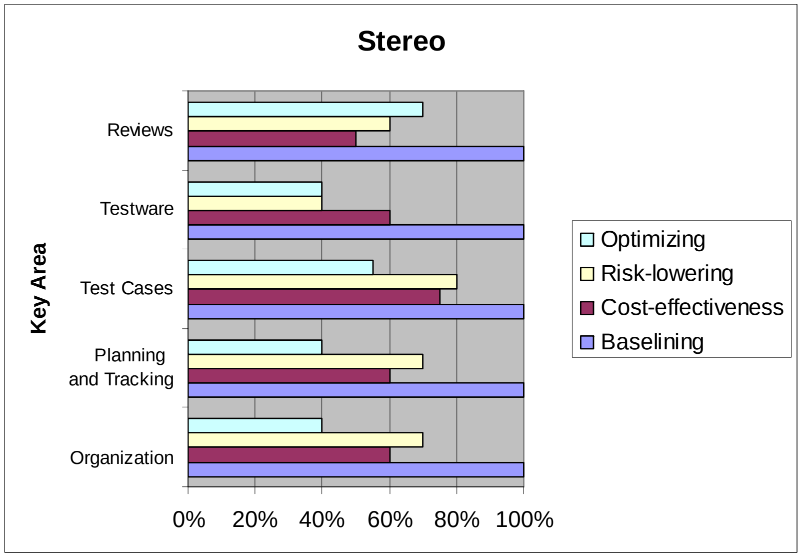
Using the Test Improvement Model (TIM)

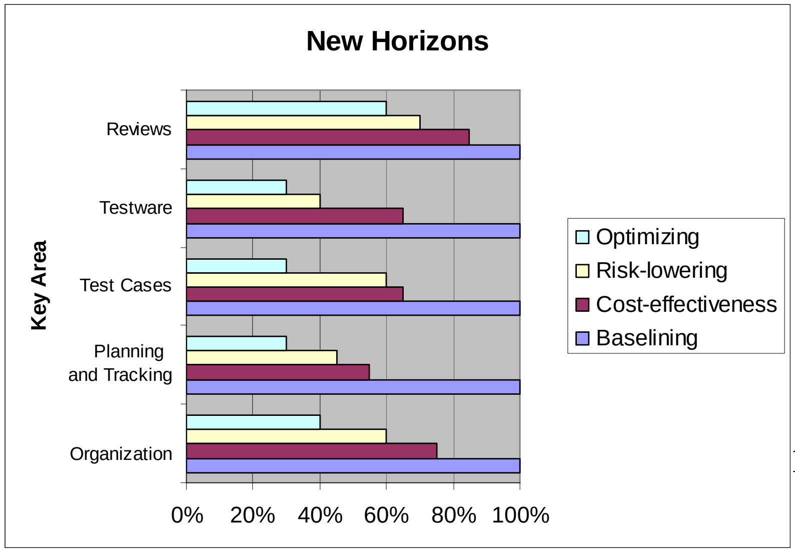
- To provide an independent assessment of our test metrics, we used the Test Improvement Model (TIM)
- This model looks at five key areas and allows the analyst to assign a current level in each area.
- The five key areas are: Organization, Planning and tracking, Test cases, Testware and Reviews.
- The four levels for each area are: Baselining (lowest - 1), Cost-effectiveness, Risk-Lowering and Optimizing (highest - 4).

Rating the Mission Specific Test Processes









TIM Results

- **Organization:** To move the next program into the cost-effectiveness level and above, we need to focus on training and building a core group of full-time testers.
- **Planning and Tracking:** To achieve Cost-effectiveness, the future test programs need to have flexible planning and a standard method for tracking.
- **Test Cases:** To achieve Cost-effectiveness, the future programs need to allow the ability for testability to influence requirements and design.
- **Testware:** To progress to the next levels, resources will have to be applied to evaluate and develop or purchase test tools to gain cost-effectiveness and lower risk.
- **Reviews:** In order to complete the optimizing level, investment needs to be made in training and adding different review techniques to our skill sets needs to be examined.

Ten Lessons Learned

- **Invest in testing –** Train a core group of testers, invest in tools, integrate testing earlier into the development process (better communication).
- **Increase focus on cost-effectiveness –** In the TIM assessment, all programs made some attempt to fulfill requirements in the risk-lowering areas, but not as much focus was placed on cost-effectiveness. Our business makes us risk adverse, and risk management is built into our processes. However, a better balance is needed between cost and risk in our testing effort.
- **Use metrics for process improvement –** Metrics are currently used to show progress on specific efforts. In addition, metrics need to be used to improve the process for future missions. For example,
  - Show cause and effect of late delivery of a functionality
  - Show overlap between deliveries and planned regression tests
  - Analyzing the impact of late or incomplete software deliveries more effectively
- **Manage resources purposefully –** Commit resources to testing and stand behind the commitments.
- **Scope the effort** – Have a plan that does not treat all requirements equally, and get it approved.

Top Ten Lessons Learned (Cont'd)

- **Involve the development team –** Make verification a joint effort between the development and test teams.
- **Leverage all test efforts –** Leverage the testing efforts of the developer, the Comprehensive Performance Test (CPT), Mission Simulations (MSIM), etc., to reduce duplication and increase effectiveness.
- **Track changes to test documentation and test tools –** Enter Change Requests (CRs) for changes to test plans that result from reviews and from actual implementation of the test. Our current process calls for changing the documentation to reflect the as-built software, but not all changes are tracked.
- **Plan for test case re-use –** Start with the objective of re-using case designs and scripts from mission to mission and across CSCs.
- **Plan for "waiting" time –** Find ways to effectively use time spent waiting for software deliveries and testbed resources.

Recommendations for Change

- Involve experienced testers heavily in requirements review, and make their membership on the requirements review panel mandatory.
- Carefully review requirements to assure testability and to filter out extraneous design information.
- Perform risk analysis to determine whether functional areas will be tested via scenario testing or traditional requirements-based testing.
- Build up scenario-based tests iteratively and incrementally.
- Maintain all test plans in Requirements Tracking Tool, link them to requirements, and capture test methods.
- Use scenarios as the basis of regression tests; use automation for cases to be included in regression; and do the remaining testing manually.
- Build and review test cases and documentation incrementally. Track changes using a COTS version control system.

Recommendations for Change (Cont'd)

- Gather and use metrics throughout the test cycles to allow dynamic process adjustments when needed.
- Plan for re-use when developing test cases, and look for ways to simplify them.
- Use verification matrix and test methods to identify where other test efforts can be leveraged to reduce duplication of tests across CSCs.

Conclusion

- The test process originally was a guideline.
- Those guidelines have become a defined, useable and valuable test process.
- The test process can be improved by continuing to identify weaknesses and implement changes to address the weaknesses.
- Changes needed are identified through evaluation of the test process.
- Test process will be re-evaluated routinely to determine effectiveness of changes and to continuously improve.

The Evolution of a Test Process for Spacecraft Software

Deborah A. Clancy, Brenda A. Clyde, M. Annette Mirantes

The Johns Hopkins University Applied Physics Laboratory
11100 Johns Hopkins Road, Laurel, MD USA 20723-6099
Deborah.Clancy@jhuapl.edu, Brenda.Clyde@jhuapl.edu, Annette.Mirantes@jhuapl.edu

Introduction

Like most starts at establishing an engineering methodology, our effort to establish an effective and efficient methodology for testing spacecraft software began as a set of guidelines and is continuing to evolve into a defined and effective process. After the success of several missions for NASA and the Department of Defense, in 2001 we determined that the Johns Hopkins University Applied Physics Laboratory (JHU/APL) Space Department software group was growing large enough that reorganization was necessary. As a result, the group was divided into three separate entities, each with its particular focus. The charter for one of these groups was the development of embedded software for unmanned spacecraft. The software development process at that time was a set of guidelines—recommendations with broad statements about how to develop and test flight software. Over the course of five years and four full spacecraft development efforts (some complete, some still "in the works"), those guidelines are becoming a defined, useable, and valuable process.

This paper focuses on our experience with the process for testing flight software. We currently have a process for testing flight software, but it is less cost-effective and efficient than we would like. To improve the effectiveness and efficiency, we did an evaluation using process metrics. We identified problems and areas where process improvements can enhance the efficiency and cost-effectiveness of flight software testing at JHU/APL. We studied the software testing process used on four recent and current missions and have used the results to recommend process improvements. After a brief description of the four spacecraft missions, this paper presents our initial approach to testing and describes how that approach changed with each mission as resources became overextended and schedules were compressed. The next section provides details about the approach we took in conducting an evaluation of the process. The final section of the paper presents the changes we believe will have the most significant impact on making our testing efforts more effective and efficient and proposes a test approach for the next mission.

Mission Overviews

Our test process was used during the flight software testing for four spacecraft missions. The missions are discussed briefly in the following sections.

CONTOUR

The COmet Nucleus TOUR (CONTOUR) was the first mission to be tested under our new process. The objective of this mission was to increase our knowledge of key characteristics of comet nuclei and to assess their diversity by investigating two short- period comets. CONTOUR launched July 3, 2002.

The flight software architecture consisted of three Computer Software Components (CSCs): Command and Data Handling (CDH), Guidance and Control (GC) and Boot. These three CSCs were also the focus of the testing effort. The CONTOUR architecture is one common to many spacecraft: one main and one backup processor for CDH, one main and one backup processor for GC, and shared Boot code. For all four missions, a 1553 bus centralized the interfaces to external subsystems, but custom interfaces were also used.

MESSENGER

The MErcury Surface, Space ENvironment, Geochemistry, and Ranging (MESSENGER) mission began its testing effort mid-way through CONTOUR's testing. MESSENGER is a scientific investigation of the planet Mercury. The MESSENGER spacecraft will orbit Mercury and perform a focused scientific investigation to answer key questions about this planet's characteristics and environment. MESSENGER was launched August 3, 2004.

The flight software architecture for MESSENGER diverged significantly from that of CONTOUR. MESSENGER had the CDH, GC, and Boot CSCs, but added a fourth CSC, Fault Protection (FP). The hardware architecture included a new processor and a new operating system. Unlike CONTOUR, the CDH and GC CSCs were located on one processor, and the FP software ran on two independent processors, one serving as backup. MESSENGER also included many new technology initiatives. Some were specific to software; others were both hardware and software. Some of these new features included a DOS-like file system for the solid-state recorder (SSR), the use of Consultative Committee for Space Data Standards (CCSDS) File Delivery Protocol (CFDP), a file-based communications protocol, an integer wavelet image compression algorithm, and a phased-array antenna control algorithm.

STEREO

Shortly after the test effort began on MESSENGER, the Solar-TErestrial RElations Observatory (STEREO) began its testing effort. The STEREO mission is aimed at studying and characterizing solar coronal mass ejections (CMEs) from their origin in the solar corona through their propagation into interplanetary space and their effects on the Earth. STEREO is scheduled for launch in 2006.

The STEREO flight software architecture was very similar to that of CONTOUR. It consisted of the CDH, GC, and Boot CSCs, but also included an Earth Acquisition (EA) CSC. The STEREO hardware architecture, like that of CONTOUR, uses one processor for CDH, one for GC, and shared Boot code, but, unlike CONTOUR and MESSENGER, STEREO had no backup processors. The processors and operating system were the same as those used on MESSENGER.

New Horizons

The final mission, New Horizons, started its test effort midway through the MESSENGER and STEREO testing efforts. New Horizons is a planned scientific investigation to obtain the first reconnaissance of the Pluto-Charon binary system and one or more Kuiper Belt Objects. New Horizons would seek to answer key scientific questions regarding the surfaces, atmospheres, interiors, and space environments of Pluto, Charon, and the Kuiper Belt Objects. NASA proposed to launch New Horizons in 2006.

The flight software architecture and most of the hardware architecture for New Horizons is identical to those used on CONTOUR, although the use of several new technologies on New Horizons increased its complexity. The new technologies used included a Flash memory SSR, a thermal control algorithm, and over 30 different science data types for recording, verifying, and downlinking.

Mission Differences and Complexities

As the mission descriptions above indicate, the four missions we studied were not sequential; there was quite a bit of overlap during software testing. Figure 1 shows the testing timeline for the four missions.

The differences among missions and their complexities varied. Table 1 lists various contributing factors to software complexity. Increased software complexity is associated with increased effort and complexity of testing.

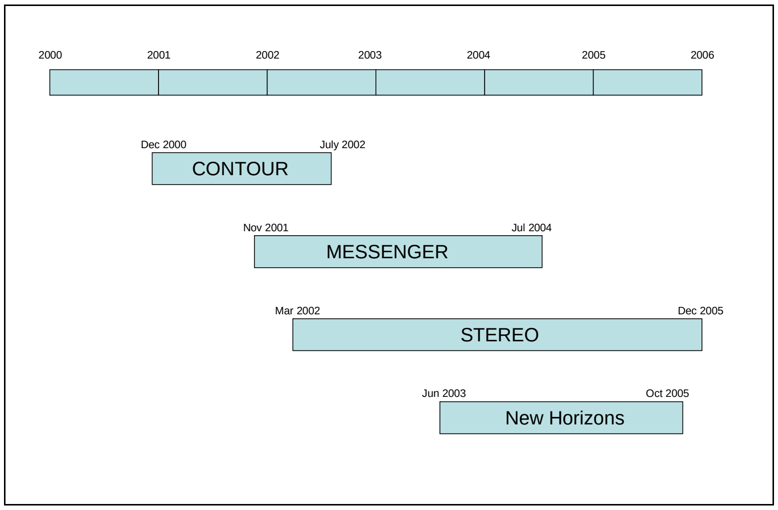


Figure 1. Software Testing Timeline for the Four Missions

Table 1. Factors Contributing to Software Complexity

Mission	Number of flight software requirements	Number of external interfaces	Number of science instruments	Lines of Code	% Software Reuse
CONTOUR	690	12	4	37893	30%
MESSENGER	1035	19	7	143121	30%
STEREO	1422	15	4	126054	15%
New Horizons	1074	12	7	145618	35%

Mission Test Processes

Starting with CONTOUR, we identified the need for more formal verification and validation and established a formal process. However, this process was above and beyond what had been baselined in the CONTOUR proposal. The CONTOUR proposal did not include plans and funding for any independent software testing to verify functional requirements. Each subsequent mission had similar issues with the requirement for increased testing; of the four missions, only New Horizons included a small effort for this formal verification and validation testing in its original plan for the mission.

Initial Test Process

The initial JHU/APL test process was based on the NASA Software Engineering Lab guidelines as specified in the *Recommended Approach to Software Development* developed by the NASA Goddard Space Flight Center (Ref. 1). CONTOUR was the first APL mission to apply this methodology. We implemented the process using dedicated test resources during the code- and unit-test phase of the development lifecycle. The efforts centered on training staff, establishing the deliverables, and verifying the functional requirements of the software. These efforts provided a foundation for future missions. Shortcomings to this process were identified. Changes were applied to the MESSENGER test process, which started the evolution. STEREO and New Horizons continued the evolution.

Evolution of the Process

Subsequent missions tailored the process to address weaknesses identified in other projects as well as to meet mission-specific requirements for testing (see Figure 2). Note: Bolded items in Figure 2 denote procedures introduced for the first time for the given mission.

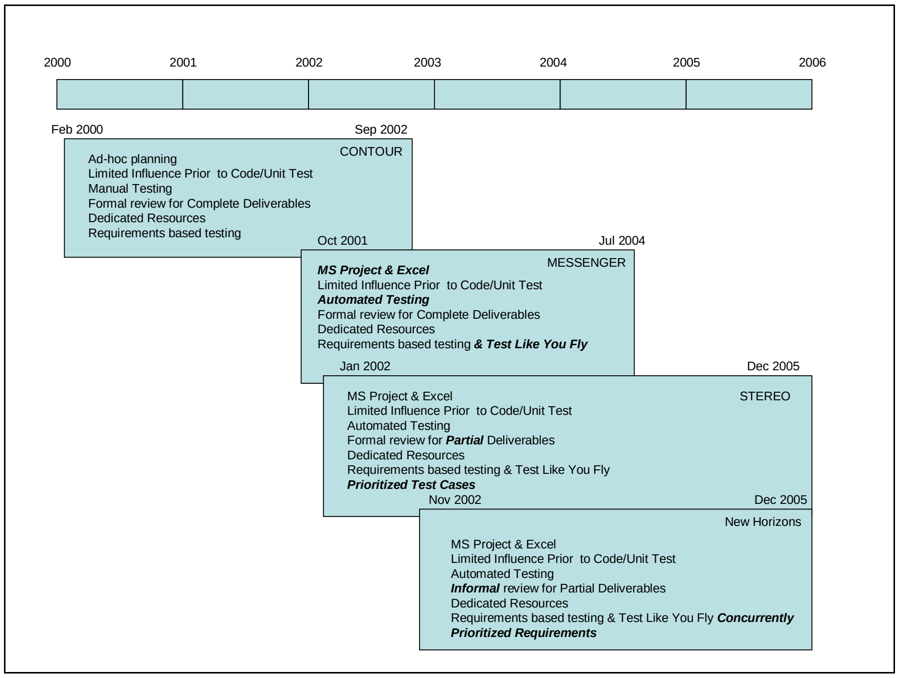

2000
2001
2002
2003
2004
2005
2006
Feb 2000
Sep 2002
CONTOUR
Ad-hoc planning
Limited Influence Prior to Code/Unit Test
Manual Testing
Formal review for Complete Deliverables
Dedicated Resources
Requirements based testing
Oct 2001
Jul 2004
MESSENGER
MS Project & Excel
Limited Influence Prior to Code/Unit Test
Automated Testing
Formal review for Complete Deliverables
Dedicated Resources
Requirements based testing **& Test Like You Fly**
Jan 2002
Dec 2005
STEREO
MS Project & Excel
Limited Influence Prior to Code/Unit Test
Automated Testing
Formal review for **Partial** Deliverables
Dedicated Resources
Requirements based testing & Test Like You Fly
Prioritized Test Cases
Nov 2002
Dec 2005
New Horizons
MS Project & Excel
Limited Influence Prior to Code/Unit Test
Automated Testing
Informal review for Partial Deliverables
Dedicated Resources
Requirements based testing & Test Like You Fly **Concurrently**
Prioritized Requirements

Figure 2. Evolution of the Software Testing Process over the Four Missions

For CONTOUR the ad-hoc planning done didn't take into account past experience or metrics since there weren't sufficient data available. The planning allocated the work to the time available and may not have been sufficiently resourced to be completed on-time.

For MESSENGER the process was tailored to use the "Test Like You Fly" (TLYF) methods described in "*Test Like You Fly" Confidence Building For Complex Systems* (Ref. 2). These methods demonstrated that the software functioned as required under realistic operational conditions and also when placed under unusual stresses. To ensure repeatability of tests, MESSENGER automated all procedures and MESSENGER also produced formal test plans for each CSC and Build pair. These test plans were formally reviewed in their entirety. MESSENGER was staffed with dedicated resources specifically for testing.

For STEREO the process was tailored to use more of a mix of dedicated and part-time resources. The test plans were incrementally generated and reviewed. The goal of incremental generation and review was to reduce the cost of generating and maintaining test documentation. Test cases were prioritized to ensure that the more complex software functions were tested first. TLYF methods and Fault Protection testing were also applied upon completion of requirements verification. Most STEREO test cases are being automated, and STEREO is performing some requirements verification through white-box or developer tests.

For New Horizons the tailoring process used more of a mix of dedicated and part-time resources. Most New Horizons test cases are automated. TLYF methods are being used concurrently with requirements verification. Requirements were prioritized to ensure that the highest-priority requirements were tested first. The test plans were incrementally generated and reviewed informally. The formality of the reviews was significantly reduced to reduce the cost of maintaining test documentation.

Current Test Process

The test process has evolved; it currently consists of 10 steps. The efforts now include steps to support increased planning and scheduling to better monitor the test progress, reduced requirements for documentation formality and maintenance, expanded use of TLYF methods, and risk-based prioritization for testing completion. Figure 3 shows the current process flow. The paragraphs below outline the basic steps.

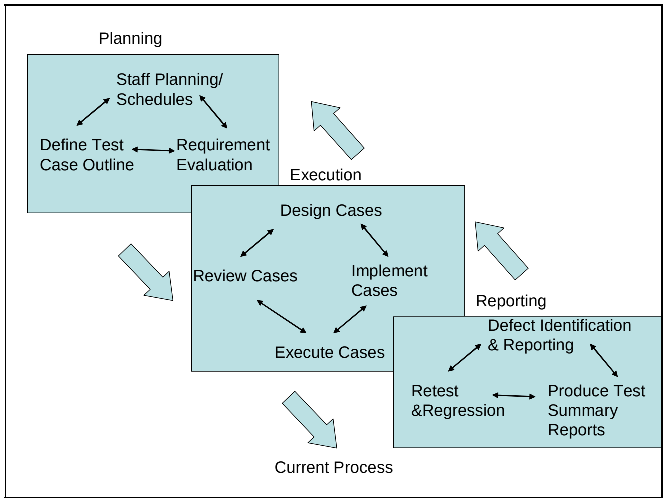


Figure 3. Current Software Testing Process

STEP 1 – *Plan the Test Effort.* The software test lead reviews the requirements, staff plan, and mission milestones. Using this material, the test lead estimates the number of test cases, using an average of 5 to 10 requirements per test case, and determines if the cases can be completed during the time available with the resources allotted.

STEP 2 – *Evaluate the Requirements.* Requirement evaluation begins with the baseline software CSC requirements, because independent acceptance test plans are written against this requirements level. All of these requirements require traceability from the Mission Software Requirements document to the acceptance test plan to show flow-down from the software system level requirements to final verification. The developer traces requirements from the Mission Software Requirements document at the definition phase of the CSC requirements, and the test engineers use this information to aid in their understanding and evaluation of the requirements.

STEP 3 – *Create the Test Plan Outline.* The introductory section of the test plan contains information such as the purpose of the test plan, test setup configurations, and any environmental criteria required to execute the test plan. The test cases are developed in outline form with placeholders for all sections. The requirements that execution of the test case will verify are traced to the test case identifier and the test case title. The identifier is a unique ID assigned to each test case.

STEP 4 – *Define the Test Cases.* A test case contains the following parts: Identifier and Name; Purpose/Objective; Inputs/Commands/Telemetry/Parameters; Steps; and Expected Results. The Test Case Number and Name uniquely identify the test case. The Purpose/Objective subsection of the test case summarizes the functionality verified. Test engineers use the Inputs/Commands/Telemetry/Parameters subsection to identify the input files and test data required to validate the test case. The Test Steps subsection provides a systematic procedure of how testing should occur. The Expected Results subsection describes the expected outcome for the tests.

STEP 5 – *Review Pieces of the Test Plan.* The acceptance test plan is peer-reviewed in segments to ensure that all requirements are being tested and that the method and approach will provide sufficient verification of the requirements.

STEP 6 – *Implement the Test Scripts*. The test steps are converted to Spacecraft Test and Operations Language (STOL) commands and included in procedures run on the test-bed for the mission. Where reasonable, the STOL procedures include code to perform automatic verification of expected values.

STEP 7 – *Execute the Test Cases.* Test engineers execute the test cases. All test material is maintained under configuration control and maintained by the software acceptance test team. During testing, a test log is maintained for each test case executed against the software. The test log allows the tester to document the day-by-day testing activities and results of each test case/procedure executed.

STEP 8 – *Create and Track Defects.* The discrepancies found during the execution of the acceptance tests are documented using a COTS defect tracking tool. The test in which the error was identified is re-executed in its entirety and the error analyzed to determine the scope and the impact to other tests that may also have to be re-executed. The Acceptance Test Team and Program Leads work together to determine the level of re-testing required when problems are identified to meet all the objectives of this Acceptance Test Plan. Defects are tracked via Change Requests.

STEP 9 - *Maintain and Report Test Status.* Status is reported weekly with a monthly roll-up. Test engineers provide the weekly status for the tasks they are currently working on. Using this status information, the test lead produces schedule, quality, and completion metrics.

STEP 10 – *Create Test Report Summary*. Test closure involves providing a report of the results of the Software Acceptance Testing activities. The Software Acceptance Test team performs black-box testing of each individual Computer Software Component (CSC) to verify that the CSC meets or exceeds its requirements. The report identifies any problems that were found and documented as Change Requests; describes the timeframe and tools used to perform the testing; and summarizes the test case execution results. The test closure report is generated at the conclusion of a test cycle.

Problems

After work began on MESSENGER, many staff both in and outside of the embedded systems group believed that the testing process was not cost-effective or efficient. In July 2003, a working group was established to discuss test process improvement. The group was a mixture of managers, developers, and testers. The goals were to use our experience with CONTOUR and the ongoing MESSENGER and STEREO projects, and use it to help the current programs finish on schedule and to plan strategy for New Horizons and future programs. After meeting bi-weekly for about 6 months, a list of problems and possible solutions was identified (see Table 2).

Table 2 - Recommendations from Test Process Working Group 2003

#	Problem	Recommended Mitigation
1	Contents of flight software builds often change unexpectedly, which can affect the test team.	a. Keep test team informed of changes. b. Assess impact of change to test team. c. Focus testing on functional areas rather than builds. d. Delay testing of functional areas that are not

#	Problem	Recommended Mitigation
		complete.
2	Requirements are often difficult to test using “black-box” methods.	a. Consider alternative test methods (e.g., inspections, analysis, inference, and others). b. Consider having development team verify these requirements during unit and integration testing. c. Assure that requirements do not contain design material.
3	Requirements contain design information or contain too much detail.	a. Change requirements to remove design detail. b. Assure that requirements review panel contains test representative.
4	Flight software and testbed documentation not available when needed.	a. Prioritize the needs of the test team. b. Put milestones in schedule for required documentation. c. Don’t deliver build without required documentation.
5	Late changes to requirements can affect test team.	a. Keep test team informed of changes (use Change Request system). b. Assess impact of change to test team before approving it.
6	Lack of communication between development and test team.	a. Encourage communication (joint meetings). b. Co-locate development and test team. c. Assign single lead for both testing and flight software.
7	Little reuse of test plans or procedures between missions.	a. Assure test team has artifacts from previous missions. b. Take steps to maintain mnemonic naming conventions between missions. c. Assess impact to testing when making changes to heritage software.
8	Developing automated procedures is time consuming and requires documentation that might not be available.	a. Reconsider the use of automated procedures. b. Automate only when it makes the effort more efficient (e.g., regression tests that will be run many times or long-duration tests that can be run during off-hours). c. Note that doing interactive testing may encourage more effort to “break the software.”
9	Feedback from test plan reviewers and Independent Verification and Validation (IV&V) may lead to greater testing than we can afford.	a. Must assess impact to cost/schedule of this feedback.
10	New test team members must climb steep learning curve before they become productive.	a. Make greater user of experienced flight software developers to test. b. Consider possible use of joint development/test team to foster greater communication and training of new team members. c. Rely less on contract employees in the future.
11	Testbed user interface is too complicated and sensitive to change.	No mitigation identified.
12	Lack of testbeds is an issue for the test team.	No mitigation identified.
13	Testbeds do not contain the functionality that is needed.	a. Need to identify missing functionality and work with testbed team to assign priorities.
1	Testing common code across proces-	a. Flight software team needs to develop com-

#	Problem	Recommended Mitigation
4	sors has not been straightforward. Common code may not have common requirements. Common code may be slightly different from processor to processor.	mand and telemetry interfaces to common code. b. Common code should be integrated by the same person to assure common approaches.

These recommendations were given to the test leads for integration into the test process for all of the ongoing missions. Some of these recommendations were implemented, but the expected improvement in cost and schedule didn't materialize. Another series of working group meetings was held in early 2004. This working group identified the problem areas and recommendations shown in Table 3.

Once again, however, the efforts didn't yield the expected improvements in cost effectiveness and efficiency. We found it very difficult to change an existing process for an ongoing program. The primary focus was on changes that could be implemented within the test process, since changing the development process requires buy-in from a larger team. Only very small adjustments to the process could be made.

Table 3. Recommendations from Test Process Working Group 2004

#	Problem	Recommended Mitigation
1	It is very time consuming to test all requirements equally by automated script.	a. Use automated test scripts for – System setup scripts – Common scripts that will be used by many testers – Instances when an artifact is required – To create regression and long-term tests – To make use of testbed "off-hours" (test can run unattended) b. Do not use automated test scripts for – Tests that won't be part of regression or long-term tests – Early builds when database and software is immature – Negative testing that will not be repeated. c. Future mission should also use some type of prioritization by risk.
2	Requirements contain too much design information, too much detail; requirements that are un-testable; entire areas of functionality with missing requirements.	a. Revisit requirements during the design phase. b. Review requirements during code reviews and during test planning. c. Provide training for writing better requirements. d. Consider having development team verify these requirements during unit and integration testing.
3	Requirements management tool being used is a complex and costly tool for requirements maintenance.	a. Alternative methods of tracking discussed, but no change made.

Evaluating the Process

Thus, we learned that to succeed in improving the cost-effectiveness and efficiency of the test process for future missions, a more objective approach was needed. We therefore evaluated the test process by looking at software process metrics. We first looked at metrics for Defect Removal efficiency, but found that this metric could not be computed reliably because of weaknesses in gathering data on when defects were discovered and by whom. We next examined the percent of effort of the life cycle. Our software process expects the lifecycle to have the partitions shown in Figure 4. The results of this metric for the four missions are shown in Figure 5.

The baseline expects 25% of the flight software lifecycle to be devoted to system testing. This data shows that all missions were within their allotted percentage of the lifecycle. However, that does not imply that they were within budget and schedule. In fact, the cost data in Table 4 tells a different story. Although actuals from previous missions are used to estimate future efforts, we continue to exceed planned costs.

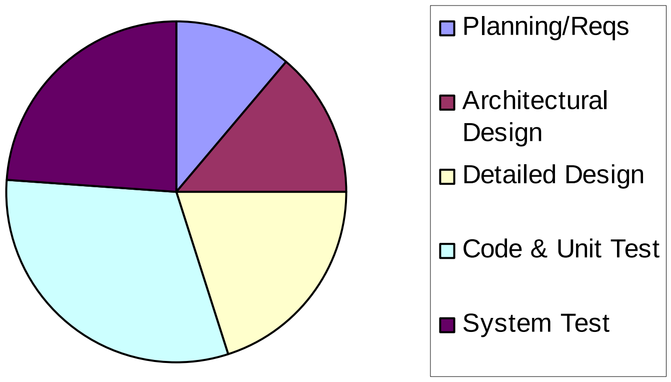


Figure 4. Software Process Baseline - Percentage of Lifecycle for Each Activity

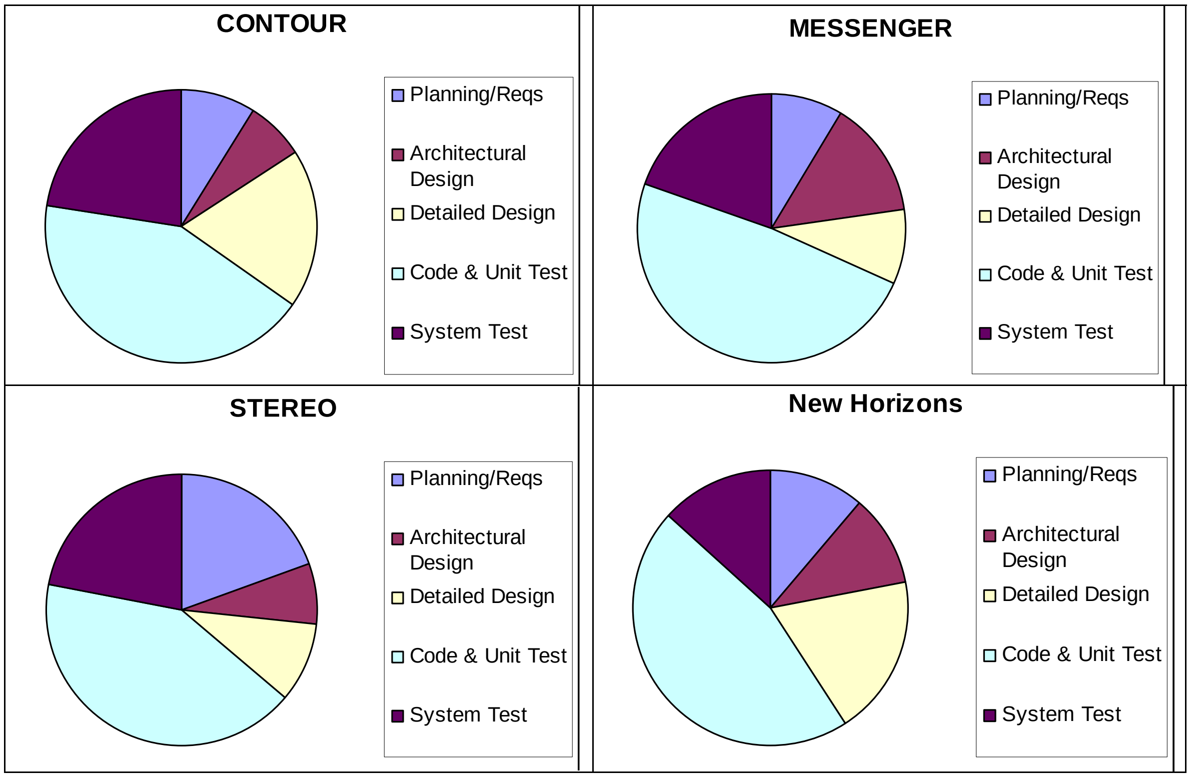


Figure 5. Actual Software Process Lifecycle Partitions for the Four Missions

Table 4. Percentage Relative to Estimate of Costs for Software Testing

Mission	Percentage Relative to Estimate: (Actual – Planned)/Planned × 100
CONTOUR	62%
MESSENGER	93%
STEREO	35% *
New Horizons	–18% *

* Based on effort to date.

Test Team Composition

Team composition is critical to performance. The cost data in Table 4 caused us to focus on staffing metrics. These provided great insight into the reasons for the cost problems shown in Table 4. Test team composition is one area where improvements can be made to achieve cost-effectiveness. The staff experience chart in Figure 6 shows that the use of part-time and contract staff is not cost-effective for three main reasons: (1) Part-time and contract staff must tackle the steep learning curve associated with testing embedded software in a complex testbed environment. Once gained, this knowledge is not being retained for use on future missions. (2) Part-time staff requires more coordination, management, and testbed resources to ensure that they are spending adequate and effective time on the program. (3) Finally, part-time and contract staff are not always available when needed.

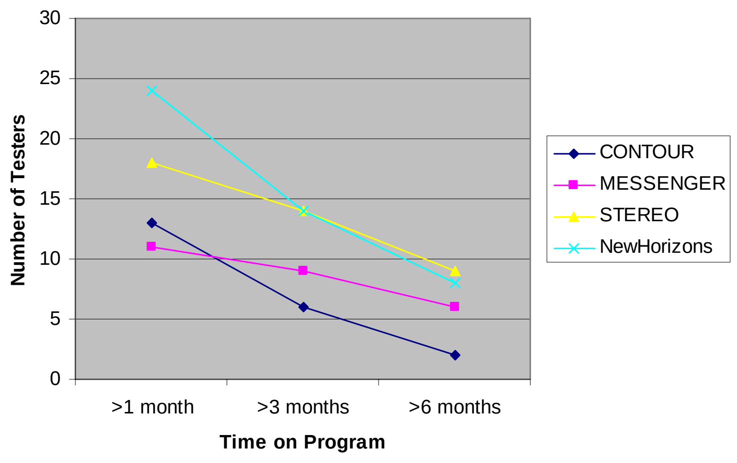


Figure 6 - Test Staff Experience.

Test Improvement Model Assessment

To provide an independent assessment of our test practice, we used the Test Improvement Model (TIM) as outlined in *TIM – A Test Improvement Model* (Ref. 3). This model has two major components: a framework and an assessment procedure. The framework has five levels (0 being the initial non-compliant level) and five key areas. Detail information on this model can be found in reference 3. In general, this framework looks at five key areas – organization, planning and tracking, test cases, testware, and reviews. The assessment is done by assigning a value based on how well we implement the items identified for each level of each key area in the framework. The assessment allows the analyst to assign a current level in each area. The four levels for each area, valued from lowest to highest, are baselining (1), cost-effectiveness (2), risk-lowering (3), and optimizing (4). Each mission's state of practice was determined using the TIM assessment procedure. People involved with each mission's test function contributed to the results given in Figure 7. A discussion of the strengths and weaknesses in each key area follows.

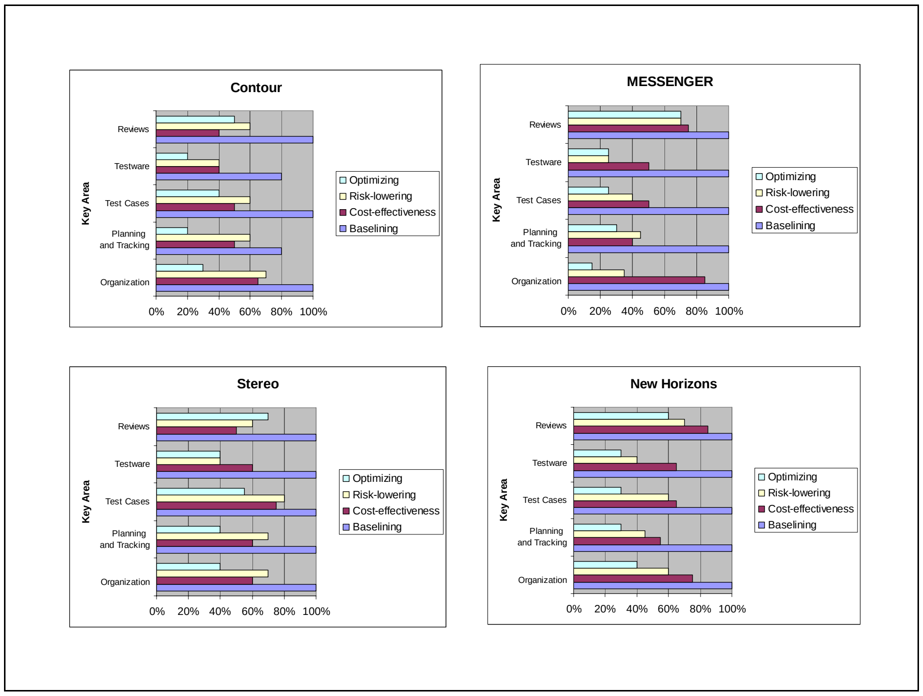


Figure 7. Ratings of the four Missions on the Basis of the TIM

Organization:

All programs fulfilled the TIM criteria for baselining, although STEREO and New Horizons did lose a number of their core, experienced testers as program priorities were realigned by department management. To move the next program into the cost-effectiveness level and above, we need to focus on training and building a core group of full-time testers. In addition, methods of improving communication between the test and development teams needs to be developed.

Planning and Tracking:

The first two programs, CONTOUR and MESSENGER, lacked some documented standards, but the subsequent programs were able to fulfill the criteria for baselining in this area. To achieve cost-effectiveness, the future test programs need to have flexible planning and a standard method for tracking. The TIM authors propose that planning should be evolutionary. This would help some of the problem areas: changing requirements, changing functionality, and availability of hardware resources.

Test cases:

All programs fulfilled the TIM criteria for baselining. To achieve cost-effectiveness, the future programs need develop the ability to allow testability to influence requirements and design. These programs will need to evaluate the testability of requirements and design and factor it into the overall software architecture and requirements definition. The test cases area did progress some into the risk-lowering level. Each program took steps to rank the criticality of either requirements or test cases or both. Significant progress has already been made toward achieving this level.

Testware:

The programs either met the criteria for baselining or were within 20% of meeting it. All programs fulfilled the requirements for baselining. However, to progress to the next levels, resources will need to be applied to evaluate and develop or purchase test tools to gain cost-effectiveness and to lower risk.

Reviews:

This is an area where we are close to the optimizing level. Each program was successful in building on the previous for each level. To complete the optimizing level for the future, investment needs to be made in training, and we should examine the possibility of adding different review techniques to our skill sets.

Ten Lessons Learned

The following are ten lessons learned from the working group results, actual experience, the review of the metrics, and the TIM assessment. They are in no particular order.

1. **Invest in testing –** Train a core group of testers, invest in tools, integrate testing earlier into the development process (better communication).
2. **Increase focus on cost-effectiveness –** In the TIM assessment, all programs made some attempt to fulfill requirements in the risk-lowering areas, but not as much focus was placed on cost-effectiveness. Our business makes us risk adverse, and risk management is built into our processes. However, a better balance is needed between cost and risk in our testing effort.
3. **Use metrics for process improvement –** Metrics are currently used to show progress on specific efforts. In addition, metrics need to be used to improve the process for future missions. For example,
    a. Show cause and effect of late delivery of a functionality
    b. Show overlap between deliveries and planned regression tests
    c. Analyzing the impact of late or incomplete software deliveries more effectively
4. **Manage resources purposefully –** Commit resources to testing and stand behind the commitments.
5. **Scope the effort** – Have a plan that does not treat all requirements equally, and get it approved.
6. **Involve the development team –** Make verification a joint effort between the development and test teams.

7. **Leverage all test efforts –** Leverage the testing efforts of the developer, the Comprehensive Performance Test (CPT), Mission Simulations (MSIM), etc., to reduce duplication and increase effectiveness.
8. **Track changes to test documentation and test tools –** Enter Change Requests (CRs) for changes to test plans that result from reviews and from actual implementation of the test. Our current process calls for changing the documentation to reflect the as-built software, but not all changes are tracked.
9. **Plan for test case re-use –** Start with the objective of re-using case designs and scripts from mission to mission and across CSCs.
10. **Plan for "waiting" time –** Find ways to effectively use time spent waiting for software deliveries and testbed resources.

Proposed Test Process Improvements for Future Missions

1. Involve experienced testers heavily in requirements review, and make their membership on the requirements review panel mandatory.
2. Carefully review requirements to assure testability and to filter out extraneous design information.
3. Perform risk analysis to determine whether functional areas will be tested via scenario testing or traditional requirements-based testing.
4. Build up scenario-based tests iteratively and incrementally.
5. Maintain all test plans in Requirements Tracking Tool, link them to requirements, and capture test methods.
6. Use scenarios as the basis of regression tests; use automation for cases to be included in regression; and do the remaining testing manually.
7. Build and review test cases and documentation incrementally. Track changes using a COTS version control system.
8. Gather and use metrics throughout the test cycles to allow dynamic process adjustments when needed.
9. Plan for re-use when developing test cases, and look for ways to simplify them.
10. Use verification matrix and test methods to identify where other test efforts can be leveraged to reduce duplication of tests across CSCs.

Conclusion

Our flight software testing process was originally just a set of guidelines – recommendations with broad statements about how to develop and test flight software. Over the course of five years and four full spacecraft development efforts (some now complete, some in process), those guidelines have become a defined, useable, and valuable process. The process has evolved and is valuable. It can be improved by continuing to identify weaknesses and implement changes to address the weaknesses. This paper highlighted some weaknesses and provided guidance on changes that can be made to improve the process. The process should be periodically reviewed and improved.

References

1. Software Engineering Laboratory, *Recommended Approach to Software Development* NASA Goddard Space Flight Center, June 1992

2. M. N. Lovellette and Julia White, *"Test Like You Fly" Confidence Building For Complex Systems*, IEEE Aerospace Conference Proceedings, 2005

3. Thomas Ericson, Anders Subotic, and Stig Ursing, *TIM – A Test Improvement Model*, URL: http://www.lucas.lth.se/events/doc2003/0113A.pdf