111149 N95- 17564

-32-61
31969
P. 8

Software Interface Verifier

Tomas J. Soderstrom, Laura A. Krall, Sharon A. Hope, Brian S. Zupke

Telos Corporation
320 N. Halstead, Suite 260
Pasadena, CA 91107
tomas.soderstrom@jpl.nasa.gov

Abstract - A Telos study of 40 recent subsystem deliveries into the DSN at JPL found software interface testing to be the single most expensive and error-prone activity, and the study team suggested creating an automated software interface test tool. The resulting Software Interface Verifier (SIV), which was funded by NASA/JPL and created by Telos, employed 92% software reuse to quickly create an initial version which incorporated early user feedback. SIV is now successfully used by developers for interface prototyping and unit testing, by test engineer for formal testing, and by end users for non-intrusive data flow tests in the operational environment. Metrics, including cost, are included. Lessons learned include the need for early user training. SIV is ported to many platforms and can be successfully used or tailored by other NASA groups.

1. Interface Testing History and Problem Statement

The Deep Space Network (DSN) Deep Space Communication Complex computer environment is highly distributed, with major functions allocated to subsystems. These subsystems are hosted in separate computers and communicate with each other and JPL via a LAN/WAN. All communications follow negotiated interface agreements which prescribe the communications protocols, data formats, and data ranges.

Over the past four years, JPL and Telos developers on the Telos DSN Task Contract fielded 40 subsystems into the DSN. Frequently, mission requirements forced subsystems to negotiate new interface agreements and to deliver asynchronously. The typical subsystem profile was:

- A telemetry, tracking, command, or supporting applications
- Communications and hardware intensive
- High reliability requirements
- 70K lines of C code, mostly realtime
- Six external LAN interfaces
- Development cost of $70K - $2M

The study team found interface testing to have been the most costly and error-prone activity. It proved nearly impossible to manually verify and all possible data ranges and data combinations for all interfaces during live tests. This was due primarily to excessive requirements for test equipment and test personnel in high demand. Consequently, interface errors sometimes were not detected until the subsystem was in operational use.

Metrics collected by the study team supported the high cost of testing. Typically, 3 - 10 attempts were necessary before the average interface was successfully tested. End-to-end interface tests required from 5 - 12 personnel, and multiple tests were necessary. Programmers spent a total of 4 - 6 work months writing unplanned interface simulation code to support the test activity. In addition, they spent another 2- 4 work months per interface in creation and testing activities.

II. SIV Goals

The study also showed that overall testing accounted for a large part of the development effort of the 40 deliveries. This agreed with an Association of Computing Machinery study of seven large software projects, which found that 50% of the resources were spent on the overall test effort. The Telos study team estimated that a comprehensive, automated, reusable test tool could save 40% of the current interface costs. The team further found that 173 DSN interfaces could benefit from this tool within the subsequent five years.

What features would be needed in such a test tool? A literature search and interviews of personnel involved in testing found that the tool should:

- Understand DSN-specific protocols
- Be flexible and extensible, yet easy to use
- Test interfaces in an exhaustive but automated manner
- Provide both realtime visibility into the testing and off line results
- Be available in time to prototype interface agreements
- Support developers' unit testing
- Support test engineers' formal testing
- Support DSN end-users' application simulation and data flow testing

In addition, the test tool should combine three types of test tools and have the following specific capabilities:

1. *Generate Test Data*
   - Control data to the bit level
   - Produce static, variable, and predicted dynamic data
   - Simultaneously run in batch mode and interactively
   - Send single data blocks at specified times and intervals
   - Send data blocks or streams to multiple destinations

2. *Capture and Compare test data*
   - Specify which streams to capture and compare to expected results
   - Specify expected data values and ranges
   - View automatic comparison of test data to expected values both on- and off line
   - Mask out data which would not require an exact match

3. *Simulate the entire application*
   - Create, run, and repeat complete application scenarios for multiple interfaces of multiple subsystems
   - Interactively change the behavior of the simulated, scripted application
   - View online and printed detailed results

Telos proposed the Software Interface Verification (SIV) tool with all the above functionality. It was to be rapidly developed and fielded with increased functionality provided in two subsequent deliveries. SIV was funded by NASA/JPL and developed by Telos. The SIV provides all the functions listed above listed above and summarized in Figure 1.

simulate an application session, such as sending the data from a typical Telemetry pass.

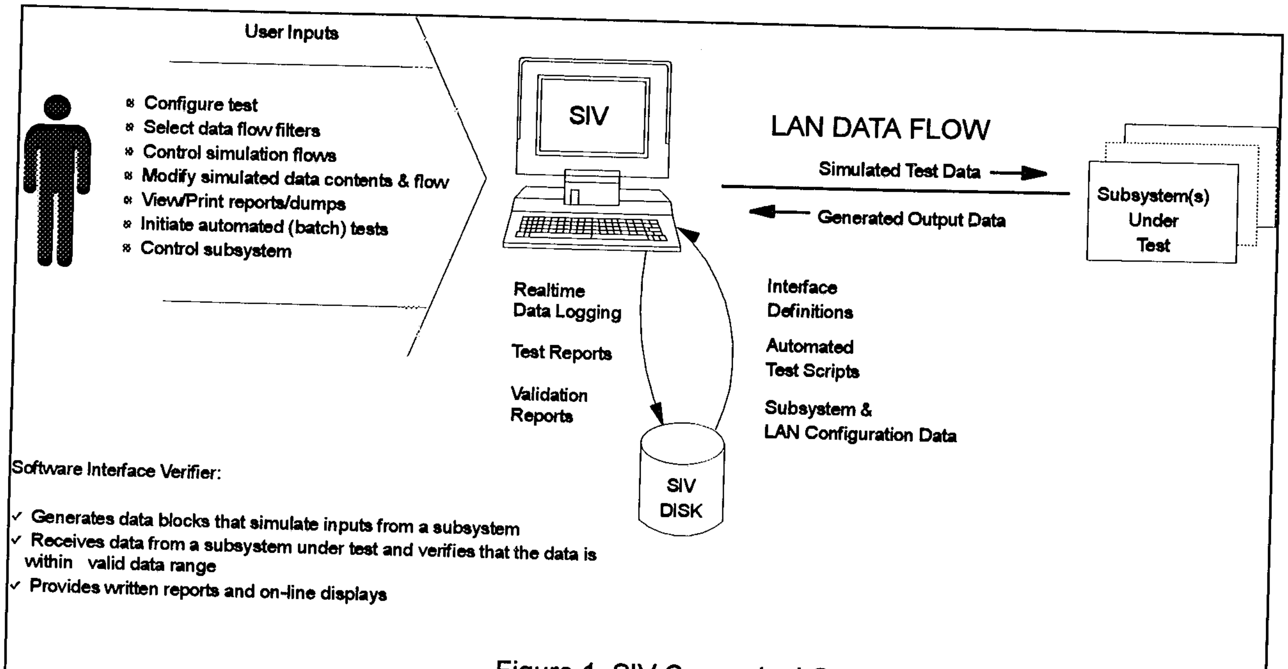
User Inputs
Configure test
Select data flow filters
Control simulation flows
Modify simulated data contents & flow
View/Print reports/dumps
Initiate automated (batch) tests
Control subsystem
SIV
LAN DATA FLOW
Simulated Test Data
Generated Output Data
Subsystem(s) Under Test
Realtime Data Logging
Test Reports
Validation Reports
Interface Definitions
Automated Test Scripts
Subsystem & LAN Configuration Data
SIV DISK
Software Interface Verifier:
✓ Generates data blocks that simulate inputs from a subsystem
✓ Receives data from a subsystem under test and verifies that the data is within valid data range
✓ Provides written reports and on-line displays

Figure 1 SIV Conceptual Operation

The following steps summarize the typical SIV user scenario.

1. Create an ASCII table describing the interface agreement (called a Rapid Interface Definition - RID). It contains interface definitions, including data types and minimum/maximum/expected values, incrementing values, etc. (In the next SIV version, this will be automatically created from the interface agreement. For now, it must be typed in once.)

2. Download the RIDs to SIV from LAN or floppy and select which RIDs to use via a type-in.

3. Select or create application simulation scripts, if desired. This will enable SIV to simulate an application session, such as sending the data from a typical Telemetry pass.

4. Select which tests to run, such as generating test data, logging and comparing test data, and/or simulating entire applications.

5. Select which online displays to view (detailed data dumps, overall status monitoring, or none).

6. Start the tests and, as desired, interactively start/stop/modify the data flows via SIV type-ins.

7. When the test is complete, or manually terminated, print the test report or download it via LAN or floppy. Note that the tests can be set up to cycle indefinitely.

III. SIV Development

The SIV development team consisted of one technical lead who interfaced with the users plus one programmer and one half-time tester. The primary obstacles to be overcome were:

- Users' reluctance to use an unproven test tool
- Requirement to support multiple operating environments
- Limited budget
- Quick results needed to meet users' schedules

In order to meet the budget, time, and multiple operating environment constraints, the development team reused a working skeleton subsystem from the Multiuse Software reuse library, which had been previously created by Telos and had already been ported seven hardware/operating system platforms. In addition, existing test software from other development efforts was adapted for use within SIV.

To overcome the users' reluctance to learn and trust new test tools, the technical lead concentrated on frequent communication with potential users. This included electronic mail, phone calls, visits, demonstrations, and presentations. In addition, the team solicited feedback and carefully folded new user requirements into subsequent demonstrations. This convinced skeptical users by providing them continual visibility and input into SIV development progress and capabilities.

Although SIV was created as a DSN-specific test tool, it was developed in a layered fashion to facilitate later porting. This could include adding new protocols, porting SIV to new hardware/operating system platforms, changing the user interface, and adding/changing SIV functionality. Figure 2 describes the SIV software architecture and major functionality. For example, to incorporate a new, low-level LAN protocol, only the LAN Protocols module of Multi-use Software need change.

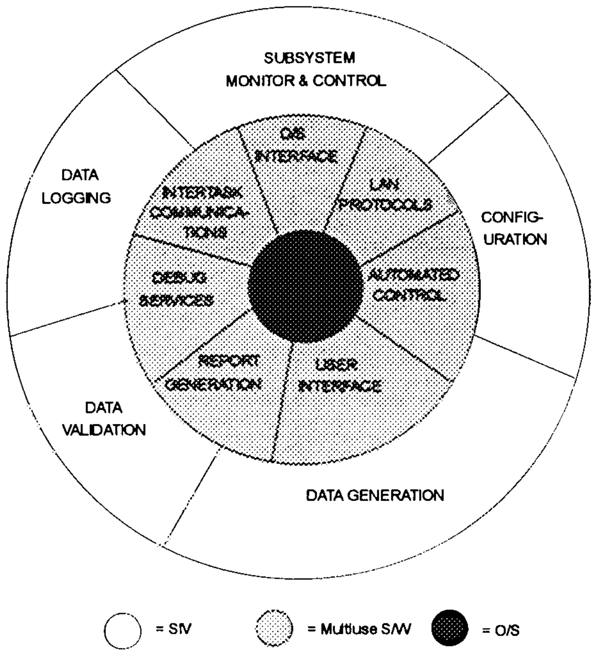

Figure 2. SIV Software Architecture

IV. SIV Results

SIV's primary goal was to reduce the cost of interface testing and the number of software interface errors in the DSN. To achieve this goal, skeptical users had to be convinced that using SIV would save them time. We originally hoped that cost savings due to SIV usage would exceed SIV total lifetime costs ($420K) during SIV's second year of use (1996).

As Figure 3 shows, the goal to obtain user acceptance was met with a wide margin. SIV was initially targeted for use by only 13 projects, or user groups, during the 1994-1999 time frame. However, within the first seven months of development, and one month after the first release, SIV had acquired 23 interested user groups, 10 of which have already used the SIV.

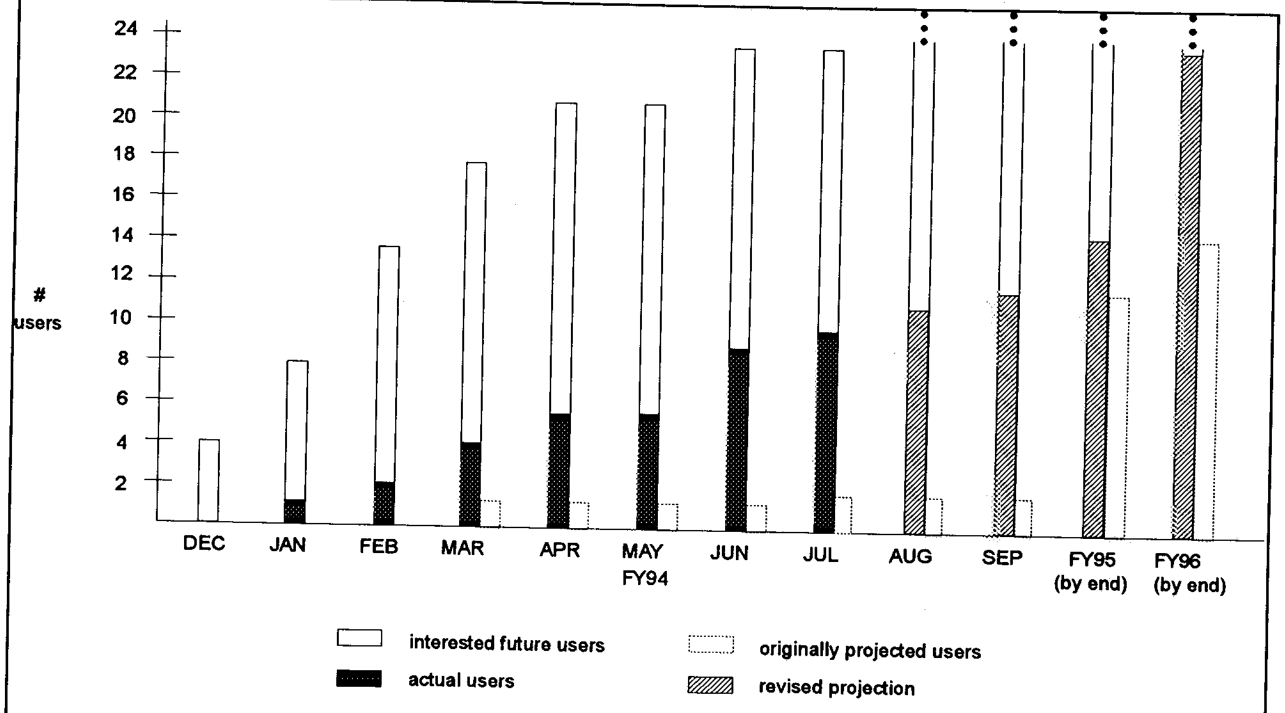

Figure 4 SIV User Interest and Involvement has Surpassed Original Goals

In these 23 user groups, there are now three distinct types of SIV users:

- Developers - use SIV to unit-test low-level interfaces in their development laboratories.

- Test engineers - use SIV to performance/stress test their applications at DSN's official test facility.

- End users - use SIV to simulate entire subsystems for data flow tests, for training, and for simulating hard-to-create error conditions at the official test facilities.

Metrics have been collected for three months: two months before official SIV release, and one month following the release of version 1. The metrics support the anticipated savings as well as ones not originally considered.

The following relates some specific user reports:

- The Metric and Pointing Assembly (MPA) group saved 50 development hours otherwise needed to write simulation code to test a new interface which would not be available until well after MPA delivered.

- The Central Monitor and Control group reported saving 20 work hours because of SIV's ability to insert predicted errors in the interfaces. This would have otherwise taken several weeks and multiple 350-mile round-trips to the DSN station to induce the interface errors, test whether the assembly reacted correctly, and return to make needed software corrections.

- Multi-use Software saved 80 hours of dedicated Test Facility resources and associated travel by using the SIV in their development laboratory to identify and correct a complicated software anomaly.

So far, SIV users have detected and corrected the following types of errors in their applications, without the need for live tests: formatting errors, data range errors, routing problems, and errors due to misinterpretations of interface agreements.

The metrics listed in Table 1 represent three months of SIV usage by six user groups.

Table 1. Initial SIV Metrics

SIV Usage	Metric
# Subsystems Interfaces Tested	11 interfaces
# Data Flows Tested	20 data flows
# Interface Definitions Generated	63 RIDs
# Code/Interface Errors Discovered	24 errors corrected
Est. Test Facility Time Saved	146 work hours saved
Est. Additional User Time Saved	190 work hours saved
Est. Simulation Code Time Saved	100 work hours saved
Est. SIV Learning Curve Total Cost	10.5 work hours invested

Let us tackle the difficult process of estimating cost savings achieved. Of the six user groups, an average of three user groups were concurrently using the SIV each month for a three month period. To estimate cost savings achieved, let us assume the DSN average development cost (including burden charges) to $67 per work hour and test facility usage to $200 per work hour (including support personnel, rent, hardware maintenance, etc.). These values when combined with the savings in the above table result in a total savings of $48.6K for the three months or $5.4K per user group per month. Applying the $5.4K to our projected Fiscal Year 1994 (FY94) and FY95 users (see Figure 3), results in a total cost savings of $216K for FY94 and $875K for FY95. This exceeds our originally projected cost savings of $51K for FY94 and $324K for FY95. In more general terms, this minimally translates into the developer having more time to work on other subsystem development areas. It also means more available test facility time to other users. Overall, SIV usage should significantly reduce the risk and cost of the typical DSN subsystem delivery.

Additional savings due to automated testing using SIV include:

- Reduced amount of travel to -- and use of expensive -- Test Facility
- Faster turn-around times when testing within development labs--no need to wait for scheduled test times or personnel availability
- Costly simulation code need not be generated nor maintained

- Fewer end-to-end test resources required since data content and protocol routing can be pre-verified with SIV
- Automated regression tests can be run at computer speed

Although the initial SIV version has just been fielded, early results clearly indicate the value of automated testing and that SIV met its goals and will help test DSN interfaces at all levels. Developers, test engineers, and end users no longer have to be "sold" on using automated test tools such as SIV. The early results indicate that automated testing will continue to pay dividends.

V. Lessons Learned

What did we do right?

We solicited user acceptance. The SIV Technical Lead spent a considerable amount of time with skeptical users to learn their test and simulation needs and teach them SIV.

We held early and frequent demonstrations. These also allowed for design refinement and identification of new requirements. When acted upon, this was especially important as it created user acceptance.

We selected an experienced staff. The developers, who were experienced with the reused packages and testing in the DSN environment, experienced no learning curve.

We employed significant reuse. The completed SIV consists of 8% (or 8K lines) application-specific code and 92% reuse from Multiuse Software and adapted simulators and test software obtained from a reuse depository. Besides for helping speed up the SIV development, the reused software had been previously proven, extensively tested, and ported to seven platforms.

What could we have done better?

We should have allocated more schedule time to the demonstrations. Although invaluable for the eventual SIV progress, the cost of each demonstration was 3-5 work days to plan and hold plus 3 work days for user requirements change requests, follow-up, and action items.

We should have provided earlier user training. This would have lessened the drain on SIV personnel for user support which we under-estimated.

We should have held smaller training classes customized to the group's needs. This would have allowed more customized training to better enable the users to recognize and use the powers of simulation and automation that SIV possesses.

VI. Applicability For Other Groups

SIV can be successfully used on all large, distributed software development efforts where computers interface over a LAN. Although standards, such as the Distributed Computing Environment, and Abstract Syntax Notation, have great promise, they are often too late to immediately benefit current, large software environments. The SIV is a flexible test and simulation tool which can test other subsystems over a LAN. It can be easily adapted to use new custom or standard high- or low-level protocols.

SIV is written in C and currently runs on a Sun under the Solaris operating systems and on Modcomp's Unix work stations running the Real/ix operating system. It can easily be adapted to run on all other platforms supported by Multiuse Software (PDOS, VxWorks, VADSWorks, and OS/2). It is currently being ported to run on Intel 80386 computers (and greater) running a shareware Unix variant called Linux. SIV is fully documented and available from Telos or JPL by request to the authors. We plan to implement TCP/IP during Fall/Winter 1994, which should make the SIV instantly usable by groups outside the DSN.

Acknowledgments

The work described in this paper was accomplished by Telos Corporation under contract to the Jet Propulsion Laboratory, California Institute of Technology and sponsored by the National Aeronautics and Space Administration.

We would like to thank Joseph Wackley, Roger Crowe, and Sheila Davis of JPL for their support of Multiuse Software and the Software Interface Verifier project and for their vision and relentless efforts to create an automated testing environment in the DSN.

Systems Development

* *Presented in Poster Session*