**SYSM 6309 Advanced Requirements**

**Software System Integration Testing**

**Submitted To:**

Dr. Lawrence Chung

Associate Professor,

Department of Computer Science,

The University of Texas at Dallas,

Richardson, TX- 75080

**Submitted By:**

Steven Ford

May 5, 2012

# ABSTRACT

Integration Testing is the process of combining and testing multiple software components together and ensuring the integrity of the complete software and/or hardware structure from unit tested modules.

# PROBLEM STATEMENT

Most, if not all, modern hardware and software systems are what we call a System of Systems (SoS). The larger system is a compilation of many smaller subsystems which need to interact with one another in order to produce a desired output. Often time’s subsystems which function as desired at the unit level do not interact as expected with other subsystems when integrated. The earlier these incompatibilities are discovered the less of an impact it will have to the project schedule and cost.

# SOLUTION

For systems of systems there are four major levels of testing. The lowest level is the unit test. At this level each individual software or hardware piece is tested to ensure it functions as desired. The next level of testing is integration testing. At this level the individual subsystems are tested to ensure they interact with one another as desired. The highest level of testing is system level testing. At this level the system as a whole is tested against customer requirements. The last level of system testing is Acceptance Testing. This is the final test the system undergoes before delivery to the customer. The primary focus of this paper is System Integration Testing but each level of testing is critical to the successful completion of a project.

## System Integration Testing

System Integration Testing is a systematic approach to building the complete software and/or hardware structure specified in the design from unit tested modules. It assures that the software units are operating properly when they are combined together. The aim of integration testing is to discover errors in the interface between the components.

# INTEGRATION FAILURE – MARS POLAR LANDER

## Mission

In January 1999 the National Aeronautics and Space Administration (NASA) launched a $165,000,000 project called the Mars Polar Lander. Its objectives were to record local meteorological conditions near the Martian South Pole. After landing at a predetermined location near the pole it was designed to do the following:

* Analyze samples of the polar deposits for volatiles, particularly water and carbon dioxide
* Dig trenches and image the interior to look for seasonal layers and analyze soil samples
* Image the regional and immediate landing site surroundings for evidence of climate changes and seasonal cycles
* Obtain multi-spectral images of local regolith to determine soil types and composition

## Destruction

After making the 11 month, 157 million mile journey to Mars the lander began its descent in December 1999. Six hours after the descent began all communication with the lander was lost. After a thorough investigation by NASA, the organization concluded that the lander’s destruction was caused by a software error which misinterpreted a message between two subsystems.

A magnetic sensor was provided in each of the three landing legs to sense touchdown when the lander contacted the surface, initiating the shutdown of the descent engines. The software logic was designed to accept these transient signals as valid touchdown events if they persisted for two consecutive readings of the sensor. Testing showed that most of the transient signals at leg deployment, which occurs long before touchdown, are long enough to be accepted as valid events.

The software, which was intended to ignore touchdown indications prior to the enabling of the touchdown sensing logic, was not properly implemented, and the spurious touchdown indication was retained.

The touchdown sensing logic was enabled at 40 meters altitude, and the software most likely issued a descent engine thrust termination at that time in response to a (spurious) touchdown indication.

At 40 meters altitude, the lander had a velocity of approximately 13 meters per second, which, in the absence of thrust, was accelerated by Mars gravity to a surface impact velocity of approximately 22 meters per second (the nominal touchdown velocity is 2.4 meters per second).

Essentially, the destruction of the Mars Polar Lander, a multimillion dollar state of the art piece of equipment was due to a BIT register being set to “1” instead of a “0”.

## Root Cause of Failure

The NASA investigation revealed that the cause of the software logic error was due to software scope creep. The software logic satisfied all initial requirements. However, testing showed hardware limitations with the magnetic touchdown sensors which could be accounted for through a software change. The original software logic requirements and requirement verification plan should have been altered to account for this change but they were not and the result was the destruction of the lander.

While scope creep initiated the system failure the root cause of the failure was the project team’s inability to adapt to this change and verify the functionality of the system as a whole. Project scope creep is inevitable on a project of this size. Additionally, requirements issues are very likely to occur on a project this massive therefore making System Integration Testing paramount.

# SOFTWARE SYSTEM INTEGRATION TESTING

Software System Integration testing is a process which consists of dynamic and static activities, which evaluate how the subsystems of a software system interact with one another in order to determine that the system will indeed meet the requirements of the end user, and also to demonstrate that the system is fit for purpose and does not have defects.

The aim of integration testing is to discover errors in the interface between the components. The interface errors and the communication between different modules are also unearthed in this testing. There are two main types of integration testing, namely incremental and non-incremental testing. Incremental testing can be performed using any one of three approaches, namely top down approach, bottom up approach, or functional incremental approach. If there are a number of small systems to be integrated to form a large system, then the system integration testing process is the same as component integration testing, as each small system is considered as a component. When system integration testing is done, the interaction of the different systems with each other is carried out, whereas in component testing the interaction between the different components is tested.

In integration testing, interface specifications are taken into consideration. This differs from system level testing because at that level requirements specifications are the main focus. In system testing, the tester does not have access to the code used to make the system. On the other hand, while carrying out software integration testing, the tester is able to see the internal code. When integration testing is carried out, some scaffolding may be required in the form of drivers and/or stubs. The same is not necessary for system testing.

# SOFTWARE INTERGRATION AND TEST APPROACH

## Integration and Test Milestones

Software integration and test milestones identify a period during which time a set of software units are verified by exercising one or more threads. Thread verification during software integration deals typically with the interfaces between software units, while thread verification during software testing deals with functional capabilities. One or more of these threads support a functional capability as identified in the software requirements specification. The build plan identifies these capabilities as milestones. A milestone may consist of one or more capabilities that can be demonstrated by executing the instructions in the software integration and test procedures. A milestone can also be associated with the release of master integration and test media. This contains all the software that can reasonably be integrated and tested in parallel for a particular milestone.

## Intermediate Integration and Test Builds

Intermediate integration and test builds may be identified, which consist of informal builds to allow for smaller increments of software to be integrated and tested. Smaller increments are encouraged, to promote faster problem identification. An incremental approach to system verification is shown in **Figure 1**. Software dissimilar in nature and/or allocated to different central processing units (CPUs) or equipment should be identified separately, and may require a separate build. Consider software that physically shares common data areas and activities that require access to common databases. Database integration and testing is an integral part of the total software integration and testing effort, and should be identified on separate builds.



**Figure 1: Top Down Incremental Design & Test Approach**

## Target Hardware/Software Configuration

Software integration and testing should be performed in an environment that represents the target hardware and software environment as closely as possible. Pay specific attention to hardware/software switch settings and specific system adaptation parameters as these parameters can vary easily from site to site.

## Software Configuration Control

A configuration control management system to control the software and software documentation configuration, storage and retrieval during the integration and testing effort should be implemented

A version description document to identify the different versions of the different software linked together for a master release should be used. The hardware and commercial software revision levels should be identified, where possible.

# MARS POLAR LANDER – INTEGRATION TESTING

As stated previously, the most probably cause for the loss of the Mars Polar Lander is that spurious signals were generated when the lander legs were deployed during descent. The spurious signals gave a false indication that the lander had landed, resulting in a premature shutdown of the lander engines and the destruction of the lander when it crashed into the Mars surface.

It is not uncommon for sensors involved with mechanical operations, such as the lander leg deployment, to produce spurious signals. For the lander, there was no software requirement to clear spurious signals prior to using the sensor information to determine that landing had occurred. During the test of the lander system, the sensors were incorrectly wired due to a design error. As a result, the spurious signals were not identified by the systems test, and the systems test was not repeated with properly wired touchdown sensors. While the most probable direct cause of the failure is premature engine shutdown, it is important to note that the underlying cause is inadequate software design and integration testing.

The Mars Polar Lander failure was the result of many individuals shortfalls in the design and test process which aligned properly to cause the complete loss of the system. Had the project team followed the iterative integration and test process outlined in Section 5 the software logic error would have most likely been discovered prior to launch and the mission could have been a success. For projects such as the lander where the equipment is designed for a one time use and the environment cannot be duplicated in a cost effective manner iterative integration testing is paramount because ability to perform system level testing is limited.

# REFERENCES

[1] Barry Shore. Mars Climate Orbiter and Mars Polar Lander. Global Project Strategy. <http://globalprojectstrategy.com/lessons/case.php?id=10> Retrieved 2012-02-07.

[2] Babu Suresh. Scope Creep Management. 2005-01-01. <http://www.projectperfect.com.au/info_scope_creep_mgmt.php>

Retrieved 2012-05-01.

[3] Bhakti Satalkar. Integration Testing vs. System Testing. 2011-06-14. <http://www.buzzle.com/articles/integration-testing-vs-system-testing.html> Retrieved 2012-02-09.

[4] Raytheon. Software Integration and Testing Guidelines. EN-04-01-02 Rev. A. 2003-06-06. Revised 2011-02-11. Raytheon Space & Airborne Systems.

[5] Thomas Young. Mars Program Independent Assessment Team Summary Report. Draft #7 2000-03-13. House Science and Technology Committee. <http://www.spaceref.com/news/viewpr.html?pid=1444>. Retrieved 2012-02-07.