This Paper was originally presented at the STAR West Conference, October 1998

AT&T Network & Computing Services

Applying Use-Case Methodology to SRE and System Testing

Date: August 10, 1998

From: Steve Meyer
Org: 1FA9T3000
OHJ740 717-2
(513) 629-7321
E-mail sameyer@att.com

Ray Sandfoss
Org: 1FA9T3000
OHJ740 718-1
(513) 629-7204
E-mail rvsandfoss@att.com

Table of Contents

Table of Figures

Table of Tables

1. Overview

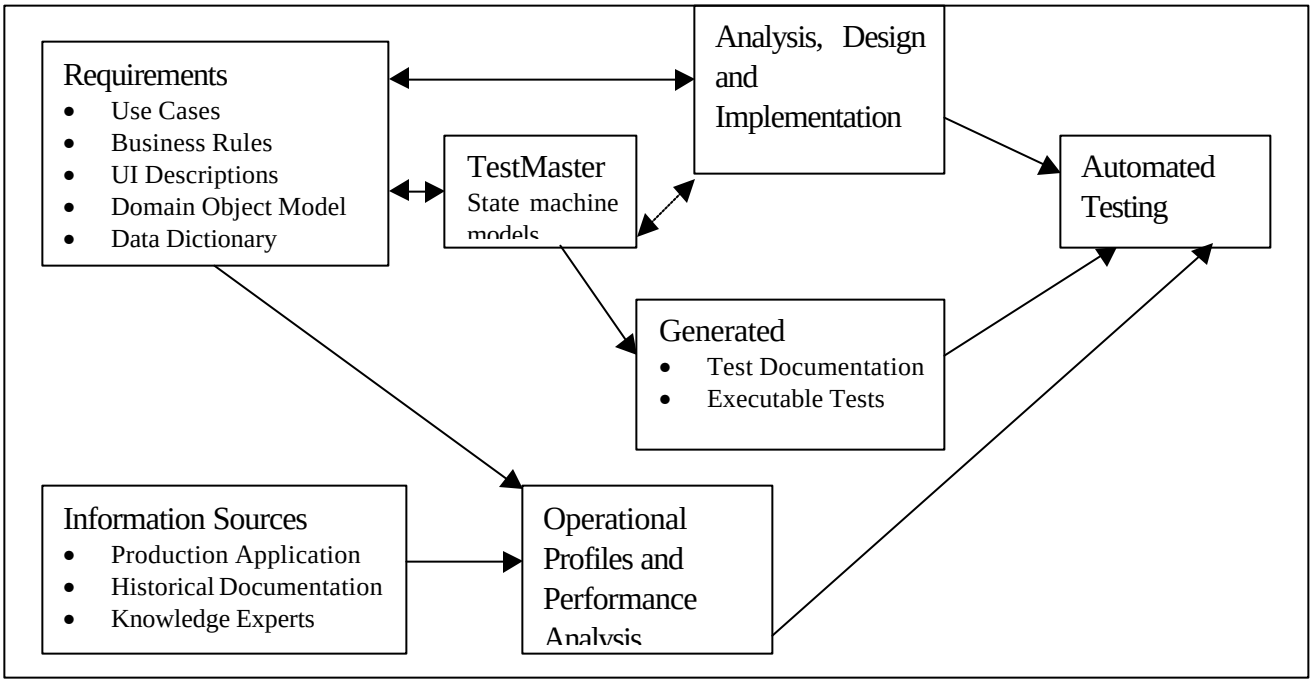
The main point of exploration in this paper is the techniques used for taking Use Case Requirements and utilizing them to expedite both Operational Profile development and Test Design and Automation activities.

The application being studied wanted to improve the quality and reliability of their system test cycle and subsequently their product releases. Past releases suffered from lack of ability to execute planned functionality tests and experienced response time, throughput and performance problems once the application was delivered to the field users. It was decided to implement Software Reliability Engineering techniques, conduct a performance analysis study and focus on automated testing as a strategy to address some of these problems. The application enlisted SRE and testing experts as consultants to guide the application on this directive.

The approach taken was:

- Develop an operational profile to more fully understand application usage and modes
- Perform additional performance analysis to better pinpoint problem areas
- Automate portions of the testing using COTS (Commercial Off-The-Shelf) tools.
  - TestMaster™ for modeling the application and generating test
  - WinRunner™ as the test execution engine.

The Operational Profile development and test automation tasks were undertaken as two separate items by internal consultants. The applications architect would conduct the performance work. The operational profile and performance work would come together to guide the testing efforts. The test automation effort would improve testing efficiency. The three tasks taken together should achieve improved testing effectiveness and better test coverage. Requirements for the current release were stated in Use Case modeling methodology. A common reference for a starting point for each of these tasks was the Use Case models.



- Figure 1: Process Flow

2. Use Cases, the Operations List and SRE Methodology

2.1 General Steps to SRE

SRE as practiced encompasses the following steps:

1. Define Reliability Objectives and Failure Intensity Objectives
2. Operational Profile Development
3. Designing Tests
4. Executing Tests
5. Interpreting Tests

Steps one and two can be completed interchangeably and it is often the case the operational profile work is done first. This allows for better definition of measurable failures during the "Reliability" steps.

2.2 General Steps to Operational Profile Development

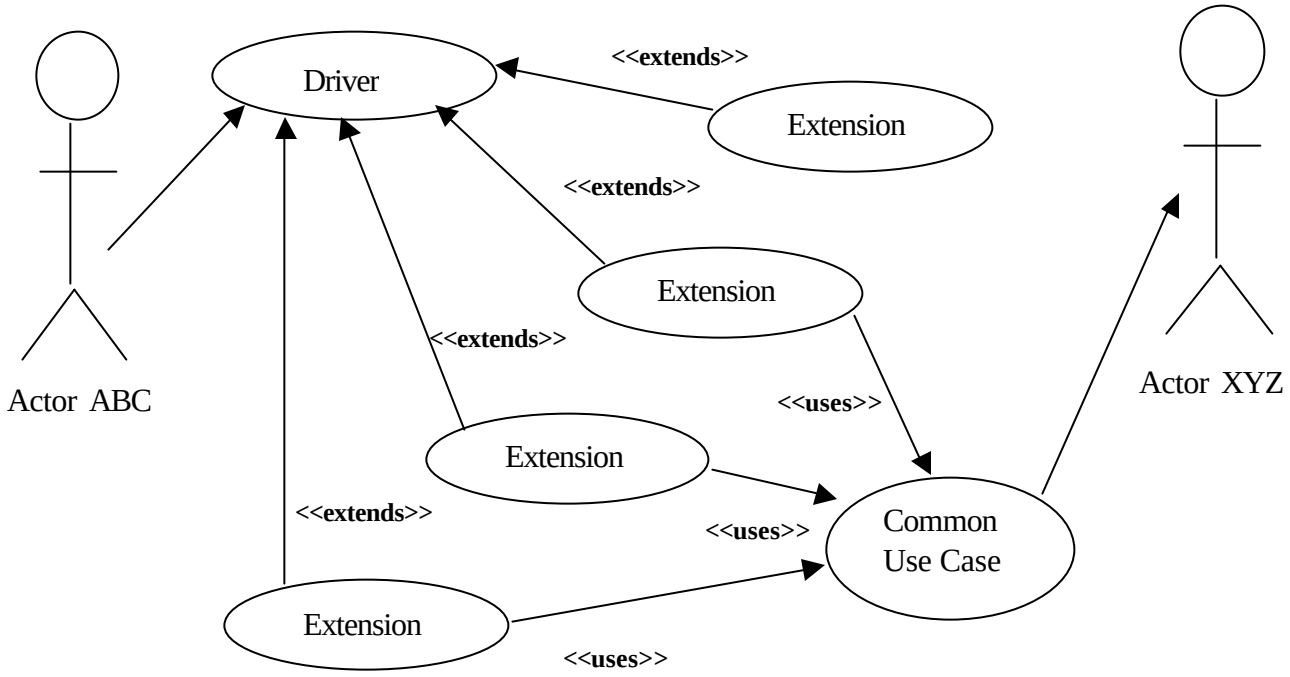
1. Identify Initiators of Operations *(Leverage Use Case Models)*
2. Create the Operations List *(Leverage Use Case Models)*
3. Identify Modes of Operations
4. Determine Operations Occurrence Rates
5. Determine Occurrence Probabilities
6. Refine Operations List *(Analyze Use Case Models)*

2.3 Use Cases for the Operational Profile

Our development organization has recently advocated Use Case Requirements Modeling as a standard methodology for requirements specification. At the same time the SRE and Test consultants realized use case modeling provides a good description of the steps needed to perform a specific task. They took an approach of identifying initiators of operations *(2.2 Step 1 above*) and creating the operations list *(2.2 Steps 2 and 6 above)* from the use case models. This was done for the new functionality introduced by this feature.

As this was new ground questions were outstanding as to how the use case models would map to pertinent parts of the operational profile. It is not always evident how various use cases interact within a developed business application solution. In general use case actors map to initiators of operations in the operational profile. Use cases map to operations in the operational profile. Where instances occur in the use case model of <<uses>> and <<extends>> stereotypes, additional analysis is necessary (*2.2 Step 6 above)* to more accurately build the operations list.

The following illustration depicts a Use Case diagram and common Use Case Relationships



- Figure 2: Common Use Case Relationships

2.3.1 Steps for Identifying Initiators and Building the Operations List

1. All use cases are identified up-front from requirements

2. All of the use cases are then mapped to operations and initiators are identified from the use case actors. The first pass assumes all use cases are operations

3. The operations list is refined by analyzing the use case <<uses>> and <<extends>> stereotypes.
   - This refinement reveals use case to operations relationships. These relationships are :
     - one to one, use case to operation, usually no stereotypes involved
     - one to many, use case to operation, often analysis of the stereotypes <<uses>> and <<extends>> yields these relationships
     - One to none, use case to operation, in case of <<extends>> the driver use case is omitted from the operations list or exists as a placeholder to give the mapping closure.

2.4 Use Cases for Test Automation

Operational Profiles provide frequency information about use cases. Each operation in the operational profile represents the invocation of a use case by a specific actor. The use case describes the operation the operational profile's frequency information provides details how often an operation is executed. The initial modeling with the TestMaster tool further refined the relationships between use cases at a high level. Related use cases were grouped that represented the process flow for a set of business tasks. The use case model diagram proved most useful for this exercise.

At some point additional information about the physical implementation was needed in addition to the requirements specifications. In this case the organization of the GUI clients aided in further detailing and annotating the model.

TestMaster is a software tool enabling test engineers to visually model their System-Under-Test using state machine based notation. By annotating the model with test execution system language, WinRunner TSL in this case, TestMaster can automatically generate test scripts for the System-Under-Test.

WinRunner is a capture/playback tool for testing GUI's. Getting TestMaster to output executable tests requires annotating the model's transitions with TSL code. The test script generator evaluates paths through the model and concatenates the code "code fragments" on individual transitions to build executable tests.

3. Application Description

The application being studied was a client server style or distributed system. Some of the components of this application include:

- Windows NT Desktop clients
- Visual C++, and WEB client implementations
- Windows NT client side WEB server and local database images
- UNIX C++ services and server side database
- Work managed across six internal networks
- High availability and site Fail-over server side configurations

4. Operational Profile

4.1 Operational Modes

Defining the operational modes was the first step in developing the application operational profile. Secondly, the use cases were mapped to the various modes. The application had 5 modes defined. They were as follows:

- **User Online** - This is the normal mode of operation. This is the mode where any or all of the users in the user organizations are utilizing the application. This mode encompasses the hours from Monday – Saturday 6:00 am to 10:00 p.m. EDT. This is all of the hours outside of the maintenance window, plus Saturdays. However, the normal User online mode is actually from 8:00 am until 6:00 p.m. EDT. The user community reserves the flexibility to extend this window to start at 6:00 am and finish at 10:00 p.m.
- **Pre-Delivery Window** - This is the mode of operation where the application is processing and preparing information and orders to be sent to various network elements. This mode encompasses the hours from Monday – Friday 6:00 p.m. to 10:00 p.m. EDT. Also, included in this mode are the system cleanup activities that take place prior to the Maintenance window.
- **Maintenance Window** - This is the mode of operation where the application is sending network element updates to the network elements. This mode encompasses the hours from

10:00 p.m. to 8:00 am EDT Monday through Friday and Friday 10:00 p.m. until 8:00 am Monday morning.

- **Software upgrade -** This is the mode of operation where the application is introducing new application software to the field. This mode encompasses the hours from 11:00 am to 1:00 p.m. EDT Monday through Friday and Saturday from 3:30 p.m. – 10:00 p.m.
- **Periodic Bulk -- ex. 15th of each month: large file processing -** This is the mode of operation where the application is processing large portions of bulk data. This mode encompasses the hours from 10:00 p.m. on the 14th to 8:00 am EDT on the 15th day of each month.

4.2 Mapping Operations List to Operational Modes

The next step in the operational profile development was to map the use cases to an operations list and then to map that operations list to the 5 operational modes that had been identified for this application. That mapping was best represented in the form of a table. The table that represented this looked like the following.

Table 1 : Application Functional Profiles mapping

Actors And Initiators of Operations	Use Cases and Operations	Operations and Use Cases Occurrence Rates by initial System Operational Modes					Total Occurrence s across all System-Operational Modes
		Mode 1	Mode 2	Mode 3	Mode 4	Mode 5	
User Group 1	10010 : operation 1	2	1				3 times /yr.
	10013 : operation 2	4	1	1			6 times/ yr.
	10020 : operation 3	1					1 time/yr.
	10030 : operation 4	3	1				4 times/yr.
	10040 : operation 5	6	1		1		8 times/yr.
	10050 : operation 6	1					1 time/wk
	10053 : operation 7	15	3	1	1		20times/yr
User Group 2	40500 : operation 1	1	1			1	1 time/day *** could occur in any of 3 modes ***
	40600 : operation 2	8	2				10 times/day
External System 1	60000 : operation 1	40	5	5			50 times/day
	70000 : operation 2	1					1 time/day

4.3 Severity Class Definition

The next step in the operational profile development was to determine the level of the severity of the failures that could occur within the application. Here are the actual definitions that were defined for this application.

> **Severity 1 failure**: a failure that occurs which renders the system release unavailable to the User community.
>
> **Severity 2 failure**: a failure that causes the system release to be available to the User community, but in degraded state.
>
> **Severity 3 failure**: a failure that is defined to cause some part of the system release to be unavailable to the User community, but with a work-around available.
>
> **Severity 4 failure**: a failure that is defined to cause some part of the system release to be unavailable but with no impact to the User communities.

4.4 Defining the Measurable failures by mode and root cause

The next step in the operational profile development was to determine the failures that could occur across the system various operational modes that were defined. In addition, root causes were first defined. The root causes that were defined were:

> **The categories or root causes are as follows:**
>
> **1.** GUI Processors (hardware)
> **2.** GUI Processors (software)
> **3.** User LANs / WANs
> **4.** Unix server Processors (hardware) used within the system platform
> **5.** Unix server Processors (software) used within system platform
> **6.** Processor LANs / WANs used between the system processors
> **7.** Networks used to connect to various network elements
> **8.** Network Element (Hardware and Software)

In addition, a table that maps these measurable failures to their respective severity levels was developed. Then these failures were mapped to the various root causes that could cause them to occur. Here is a table that depicts these mappings.

Table 2: System Measurable Failures by Root Cause and Operational Mode

Measurable Failure	Severity Class	Operational Modes	Root Causes
Cannot Retrieve list of required information	1	1,2,3	1,2,3,4,5,6
Retrieved required information inaccurate	1	1,2,3	2,5
Takes "long" time to retrieve required information	2	1,2,3	2,3,4,5
Cannot Retrieve list of low level details required	1	1,2,3	1,2,3,4,5,6
Retrieved list of low level details inaccurately	1	1,2,3	2,5
Takes "long" time to retrieve list of low level details	2	1,2,3	2,3,4,5
Cannot send necessary information to external system	1	1,2,3,5	1,2,3,4,5.6
Necessary information sent to external system takes a long time	2	1,2,3,5	2,3,4,5
User Interface Fields rejected	2	1,2,5	2,5
User interface fields take a long time to accept	2	1,2,5	2,3,4,5
Cannot notify external system of the updates	3	1,2, 5	1,2,3,4,5,6
Notify to external system takes a long time	2	1,2, 5	2,3,4,5
Cannot receive events from external system	1	1,2,5	3,4,5,6
Receive events from external system in a degraded fashion	2	1,2, 5	2,3,4,5
Download of events to NE's takes a long time	2	1,2,3,4,5	2,3,4,5,6,7,8
Events received from NE's takes a long time	2	1,2,3,4,5	2,3,4,5,6,7,8
Maintenance work cannot be	1	1,2,3,4,5	5

Measurable Failure	Severity Class	Operational Modes	Root Causes
performed on system			
Maintenance work can be done, but isn't correct	2	1,2,3,4,5	5
Cannot print reports	1	1,2,3,5	2,3,4,5,6
Can print reports, but they are inaccurate	2	1,2,3,5	2,5
Print of reports takes too long	2	1,2,3,5	2,3,4,5
User LAN / WAN slow to transmit data	2	1,2,3,4,5	3
Server LAN / WAN slow to transmit data	2	1,2,3,4,5	6
User LAN / WAN not available	1	1,2,3,4,5	3
Server LAN / WAN not available	1	1,2,3,4,5	6
Cannot Upgrade server software	1	4	4,5,6,7
Server software upgrade takes too long	2	4	4,5,6,7
Cannot perform Database conversion	1	4	4,5,6,7
Database conversion takes too long	2	4	4,5,6,7
User cannot login	1	1,2,3,5	1,2,3,4,5
User login takes too long	2	1,2,3,5	1,2,3,4,5,
Cannot run cron jobs	1	1,2,3,4,5	4,5
Cannot clean up object space	1	1,2,3,4,5	4,5
Cleanup of object space takes too long	2	1,2,3,4,5	4,5

5. Test Automation

5.1 Aspects of Modeling

TestMaster Models are comprised of States and Transitions. Where behavior gets more complex a sub-model is substituted for a state. Thus, as models grow in complexity layers are added. The higher levels are characterized by transitions to sub-models. At the lowest level there will only be states and transitions. From the modeling approach being used I would expect the System Engineers models to have three to four layers and the Test Engineers work to add three to four layers. Indications are most applications will have multiple models. In our case the first breakout would be one of application's GUI clients. It may also become necessary to further partition the more complex clients into more than one model. Partially modeled behavior can be easily disabled while

other parts of the model are being worked to completion. The partially modeled functionality can be revisited as modeling work progresses.

5.2 Building the Model

The subset of functionality initially modeled incorporated behavior from six fairly complex use cases. The GUI implementation contained many common objects: menus, buttons, pick lists, … Approximately eight screens with approximately 25 data elements total were modeled completely or in part. Screen level and field level edits were in place along with “Back", “Next” and “Cancel” buttons.

A high level model of the new functionality was created from Use Cases (Systems Engineering Model) This model was then extended (Test Engineering Model). The extensions concentrated on a subset of the functionality. As work progressed more of the functionality would be completed within the model. This subset was identified as appropriate for the initial modeling effort. To complete the extended model, more detailed information of the physical implementation was necessary. For this an integration load of the application was used. This provided the physical implementation details (screens, data elements and workflow) to complete the model to a point where it fully captured the applications behavior.

First the model was annotated and constrained to produce test documentation paths and steps. TestMaster algorithms found paths through the model and each step in a path was numbered. The output was reviewed and the model iteratively refined. The model subset produces 9 – 16 paths in transition coverage and 50 – 100 in full coverage. The variation is based on the level of constraints that are enabled.

Next a second test script output stream was defined. This stream would produce executable WinRunner Scripts. One capture playback session exercising the GUI provided enough WinRunner TSL to allow a good portion of the model to be annotated for the executable output stream. Some of the TSL gathered in this session included: NULL entries in required fields, invalid entries, button presses, menu-selections …

As a measure of relative complexity note the following hash counts for various TestMaster model components.

- 1 Root model
- Deepest sub-model layering -- 6
- two test output streams
  - test documentation for manual test execution
  - WinRunner executable test scripts
- 38 sub-models
- 98 states
- 178 transitions –
  - 87 transitions were annotated

5.3 Sample Output

The following are sample test output streams from TestMaster. Each numbered step is a transition. Transition information in a path is concatenated to produce a test scenario. The step numbering was a characteristic added by the modeler. This makes it easy to recognize separation between transitions and enhances the ability to cross-reference multiple output streams. A sample script for each output stream path04 is provided. These stream have been modified to protect sensitive information

5.3.1 Test Output Stream for manual test execution

path4(05/19/1998)

1. Select Add XXXX from List

2. Task Manager Create Button to Menu select Node
   To Add XXXX
3. Verify Add XXXX Window Appears with Blank fields
4. Select Valid Variable value from pick list;
5. Enter Valid XXXX into Field;
6. Select Variable value from list;
7. No Optional variable value Entered
8. Press Next Button
9. Verify Summary Window Information
   Verify Task_Name: ADD XXXX
   Verify XXXX
   Verify YYYY
   Verify ZZZZ
   Verify User Id is: abcdefg
10. Back Button Pressed from Summary Window
11. Verify Add XXXX Window Appears with previous Fields Populated
12. Select Valid variable value from pick list
13. Enter Valid XXXX into LLI Field
14. Select Variable value from list
15. No Optional variable value Entered
16. Press Next Button From Add XXXX Window
17. Verify XXXX Summary Window
    Verify Task_Name: ADD XXXX
    Verify XXXX
    Verify YYYY
    Verify ZZZZ
    Verify User Id is: abcdefg
18. Press Finish Button from Summary Window

smeyer -- 05/19/1998 14:54:40}
END PATH 4

5.3.2 Test Output Stream for automated test execution

```
# path(4) -- (05/19/1998)

#    1. Add XXXX

#    2. From Task Manager navigate through menu's to ADD XXXX;
         win_activate ("Task Manager - XXX");
         set_window "Task Manager - XXX", 10);
         toolbar_button_press ("ToolbarWindow32", "#1");
         win_activate ("Tasks Menu");
         list_expnd_item ("SystreeView32", "Tasks;Node"); # Item Number 2;
         list_select_item ("SystreeView32", "Tasks;Node;ADD XXXX"); # Item Number 6;
         button_press ("OK");
         # Add XXXX Appears Focus on Window;
         win_activate ("Add XXXX");
         set_window ("Add XXXX", 10);

#    3. Add GUI Checkpoint WIndow Verification here;

#    4. Select Valid Variable value from List
         edit_set_insert_pos ("XXXX *:_1" 0, 0);
         list_select_item ("XXXX *:_0", "22"); # List Item Num 4;

#    5. Eneter Valid XXXX;
         edit_set_insert_pos ("XXXX *:", 0, 0);
         type ("XXXXXXXXXXX");

#    6. Select Variable value from list;
         edit_set_insert_pos ("XXXX *:_1", 0, 1);
         list_select_item ("XXXX *:_0", "5");

#    7. No Optional Variable value Entered;

#    8. Next Button Pressed From Add XXXX Window;
         set_window (Add XXXX", 10);
         button_press ("Next >");
         # Summary Window Appears;
         win_activate ("Summary");

#    9. Verify Data entered on Add XXXX Window;
         list_select_item        …..
         list_select_item        …..
         list_select_item        …..
         list_select_item …..

#   10. Back Button Press From Summary Window;
```

```
        button_press ("< Back");
        # Add XXXX Window returns;
        win_activate (Add XXXX");
        set_window ("Add XXXX", 10);

#   11. Add GUI Checkpoint Window Data Verification TSL here;

#   12. Select Valid Variable value from List
        edit_set_insert_pos ("XXXX *:_1" 0, 0);
        list_select_item ("XXXX *:_0", "22"); # List Item Num 4;

#   13. Eneter Valid XXXX;
        edit_set_insert_pos ("XXXX *:", 0, 0);
        type ("XXXXXXXXXX");

#   14. Select Variable value from list;
        edit_set_insert_pos ("XXXX *:_1", 0, 1);
        list_select_item ("XXXX *:_0", "5");

#   15. No Optional Variable value Entered;

#   16. Next Button Pressed From Add XXXX Window;
        set_window (Add XXXX", 10);
        button_press ("Next >");
        # Summary Window Appears;
        win_activate ("Summary");

#   17. Verify Data entered on Add XXXX Window;
        list_select_item        ….
        list_select_item        ….
        list_select_item        ….
        list_select_item        ….

#   18. Press the Summary Window FInish Button;
        button_press ("Finish");

# smeyer -- 05/19/1998 14:54:40;
# END PATH path(4)
```

6. Conclusions

Use Cases lend themselves particularly well to Operational Profile development. There is a natural mapping of Actors in the Use Cases to Initiators of Operations in the Operational Profile. Mapping use cases to operations is an efficient and effective way to build the operations list for the operational profile. A necessary step to the translation of use cases to operations is analysis of the use cases diagrams and understanding how use cases relate. Use Case <<uses>> and <<extends>> stereotypes should be given attention when refining the operations list. By leveraging

use case requirements modeling the work needed to produce an operational profile can be greatly reduced.

System engineers regard modeling the Use Cases with TestMaster as another way to further nail down requirements. It provides clarity for both testers and developers and can be used as a way to uncover requirement inconsistencies. It gives organization to the use cases that is sometimes needed to understand how various use cases interact.

The initial modeling from requirements gives the Test engineers a good framework to start their work of test design and creation. At some point knowledge of physical implementation beyond what is provided by the use cases is necessary. Use case requirements modeling and extension into both the operational profile and modeling provides a consistent point of reference from requirements through testing.

TestMaster's various coverage schemes include profile coverage. Relative likelihood information can be annotated on various transitions. This allows operational profile occurrence information to be recorded within test models. This information when provided in the model will cause the generated tests to reflect a mix of operations tests as documented in the operational profile.

Use case requirements modeling provides a strong point of focus for subsequent operational profile development. Use cases provide a common point of reference connecting the operational profile to test design and creation. These approaches are an effective way to strengthen the realization of requirements through development, test and into field release.