9.0 OPERATIONAL PROFILES

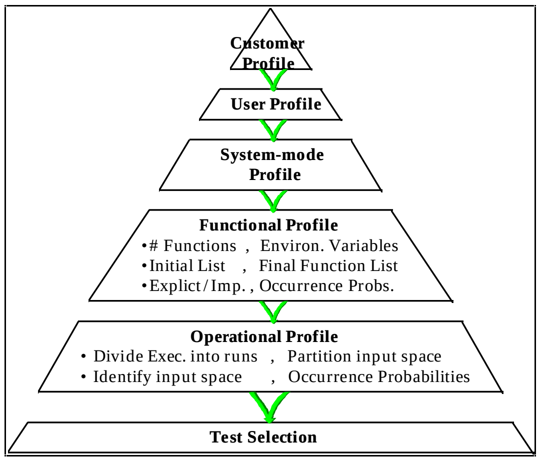
Similar to hardware, software performance is significantly dependent on the environment in which it operates. The similarity ends there. With hardware, the environment physically changes a piece of equipment. This physical change is mainly responsible for faulty behavior. A software system doesn't change, but can still fail due to the inputs it receives from the external environment.

The reliability of a software-based product depends on how the computer and other external elements will use it[1]. Making a good reliability estimate depends on testing the product as if it were in the field. The *operational profile* (OP), a quantitative characterization of how the software will be used, is therefore essential in any Software Reliability Engineering (SRE) application. It is a fundamental concept which must be understood in order to apply SRE effectively and with any degree of validity. This section provides a detailed description of the OP.

A *profile* is a set of independent possibilities called *elements*, and their associated probability of occurrence. If operation *A* occurs 60 percent of the time, *B* occurs 30 percent, and *C* occurs 10 percent, for example, the profile is [*A*, 0.6...*B*, 0.3...*C*, 0.1]. The *operational* profile is the set of independent operations that a software system performs and their associated probabilities. Developing an operational profile for a system involves one or more of the following five steps:

1. Find the customer profile
2. Establish the user profile
3. Define the system-mode profile
4. Determine the functional profile
5. Determine the operational profile itself

The process for developing the operational profile is depicted in Figure 9-1.



Customer Profile
User Profile
System-mode Profile
Functional Profile
•# Functions , Environ. Variables
•Initial List , Final Function List
•Explict/Imp. , Occurrence Probs.
Operational Profile
• Divide Exec. into runs , Partition input space
• Identify input space , Occurrence Probabilities
Test Selection

FIGURE 9-1. Operational Profile Development[2]

[1] Musa, John D;Iannino, A.; Okumoto, K; "Software Reliability Measurement", Prediction, Application, McGraw-Hill, 1987.
[2] Musa, J.D., "Operational Profiles in Software Reliability Engineering," IEEE Software Magazine, March 1993,

9.1 Customer Profile.

A customer is the individual, group or organization that is purchasing the software system, the same as any other product. A *customer profile* consists of an array of independent customer types. A *customer type* is one or more customers in a group that intend to use the system in a relatively similar manner, and in a substantially different manner from other customer types.

An example of a software system with different customer types would be a spreadsheet package. Various customers could be educational institutions, businesses, and individual home users. Each of these types of customers may be expected to utilize the spreadsheet in a substantially different way. For instance, schools might use them for tabulating and updating student grades. Businesses might use them mainly for financial and operations controls. Home users could keep track of their monthly income and expenses, as well as investments and savings plans. The customer profile is the list of customer types and the associated probabilities. These probabilities are simply the proportions of time that each type of customer would be using the system. A customer profile for the example here might be as in Table 9-1.

TABLE 9-1. Sample Customer Profile

Customer	Occurrence Probability
Educational Institution	0.45
Business Organization	0.35
Individual Home User	0.20

9.2 User Profile.

A system's users may be different from the customers of a software product. A user is a person, group, or institution that operates, as opposed to acquires, the system. A user type is a set of users that will operate the system similarly. Identification of different user types allows the task of operational profile development to be divided among analysts. The *user profile* is the set of user types and their associated probabilities of using the system.

9.3 System Mode Profile.

A system mode is a way that a *system* can operate. The system includes both hardware and software. Most systems have more than one mode of operation. For example, system testing may take place in batch mode or user-interactive mode. An airplane flight consists of takeoff and ascent mode, level flight mode and descent and land mode. An automobile may be in normal mode or four-wheel drive; it may also be in normal mode or cruise control. System modes can be thought of as independent segments of a system operation or various different ways of using a system. A system can switch among modes sequentially, or it can permit several modes to operate concurrently, sharing the same system resources. For each system mode, if there are more than one or two, an operational profile (and sometimes functional profile) should be developed. There are no technical limits on how many system modes may be established.

The system mode profile is represented by the list of system modes and their corresponding occurrence probabilities. Table 9-2 shows an example of a system mode profile.

TABLE 9-2. System Mode Profile

System Mode	Occurrence Probability
Batch Mode	0.65
User-Interactive Mode	0.35

9.4 Functional Profile.

After a good system mode profile has been developed, the focus should turn to evaluation of each system mode for the functions performed during that mode, and then assigning probabilities to each of the functions. Functions are essentially tasks that an external entity such as a user can perform with the system. For instance, the user of an e-mail system would want the following functions: create message, look up address, send message, open message, etc. Functions are established during requirements based on what activities the customer wants the system to be able to perform. Developing a functional profile is, in that respect, a part of developing requirements.

A functional profile need not have a defined number of functions, but generally contains 20 to more than a hundred. The number will vary based on project size, number of system modes, environmental considerations, and function breadth.

The functional profile can be either *explicit* or *implicit*, depending on the key input variables. A *key input variable* is an external parameter which affects the execution path a software system traverses based on the different values the parameter takes on. These key parameter variables in many cases consist of ranges of variables that cause different operations to be performed. These various ranges are referred to as *levels*. A profile is explicit if each element is designated by simultaneously specifying the levels of all key input variables needed for its identification. A profile is implicit if it is expressed by subprofiles of each key variable. That is, each key environmental parameter is assigned probabilities associated with the ranges it can legally use.

Suppose there are two key independent parameters, X and Y, each taking on three discrete values. Nine operations can be defined based on the combinations of the variables. Example implicit and explicit operational profiles are shown in Table 9-3. The main advantage of using the implicit profile is that a significantly smaller number of elements need to be specified, as few as the *sum* of the number of levels of key input parameters. The explicit profile can have as many as the *product* of the number of levels for each variable. For five variables with five levels, assuming complete independence, the implicit profile requires only 25 elements whereas the explicit profile would call for 5^5, or 3,125 elements. In most cases it is not necessary to generate the explicit profile, because it exists by default from the implicit profile.

TABLE 9-3(a). Sample Implicit Operational Profile

Subprofile C		Subprofile D	
Key input variable value	Occurrence probability	Key input variable value	Occurrence probability
X1	0.6	Y1	0.7
X2	0.3	Y2	0.2
X3	0.1	Y3	0.1

TABLE 9-3(b). Sample Explicit Operational Profile

Key input variable values	Occurrence probability
X1Y1	0.42
X2Y1	0.21
X1Y2	0.12
X3Y1	0.07
X1Y3	0.06
X2Y2	0.06
X2Y3	0.03
X3Y2	0.02
X3Y3	0.01

Procedure 9.4.1 - Generating a Functional Profile.
Development of the functional profile generally involves the following four steps:

1. Generate an initial function list
2. Determine environmental variables
3. Create final function list
4. Assign occurrence probabilities

The initial function list should be comprised of features and capabilities needed by the users. This list can be organized by functions relevant to each key input variable if an implicit profile is used. Features should be obtained from the customer and/or users, and may be stored in a system requirements specification. The functions should be identified rather easily if a good job of requirements evaluation has been done. If functions are difficult to identify, the requirements may be incomplete or unclear. A Request For Proposal (RFP) is often issued for government programs which contains a list of capabilities or features that a system must have.

The next step is to define the *environmental* input variables and their value ranges that segregate development. Functions will many times be broken up and allocated to different modules for development based on environmental variables. These environmental variables characterize the conditions that influence the paths traversed by a program, but do not correspond directly to features. Examples of environmental variables include hardware configuration and traffic load. The system design team should work together to brainstorm a list of environmental variables that would cause different behaviors of the software program, and pare this list down.

Prior to creating the final function list the key environmental and feature variables should be examined for dependencies. The list should be as orthogonal as possible. If one variable is significantly dependent on another, it can be eliminated from the final function list. Partial dependencies can cause difficulties because all possible combinations of levels of both variables may need to be listed. Those interactions which cannot occur and which have insignificant likelihood of ever occurring should be removed from the list. The final number of functions in the list is then calculated as the product of the number of functions in the initial list and the number of environmental variable levels, minus the combinations of initial functions and environmental variable values that do not occur. The final function list consists of the functions and environmental variables for each function.

TABLE 9-4. Sample Final Function List

Function	Environmental Variable
Standard Deviation	X
	Y
Correlation	X
	Y
Analysis of Variance	X
	Y
Regression	X
	Y

The final step in functional profile development is the assignment of occurrence probabilities. The ideal data source for these values consists of usage measurements taken on the latest release or a similar system. These measurements may be obtained from system logs or data storage devices. Occurrence probabilities computed with the historical data should be updated to account for new functions, users, or environments. In the event that a system is completely new the functional profile might be very inaccurate. It should still be developed, however, and updated as more is known about how the system will be operated. The process of predicting usage forces interaction with the customer, which can be very important. The required dialogue may highlight the relative importance of the various functions, indicating that some functions may not be necessary while others are most significant. Reducing the number of functions should increase reliability[3].

TABLE 9-5. Sample Functional Profile Segment

Function	Chi-Square System Mode Occurrence Probability	Overall Occurrence Probability
Standard Deviation	.60	0.12
Correlation	.22	0.044
Analysis of Variance	.10	0.02
Regression	.08	0.016

[3] Lyu, Michael R. "Handbook of Software Reliability Engineering", IEEE Computer Society Press, 1996.

TABLE 9-6. Sample Environmental Profile

Variable Count	Occurrence Probability
One (X)	0.6
Multiple (Y)	0.4

TABLE 9-7. Sample Final Functional Profile Segment

Function	Chi-Square System Mode Occurrence Probability	Overall Occurrence Probability
Standard Deviation	X	0.072
	Y	0.048
Correlation	X	0.0264
	Y	0.0176
Analysis of Variance	X	0.012
	Y	0.008
Regression	X	.0096
	Y	.0064

9.5 Operational Profile.

Figure 9-2 shows the elements involved in determining operational profiles from functions. A function may comprise several operations. In turn, operations are made up of many run types. Grouping run types into operations partitions the input space into domains. A domain can be partitioned into subdomains, or run categories. To use the operational profile to drive testing, first choose the domain that characterizes the operation, then the subdomain that characterizes the run category, and finally the input state that characterizes the run[4].

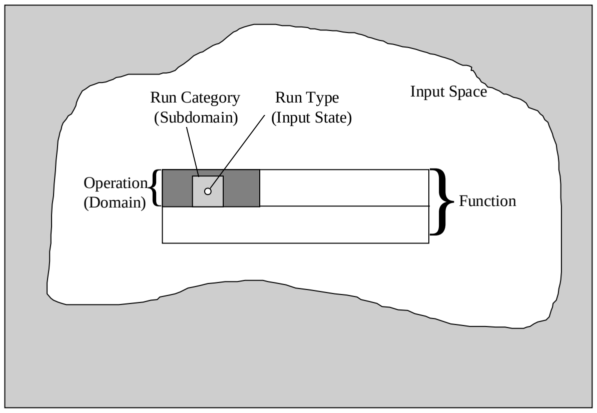


FIGURE 9-2. Operational Elements[5]

The functional profile is a *user*-oriented view of system capabilities. From the developers' perspective, it is *operations* that actually implement the functions. Operations are usually the focus of testing. An operation represents a task being accomplished by the system from the viewpoint of the people who will test the system. To allocate testing effort and develop a test description, the operational profile must be available for the purposes of test planning.

The *operational architecture* describes how the user will likely employ operations to accomplish functions. In general there are more operations than functions, and, as Figure 9-2 shows, operations tend to be more refined. The primary objectives in determining the operational profile include listing the operations and determining the occurrence probabilities.

The list of operations can be extracted from the functional profile, mapping functions to operations using the operational architecture of the system. If possible, functions should be defined so that there is a one-to-many correspondence between functions and operations. This greatly simplifies the derivation of operational profile from functional profile.

The three steps in determining the operations list are to divide the execution into runs, identify the input space, and partition the input space into operations.

[4] Musa, J.D., "Operational Profiles in Software Reliability Engineering," IEEE Software Magazine, March 1993, p.14-32.
[5] Musa, J.D., "Operational Profiles in Software Reliability Engineering," IEEE Software Magazine, March 1993,
© 1993 IEEE, reprinted with permission.

Divide execution into runs. Operations are characterized by runs. A run is essentially a discrete task performed by a system in response to external variable(s). Runs from the same class form a run type. To be in the same class the runs should take the same execution path through the software. Any variation in the *level* of any input variable results in a separate run type. Each run type has an associated input state, a set of input variables and their associated levels, that generates the run. The input state from an input *space* uniquely determines the path of control of an execution of a run. An input parameter is allowed only one value for each run. Externally initiated interrupts would be considered input variables because they are parameters which exist external to the program. Intermediate data items, obtained within a run, are not input variables.

Identify input space. A software system's input space, combinations of inputs and the number of different values they can take on, is prohibitively large for a program of even low complexity. An input state profile is the set of input states and their associated occurrence probabilities. The complete input state profile is never really measured in practice because any continuous variable can potentially take on an infinite number of values. It is important to understand because it can help in identifying approximations that cost-effectively approach the ideal scenario. An input space can be identified by simply listing the set of input variables involved.

Partition input space. In practice, the profiles should be limited to several hundred elements due to cost restrictions. The input space can be dissected by selecting ranges for input variables that belong to the same run type. By grouping run types into operations, the input space can be partitioned to significantly reduce the number of elements. The larger the ranges (less levels), the smaller the input space. The ranges should be selected so that each range represents a different execution path. Each combination of ranges then would represent a run type. Partitioning in this way provides the framework for sampling non-uniformly across the input space. If operations are selected randomly according to the operational profile and input states drawn randomly from the input space, non-uniform random tests matching operational profile will be selected.

Reducing the number of operations. The operational profile may create an unrealistic set of tests because the list of operations is too long. There are at least three ways to restructure it:

1. Reduce the number of run types.
2. Increase the number of run types grouped per operation.
3. Ignore the remaining set of run types expected to have total occurrence probability appreciably less than the failure intensity objective.

The number of run types can be reduced by reducing either the size of the input variable list or the number of levels of the input variables. To potentially reduce the number of input variables, one of the following can be performed:

1. Reduce functionality.
2. Reduce the number of possible hardware configurations.
3. Restrict the environment the program must operate in.
4. Reduce the number of fault types.
5. Reduce unnecessary interactions between successive runs.

The first four of these alternatives change the system's features, thus impacting the customer and reducing flexibility, robustness, and possibly reliability. The fifth option, however, is a desirable one. Design changes can be implemented which should not impact functionality for the customer.

Methods for reducing run interactions are as follows:

1. Minimize the input variables that application programs can access at any one time.
2. Reinitialize variables between runs.
3. Use synchronous, as opposed to asynchronous, design.

Although reducing interactions can effectively reduce input space, it adds another level of complexity. It is much more risky than the other approaches to reducing input space.

<u>Occurrence Probabilities.</u> After each input variable is partitioned into ranges, probabilities associated with each range for each variable must be identified. In some instances field data may already exist on the frequency of key input variable ranges. It is highly recommended that an attempt to find this data or data on a similar system be undertaken. In the long run, this may be more cost effective than attempting to estimate probabilities with no usage background. Validity of reliability estimates is directly related to the proximity of the probability estimates to the actual occurrences in the field.

To minimize the risk of obtaining inaccurate reliability estimates, start by taking measurements of input variable ranges on an initial release of a system, updating the occurrence probabilities used for testing associated with later releases if it is determined that the estimates made during testing differ from those observed in a target environment in the field. Beta testing may be useful.

The initial estimation effort should be performed by an experienced systems engineer or someone who has a thorough understanding of both the system and user needs. Experienced users should be interviewed to verify that estimates are within reason.

It may also be helpful to create an interaction matrix of input variables plotted against other key input variables. The matrix should reveal combinations of variables that do not occur or contain dependencies. The remainder of the matrix contain independent combinations where the estimates of occurrence probabilities are the product of individual input variable probabilities.

<u>Example 9.1 Data-driven system:</u> Financial and billing systems are commonly data driven. Suppose a cable television billing system was designed as an account processing system. This system processes the charge entries for each account for the current billing period and generates bills. The reliability to evaluate is the probability of generating a correct bill. This involves determining the reliability over the time required to process the bill and its entries.

Assume that the design was not anticipated when the functional profile was developed, so the relationship between the functional profile and operational profile is complex. For instance, typical functions might have been bill processing, bill correction, and delinquency identification.

The account-processing system has an operational profile that relates to account attributes. Its operations are classified by customer type (business or residential), service type (basic, expanded basic, premium package), and payment status (paid, delinquent).

Assume that 90 percent of the customers are residential and 10 percent are businesses. Forty percent of the customers subscribe to the basic cable service. Half of all customers receive expanded basic, and the remaining 10 percent pay for the full premium package. History shows that 2 percent of the accounts are delinquent, on average. Table 9-8 shows the set of operations and the associated probabilities.

TABLE 9-8. Operational Profile for Account-Processing Billing System

Operation	Occurrence Probability
Residential, Expanded Basic, Paid	0.4410
Residential, Basic, Paid	0.3528
Residential, Premium, Paid	0.0882
Business, Expanded, Paid	0.0490
Business, Basic, Paid	0.0392
Business, Premium, Paid	0.0098
Residential, Expanded, Delinquent	0.0090
Residential, Basic, Delinquent	0.0072
Residential, Premium, Delinquent	0.0018
Business, Expanded, Delinquent	0.0010
Business, Basic, Delinquent	0.0008
Business, Premium, Delinquent	0.0002

This concludes the general discussion on operational profiles.

Following is an example of an operational profile development for an actual system. The system name is not revealed and some data has been altered, but the general concepts have been retained.

Example 9.2 - Missile Application:
The reliability requirement for this system stated that the missile must land *within 10 meters of target* not less than 95 percent of the time. In addition, captive carriage reliability should be at least 0.98. Captive carriage reliability is defined as the probability that the air vehicle will pass an Initiated Built-in-Test (IBIT) between takeoff and air vehicle release. These system level requirements were allocated to the hardware configuration items and the flight software. The flight software received an allocation of 0.97 for free-flight, meaning that failure of the missile to land within 10 meters of the target due to a software fault could occur no more than three out of 100 flights. The software reliability requirement for IBIT was 0.995.

The Operational Flight Software (OFS) represents one computer software configuration item (CSCI). To develop the operational profile, the process in Section 9 and depicted in Figure 9-1 was followed. The first step was to determine the customer profile. There were two military customers for this product, the Air Force and the Navy. It was estimated that the Air Force would acquire about 73 percent of the missiles. The customer profile is shown in Table 9-9.

TABLE 9-9. Missile Customer Profile

Customer	Occurrence Probability
Air Force	0.73
Navy	0.27

The user profile was developed by looking at the types of aircraft on which the missile would be carried. The Air Force had three fighter aircraft which could launch the missile, the F-100, F-200 and F-300. The Navy had two attack planes, A-25 and A-50, which could carry the missile. Estimates were made for how often the missile would be launched from each type of aircraft. The user profile established is shown in Table 9-10. An operational profile should be considered for each type of aircraft because flight parameters varied significantly among them.

TABLE 9-10. Missile User Profile

Aircraft	Occurrence Probability
F-100	.38
F-200	.22
A-25	.21
F-300	.13
A-50	.06

The next step was to determine a system mode profile. There were two main system modes identified in the requirements, free-flight and IBIT. The OFS operated in other modes during a mission, but those modes were not considered critical to mission success, so the system mode profile, shown in Table 9-11, was limited to the two modes in the requirements. The occurrence probabilities were obtained by estimating the average time the system would be in each mode during a mission. The IBIT mode was generally expected to last 20 seconds. The free-flight portion was estimated to be 90 seconds. Each mode would occur once during each mission. For more granularity, a system mode profile could have been developed for each aircraft user.

TABLE 9-11. Missile System Mode Profile

Mode	Occurrence Probability
Free-flight	0.818
Initiated built-in-test (IBIT)	0.182

The software modules (CSCs) which function during IBIT and flight are shown in Table 9-12. Note that some of the modules are operating for each system mode.

TABLE 9-12. Missile Software Modules

Free-flight	IBIT
Aided Navigator	IMU Controller
Strapdown	GPS Controller
Navigation Solution	LP Controller
Kalman Filter	Mission Sequencer
IMU Controller	Message Handler
GPS Controller	Core/Alternate
Autopilot	TAS Controller
TAS Controller	JPF Controller
JPF Controller	Vehicle Status
Guidance Processor	Discretes
Vehicle Status	Commun. Controllers
Discretes	1553B Controller
Commun. Controllers	GEM Interface Controller
GEM Interface Controller	Utilities
Telemetry	

For this particular system it was not necessary to develop a functional profile by enumerating the probabilities associated with a set of functions. Instead, the systems engineers decided that the operational variables relevant to each system mode were easily determined. The operations used were simply the system modes themselves. An operational profile could be developed from the list of key input variables. The variables for IBIT and Free-flight are shown below. If these parameters are varied, the OFS will operate differently (take a different path of control).

IBIT Factors	**Free-Flight Factors**
- IMU BIT Result	- GPS Available
- IMU Result Time	- Target Orientation
- GPS BIT Result	- Axis Alignment
- GPS Result Time	- Trajectory
- TAS BIT Result	- Speed
- TAS Result Time	- Delay Time
- JPF BIT Result	- Alignment Duration
- JPF Result Time	- TA Interrupt Time
- ATE BIT Result	- Release Level
- ATE Result Time	- Angle of Impact
- Discrete Results	- Down Range Distance

Armed with a list of environmental variables, the next step was to establish ranges for each of the parameters, and the number of levels each variable should be partitioned into. For simplicity, it was decided to limit every parameter to just two levels. This was also done to allow another approach to testing using *Design of Experiments*. The parameters with associated variable ranges and estimated probabilities of occurrence for IBIT are shown in Table 9-13.

TABLE 9-13. IBIT Operational Profile for Missile OFS

Parameters	Values	Probabilities
IMU BIT Result	(Pass, Fail)	(.99 , .01)
GPS BIT Result	(Pass, Fail)	(.99 , .01)
TAS BIT Result	(Pass, Fail)	(.99 , .01)
JPF BIT Result	(Pass, Fail)	(.99 , .01)
ATE BIT Result	(Pass, Fail)	(.99 , .01)
Discrete Results	(Pass, Fail)	(.99 , .01)
IMU Result Time	(2-10, 10-20)	(.05 , .95)
GPS Result Time	(2-10, 10-20)	(.05 , .95)
TAS Result Time	(2-10, 10-20)	(.10 , .90)
JPF Result Time	(2-10, 10-20)	(.15 , .85)
ATE Result Time	(2-10, 10-20)	(.50 , .50)

The parameters with associated variable ranges and estimated probabilities of occurrence for Free-flight are shown in Table 9-14.

TABLE 9-14. Free-flight Operational Profile for Missile OFS

Parameters	Values	Probabilities
GPS Available	(Yes, No)	(.75 , .25)
Target Orientation	(Horiz., Vert.)	(.90 , .10)
Axis Alignment	(On, Off)	(.70 , .30)
Trajectory	(Normal, Loft)	(.80 , .20)
Speed	(Slow, Fast)	(.60 , .40)
Delay Time	(Short, Long)	(.95 , .05)
Alignment Duration	(Short, Long)	(.75 , .25)
TA Interrupt Time	(Short, Long)	(.85 , .15)
Release Level	(40K, 18K)	(.90 , .10)
Angle of Impact	(Low, High)	(.65 , .35)
Down Range Dist.	(Short, Long)	(.50 , .50)

Each of these profiles are *implicit*. A specific input state would be generated by randomly selecting a value for each one of these parameters. A range would first be selected in accordance with the probability of occurrence. For example, there is a 60 percent chance that a slow speed would be selected. A slow speed may be in the range of 0.4 to 0.9 Mach. Then a value within this range would be randomly selected. All values within a range have equal likelihood of being selected.

These operational profiles were used to generate test cases for the OFS during system test. Using random testing, a good representation of the expected field operation can be tested. Figure 9-3 shows the distribution of points selected for Speed and Release Level using the Free-flight operational profile. Running a substantial number of tests allows a good level of input space coverage to be achieved.

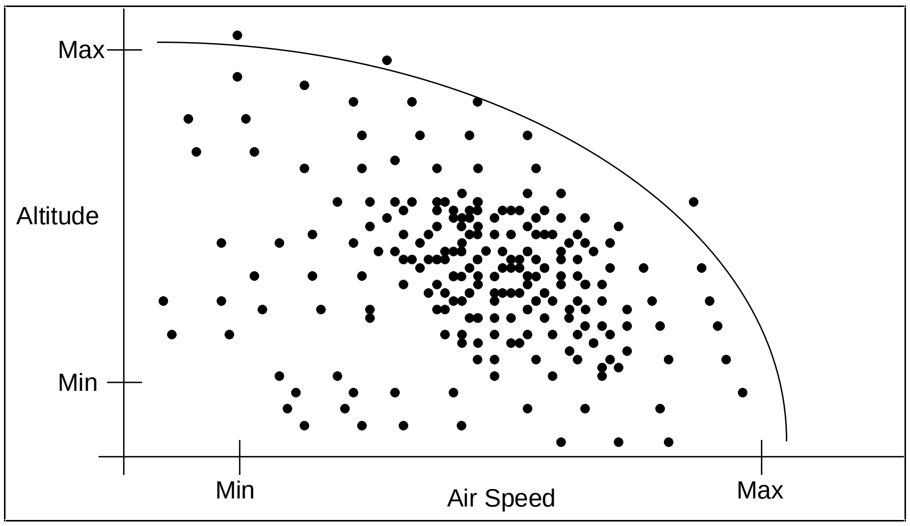


FIGURE 9-3. Plot of Selected Parameters from Free-flight Operational Profile

There are a few points regarding the operational profile that this example helps to bring out. One has to do with how reliability requirements are stated. Another is concerned with the meaning of user profiles. Finally, the topic of functional profiles is addressed.

The requirements stated that certain levels of reliability had to be achieved for the system. There was no separate requirement for the Air Force and the Navy. From this an assumption may be made that both customer requirements are the same. If the system is tested, and reliability modeling shows that the requirements have been met through growth testing, is the system ready to be released to both customers? What happens if the system gets out to the field and the reliability experienced by the Air Force is 0.98 while the Navy is 0.96 for the Free-flight software? The point here is that separate reliability calculations may be needed for each customer.

What purpose, if any, does the user mode profile serve in this example? The user mode is actually very significant here. Environmental variables such as altitude are significantly different for each of the aircraft types. Nominal values were taken from each aircraft for the parameters. Then the operational profile was generated by taking a weighted sum of key variables using the occurrence probabilities from the user profile. An alternative method would have been to generate operational profiles for each aircraft (users) individually and tested them separately.

A functional profile was not developed for this system. A functional profile, remember, is a system view from the perspective of a user. In this case there is really no human user that the system interacts with. It would be difficult to determine a functional profile in this case, although the functions are known (modules in Table 9-10). There was simply no reason to develop one.

9.6 Operational Profile Development from Object-Oriented Analysis/Design.

Object-Oriented Analysis and Design (OOA/D) provides a sound structure for developing the operational profile. Because the use of object-oriented techniques are widespread, it is important to understand the linkage between object design and the operational profile. This section assumes a general understanding of object-oriented analysis.

Starting with the *use case* model in the object paradigm, an operational profile can be developed. Use cases consist of scenarios, actors and entities. Scenarios are developed by selecting typical interaction sequences of actors and external systems with the system. The goal is to examine the role of a system. Specific interactions between the system and other components should be modeled as use cases. As an example, consider Figure 9-4 where a customer would walk up to a vending machine, insert two quarters, press the Pepsi button and remove the Pepsi.

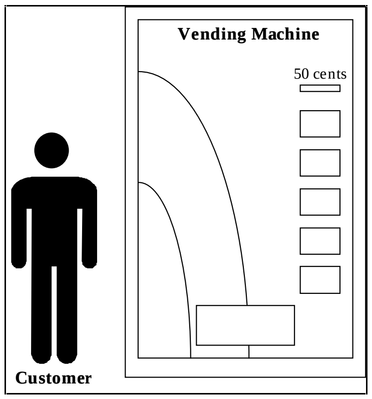


FIGURE 9-4. Vending Machine Object Representation

A customer purchasing a soda describes the general use case. A scenario is an instance of a use case (e.g., John inserts one quarter, two dimes and a nickel, presses Sprite and gets Sprite). The customer is an external entity. An external entity is a user or external system that interacts with the system. Here the external entity is an actor who plays the role of a customer.

Each way that a system can be used represents a use case. Figure 9-5 shows two use cases involving a stereo system. A use case describes the potential sequences of interaction between the system, initiator and other actors. The system is treated as a black box from a user's perspective.

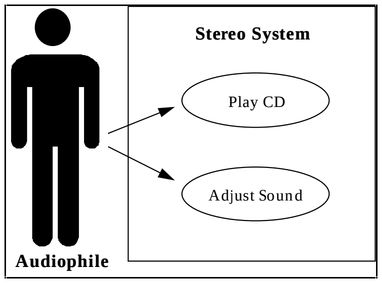


FIGURE 9-5. Stereo System Use Cases

Documentation of a use case usually includes a textual description of the transactions involved, described from a user's point of view. Figure 9-6 shows an example use case description.

Use Case: Play CD

Actors: Audiophile
Description:
(1) Audiophileselects a CD
(2) Audiophile inserts the CD
(3) Audiophile selects a track
(4) Audiophile presses play
(5) System plays the CD

Precondition: System turned on

Postcondition: Stop at end of CD

FIGURE 9-6. Documented Use Case

For the object model to be useful it needs to be dynamic. A dynamic model is characterized by *events* and *operations*. An event is an external stimulus to an object. A sequence of events forms a *scenario*. The object interaction of a scenario can be described by an *event trace* (Figure 9-7).

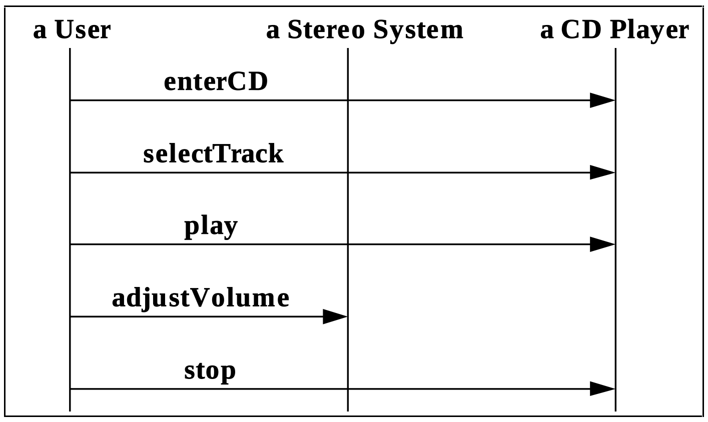


FIGURE 9-7. Event Trace Example

While an event causes some behavior of an object, it has no time associated with it. For something to occur over time, the object paradigm provides *operations*. There are two types of operations. An *activity* is associated with a state of the system and lasts for some duration. An *action* is associated with a transition between states and has no significant duration. An operation either occurs for some time span in a static system state, or it performs some action that transitions between states. This is important in consideration of the operational profile.

The elements of the object paradigm presented thus far can be mapped to the operational profile concept. A *use case* is synonymous with the operational profile *function*. A set of *events* is similar to an operational scenario. *Event traces* help to identify *key environmental variables*. An *operation* in the object paradigm is conveniently the same as the *operation* in the profile. Finally, a test case (not described here) is similar to a *run* for the operational profile.

A use case is a user-oriented perspective of what functions the system can perform. Events make up an operational scenario. They trigger certain operations to occur. These operations are what are tested. Use cases as defined in the object-oriented analysis approach to software design are not yet ready for choosing test cases. Starting with the use case, an inventory of operational variables needs to be identified. The domain constraints are then determined, along with the relative frequency of each use case. Then tests are generated from this *extended* use case profile.

The system test suite is derived from extended use-cases and event trace diagrams. A use-case is extended by adding an operational relation and estimates of relative usage frequency. Frequency estimation allows testing under the operational profile. This maximizes field reliability subject to available testing resources by testing most frequently used capabilities first.

In a reliability-driven, test-to-MTTF project, testing is conducted in accordance with the operational profile until an acceptably low failure rate is observed. Three levels of system testing may be performed: (1) use-case compliance (extended use-case coverage), (2) reliability optimization (level 1 under control of the operational profile), and (3) integrity verification[6].

The steps involved in Reliability Optimization testing using OOA/D are as follows:

1. Develop a complete object model, including all use cases.
2. Identify key operational variables.
3. Establish operational relationships for each use case.
4. Develop an operational profile for all use cases.
5. Estimate the frequencies of uses cases, then operations.
6. Estimate testing productivity.
7. Evaluate system test effort budget.
8. Evaluate coverage, add more tests, if necessary.

Table 9-15 shows a couple of use-case examples.

[6] Binder, Robert V. "Using Operational Profiles with Object-Oriented Specifications", Presented at the AT&T SRE Users Group Meeting in Naperville, IL., June 6, 1996.

TABLE 9-15. Use Case Examples[7]

Use Case	Actor	Scenario
Cash Withdrawal	Bank Customer	Wrong PIN entered once, request $75
	Bank Customer	PIN OK, deposit $300, request $50
	Crook	Stolen card inserted, valid PIN entered
ATM Cash Restocking	Operator & Guard	ATM opened, cash dispenser empty, $15,000 is added
	Operator & Guard	ATM opened, cash dispenser is full

With the use cases established, the next step is to identify operational variables. Table 9-16 shows a set of operational relationships.

TABLE 9-16. Operational Relationships[8]

Operational Variables				*Expected Result*	
Card PIN	**Entered PIN**	**Customer Bank Reply**	**Customer Acct. Status**	**Message Displayed**	**Card Action**
Invalid	-	-	-	Insert ATM Card	Eject
Valid	Matches Card PIN	OK	Closed	Account Closed	Eject
Valid	Matches	OK	Open	Enter Amount	Keep
Valid	Matches	No Reply	-	Try Later	Eject
Valid	Doesn't Match	-	-	Reenter PIN	Keep
Revoked	-	Bank Replies	-	Card Revoked	Retain
Revoked	-	No Reply	-	Card Invalid	Eject

This information can be used for test planning. Assume the following defaults for test variables:

- It takes 1 hour to design and run one test
- 5 percent of tests reveal faults
- It takes 4 hours to correct each fault
- Test budget = 1000 hours

[7] Binder, Robert V. "Using Operational Profiles with Object-Oriented Specifications", Presented at the AT&T SRE Users Group Meeting in Naperville, IL., June 6, 1996, reprinted with permission.
[8] Binder, Robert V. "Using Operational Profiles with Object-Oriented Specifications", Presented at the AT&T SRE Users Group Meeting in Naperville, IL., June 6, 1996, reprinted with permission.

From this information, the number tests to be performed is obtained as:

$$T + (0.05 * 4T) = 1000,$$

$$T = 833.$$

These 833 tests should be allocated among the operational profile shown in Table 9-17.

TABLE 9-17. Test Planning Based on Operational Profile[9]

Use Case	Occurrence Probability	Number of Tests
Cash Withdrawal	0.53	441
Checking Deposit	0.15	125
Savings Deposit	0.14	117
Funds Transfer	0.08	67
Balance Inquiry	0.06	50
Restock	0.02	17
Collect Deposits	0.02	16
Total	**1.00**	**833**

[9] Binder, Robert V. "Using Operational Profiles with Object-Oriented Specifications", Presented at the AT&T SRE Users Group Meeting in Naperville, IL., June 6, 1996, reprinted with permission.