Chapter

5

The Operational Profile

John Musa, Gene Fuoco, Nancy Irving, Diane Kropfl
AT&T Bell Laboratories

Bruce Juhlin
U.S. West

5.1 Introduction

A software-based product's reliability depends on just how a customer will use it [Musa87]. Making a good reliability estimate depends on testing the product as if it were in the field. The operational profile, which is a quantitative characterization of how a system will be used, is thus essential in software reliability engineering (SRE). Operational profile theory can be applied to hardware as well as software, and even to human components. Thus it is applicable to complete systems. The operational profile shows you how to increase productivity and reliability and speed development by allocating development and test resources to functions on the basis of use. It helps you plan test activities, generate test cases, and select test runs.

Using an operational profile to guide system testing ensures that if testing is terminated and the software is shipped because of imperative schedule constraints, the most-used operations will have received the most testing, and the reliability level will be the maximum that is practically achievable for the given test time. In guiding regression testing, it tends to find, among the faults introduced by changes, the ones that have the most effect on reliability.

An example of the benefits from developing and applying the operational profile is shown by AT&T's International Definity® project (a PBX switching system). This project combined the operational profile with other quality-improvement techniques to reduce customer-reported problems and maintenance costs by a factor of 10, system-test interval by a factor of 2, and product-introduction interval by 30 per-

cent [Abra92]. The system experienced no serious service outages in the first two years of deployment; customer satisfaction improved significantly. The marked quality improvement and a strong sales effort resulted in an increase in sales by a factor of 10.

In a similar quality-improvement program, Hewlett-Packard applied software reliability engineering and the operational profile to reorganize their system-test process for a multiprocessor operating system. With automated test and failure recording and using the operational profile to guide testing, they reduced system-test time and cost by at least 50 percent.

The cost of developing an operational profile varies. Our experience indicates that the effort to construct the operational profile for an average project—about 10 developers, 100,000 source lines, and a development interval of 18 months—is about one staff month. Large projects can cost more, but the increase is clearly less than linear with project size. International Definity invested two to three staff years in extensive customer study that led to an operational profile. Of course, every project requires good knowledge of the customer base, so only a portion of this effort can reasonably be charged to the operational profile. Also, the work can be written off over several releases, with only minor updating needed between them.

Experience to date indicates that operational profiles are beneficial even when very simple and approximate. For example, one project used an operational profile defined on only five operations. However, defining operational profiles in the range of 50 to 200 operations has usually been definitely worth the extra effort.

5.2 Concepts

A *profile* is simply a set of disjoint (only one can occur at a time) alternatives called *elements,* each with the probability that it will occur. If *A* occurs 60 percent of the time and *B* 40 percent, for example, the profile is *A,* 0.6 and *B,* 0.4.

In order to select and define the terms we will use in analyzing a work process and developing the operational profile for the system that will implement the activities of the work process, we must recognize two practical needs.

The first is the need to distinguish between the view of the system taken in specifying its requirements and the view of the system as it is built. We will use terms containing the word *function* at the requirements stage and terms containing the word *operation* when dealing with the system as it is being built.

The second need arises from the situation that is common to most work processes. There is a need to talk about tasks that represent the

smallest divisions of work that can be initiated by external intervention, either human or by a system external to the one being analyzed. Although work can always be divided into smaller packages, there is no useful purpose and only greater cost and reliability risk involved in allowing initiation access to these packages if the same sequence of work packages will always be followed. We will call these smallest initiable divisions *runs.* In a switching system, a run might be a telephone call. In interactive systems, it might be a command input by a user. In transaction-based systems, it might be a transaction. In an aircraft-control system it might be a maneuver.

Runs are associated with input states. The *input state* is the complete set of input variables of the system, an *input variable* being any data elements that exist external to the system and influence it. For example, externally initiated interrupts, such as interrupts generated by the system clock, by operator actions, and by other components of the system outside the program, are input variables. Intermediate data computed by the program during a run and not existing external to the program are not input variables. Hence, interrupts generated directly by a run or interrupts that are determined from other input variables (for example, overflow and underflow) are not input variables.

Input variables don't just include the parameters that are explicitly transmitted. For example, data elements that affect a system may be unknown to the designers; nevertheless, they are input variables. The concept of an input variable is logical, not physical. For example, a physical memory location can be time-shared by different input variables.

Runs that have identical input states are said to belong to the same *run type.* Airline-reservation transactions that are exact duplicates have the same run type. However, reservations made for different people, even on the same flight, are different run types. If you test all run types, you have exhaustively tested the system. You do not need to execute repeated runs or instances of the same run type.

A *function* is a set or grouping of run types, as conceived at the requirements stage. An *operation* is a set or grouping of run types for the system as built. One might ask why we should group run types. The main reason is that practical systems usually have astronomical numbers of run types. It is totally impractical to collect usage on run types, unless you do it only for a very small and important subset. We need a much smaller number of elements (generally not more than hundreds) on which we will collect usage data. Functions and operations are defined and used for that purpose. The *functional profile* is a profile of functions; *operational profile,* of operations. A run (and hence function or operation) is a logical rather than physical concept. It can extend over multiple machines (for example, in distributed systems). It can execute in segments of time rather than continuously.

It may be helpful at this point to view some of these concepts graphically. The set of all possible input states for a system is called the *input space,* as shown in Fig. 5.1. Each input state and hence run type is represented by a point in this space. An operation is represented by a domain in the input space. A run type can belong to only one operation. The domains of different operations do not overlap. A function can't be directly represented in the input space because inputs are not defined until design is complete. However, you could look at a function as implying a domain in the input space. Such a domain would not be identical to the domain of an operation unless the function maps directly one-to-one to the operation.

Execution of runs is represented by selection of points from the input space. If you repeat the same run type, you are selecting the same point again.

5.3 Development Procedure

The operational profile is usually developed by some combination of systems engineers, high-level designers, and test planners, with strong participation from product planning, marketing professionals, and customers.

To determine an operational profile, you look at use from a progressively narrowing perspective—from customer down to operation—and, at each step, you generate an intermediate profile by specifying

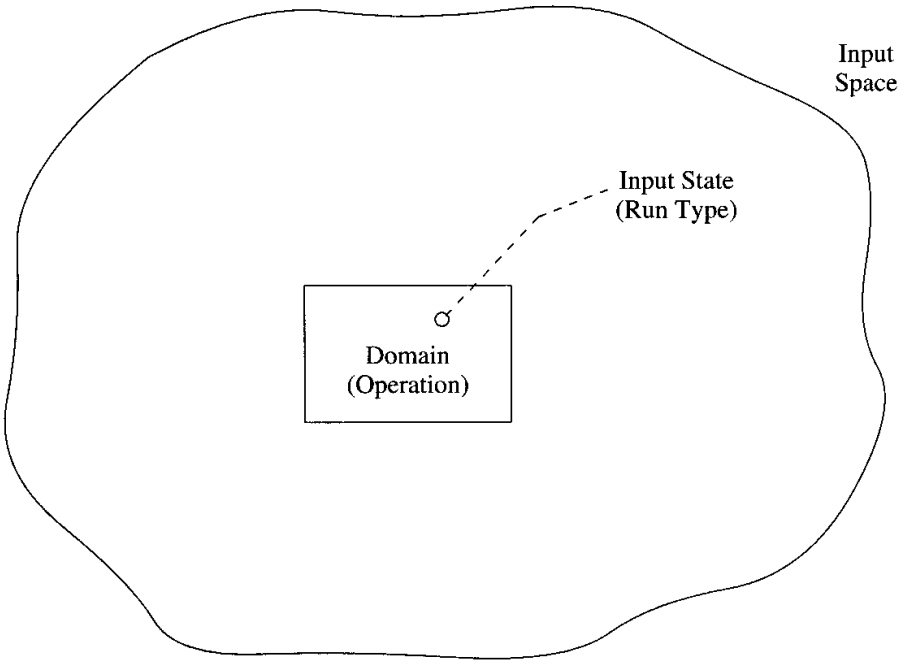

Figure 5.1 Some operational profile concepts.

the probability that each of the elements in that step will be used. This procedure represents the best current approach, based on experience with a variety of projects. We illustrate the procedure by developing, step by step, a simple operational profile for a private branch exchange (PBX). However, the procedure is generally applicable to all kinds of applications, with just minor tuning.

In many cases, usage information is available or can be estimated, most easily in terms of rates like transactions per hour. But these data are not true profiles until you convert them to probabilities by dividing by the total transactions per hour. Converting to probabilities is helpful because you can make a quick completeness check by seeing if the probabilities add to 1. On the other hand, the raw data are useful to recreate actual traffic levels in test.

You will need to establish the granularity (number of elements for which you collect occurrence probabilities) and accuracy (error in estimated occurrence probabilities with respect to those actually experienced in the field) needed for the operational profile. You should base these decisions on the net economic gain resulting from the better decisions that result from more fine-grained and accurate data, minus the extra cost of gathering and analyzing these data being considered. In many cases, however, you will use informed engineering judgment rather than a formal economic analysis. The engineering judgment will be based on such factors as product complexity, product maturity, product cost, and schedules. In practice, you must limit profiles to several hundred (several thousand, at most) elements, because the cost of developing them increases approximately in proportion to the number of elements. Reliability measurements are generally robust with respect to inaccuracies in determining the operational profile [Musa94].

Is use the only factor you should consider in developing operational profiles? What about infrequently executed functions whose failure might lead to disastrous results, such as the function that shuts down an overheating nuclear reactor? This concern can be handled by considering criticality, which we cover in Sec. 5.3.3.

Developing an operational profile to guide testing involves as many as five steps:

- Developing a customer type list
- Developing a user type list
- Listing system modes
- Developing a functional profile
- Converting the functional profile to an operational profile

In the first four steps, you progressively break down system use into more detail. Customer types (large retail stores) break down into user types (sales clerks and information-system specialists). A single user type may invoke several system modes (information-system specialists perform both database cleanup and generate reports). In turn, each system mode has several functions (the generate-reports mode has several types of reports). In the fifth step, functions evolve into operations as the system is implemented.

Some steps may not be necessary in a particular application. A customer list is unnecessary if you have only one customer or if all customers use the system the same way. Sometimes you can skip the functional profile, especially when the requirements are so detailed they specify how users will execute the system to accomplish their tasks. At some other times the information you need to develop the operational profile is not available until the design or even a substantial part of the implementation is complete, so you must develop the functional profile if you want to have guidance in allocating development resources and determining priorities during the design and implementation phases.

Even if you have all the information you need to develop the operational profile before design commences, you may prefer to generate the functional profile first. Because it generally has fewer elements, it is easier to develop and use for guiding pretest development than an operational profile. Should you decide to develop the operational profile directly, you should consider the tasks outlined in the functional-profile step in combination with those in the operational-profile step.

The procedure can conceivably be iterative. If you have an existing system, you may determine the current operational profile, then analyze your current and prospective future customer groups, and then proceed through the steps to obtain a modified operational profile.

At each step, you must determine the level of detail you need. Whether you distinguish and treat an item differently at a given step will depend on the net economic gain for doing so. For example, in developing a customer list broken down by industry, it may be cost-effective to distinguish certain important customers with the intention of performing special testing on the product supplied to them. The degree of detail does not have to be uniform across the system. For example, some important system modes may require greater levels of detail.

When attaining an average reliability over all of a system's applications is acceptable, there may be no need for more than one operational profile. You must still separately identify different customers, users, and system modes so that you can weight the contribution each makes to the operational profile. If it is important to assure a particular reliability for a particular use (even if all reliability objectives are the

same), then you must determine multiple operational profiles. You may also need multiple operational profiles if the system will operate with different hardware configurations. Finally, lab and test resource limits may force you to divide testing, using different operational profiles.

Sometimes a software product is part of a network of systems. In that case, it may be useful to develop an operational profile for the entire network before developing the operational profile for the product. This supersystem profile can be very useful to determine which systems in the network are most important and should receive the most attention. It is also possible to decompose a system into subsystems and to develop an operational profile for each.

All profiles that are developed should be baselined and placed under change control, with appropriate traceability requirements.

As far as we now know, the operational profile is independent of design methodology—its determination will not be affected by an object-oriented approach, for example. The one exception might be a case in which functions designed with one methodology map to a considerably different set of operations when designed with another.

5.3.1 Customer type list

A customer is the person, group, or institution that is acquiring the system. A customer type, the key concept here, is a set of customers who will use the system in the same way. Large pharmacies use a switching system very much like other large retailers and thus could be grouped with them, even if they are not in the same market segment. The customer type list is the complete set of customer types.

You obtain information on potential customers from marketing data for related systems, modified by marketing estimates that take into account the new system's appeal. The business case developed for a proposed product usually includes the expected customer base. It is a valuable source for developing operational profiles, analyzing performance, and reviewing requirements.

Example 5.1 As an example, consider a hypothetical PBX that is sold to institutions for internal use and, of course, external connections. Assume there are two customer types, large retail stores and hospitals.

5.3.2 User type list

A system's users are not necessarily identical to its customers. A user is a person, group, or institution that employs, not acquires, the system. A user type is a set of users who will employ the system in the same way. By identifying different user types, you can divide the task of developing the operational profile among different analysts, each an

expert on their user type. The user type list is the set of user types. Sometimes the users are customers of your customers; sometimes they are internal to your customer's institution. In any case, different users may employ the system differently. The differences may be the result of job roles—an entry clerk will view an insurance company's claim-processing system differently than an actuary.

You derive the user type list from the customer type list by refinement: looking at each customer type and determining which user types exist. If you find similar user types among different customer types, you should combine them.

Example 5.1 (cont.) In our example, user types in each customer type include telecommunications users (people making calls and sending data), attendants (internal operators who answer the main number), a system administrator (who manages the system and adds, deletes, and relocates users), and maintenance personnel (who test the system periodically and diagnose and correct problems).

Each of these user types employs the system differently.

5.3.3 System mode list

A system mode is a set of functions or operations that you group for convenience in analyzing execution behavior. A system can switch among modes so that only one is in effect at a time, or it can allow several modes to exist simultaneously, sharing the same resources.

For each system mode, you must determine an operational (and perhaps functional) profile. Thus multiple system modes means multiple operational and perhaps functional profiles. The same function or operation can occur in different system modes.

There are no technical limits on how many system modes you can establish. You must simply balance the effort and cost to determine and test their associated operational profiles against the value of more specialized information and organizational convenience they provide.

Some bases for characterizing system modes, with examples, are

- *Relatedness of functions / operations to larger task.* System administration, data entry, customer representative queries, transaction processing, report generation.
- *Significant environmental conditions.* Overload versus normal traffic; initialization (start-up or reboot for failure recovery) versus continuous operation (includes warm start after an interruption); system location; time.
- *Operational architectural structure.* Online retail sales mode versus after-hours billing mode.

- *Criticality.* Shutdown mode for nuclear power plant in trouble.
- *Customer or user.* Customer group or user group requiring special functions/operations.
- *User experience.* Novice versus expert mode.

When a system has different priorities for its tasks, it is particularly important that system modes be defined for both normal traffic and heavy traffic conditions, because their operational profiles will differ. For example, when you have a normal load of feature-oriented tasks, more low-priority tasks such as audits and housekeeping operations will execute. Occurrence probabilities of such low-priority operations may be zero under heavy traffic conditions.

Defining different system modes is a convenient way of accommodating changes in the operational profile as users become more experienced. In practice, you can capture the variations in experience with two extremes, novice and expert, mixed in different proportions.

Some systems control the operations they will accept on the basis of environmental variables, such as traffic level and system-capability status, in order to reject noncritical, nonfunctioning operations and dedicate capacity to more critical ones. If a system must function in these conditions, each of these situations should be established as a system mode and tested with the guidance of separate operational profiles.

It is most convenient to group critical operations into one or more system modes, where each system mode incorporates operations of the same criticality. Critical system modes will then receive accelerated or increased testing. The factor of acceleration or increase is usually selected to yield enough execution time for the critical functions to assure achieving a desired level of failure intensity with acceptable confidence. Failure intensities measured in accelerated testing can be transformed to the values they would have had without the acceleration. The case study in Sec. 5.8 deals with critical but infrequent operations.

To measure criticality, consider value added (increased revenue or reduced cost) by an operation or the severity of the effect when it fails. Effects include risk to human life, cost, or reduction in capability. Cost consists of direct and indirect (damage to reputation) revenue loss and the cost of failure workaround, resolution, and recovery. In some cases, an operation can fail in different ways, with different severities. We use the average of the severities, weighted by relative probability, as the operation's criticality.

In considering financial effect, old operations can be more critical than new operations because their failure disrupts existing capabilities that users rely on. At least part of this effect may be captured by

higher occurrence probabilities for these operations. On the other hand, new operations may be critical to the success of a new product, yet may not have a very high estimated occurrence probability. You may have to assign them a high criticality to reflect their importance.

It is common to use four criticality categories, each separated from the next most critical by one order of magnitude of effect. You must define a failure intensity objective for each criticality category and all must be met to ensure satisfactory operation.

The margin for error in reliability estimates is almost always smaller for the most critical category. So the effects of environmental input variables, such as traffic fluctuations and entry errors, will often be material for the critical category; testing for operations in that category must cover them.

Example 5.1 (cont.) The sample PBX has five system modes:

- Telecommunications business use
- Telecommunications personal use
- Attendant use
- Administration
- Maintenance

The last three system modes represent user types. They are disjoint in that they do not share functions or operations. The first two modes share most functions or operations and could be combined. However, both the functional and operational profiles for the two modes are expected to be very different. Hence we will separate these modes so that all modes may be viewed as disjoint. Note that we are separating the modes only to ease the job of analyzing them. It does not imply that different modes can't execute simultaneously.

5.3.4 Functional profile

The next step is to break each system mode down into the functions it needs—creating a function list—and determine each function's occurrence probability.

Functions, as noted, are defined from the user's perspective; they do not necessarily consider architectural or design factors. They are established during the requirements phase and are closely related with requirements. In general, developing a functional profile is considered part of the job of developing the requirements. The functional profile is used in the management of the architecture and design phases and in the design of the architecture itself. The functional profile is baselined and placed under change control, with appropriate traceability requirements, just as the requirements are.

Because you determine the functional profile before design begins, it can help guide the allocation of resources during design, coding, unit test, and possibly subsystem test. Of course, to allocate resources and

set priorities, you must consider other factors as well, such as risk and developer expertise.

5.3.4.1 Number of functions. A functional profile does not have a set number of functions, but it typically involves between 20 and several hundred. The number generally increases with project size, the number of system modes, the number of major environmental conditions, and function breadth—the extent to which a function accommodates task variations.

Functions should be defined such that each represents a substantially different task, in the sense that we are likely to assign a different priority and allocate different resources to the development of, or design a different architecture for, that part of the system that supports that task. The task can be substantially different either as a result of work accomplished (most common) or the environment encountered. Examples of different environments might be different equipment or different traffic levels (normal and overload). You are likely to need to define different functions if tasks differ considerably in criticality; the run types you group in a function should have approximately the same criticality, because they will be treated as if they did.

5.3.4.2 Explicit versus implicit. Functional, operational, functional scenario, and operational scenario profiles can all be expressed in two forms, explicit and implicit, although it is usually easiest to express the latter two profiles in implicit form. *Explicit* and *implicit* refer to different ways of specifying functions and operations and hence selecting them for execution. At this point you must choose between an explicit or implicit operational profile or some combination of the two because that determines if you should develop an explicit or implicit functional profile.

In order to distinguish the two forms, we first need to define what we mean by "key input variable." We will, for brevity, give the definition in terms of operations, but it is equally applicable to functions. A *key input variable* is an input variable that is common to the input states of two or more operations, and whose value is needed to differentiate among some of these operations. In many cases, the values of a key input variable that differentiate operations are actually ranges, which are called *levels.* The name of the operation is a key input variable. A parameter may be a key input variable if two or more operations have the same name and the value of the parameter is the only way of distinguishing between them.

A profile is *explicit* if each element is designated by simultaneously specifying the values of all the key input variables necessary to identify it. A profile is *implicit* if it is expressed in terms of sequences of subpro-

files, each subprofile representing the possible values of one key input variable and their conditional probabilities of occurrence, given the values specified for the previous key input variables in the sequence.

Suppose you have two key input variables, *C* and *D*, each with three values. For simplicity, assume that the variables are independent (if not, the subprofiles become more complex since all the preconditions must be stated). We can define nine operations based on the values of these key input variables. Example implicit and explicit operational profiles for these operations are given in Table 5.1.

An implicit profile is most conveniently expressed as a directed graph or a tree with the nodes representing key input variables and the branches, their values, and the associated conditional probabilities. For example, Fig. 5.2 shows sample "call trees" used by International Definity. It represents an implicit operational profile. Instead of selecting test cases from a complete list of all possible paths with associated probabilities (explicit profile), you select from each set of branch alternates with their associated branch probabilities. An explicit profile can always be determined from an implicit profile by tracing all paths through the directed graph or tree, multiplying the conditional probabilities of the branches together. Note how a test call is generated by pairing the call trees for a calling and a receiving party.

The chief advantage of the implicit profile is that it usually requires you to specify fewer elements—as few as the *sum* of the number of levels of the key input variables, depending on the amount of indepen-

TABLE 5.1 Sample Implicit Operational Profile

Subprofile C		Subprofile D	
Key input variable value	Occurrence probability	Key input variable value	Occurrence probability
C1	0.6	*D1*	0.7
C2	0.3	*D2*	0.2
C3	0.1	*D3*	0.1

Sample Explicit Functional Profile	
Key input variable values	Occurrence probability
C1D1	0.42
C2D1	0.21
C1D2	0.12
C3D1	0.07
C1D3	0.06
C2D2	0.06
C2D3	0.03
C3D2	0.02
C3D3	0.01

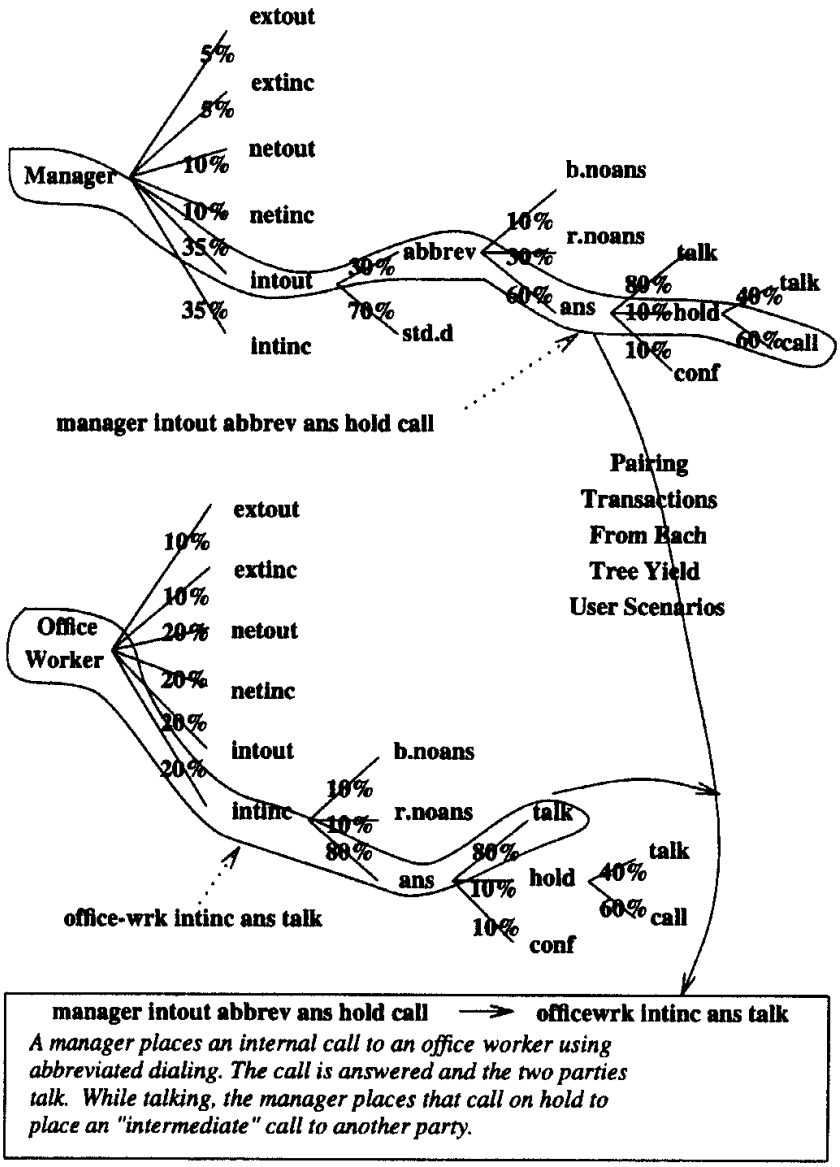

Figure 5.2 Sample call trees.

dence among these variables. Using an explicit profile, the number of elements you must specify can be as high as the *product* of the number of levels of the key input variables. The implicit profile can then be used in cases where the number of elements of an explicit profile would be too large. Alternatively, use of the implicit profile may give you finer granularity in measuring usage for the same effort you would use for an explicit profile.

The implicit approach is suited to transaction-based systems, in which processing depends primarily on transaction attributes that have known occurrence probabilities. A system to generate personalized direct mail, for example, depends on customer attributes such as location, income, and home ownership.

An implicit profile is particularly easy to construct when the key input variables that characterize a function or operation tend to present themselves sequentially in the description of the task that is being accomplished, such as in the call trees of Fig. 5.2.

5.3.4.3 Initial function list. The initial function list focuses on features, which are functional capabilities of interest and value to users. It consists of one list if you are developing an explicit profile, and one list for each key input variable if you are developing an implicit profile. System requirements are usually the best source of information on features. If you have trouble identifying the function, it is often because the requirements are incomplete or fuzzy. User input is vital in creating the initial function list. Only those who have practical experience with the existing work process can uncover problems with a proposed process. For military projects, part of the request for proposal may contain a "Design Reference Mission Profile" which, in turn, may contain descriptions of system modes, functions, and tasks performed and environmental factors. The mission profile serves as a good basis for developing a functional profile.

You can use a prototype as a second verifying source, but you must use it with care, because often a prototype implements only some functions. The most recent version of the product can also serve as a valuable check, but of course it lacks the new functions planned for the next version. Sometimes there is a draft user manual, written to enhance communication with users, that you can check, but it will probably emphasize functions activated by commands, not functions activated by events or data.

Example 5.1 (cont.) In the PBX example, we will generate an explicit profile for the administration system mode functions. The initial function list has four elements because the system-administration mode has four principal functions: adding a new telephone to the exchange, removing a telephone, relocating a telephone or changing the service grade provided, and updating the online directory.

5.3.4.4 Environmental variables. Now you should identify the environmental input variables and their values or value ranges that will require separate development efforts, such as substantial new modules. Environmental variables describe the conditions that affect the way the program runs (the control paths it takes and the data it accesses), but do not relate directly to features. Hardware configuration and traffic load are examples of environmental variables. Probably the best approach is to have several experienced designers brainstorm a list of environmental variables that might cause the program to respond in different ways, and then decide which of these would likely have a major effect on the program.

Example 5.1 (cont.) In the PBX example, telephone type is an environmental variable that has a major effect on processing. Although telephone type can have several values, here only analog (*A*) and digital (*D*) telephones have substantially different effects on processing. So there are two levels for the environmental input variable, *A* and *D*.

When environmental variables and their values are associated with their occurrence probabilities, you have an *environmental profile.* The configuration profile used by DEFINITY (Sec. 5.7.3) is an example of an environmental profile.

5.3.4.5 Final function list. Before you create the final function list, you should examine dependencies among the key environmental and feature variables. If one variable is totally or almost totally dependent on another, you can eliminate it from the final function list. If one variable is partially dependent on another, you must list all the possible combinations of the levels of both variables, along with all the independent variables. You will have one final function list for an explicit profile. For an implicit profile, the number of function lists will equal the number of feature and environmental key input variables.

The number of functions in the final function list is the product of the number of functions in the initial list and the number of environmental variable values, minus the combinations of initial functions and environmental variable values that do not occur.

Example 5.1 (cont.) The final function list for the PBX, shown in Table 5.2, is developed from the initial function list enumerated above and the telephone type environmental variable. It has seven elements: the three initial functions with two environmental variable values, plus the initial function "online-directory updating," which is not affected by telephone type.

5.3.4.6 Occurrence probabilities. The best source of data to determine occurrence probabilities is usage measurements taken on the latest release, a similar system, or the manual function that is being automated. Usage measurements are often available in system logs, which

TABLE 5.2 Final Function List

Function	Environmental variable
Relocation/change	*A*
	D
Addition	*A*
	D
Removal	*A*
	D
Online-directory updating	

are usually machine-readable. Note that these measurements are of operations, not functions, so they must be combined (usually by simple addition) when a function maps to more than one operation.

Occurrence probabilities computed with these data must be adjusted to account for new functions and environments and expected changes due to other factors. Most systems are a mixture of previously released functions, for which you may have measurements, plus new functions, for which you must estimate use. Although estimates are less accurate than measures, the total proportion of new functions is usually small, perhaps 5 to 20 percent, so the functional profile's overall accuracy should be good.

In the rare event that a system is completely new and the functions have never been executed before, even by a similar system or manually, the functional profile could be very inaccurate. However, it is still the best picture of customer use you have and so is valuable.

The process of predicting use alone, perhaps as part of a market study, is extremely important because the interaction with the customer that it requires highlights the functions' relative value. It may be that some functions should be dropped and others emphasized, resulting in a more competitive product. Reducing the number of little-used functions increases reliability, speeds delivery, and lowers cost.

Example 5.1 (cont.) In the sample PBX, there are 80 telephone additions, 70 removals, and 800 relocations or changes per month. Online-directory updating represents 5 percent of the total use in the system-administration mode.

We will assume that the occurrence probability for the system-administration mode is 0.02. Thus the overall occurrence probability for each of these functions, without consideration of environmental factors, is the product of their occurrence probability and the system-administration mode's occurrence probability in the overall system. Table 5.3 shows the resulting segment or part of the initial functional profile.

To take into account environmental factors, assume that 80 percent of the telephones are analog and 20 percent are digital. The environmental profile is shown in Table 5.4.

Also assume that the occurrence probabilities of the first three functions and telephone type are independent. To determine the final functional profile, you

TABLE 5.3 Sample Initial Functional Profile Segment

Function	System-administration-mode occurrence probability	Overall occurrence probability
Relocation/change	0.80	0.0160
Addition	0.08	0.0016
Removal	0.07	0.0014
Online-directory updating	0.05	0.0010

TABLE 5.4 Sample Environmental Profile

Telephone type	Occurrence probability
Analog (*A*)	0.8
Digital (*D*)	0.2

multiply the values of the environmental profile by the values of the initial functional profile to obtain the final functional profile segment for the system-administration mode in Table 5.5.

5.3.4.7 Functional and operational profiles with correlated elements. It is desirable to define functions and operations such that their probabilities of selection are independent of previous functions/operations selected. Then you only need to determine occurrence probabilities for the functions/operations in your list. The foregoing implies that if you have a sequence of functions/operations that are highly correlated, you should try to bunch them into one function/operation.

You can assume that functions/operations with little correlation are independent, but you must recognize the fact of your approximation. The risk here is that independent random selection will distort the true occurrence probabilities to some extent. Positively correlated sequences will occur less frequently than they do in reality, increasing the risk that failures occurring in them may be missed. Negatively correlated sequences will occur more frequently than they do in reality, wasting test and debugging resources by overemphasizing failures associated with them.

5.3.5 Operational profile

The functional profile is a user-oriented profile of functions, not the operations that actually implement them. But it is operations, not functions, that you test. An operation represents a task being accomplished

TABLE 5.5 Sample Final Functional Profile Segment

Function	Environment (telephone type)	Overall occurrence probability
Relocation/change	*A*	0.0128
	D	0.0032
Addition	*A*	0.00128
	D	0.00032
Removal	*A*	0.00112
	D	0.00028
Online-directory updating		0.00100

by the system, sometimes in a particular environment, as viewed by the people who will run the system (also as viewed by testers, who try to put themselves in this position). To allocate testing effort, select tests, and determine the order in which tests should be run, the operational profile must be available when you start test planning.

Functions evolve into operations as the operational architecture of the system is developed. The *operational architecture* is the way the user will employ operations to accomplish functions. There is often some, but rarely complete, correlation between the operational architecture and the system architecture, which determines how modules and subsystems combine into the system.

A function may evolve into one or more operations, or a set of functions may be restructured into a different set (and different number) of operations. Thus the mapping from functions to operations is not necessarily straightforward. For example, an administrative function in a switching system might be to relocate a telephone. This single function may be implemented by two operations, removal and installation, because these tasks may be assigned to different work groups. Generally, there are more operations than functions, and operations tend to be more refined. An operation is usually more differentiated than a function.

The principal steps in determining the operational profile are to list the operations and determine the occurrence probabilities.

5.3.5.1 Listing operations. You use the functional profile to develop the list of operations, mapping functions to operations by following the operational architecture of the system (the way the operations combine to accomplish the functions). Since functions differ from one another by virtue of implementing substantially different tasks, operations will generally be distinguished from one another by substantially different processing. Different operations will usually execute substantially different code paths. Thus the definition of *operation* can be affected by program structure. You will usually want to select at least one test case for each operation, as each has a substantial risk of experiencing a failure that is not experienced by other operations. As with functions, operations may differ as the result of work accomplished or environment encountered.

If possible, it is desirable to define functions such that functions and operations will map one-to-one. This will greatly simplify deriving the operational profile from the functional profile. It requires, concurrent with requirements development, consideration of architectural factors, design factors, and practical constraints on the execution time of operations. If you are developing an operational profile for an existing system, there will be a natural tendency for functions and operations to be similar.

Creating and examining a list of those events that initiate program execution (such as commands or transactions) may help you create the operations list. If the events have parameters, consider the effects of different parameter values. If a value of a parameter causes significantly different processing to occur, you may want to define a new operation. If there are N significantly different kinds of processing associated with an event as a result of different parameter values, then the original function associated with the event should be replaced by N operations. In creating the list, you must also consider environmental variables.

The definition of a run should imply an execution time that is short enough so that sufficient executions exist during:

1. Field operation to permit satisfactory accuracy in characterizing usage
2. Test to permit satisfactory accuracy in reproducing usage

Also, the run should be short enough so that the input state needed to characterize its interaction with the environment is not excessively long. For example, a complete flight of a space vehicle or aircraft might be defined as a run, but the execution time would be too long to meet the constraints just noted. You might then define the runs such that their time durations correspond to the time of a flight maneuver for the vehicle. Examples of grouping of such runs into operations might be "turn," "climb" (including dive), and "steady flight." These operations are probably not correlated with each other. However, the input states and hence runs *are* because the vehicle's position and velocity can change only a limited amount during the time of a maneuver. This must be handled in test by selecting a sequence of runs (input states) within the sequence of operations chosen that recognizes the limitations.

Avoid excessive interaction between operations. When sequences of tasks occur such that substantial amounts of data must be passed between them, consider defining the entire sequence as a run. If the system is reinitialized from time to time, it is a good idea to define runs so that they do not cross reinitialization boundaries.

Now you need to verify that the list of operations is as complete as you can reasonably make it. First, you develop a list of input variables and their ranges of values that is practically complete. A "practically complete" list identifies all input variables except those that take on one value with very high probability. You are ignoring and thus won't be testing the alternate values. This is acceptable because they occur so rarely that they have little effect on reliability even if they fail. The degree to which you can do this decreases for systems requiring higher reliability.

You then create a representation of the input states, also making it practically complete. Because the identification process will never be perfect, you should employ other strategies, such as reducing the number of input states and using indirect input variables, described later, to ensure that you handle hidden input variables properly. The amount of effort you put into this should be based on reliability requirements, the cost of the extra effort, and any information you have on the probability that these interactions will occur.

5.3.5.2 Reducing number of operations. If developing the operational profile is burdensome because the operations list is too long, you essentially have three options in redefining it (we are not discussing test selection here, which will be covered in Sec. 5.4):

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

Reducing the number of run types has the added benefits of reducing the testing effort and perhaps design and implementation costs. Practically speaking, you can reduce the number of run types either by reducing the number of input variables or the number of values for each input variable. In general, the number of input variables is more likely to influence program control flow and failure behavior than the number of values, so you should give this more attention. Some ways to reduce the number of input variables are:

- Reduce functionality.
- Reduce the number of possible hardware configurations.
- Restrict the environment the program must operate in.
- Reduce the number of types of faults (hardware, human, software) the system must tolerate.
- Reduce unnecessary interaction between successive runs.

All these approaches have costs, in addition to the costs of analysis and redesign. The first four, which change the system's features, may involve customer objections, less flexibility, less robustness, or reduced reliability, respectively. The disadvantage associated with reducing operations may be more apparent than real. It may be possible to build systems with the same functionality but fewer operations by applying the *reduced-operation software* concept [Musa91a], analogous to

reduced-instruction-set computing. With this approach, you do not implement operations that occur rarely. Instead, they are accomplished by executing sequences of other operations or combining other operations with manual intervention. To decide which operations should not be implemented, look at the economic trade-off between reduced development cost and potentially higher operating costs.

The fifth option, reducing unnecessary (not required for functional reasons) interaction between successive runs, is highly desirable. It requires only changes in design or operational procedures, not negotiation with customers. Reducing interaction not only simplifies test planning, but substantially reduces the risk of failure from unforeseen causes. You must recognize, however, that the extent to which you can do this may be limited. And there is usually some cost in greater execution time.

Some ways to reduce unnecessary interaction are as follows:

- Design the control program to limit the input variables that application programs can access at any one time (information hiding).
- Reinitialize variables between runs. Because it may be difficult to determine with high confidence which input variables are influencing runs, it may be simplest to reinitialize all of them between runs. If the resulting overhead is excessive, reinitialize periodically, which reduces overhead but allows more interaction.
- Use synchronous (time-triggered) instead of asynchronous (event-triggered) design. Synchronous design lets you better control the input variables that are in play at any time. However, it may add overhead; it requires extra measurement and planning to prevent functions from being aborted when deadlines are missed; and it may be a less natural fit with the problem being solved, resulting in a less compact design.

Reducing interactions has a higher risk than the other approaches to reducing input space. It is more complex and hence more error-prone, so "sneak" interactions may remain. Also, we know less about how to best exploit reduced interactions to reduce testing.

The second option to reducing the number of operations, increasing the number of run types grouped per operation, involves increasing the difference in occurrence probabilities among run types required to establish separate operations. Operations that do not meet the higher differentiation standard are merged, provided they share the same input variables. If the number of randomly selected tests in the merged operation equals the sum of the number of tests for its components, the risk of missing a failure is not substantially changed, providing the failure probability of a run type in each of the components is approximately the

same [Haml90]. Greater grouping does not reduce the amount of testing required; it just decreases the amount of effort required to develop the operational profile. There is an extra cost for analysis, however.

The run types you group should have approximately the same criticality and probability of occurrence, because you will give them the same priority in test and test them to the same degree of intensity. If they also share the same input variables and execute the same code path, the job of test selection (see Sec. 5.4) will be simplified, but this condition is not essential. Consider an airline reservation system. We will group all the single-leg flight reservations, since their run types share the input variables of passenger name, flight number, originating city, terminating city, and so on. A two-leg reservation, on the other hand, is a different operation with a different set of key input variables, which include the second flight number and the connecting city.

If you have different operations with the same input variable sets, you should consider merging them unless they have substantially different criticalities or occurrence probabilities. In the latter case, maintaining them separately provides the basis for nonuniform testing, that is, testing the more frequently occurring operation more intensely.

The reduction in operations may even be on a temporary basis. You may develop an operational profile with a moderate number of elements for the first version of a software product, refining it for later versions only if you discover in the field that the reliability predictions from test are in error.

The third option, excluding the remaining set of run types expected to have total occurrence probability appreciably less than the failure intensity objective of the system, can happen automatically if the number of test cases is limited and they are selected in accordance with the occurrence probability of the operations. Let the sum of the occurrence probabilities of the excluded run types equal p_E. Assume the failure intensity at the start of test is λ_0; at the end, λ_F. Assuming that faults initially are distributed uniformly with respect to operations, then operations contribute to the failure intensity in proportion to their occurrence probability. The excluded operations will contribute $p_E\lambda_0$ to the failure intensity at the start of test. This number will be the same or less at the end, assuming that no faults are spawned that could cause any excluded operation to fail. Because this contribution will not be measured, λ_F will be low by, at most, this amount. Let ε be the maximum acceptable error in measuring failure intensity. Then, setting ε equal to $p_E\lambda_0$, you obtain the allowable value of

$$p_E = \frac{\varepsilon}{\lambda_0}$$

Suppose the failure intensity objective is 10 failures per 1000 CPU hours. It might be reasonable to set ε equal to one failure per 1000 CPU hours. If $\lambda_0 = 10^5$ failures per 1000 CPU hours, then $p_E = 10^{-5}$. This means that once the total occurrence probabilities of the operations you are testing reaches $1 - 10^{-5}$, you can ignore the rest. The higher the failure intensity objective (the lower the reliability), the more operations you can exclude. Obviously, as criticality increases, the degree to which functions can be excluded diminishes.

The set of operations should include all operations of high criticality, even if they have low use. The effect of not including operations of low criticality and low use will be negligible unless reliability requirements are very high. To increase the likelihood that all high-criticality operations are included, you should focus on tasks whose unsatisfactory completion would have a severe effect and carefully consider all the environmental conditions in which they may be executed. Postmortems of serious failures in previous or related systems often suggest some of these situations.

5.3.5.3 Occurrence probabilities. Since different system modes have different operational profiles but may share common operations, you may have to determine multiple occurrence probabilities for the same operation.

In some cases, there may be field data already existing on the frequency of events the system must respond to. You should expend some effort searching for such data, because it may provide the most cost-effective approach to determining occurrence probabilities. This is especially true if the alternative is building special measurement and recording tools.

There are two general ways to determine occurrence probabilities for operations:

- Count the occurrence of operations in the field.
- Rely on estimates derived by refining the functional profile.

The first is more accurate, but obviously can be done only if a previous release exists. When adding new operations to an existing system, you must supplement use records with estimates. You can also view a system modification as adding a new operational profile to an old one. The operational profile for the old system can be measured; for the new operations, estimated. The two parts are joined by weighting each one's occurrence probabilities by the proportion of total system usage that part represents.

It may take some effort to develop recording software, but you may be able to develop a generic recording routine that requires only an interface to each application. The recording software must instrument the system so that it extracts sufficient data about input variables to identify the operations being executed. If operations are independent and do not depend on the history of preceding operations, you need only to count the execution of each operation. If they are not independent, you must record the sequence of operations. You can later process the sequence to determine conditional probabilities. An operational profile can be recorded in either explicit or implicit form.

The recording process usually adds some overhead to the application. As long as this overhead is not excessive, it may be feasible to collect data from the entire user community. However, if the overhead is large, you will probably have to employ a user survey instead of recording. In sampling users, the same guidelines and statistical theory used for polling and market surveys apply; a sample of 30 or even fewer users may suffice to generate an operational profile with acceptable accuracy.

If the costs of obtaining occurrence probabilities are an issue, you may make measurements of moderate granularity and accuracy for the first version of a software product, refining them for later versions only if you discover in the field that the reliability predictions from test are in error.

If you will be using usage data for testing only, it is acceptable to directly drive testing by recording complete input states in the field. You need not determine the operational profile or create test cases. This saves time and effort, but these savings will be reduced by the extent to which you add new operations. For the new operations, you must estimate occurrence probabilities and develop test cases.

In order to estimate by refining the functional profile, determine the function to operation mapping. Then allocate the function occurrence probabilities to operations. In doing this, you may need to obtain occurrence probabilities of environmental variables whose values distinguish different operations. Sometimes the occurrence probabilities of certain key input variables can be found by simulating the operation of associated systems that determine them. For example, the operational profile of a surveillance system will depend on the frequencies of certain conditions arising in the systems being monitored.

The estimation effort is usually best done by an experienced systems engineer who has a thorough understanding of the businesses and the needs of the expected users, and how they will likely take advantage of the new functions. It is vital that experienced users review these estimates. Often, new functions implement procedures that had been performed manually or by other systems, so there may be some data available to improve the accuracy of the estimates.

It often helps to create an interaction matrix of key input variables plotted against other key input variables. This matrix reveals combinations of key input variables that do not occur or that interact. The remaining areas of the matrix represent regions where you can assume key input variables are independent and estimate the occurrence probability as the product of individual key input variable probabilities.

Let's examine two common types of systems, command-driven and data-driven (also called *transaction-based*).

Example 5.2 Command-driven system The PBX is a *command-driven* system. In implementing the features of the system-administration mode, assume that this command set was developed:

```
relocate <old location> <new location>
add      -s <service grade> <location>
remove   <location>
update
```

The designers decided to handle the function "change of service grade" by removing the old service and adding a new one. As you consider the parameters, you note that location does not affect the nature of the processing. However, the service grade does because the features provided are substantially different for staff, secretaries, and managers. So you refine the add command into three operations and obtain the operations list:

```
relocate <old location> <new location>
add      -s staff <location>
add      -s secretary <location>
add      -s manager <location>
remove   <location>
update
```

All these commands, except update, account for 0.019 of the occurrence probability; update accounts for 0.001. Suppose that the expected 80 additions of service per month break down into 70 staff, 5 secretaries, 5 managers. There will be 780 relocations and 20 changes of service grade each month, the latter representing promotions to manager. There will be 70 removals proper and 20 removals created as the result of change of service grade, yielding a total of 90 removals. The part of the operational profile for the system-administration mode is shown in Table 5.6.

TABLE 5.6 Operational-Profile Segment Based on Features

Command	Transactions per month	Occurrence probability
relocate	780	0.0153
remove	90	0.0017
add -s staff	70	0.0014
update		0.0010
add -s manager	25	0.0005
add -s secretary	5	0.0001

You proceed in this fashion until you have accounted for all the ways the system can be employed. Now you must consider the possible expansion of the operation list to account for environmental variables that could change the processing (and thus result in different failure behavior). There will usually be environmental variables that affect the processing sufficiently to require testing based on some of their values that you must now consider, even though they were not sufficiently major to be considered in the development of the functional profile. For simplicity, assume that the environmental variables may interact with feature variables in determining occurrence probabilities. For example, certain features may be executed at constant occurrence *rates* but as traffic increases, their occurrence *probabilities* decrease.

Example 5.2 (cont.) In our sample system, the environmental variable is telephone type: the system must handle both analog and digital telephones. The operational profile in Table 5.6 will thus expand under this environmental variable into 11 operations (online-directory update is not affected by telephone type). Let's exclude directory update and consider the part for analog telephones *A*, which will have occurrence probabilities that are 80 percent of those for all configurations.

Assume that system load is such an environmental variable. If system-administration functions are performed when the system is in an overload condition because of heavy communication traffic, processing may be affected (administrative requests might be queued, for example). Assume that this occurs 0.1 percent of the time.

To generate the segment of the operational profile we are considering, first multiply all values in Table 5.6 by 0.8 to give the occurrence probabilities for analog telephones. Then multiply by 0.999 to obtain the occurrence probabilities for normal load, or by 0.001 to obtain the occurrence probabilities for overload. Table 5.7 is the new operational profile segment.

TABLE 5.7 Operational Profile Segment Based on Features and Environment

Command	Environment	Occurrence probability ($\times 10^{-6}$)
relocate	Normal load	12,228.00
remove	Normal load	1,359.00
add -s staff	Normal load	1,119.00
add -s manager	Normal load	400.00
add -s secretary	Normal load	79.90
relocate	Overload	12.24
remove	Overload	1.36
add -s staff	Overload	1.12
add -s manager	Overload	0.40
add -s secretary	Overload	0.08

Some operations in Table 5.7 occur very infrequently. You should seriously question if it is really necessary to test all of them. Consider eliminating the "add -s secretary under overload conditions."

Example 5.3 Data-driven system Financial and billing systems are commonly *data-driven*. Suppose a telephone billing system was designed as two subsystems. The sort subsystem receives call transactions and sorts them by billing period and account number, grouping all the items for one account for the current billing period. The account-processing subsystem processes the charge entries for each account for the current billing period and generates bills.

The reliability you want to evaluate is the probability of generating a correct bill. This involves determining the reliability of each subsystem over the time required to process the bill or the entries associated with the bill, and then multiplying the reliabilities. You must determine an operational profile for each subsystem.

Because this design was not anticipated when the functional profile was developed, the relationship between the functional profile and the two operational profiles is complex. For example, typical functions may have been bill processing, bill correction, and the identification of delinquent customers. The bill-processing function relates to operations in both subsystems, but the other two functions relate only to the account-processing subsystem.

The first subsystem, the sort subsystem, will have relatively few operations and a simple operational profile. The operation for processing correct charge items has an occurrence probability greater than 0.99; other operations handle missing data, data with recognizable errors, and so on. You should be able to estimate occurrence probabilities from past data on the frequency and type of errors.

The second subsystem, the account-processing subsystem, has an operational profile that relates to account attributes. Its operations are classified by service (residential or business), use of a discount calling plan (non, national, or international), and payment status (paid or delinquent), resulting in 12 operations.

Assume that the service classification is 80 percent residential and 20 percent business. A national discount calling plan is used by 20 percent of subscribers; international, 5 percent. Only 1 percent of accounts are delinquent. Table 5.8 shows the set of operations and their associated probabilities.

TABLE 5.8 Operational Profile Account-Processing Subsystem of Billing System

Operation	Occurrence probability
Residential, no calling plan, paid	0.5940
Residential, national calling plan, paid	0.1584
Business, no calling plan, paid	0.1485
Business, national calling plan, paid	0.0396
Residential, international calling plan, paid	0.0396
Business, international calling plan, paid	0.0099
Residential, no calling plan, delinquent	0.0060
Residential, national calling plan, delinquent	0.0016
Business, no calling plan, delinquent	0.0015
Business, national calling plan, delinquent	0.0004
Residential, international calling plan, delinquent	0.0004
Business, international calling plan, delinquent	0.0001

If transaction use is described in terms of transaction rates, you obtain the occurrence probabilities by dividing the individual transaction rates by the total transaction rate and multiplying this by the probability of transaction occurrence (with respect to other operations). If all the operations are transactions, the last step is not necessary.

5.4 Test Selection

The operational profile is used to select operations to execute in test in accordance with their occurrence probabilities. Testing driven by an operational profile is very efficient because it identifies failures (and hence the faults causing them), on average, in the order of how often they occur. This approach rapidly increases reliability—reduces failure intensity—per unit of execution time because the failures that occur most frequently are caused by the faulty operations used most frequently. Users will also detect failures in the order of their frequency if they have not already been found in test.

Since an operation represents a group of run types, the *coarse grain* selection of the operation must be followed by the *fine grain* selection of a run type. Although selection of the operation can be based on usage (and criticality), the fine grain selection must be very simple and easy to implement because of the large number of elements in the population from which you are selecting. Of course, you also want it to be efficient in the sense of requiring the smallest possible number of tests to assure a specified reliability at the required confidence level.

Random selection of run types within an operation is a common strategy; it is probably best when you have little information that might affect the distribution of failing run types within an operation. An example of pertinent information would be a processing difference between groups of run types, where it is known that the code associated with one processing alternative is considerably more complex.

However, when some information is available, it is useful to profit from it by dividing the operation into run categories. A *run category* is a group of run types that represent part of an operation. The information that might affect the distribution of failing run types is used to attempt to select run categories or sets of run types that are *homogeneous.* Run categories are homogeneous if any one run type represents the entire set in the sense that all have the same failure behavior. Thus if a test of one of them fails, all will fail. Similarly, a successful execution of one run type means the entire homogeneous set will execute successfully. Clearly, identification of homogeneous sets of run types will reduce the amount of testing required for a specified level of reliability. In practice, it is very difficult to identify homogeneous sets with certainty; hence we define a run category as being *near homogeneous.*

You should establish run categories such that they have approximately equal occurrence probabilities. This is because selection of run categories from operations is done on a uniform basis, usage information not being available at this level.

To define run categories that approach homogeneity, look for run types that at least share the same input variables and execute the same code path. Try to find ranges of values for each input variable over which essentially the same processing occurs.

Although you could in theory select only one run type from a run category, it is probably better to pick two or more and do it randomly to counter the risk that homogeneity is often not achieved. Thus, the sequence of selections that occurs when using run categories is operation, run category, run type.

Selection should be with replacement for operations, run categories and run types. *Replacement* means that, after selection, an item (for example, operation or run type) is returned to the list from which it was chosen so that it is not excluded from reselection. One might argue that replacement for run types wastes test resources because of the possibility of duplication. However, the number of run types is so large that the probability of this happening is infinitesimal.

Selection could be performed without replacement, in which an element can be chosen only once. This is unwise for operations because they can be associated with multiple faults. There is a high risk that different run types within an operation may show different behavior.

5.4.1 Selecting operations

With an explicit operational profile, you select operations directly in accordance with their occurrence probabilities. With an implicit operational profile, you select operations by choosing the level of each key input variable in accordance with its occurrence probability, which implicitly selects the operation at the conjunction of these values.

If different profiles (system modes) occur at different times in the field, you should conduct separate tests. However, if they occur simultaneously, testing should be concurrent, because system modes running simultaneously can interact. The execution time allocated to each system mode should be proportional to its occurrence probability. Concurrent testing in effect combines multiple operational profiles into a single one.

If different versions of the software product are supplied to different customers, they may differ primarily in system-mode profiles. If interaction among system modes is nonexistent or small, you can test each system mode independently. Failure intensities for the different customers can be obtained by weighting the system-mode failure intensi-

ties by the occurrence probabilities. The result is substantial savings in test time. You may also want to test several operational profiles that represent the variation in use that can occur among different system installations to determine the resulting variation in reliability.

If possible, you should select operations randomly to prevent some unrealized bias from entering into the testing process. Data corruption often causes such a bias. Data corruption increases with execution time since the last reinitialization, so if one operation is always executed early and another always late, your tests may miss significant failure behavior.

It is wise to randomly select as many key input variables as possible. Random selection is feasible for operations with key input variables that are not difficult to change. However, some key input variables can be very difficult and expensive to change, such as one that represents a hardware configuration. In this case, you must select some key input variables deterministically, because changing these variables during system test must be scheduled. Carefully consider the bias that might result from those you select deterministically and try to counter it. For example, reinitialize the system at random times to avoid data-corruption bias.

5.4.2 Regression test

Regression testing can be a substantial portion of the overall test effort. Regression tests are run after changes have been made to uncover spawned faults. Spawned faults are faults introduced while removing other faults. Because changes are frequently grouped and introduced periodically, regression testing is also usually periodic. A week is a common interval, although intervals can be as short as a day or as long as a month.

Some testers say regression testing should focus on operations that contain the changed code. This view makes sense only if you are sure the possible effects of the changes are isolated to those operations or if system reliability requirements are low so that cross-effects to other operations do not matter. However, in most cases you cannot rely on isolation, and potential cross-effects can cause unacceptable deterioration in system reliability. So all operations should be considered when planning a regression test. However, a change generally results in a smaller probability of failure than a new program, so it isn't really necessary to retest every operation after every change.

It is inefficient to cover operations of unequal occurrence frequency with equal regression testing; hence, operations should be selected in accordance with the operational profile. Now the possibility exists of

integrating regression testing with regular system testing. You can achieve substantial savings in test resources and time by making your tests do double duty.

5.5 Special Issues

As we used the operational profile on several projects, we encountered some special situations for which solutions had to be researched. We address the most important of these cases here.

5.5.1 Indirect input variables

Sometimes the relationship among observable task and environmental variables and input states is not clearly discernible, at least not without an expensive effort. In this case, you can establish and employ indirect input variables to control test selection. An indirect input variable is believed to affect processing, but is not used by the program directly. In many cases, you will be interested in establishing indirect *key* input variables.

Consider traffic load. It is neither practical nor enlightening to determine which input-variable values are caused by heavy traffic and directly affect processing. It is better to actually generate a heavy traffic load and observe the results. The program accesses no traffic-level variable, but you can consider the traffic level generated as an indirect input variable. You can select levels of these indirect input variables randomly, in accordance with estimated occurrence probabilities.

Indirect input variables are particularly useful for handling the effects of data corruption. Data corruption is the accumulated degradation in data with execution time that results from anomalies in intermediate variables that do not represent failures. Some of these data are in reality input variables for other operations, but the interaction is often not known. In this case, you can define an indirect input variable called *soak time* in terms of hours of execution and plan to test several different values of this variable. You may implicitly select the values by performing a *soak test,* in which you continuously increase soak time up to a limit, with operations randomly chosen in accordance with the operational profile in this interval. In either case, you should include a soak time slightly less than the reinitialization interval you expect to use in the field.

5.5.2 Updating the operational profile

An operational profile can change during the life of a product, especially when new features are regularly made available through new

releases. Each new release will necessitate modifying the operational profile. Because it is best to base the modified operational profile on measured data, a regular operational-profile measurement program is recommended.

In the long run, the simplest way to do this is to build the measurement capability into the system. This involves counting and recording the number of runs of each run type. You can combine this measurement system with a failure-detection and -recording function or other performance-measurement system. The most economic and reliable way to collect data is through periodic reporting over telecommunication channels to a central location. If this is not feasible, you can collect data on a removable medium that is mailed periodically to a processing center.

Some designers may be concerned about performance degradation caused by the recording function. For the amount of data needed and the length of the runs involved, performance degradation is unlikely, but recording excessive amounts of data could cause a problem. If performance does become a problem, you can sample a set of sites or a set of time periods, as long as you are careful to sample randomly. Built-in recording may not happen until suppliers and customers learn through experience to appreciate its value. In this case, the best approach is to take measurements at a randomly selected sample of sites. Root-cause failure analysis may also provide data, because it sometimes leads to uncovering system uses that were not known and hence not tested.

Generally, operational profiles should be updated when there are major releases that represent substantial changes in capabilities and expected use. As a system passes through different versions or releases, the functional profile usually needs updating less frequently than the operational profile, because it is used mainly to prioritize tasks and allocate resources and thus can be less refined.

If new functions are added, their occurrence probabilities must be estimated. Suppose that the new functions' usage totals p_N. If the new functions do not affect the occurrence probabilities of the old functions except by adding to the overall set of functions, the old functions' probabilities are adjusted by multiplying by $(1 - p_N)$. However, if some old functions are replaced or otherwise affected, the probabilities are adjusted individually.

5.5.3 Distributed systems

You can apply operational-profile techniques to distributed or networked systems if they are engineered, tested, and managed as a whole. This implies that "operation" refers to a task that involves part or all of the

total system, not just one component. There is nothing that restricts the concept of an operation to a program that executes on a single machine.

All the concepts relating to input space—run, run type, run category, operation, and function—and failures are logical, in the sense that they can span a set of software, hardware, and human components. For example, a run can consist of a series of segments, each executed as a process by a server, with the servers being implemented on the same or different machines.

The functional profile can be used to guide resource allocation and set priorities in development with respect to the entire system. The operational profile can guide testing of the entire system as a unit.

Delineating run types, run categories, and operations for distributed and networked systems can be more complex because the set of environmental variables can be appreciably larger. Although going from a centralized to a distributed system does not increase the number of task variables, it often increases the effective number of environmental variables, such as traffic load and soak time, because you may have to specify and measure them with respect to individual machines.

You can counter the foregoing proliferation of environmental variables by carefully designing both the system and its operating procedures. For example, you can design a system so that all machines have similar traffic loads (as a percentage of capacity). And you can equalize soak time by synchronizing reinitializations.

You can, of course, also apply the operational profile to any subsystem as long as that subsystem has operations that relate directly to users.

5.6 Other Uses

Although it was developed to guide testing, the operational profile can also guide managerial and engineering decisions throughout the life cycle, including requirements specification, design, implementation, and testing. Because it ranks features by how often they will be used, it suggests development priorities.

A prioritized *operational development* approach is potentially a very competitive, customer-oriented way to sequence new-product introduction: make the most-used features (operations) available very quickly and provide less-used features in subsequent releases. Development of a specified release proceeds incrementally by operations. This approach is different from traditional incremental development approaches, which proceed incrementally by modules. Since operational development is relatively new, experience will be required to identify and resolve development issues that may arise.

The functional profile improves communication between developer and customer and within the customer organization by making expres-

sion of needs more precise. It may highlight types of use not anticipated by the developers. It may cause users to think about their needs in greater detail. For example, when a developer asks which functions are needed to support maintenance and how often they will be used, it stimulates users to think about, discuss, and study what the maintenance procedures should be.

The operational profile can also be used in performance analysis. If you multiply each operation's occurrence probability by the system's overall run or transaction-execution rate, you obtain the run or transaction rates for each operation. This information is used for performance analysis and performance testing. Among other uses, it can help determine the number of servers a client-server system requires.

Finally, the operational profile is an educational aid. It organizes work in a manner that is closely related to user work processes. It can direct the customer's training efforts toward the most-used operations. For user manuals, the operational profile suggests the order in which material should be presented (most-used first) and the space, time, and care that should be devoted to preparing and presenting it.

By employing an operational profile for multiple purposes, we lower its cost per use.

5.7 Application to DEFINITY®

5.7.1 Project description

The Global Business Communications System (GBCS) division of AT&T provides private branch exchanges (PBXs) to businesses. A PBX is a telecommunication system for businesses that provides phone services within buildings and out into the public telephone network. The DEFINITY G3 line of PBXs comes in a variety of models that serve the needs of customers from all areas of industry and in sizes from around 80 lines to tens of thousands.

DEFINITY has a central processor running a program of about one and a half million lines of source code. In addition, there are as many as several hundred distributed processors (line, trunk, and other interfaces) each running programs ranging from 2000 to 250,000 lines of source code.

5.7.2 Development process description

In 1989, DEFINITY development processes were reengineered to focus more on customer satisfaction. During this reengineering process, research indicated that SRE principles could form the base of a complete product development process that was oriented toward satisfying customers. The resulting development process is called Customer Satisfaction Based Product Development. Its components include rigorous

software development techniques (including design teams, code inspections, and developer testing), an incremental development life-cycle model, and quality factor assessment (the assessment of customer-oriented product quality metrics during the development cycle). These processes are now applied to all DEFINITY releases and other products besides PBXs.

SRE is woven into this development process in the following way:

- Customer satisfiers are determined (reliability is a key quality requirement).
- Metrics to measure those quality requirements during product development are defined (various failure-rate metrics are used for reliability measures).
- Customer usage of the DEFINITY product is documented in detail (operational profiles are defined).
- Methods for assessing quality (reliability) during development are devised.

5.7.3 Describing operational profiles

Multiple separate operational profiles are created for the DEFINITY product, rather than one all encompassing profile. This is because a single test environment that represents all DEFINITY customers cannot be effectively, efficiently, or realistically created. Instead, an operational profile is created for each customer type in the customer profile.

Operational profile definitions start with customer models derived from marketing, sales, manufacturing, user groups, and other data sources. These models define typical or generic customers from each of about 12 key business/industry areas (e.g., banking, universities, factories). This forms a customer profile. For each customer type, the set of users in that profile is described. Table 5.9 is part of the user profile for a bank.

The user percentage and call rate are determined for each user in the profile. Together, these allow a usage distribution for those users to be computed. As work was done to define the users for each customer in the profile, the DEFINITY project recognized that users with similar usage behaviors could be grouped into a common usage description. These user descriptions are called *generic users.*

Each operational profile created has essentially one system mode called a *busy hour.* This represents a customer's busiest hour of usage across a typical week. More stressful usage modes are studied independently via stress tests designed by system test engineers.

Initially, the functional profile was defined to be simple and focus on only a single basic PBX function—making and/or receiving a phone call

(though with more than 150 features on a DEFINITY PBX, there's no such thing as a simple phone call). This decision was based on the fact that the vast majority of a PBX's users were telephone users and that the main PBX function they employed was making/placing calls. Thus, the major component of system usage (as measured by transactions, system hours, or CPU hours), and therefore software usage, was accounted for by modeling phone calls. Since that time, the project has subsequently defined other users (e.g., system administrators and maintenance technicians) and their corresponding functional profiles.

An environmental profile, called a *configuration profile,* is also defined. It is a set of configuration descriptions representing those used by the customers in the customer profile. The descriptions specify the parameters needed to describe the hardware and software environment in which the functional profile takes place. The parameters include such things as number of PBXs, types of processors used, types of trunks between PBXs, and number of analog stations.

To describe the operational profiles, a tree-based approach is used. For each user, a *call tree* is defined that describes their telephone usage. The tree-based approach allows for expression of context-dependent occurrence probabilities (i.e., conditional probabilities). This means that the probability of a feature's usage can change depending on when in a call the feature is used. A set of tools is used to automatically create complete telephone calls (run types) from the call trees. Figure 5.2 shows an example of two simplified call trees yielding a simple run type. The tools, using the frequencies defined in the call trees, generate samples that are proportioned according to customer usage.

This method substantially automates the generation of customer-oriented test cases and, of course, provides the customer-like testing (operational-profile-based testing) needed to estimate a system's relia-

TABLE 5.9 A User Profile

Customer Type = Bank				
User name	Generic user type	User percentage	Call rate (c/hr)	User prob.
Secretary	Secretary	20	10	0.379
Attendant	Attendant	1	24	0.046
Night-service	Attendant	1	8	0.015
Agent	Call-center agent	1	15	0.028
Supervisor	Call-center supervisor	1	3	0.006
Executive	Manager	3	3	0.017
Tellers	Worker	15	2	0.057
Office workers	Worker	52	4	0.395
Investment counselors	Administrators	6	5	0.057

bility. An added benefit is greatly enhanced tester productivity since the vast majority of run types and test cases are automatically produced by this method.

Call trees are implicit operational profiles—selections are made directly from the frequency/probability distributions of customers, users, functions, and operations rather than from a listing of all possible operations (an explicit approach). Since each customer may have as many as 20 different users, and each user tree may have hundreds of branches (complete paths through the trees), the possible number of run types is extremely large (estimated to be in the 10^8 range). Consequently, enumerating all run types to create an explicit profile is not feasible.

5.7.4 Implementing operational profiles

A key to this operational profile implementation is the selection methods used across the various profiles. To select customers and configurations, this technique selects ones that are both of high probability and (more) likely to fail. These configuration and customer choices are made independently since experience has indicated that (nearly) all combinations exist in the DEFINITY customer base (or are at least possible).

The customers/configurations selection strategy begins by deterministically (not randomly) selecting those with the highest usage probabilities. The selections are then modified by adding or removing configuration components based on knowledge of the likelihood of failure of the components. (Though perhaps difficult to quantify, knowledge of product failure behavior is available during the development process. It includes such things as which code has changed, which circuit packs are new, which functional interactions exist, what the "traditional" trouble areas are, and so on.)

This is a good strategy from the test resource perspective: by following this strategy, the largest proportion of customers are covered with the fewest selections. It is also good from the quality-improvement perspective since it uses failure- and usage-likelihood information in combination. This will most rapidly improve customer-perceived quality.

Once the customer and configuration are selected (deterministically), the run types to execute are selected randomly (as explained earlier) from that customer's user call trees according to usage probabilities.

The system test organization implements this operational profile approach by first documenting customer, user, and configuration selections in the system test environment and strategy planning documents. These documents guide the creation of individual test plans, one for

each quality (reliability) assessment done during the development interval. The test plans consist of configuration information (environment), usage information (traffic loads and user run types), and other system-level test cases. Using the information in the test plan, the basic execution process is to set up the test lab into a customer-like configuration and operate it as a customer would by running traffic and executing user run types. Failure data observed from this process are fed into reliability analysis tools that plot reliability and other quality metric graphs.

5.7.5 Conclusion

As described earlier in this chapter (see Sec. 5.1), initial experience with SRE-based product development for DEFINITY was quite successful (other factors also influenced these results—see [Abra92]). Subsequent applications of these techniques continue to yield extremely beneficial results both in terms of product quality as well as development time and cost. You can find out more about these operational profile techniques in [Juhl92a] and the overall SRE approach in [Juhl92b] and [Juhl93].

5.8 Application to FASTAR^{SM} (FAST Automated Restoration)

5.8.1 System description

AT&T's long distance network provides ultrareliable telecommunications services. The network is composed of fiber light-guide systems connected by Digital Cross Connect Systems. With the advent of fiber-optic technology, the capacity of the network has increased along with the consequences of events such as cable cuts caused by backhoes, train derailments, and ice storms. Fiber systems are vulnerable to damage due to cable cuts, and a large cut could easily affect over 100,000 telephone circuits. FASTAR, AT&T's Fast Automated Restoration Platform, alleviates the impact of fiber cable cuts by reducing the time it takes to restore service from hours to minutes.

In November 1988, prior to FASTAR, a fiber cable cut in the Newark, New Jersey, area caused the loss of over 270 DS3 paths. A DS3 path consists of 672 telephone circuits and is the unit of transmission and restorability in the AT&T long-distance network. Thus more than 180,000 telephone circuits we affected. It took more than 15 hours to repair the cable and reestablish service. After FASTAR was deployed, a cable cut between Kansas City and St. Louis severed over 250 DS3 paths. It took considerably less than the 5-minute restoration objective to restore service using FASTAR technology.

Two key systems manage FASTAR restoration capabilities. These are the Restoration Node Controller and the Central Restoration System. The Restoration Node Controller is a UNIX®-based system with instances in more than 200 AT&T digital central offices. It monitors the Lightguide Terminating Equipment and Digital Cross Connect Systems for alarm conditions indicating a fiber cut or component equipment failure. The Central Restoration System, also a UNIX®-based system, controls the overall restoration process. This system utilizes unused capacity in the AT&T network to dynamically reroute traffic around failures. Users interact with the Central Restoration System to monitor restoration progress and handle exception conditions.

Figure 5.3 provides a diagram of the FASTAR architecture. If a cable between office A and office C is cut, the Restoration Node Controller in each of these offices detects and reports the failure to the Central Restoration System. Next, the Central Restoration System computes optimal DS3 restoration paths between offices A, B, and C and implements these paths by sending appropriate commands to the Digital Cross Connect System in each office.

Software validation of the UNIX®-based systems that control the restoration process was critical to the success of FASTAR. Since the Restoration Node Controller and Central Restoration System have the power to rearrange the AT&T network, software defects in these

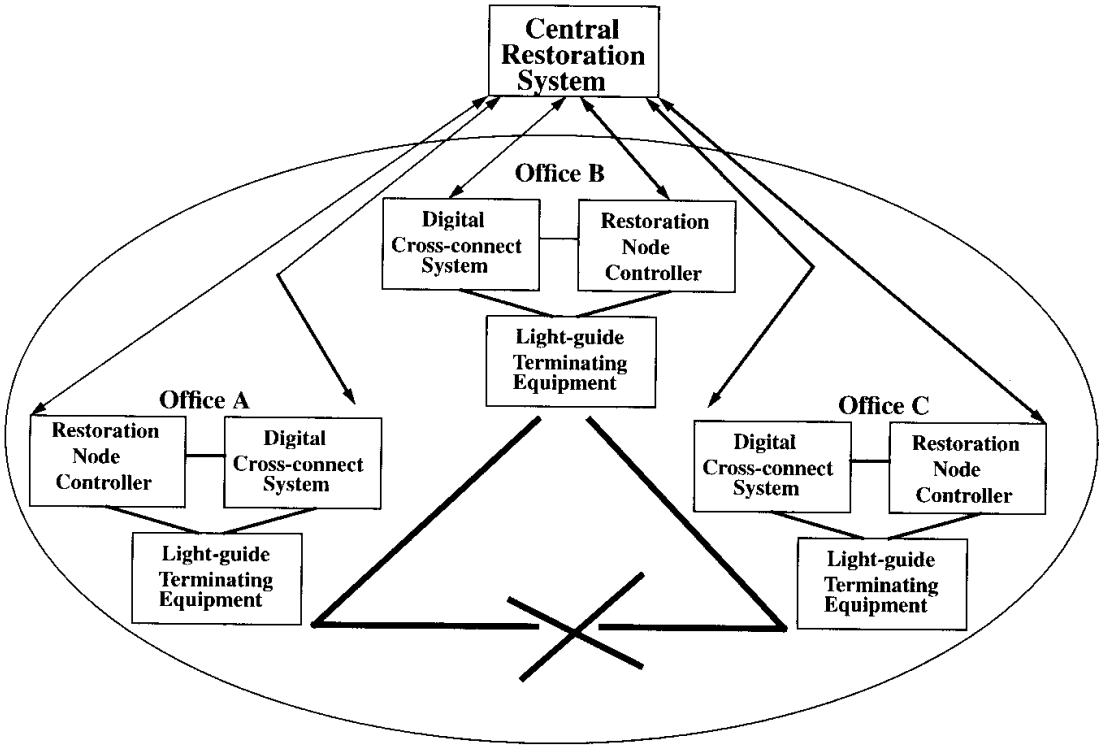

Figure 5.3 FASTAR[SM] architecture.

systems could cause severe network service interruptions. As the development team for FASTAR, we selected software reliability engineering (SRE) to verify that stringent reliability objectives had been met. (Throughout the remainder of this section, *we* refers to the FASTAR development team.)

The success of the FASTAR SRE program was based on our focus in four key areas: understanding customer expectations, deriving the operational profile, developing the test environment, and executing the load/stress/stability test program.

5.8.2 FASTAR: SRE implementation

5.8.2.1 Understand customer expectations. The customers for FASTAR are the AT&T business units that provide business and consumer long Distance and 800 services and the Network Operations group that maintains the AT&T network. The job of these AT&T organizations is to provide a reliable network for the millions of business and residential consumers who use AT&T telecommunications services every day. FASTAR helps them meet the demands of these consumers.

These customers demand high levels of quality, reliability, availability, and performance. They expect the systems that compose FASTAR to not only meet performance objectives but also operate continuously and flawlessly. They want to be absolutely certain that the system has no defects that could negatively impact long-distance service.

Our goal as the development managers for the Central Restoration System and the Restoration Node Controller is to ensure that our organization delivers on our customers' high demands. Our SRE program allows us to do just that. With SRE, we are able to verify FASTAR operational reliability and assess performance under projected field operation. To do this, we developed an operational profile.

5.8.2.2 Derive operational profile. As previously noted, an operational profile characterizes the operating conditions for the system—that is, how the software will be used in the field. We derived the initial version of the FASTAR operational profile using a computer simulation model of FASTAR. This first profile focused solely on cable-cut scenarios. We defined the typical and worst-case cable-cut incidents from simulation results, and derived the frequency of various cable-cut scenarios from historical data. Figure 5.4 displays the FASTAR operational profile for cable-cut scenarios. It describes the probability of small, medium, and large cable cuts occurring. We estimate that 10 large cable cuts occur in the AT&T fiber network each year.

As development proceeded, we recognized the need to define a richer operational profile that described all the system inputs and outputs in

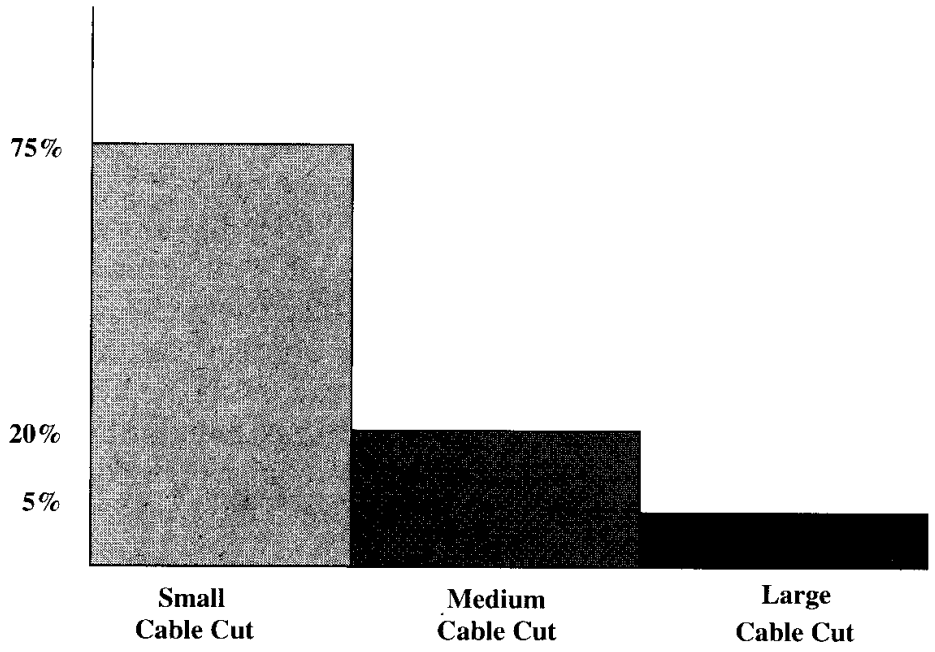

Figure 5.4 FASTAR[SM] operational profile. Probability of small, medium, and large cable cuts.

order to adequately test the system. We added background events to the profile. We defined *background events* as all non-cable-cut activities that could occur on the system. These included activities such as database provisioning, maintenance operations, network management operations, and single DS3 path failures caused by component equipment failures. While major cable cuts occur approximately 10 times during a year, background events occur continuously.

After close examination of the FASTAR operational profile, we determined that the operating conditions for the Central Restoration System were quite different than that of the Restoration Node Controller. We viewed the operating scope of the Central Restoration System as the entire AT&T network. For the Restoration Node Controller, we viewed the operating scope as a single digital central office. In order to take these important differences into account in our test program, we decided to define distinct operational profiles for each of the two systems.

In the operational profile for the Central Restoration System, we were concerned about the overall FASTAR profile and the impact of receiving data from over 200 Restoration Node Controllers, maintaining communications to over 200 Digital Cross Connect Systems and processing commands from 50 simultaneous users. Special attention was paid to collecting data that described how users actually interacted with the system. For the Restoration Node Controller, we developed one operational profile that represented all the offices and focused on the worst-case load in the largest AT&T digital central offices.

We derived the operational profile for each of the systems from discussions with customers and from log data collected from Beta Test versions of the system. The frequency of event occurrences in the Central Restoration System dwarfed that of the Restoration Node Controller. For the Central Restoration System, 10 major cable-cut events are expected in a year. For the Restoration Node Controller, any single system can expect to participate in less than one cable-cut event each year. The difference in background events is even more dramatic. Forty thousand background events are executed on the Central Restoration System every day due to the many users who interact with the system. For the Restoration Node Controller, 500 background events are performed each day. After fully understanding the operating conditions for the software, our next challenge was to build a test environment to take advantage of the operational profile.

5.8.2.3 Develop test environment. We began FASTAR testing by using real network facilities and central office equipment in a laboratory. However, this did not allow us to test the field operational profile. It was not possible or feasible to test the system's maximum configuration. We augmented the test lab with a simulated test environment that could handle all the inputs and outputs identified in the operational profile.

For the Central Restoration System, we developed a Network and User Simulator (see Fig. 5.5). It can simulate the actions of more than 200 Restoration Node Controllers and Digital Cross Connect Systems and over 50 simultaneous users. It provides an environment to test the Central Restoration System with the worst-case cable-cut scenarios that can occur in the AT&T network in the safety of our lab environment.

For the Restoration Node Controller, we developed an Office Simulator (see Fig. 5.3). It simulates the maximum office configuration. The Office Simulator supports the simulation of direct connections to Lightguide Terminating Equipment and Digital Cross Connect Systems. It provides an environment to test the Restoration Node Controller with worst-case cable-cut events in AT&T's largest offices in the safety of our lab.

Both simulation environments were designed with a programmable interface. This allowed us to develop automated test scripts to drive the test runs. We wrote automated test scripts in the Network and User Simulator environment to simulate the operational profile of the Central Restoration System. For example, user scripts mirror the operations of the various users, and alarm scripts simulate the cable-cut alarms from the Restoration Node Controller. We wrote automated test scripts in the Office Simulator environment to simulate the operational profile of the Restoration Node Controller. For instance, alarm scripts simulate cable-cut alarms from Lightguide Terminating Equipment.

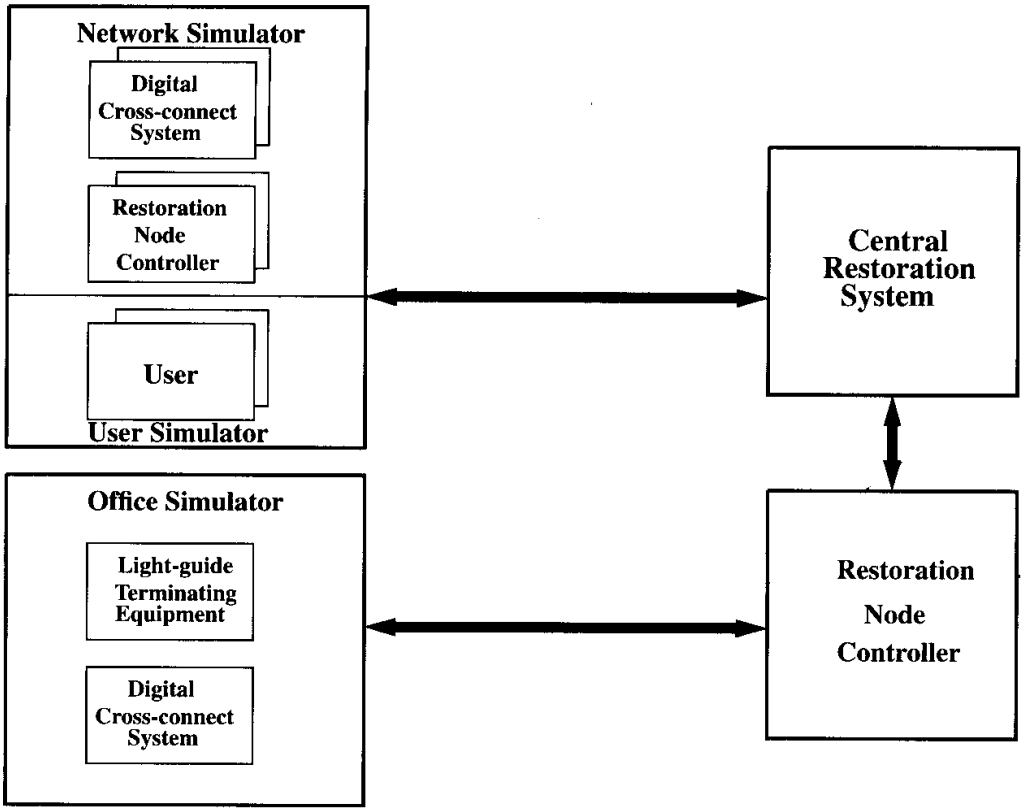

Figure 5.5 Simulator test environment.

Once the test environment was developed, we had the tools in place to execute our load/stress/stability test program.

5.8.2.4 Execute load/stress/stability test program. The purpose of our load/stress/stability test program was to ensure that the Central Restoration System and Restoration Node Controller met our customer's stringent reliability objectives for FASTAR. We wanted to be certain that FASTAR did what it was supposed to do and, more important, did not do anything that it was not supposed to do. We orchestrated load/stress/stability tests by executing each system's operational profile in its respective test environment. We found and fixed defects and reran the tests. Using this process, failures occurred less frequently and the reliability of FASTAR improved over time.

During our load/stress/stability test program, we tested the Central Restoration System and Restoration Node Controller under the same conditions that would be encountered in the field. These tests were performed after the individual commands and operations had been functionally verified. Load tests mirrored the operational profile of the two systems. Stress tests executed worst-case cable-cut scenarios on each of the systems. Stability tests verified each system with its operational profile for a minimum of 100 hours of continuous operations. Often, stability tests ran for over one month.

The FASTAR operational profile showed that cable cuts happen very infrequently. Restoration of cable cuts was certainly the most critical operation of the system. Since it was not feasible to execute the test program for a year to simulate the expected 10 cable cuts, we designed accelerated test runs. In the case of the Restoration Node Controller, load was accelerated by a factor of 24 during stability tests. This allowed us to execute an entire day's activities in one hour. Hence, we could execute a year's operational profile in the test environment within a matter of weeks. For the Central Restoration System, load was accelerated by a factor of 1.25 during stability tests and by a factor of 50 during stress tests.

We measured test results for each load/stress/stability test run and tracked the growth of reliability over time. The metrics we used for tracking results were DS3 paths correctly reported by the Restoration Node Controller and DS3 paths correctly rerouted by the Central Restoration System. When we started obtaining load/stress/stability test results, 90 percent of DS3 paths were correctly reported and rerouted. We continued to improve the system, run tests, measure results, and fix defects. When we were ready to release FASTAR, reliability had grown above the 99 percent mark. Still we continued to improve the system until our results reached nearly 100 percent.

5.8.3 FASTAR: SRE benefits

SRE was instrumental to the success of FASTAR by ensuring that reliability objectives had been satisfied prior to introducing the system into the AT&T network. SRE results guided us in making the critical decision on when to release FASTAR.

The major benefit from SRE was that it gave us the confidence to deploy the system. In fact, we believe that without SRE we would have not been able to release FASTAR, because we had no other way to be certain that the system would not have a defect that could seriously impair the operation of the AT&T network. With the results we obtained from SRE, we were able to decide when the system was ready to be released, and we were confident that FASTAR would dramatically improve network operations. Our SRE prediction has proven to be accurate. FASTAR handles single DS3 restorations on a daily basis and major cable cuts throughout the year.

5.9 Application to the Power Quality Resource System (PQRS)

5.9.1 Project description

In recent years, AT&T has placed increased emphasis on the role of AC and DC power systems and other network infrastructure elements

with respect to network reliability. While normally maintaining a low profile among the more exciting, high-visibility technologies in use today, power and infrastructure systems must be in place, engineered correctly, and maintained appropriately to ensure reliable operations of the network elements they support.

In mid-1992, AT&T's Network Services Division, which operates the core network, deployed a new internal application to support network operations and engineering of power and infrastructure systems. The Power Quality Resource System (PQRS) maintains an inventory of the major units of AC and DC power systems and infrastructure elements, including building environmentals such as chillers. In addition, PQRS contains algorithms to calculate fuel and battery reserve time. PQRS provides the capability to calculate the number of hours the office can run on auxiliary power generated by its standby engines and by battery backup.

A typical use of PQRS involves support of the demand (or reactive) maintenance process. In the event of a power failure alarm, for example, AT&T's surveillance and alarm center personnel will determine the appropriate response, given a number of factors including battery reserve time, fuel reserve, or availability of a portable engine, and so on—information extracted from PQRS upon demand.

The project team applied state-of-the-art methods and tools in the development of PQRS. In the preliminary planning stages for the project, a decision was made to apply SRE—in particular, operational profile testing.

5.9.2 Developing the operational profile

5.9.2.1 Initial functional profile derived from DFDs. The PQRS requirements team used dataflow diagrams (DFDs) and entity relationship diagrams (ERDs) to model the application's functions and stored data. The level 0 DFD is used to identify the major processes and functions of the system. The level 0 diagram is the DFD directly beneath the context diagram in a leveled set of DFDs, and it represents the highest-level view of the major functions within the system and the major interfaces between those functions [Your89]. The initial functional profile was derived from PQRS's level 0 DFD.

At this early point in the development cycle, the team estimated transaction volume in terms of broad ranges and used the information as input to the formulation of the hardware and software architecture of the system.

5.9.2.2 Final functional profile. The project team used a process known as *architecture discovery* to understand the user's expectations, to identify the problem to be solved, and to determine if a solution (par-

tial/whole) already exists. As the software architecture solution began to be defined, logical groupings of transactions emerged. The transaction classes were groups of transactions which were roughly equivalent in terms of CPU utilization, software architecture elements applied, user think time, and performance/response time.

The architecture team turned over a list of 21 transaction classes to the requirements team to develop estimates of daily system-usage volumes (best and worst case) by user group for the first and fourth years of operation. The requirements team pulled in a subject matter expert from the user community to help produce the estimates.

The architecture team used the functional profile to finalize the resource budgets and system configuration. The team also established a plan to instrument the system so that field data could be collected to assess the system's performance, to identify usage trends, and to validate/update the estimates that were used for the performance model and for the operational profile.

5.9.2.3 Operational profile. Over time, PQRS's project-specific definition of *operational profile* has evolved to the following: listing of each transaction, mapping of each transaction to an operation, execution frequency of each operation, and usage characteristics (time of day, number of users). The definition does not include environmental and customer-type variables, as they are not factors for this internal product.

About 3 months prior to system test, members of the project team met to finalize the operational profile. By that point, the application itself was well understood. With much discussion and lively interchange, 56 individual transactions were identified and mapped to 12 operations. The rationale for aggregating the transactions followed the logic of the architecture phase: that roughly equivalent transactions could be grouped based on CPU utilization and software architecture elements involved. Further, due to the difficulty (and inherent inaccuracy) in estimating transaction volumes at the lowest level (add battery record, view battery record, modify battery record, etc.), the team agreed that the aggregate approach was appropriate for this data-intensive application.

PQRS's heaviest users (in terms of numbers of transactions they were expected to generate) work regular office hours, and most are in the eastern time zone. The project team settled on three system modes for the initial operational profile: off hours, prime-time peak, and prime-time nonpeak. Experience with the production system has validated these assumptions.

With a total number of transactions expected during an average business day and considering the predominant daytime usage, it was easy to estimate the off-hours volume by class. To segregate the re-

mainder between peak and nonpeak hours, the team approximated prime-time peak as 50 percent higher than prime-time nonpeak.

5.9.2.4 Updating the operational profile. With the instrumentation provided in PQRS's software to log transaction volumes at the lowest level, including their performance, it has been simple for the project team to update the operational profile for each new release. Following the initial operational profile testing of release 1.0, the team has further condensed the number of operations. Every transaction is mapped to one of seven basic operations. For each new release, the seven basic operations are updated to include functionality delivered in the previous release, and the new functionality is isolated in three separate, additional operations to ensure thorough test coverage of the new capabilities, but in proportion to their expected usage.

PQRS's logs are shipped regularly from the production site to the development machine. Numbers of login sessions and transactions are tracked by class, by user, and by hour of the day. The data are also analyzed for transaction mix and volumes during network emergencies such as Hurricane Andrew.

5.9.3 Testing

The primary objective of the project team in testing in accordance with the operational profile was to certify PQRS's software quality from the users' viewpoint. The team expected to identify software failures occurring as a result of combinations of transactions or due to hours of continuous operation, failures not observable through traditional functional testing.

5.9.3.1 Tools. A high degree of automation has always been an assumption with respect to operational profile testing for both test generation and test execution. The test team developed a test executive that controls generation of test scripts and executes them based on the operational profile. Every hour, a scheduled job begins executing, and it performs the following:

1. Determine system mode for the next hour and number of users who will log in.
2. Determine number of transactions by class that will be executed; randomly select individual transactions within the class to be executed; generate test cases in real time based on a set of fixed, valid data.
3. Randomly map test cases to users and determine a random minute within the hour for the start of each user session.
4. Create an execution script for each user.

During each hour, the test executive provides overall control for execution of the scripts and monitoring mechanisms.

5.9.3.2 Varying occurrence probabilities. Due to the simplicity of the design of the test executive, the test team has been able to vary the occurrence probabilities and transaction volumes by class. Since the operational profile is only an estimate at best, the team has found it useful to vary it a bit to reflect the production environment, i.e., every day in the field will not be an average day. This has resulted in the detection of at least one failure that may not have been detected in other phases of testing. This follows from the expectation that reliability will change when the environment changes [Musa87].

With such a data-intensive product, and each new release delivering additional inventory capability (i.e., new screens and tables), the team has at times intentionally increased the rate of inventory transactions significantly above the norm to evaluate the results. And due to heavy usage of PQRS's reporting capabilities during network emergencies, the test team has also varied the mix of transactions to simulate non-average field conditions.

5.9.3.3 Extension to performance and stress testing. Because PQRS was instrumented to track transactions and performance, performance testing is not a separate testing discipline for the project. During the intervals when the operational profile is being executed, the system logs are monitored closely for anomalies in performance.

The simplicity of the design of the operational profile test executive has also made it extensible to stress testing. Transaction volumes, transaction mix, and numbers of users are systematically intensified and the logs examined for evidence of failures and to evaluate performance.

5.9.4 Conclusion

Development of the operational profile was an extension of work already performed for understanding the customers' requirements. With the inclusion of application measurements and logging, it has been a relatively simple matter to keep the profile updated for each release.

The benefits realized from testing in accordance with the operational profile include detection of several software faults not observed (and likely not observable) during traditional functional testing. It has certified the quality of the delivered product.

The investment made in automation continues to pay back, in its ability to run load testing continuously for two weeks without intervention and in its extensibility to stress testing and application for regression testing.

5.10 Summary

At AT&T, software reliability engineering, which includes the operational profile, was approved as a "best current practice" (see Chap. 6) in 1991. To qualify as a best current practice, a technique must have substantial project application with a documented, favorable benefit-to-cost ratio, support by world-class technology, and mechanisms for technology transfer (courses, reference material, jump starts and consulting, tools) in place. It must pass probing reviews by two committees of senior software managers [Ever93].

AT&T has used SRE and the operational profile to ensure the success of many projects, including such critical ones as DEFINITY, FASTAR, and PQRS. In one large software-development organization, it is being fully integrated into the development process. AT&T and other companies (for example, Hewlett-Packard) have found that application of the operational profile yields substantial savings in test costs and hence total project cost. You can expect to save even more if you use the operational profile to guide other development phases as well.

Problems

5.1 What simplifications should be made in the procedures for developing an operational profile for a software-based product for internal use in a company?

5.2 When can you go directly to an operational profile without creating a functional profile?

5.3 Can different customer groups have the same user groups?

5.4 Systems in heavy traffic conditions often will not allow the execution of low-priority operations such as maintenance and backup. How do you handle the effect on the operational profile?

5.5 Is it acceptable to merge a number of noncritical, rarely occurring operations into one operation?

5.6 What is a common failing in creating the function list?

5.7 Can an operational profile be developed for operations that span multiple machines?

5.8 What is *reduced-operation software?*

5.9 Assume you are the development manager on a software project for an airplane flight control system. How would you implement an SRE program for

this software development effort? Include the major steps you would undertake to ensure the success of this program.

5.10 Discuss the advantages and disadvantages of aggregating individual transactions in the formulation of the operational profile.

5.11 Discuss the benefits of instrumenting your application to record field data.

Part

2

Practices and Experiences