Abstract

This paper provides an overview of a step-by-step approach for developing the operational profile used in software reliability engineering. The operational profile can help guide managerial and engineering decisions throughout the product life cycle, including requirements specification, design, implementation and testing. It is particularly valuable in guiding test planning and testing. The development and use of this profile typically involves product and market managers and systems engineers. It is used by software developers and managers, system test planners and testers, and quality assurance and reliability engineers. The material presented here has been drawn from experience in developing operational profiles for a variety of applications.

Keywords: software reliability engineering, operational profile, functional profile, customer profile, user profile, system modes

1. Importance of the Operational Profile

The operational profile, which is a quantitative characterization of the way a system will be used, plays an essential role in software reliability engineering. The reliability of software-based products depends on just how a customer will employ them (Musa, Iannino, and Okumoto, 1987). Good estimation of reliability during test depends on knowing how the product will be used, so that tests can be planned to represent the field environment, or appropriate adjustments made to the estimates. In addition, knowledge of usage patterns is invaluable in guiding software development decisions.

1.1 Benefits

The operational profile offers great benefits in increasing productivity, speeding up development, and increasing reliability by indicating where and to what extent development effort can be differentially assigned on the basis of usage. Typically, the same development effort would be retained for the most-used part of the system and reductions would be made for lesser-used parts, with the amount of reduction related to the difference in usage. Although savings may vary somewhat, among projects, the estimate of potential savings in system test for a "typical" project is 56%. This is 11.5% of total project cost. This estimate is supported by results from the International DEFINITY® system project (Abramson et al., 1992) and a Hewlett Packard multiprocessor operating system project (see below). Additional cost savings can be expected if the operational profile is used to guide other phases of software development as well.

The operational profile can support a very competitive way of introducing a new product: make a few highly-used features available quickly, and provide lesser-used features in subsequent releases. Using the operational profile to guide test will insure that if testing is terminated and the software is shipped due to imperative schedule constraints, the most-used operations will have received the most testing and the reliability level will be the maximum practically achievable for the given test time. When the operational profile is used to guide regression test, it allocates the limited test cases available in accordance with how customers will use the system, so that the faults introduced during program change that are most likely to be found are those that have the most effect on the reliability.

The operational profile improves communication between supplier and customer, and among different parts of the customer organization by structuring the specification of needs and making their expression more precise. It can benefit customers by directing their operator training efforts toward the most-used operations. The operational profile is frequently weighted by criticality (see Section 5), so it can reflect both how the

---

1. We will refer to "the" operational or functional profile for simplicity throughout, although there can be multiple profiles.

2. "DEFINITY®" is a registered trademark of AT&T.
system is being used and the relative importance of the uses.

The AT&T International DEFINITY® system project (a private branch exchange telecommunications switching system) implemented a development process quality improvement program, using software reliability engineering and the operational profile (Abramson et al., 1992). Automated test case generation based on the operational profile, incremental development with quality factor assessments that included software reliability, and cleanroom development techniques were also applied.

The set of improvements resulted in a factor of 10 reduction in customer-reported problems and in program maintenance costs, a factor of 2 reduction in the system test interval, and a 30% reduction in new product introduction interval. There were no serious service-affecting outages in the first year 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 the system test process. They set quantitative objectives and tracked quantitative progress against the objectives, using the results to improve project management. They used automated test and failure recording, with tests guided by the operational profile. As a result, system test duration and system test costs decreased at least 50%.

Software reliability engineering, including the operational profile, was approved as a best current practice in AT&T in May, 1991, based on substantial project application with a documented strong benefit/cost ratio.

1.2 Costs

The cost for defining an operational profile depends on the number of elements. Experience to date has indicated that the effort to construct an operational profile for an “average” project is typically about one staff month. An “average” project is defined as one with about 10 developers, perhaps 100,000 source lines, and a development interval of about 18 months. Large projects can cost more, but the increase is clearly less linear with project size. The benefit to cost ratio in developing and applying the operational profile is typically 10 or greater.

1.3 Need for procedure

In view of the importance of the operational profile, the need for an organized procedure to determine it has become paramount, although not every project will necessarily need to implement every step. This paper attempts to provide the "best" current approach in what is an evolving methodology, distilled from experience with a variety of projects.

We will use various specific examples in presenting this material. This does not imply that application of the operational profile is limited to these areas. In fact, we have not found a software system to which it could not be applied.

2. Concepts

A profile is simply a set of disjoint (only one can occur at a time) alternatives and their associated probabilities of occurrence or weighting factors (which add to 1). Profiles can often be conveniently conveyed as plots. Often the logarithm of the probability of occurrence is plotted because of the wide range of probabilities encountered.

In many cases, usage information is available or can be estimated most easily in terms of rates, such as transactions or commands per hour. This information does not represent a true profile until converted to probabilities by dividing by total transactions or commands per hour. Converting to probabilities is important because you can make a quick completeness check by seeing if the probabilities add to 1.

We will follow the convention of naming the profile by the generic name of the members of the set of alternatives. The operational profile is based on operations. Other profiles, such as the functional profile, may be generated in the course of developing the operational profile. The functional profile is based on functions.

A function represents a task or part of the overall work to be done by the projected system in a particular environment, as viewed by the system engineer and high-level designer. An operation represents a task being accomplished by the implemented system in a particular environment, as viewed by the people who will run the system (it is also the view of testers, who try to put themselves in this position).

Environment is described by environmental variables. These are conditions that affect the way in which the program runs (the control paths it takes and the data it accesses) and hence can fail, but which do not relate directly to features, which are usually considered as operational functional capabilities that are of interest and value to customers. For example, hardware configuration and traffic load are environmental variables.

Functions evolve into operations as the operational architecture of the system is developed. By "operational architecture," we mean the way that operations are defined and combined together in execution to accomplish the functions that have been specified for 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 simple. For example, a function in the administration of a telecommunications switching system might be the relocation of a telephone from one room to another. This function might be implemented by two operations, removal of a telephone and installation of a new one, each associated with a run, because this design is more convenient with respect to system and operational architectures. Generally there are more operations than functions, and operations tend to be more refined.

Operations are associated with runs. A run is a subdivision of the time period of execution of a program, usually based on the accomplishment of a user-oriented task. For example, a run might be a command entered by a user, a transaction, or an external signal to be processed. Functions have some relationship with time since they are related to operations, but in a much looser sense, because of the potentially complex mapping from functions to operations, as noted above.

Functions tend to be independent of the system and operational architecture (they are defined at the system requirements stage before the architecture is known), operations do not.

An operation represents a set of run types, or specific executions of the operation associated with specific input states or sets of values for the input (including environmental) variables. For example, the administrative operation 'removal of a telephone' in a telecommunications switching system represents many run types. A run type would refer to the processing involved with removing a specific telephone under specific environmental conditions: a particular number, location, type, feature set, administrative traffic load, etc. A run type is associated with an input state. A run is an instance or repeated execution of a run type. For example, if a telephone were erroneously removed too early and then replaced, but finally removed at the correct time, two runs of the removal run type for the specific telephone would be experienced.

The functional profile is usually developed during the system engineering or requirements definition phase, possibly extending into the beginning of high-level design. It should be viewed as part of the system requirements document. It is not dependent on design methodology. The functional profile can be used as a guide in allocating resources during design, coding, unit test, and possibly subsystem test.

The operational profile is needed as we start test planning, because tests must be based on the actual operations implemented in the system rather than the functions conceived during system definition. It should be viewed as part of the system test plan. The operational profile is used to allocate effort during test, to select the tests to be run, and to determine the order in which they should run.

Operational-profile-driven testing is a very efficient method of testing, in the sense that on the average (but not necessarily for any particular run) it increases the reliability (reduces the failure intensity) rapidly per unit execution time. Its efficiency is based on the fact that failures are identified on the average (and hence the faults causing them can be removed) in order of their frequency of occurrence. That is, since failures occur most frequently for the faulty operations that are used the most frequently, testing the most frequently used operations first exposes the most frequently failing operations first. Removing the faults that cause failures first increases the reliability rapidly. If the operational profile is weighted by a financial measure of criticality, then life cycle failure cost will be reduced rapidly with execution time. Note that the user in the field will detect failures on the average in order of their frequency of occurrence as well, if they have not already been found in test.

In many projects it is possible to develop the operational profile directly during system definition, without passing through the intermediate stage of the functional profile. This is particularly true when requirements are detailed to the point of incorporating the operational architecture: detailing the commands or the transactions the system will respond to. However, in some cases, the information required to develop the operational profile may not be available until the design or even a substantial part of the implementation is complete. If you should decide to develop the operational profile directly, you should consider the steps outlined in the procedure for developing the functional profile in combination with the procedure for developing the operational profile.

3. Developing the Operational Profile

In cases where an "average" reliability objective and "average" reliability measurements over all applications are acceptable, there will be no need for more than one operational profile. However, if a product has specially tuned versions (i.e., different reliabilities for applications that represent significantly different use for particular customers or sets of customers), then it will be necessary to determine multiple operational profiles. Similarly, if the differences between operational profiles are such that they significantly affect the reliability measurements, then you will need to determine and test with multiple profiles. In all situations different customers, users, and system modes need to be identified separately for the purpose of
weighting differently the contributions each makes to the operational profile or profiles.

The operational profile is developed in up to five steps:
1. Find the customer profile,
2. Establish the user profile,
3. Define the system modes,
4. Determine the functional profile,
5. Determine the operational profile.

Some steps may not be necessary in a particular application. For example, the customer profile is not needed if you only have one customer. In some cases, sufficient information is available early in development to skip the functional profile step.

At each step, determine the level of detailed breakdown desired. This depends on the net economic return from giving differential treatment or weighting to the items at that step with respect to software development. For example, consider a customer profile that is broken down by industry. Identifying certain important individual customers may be worth the extra cost of data collection and differential treatment (for example, special testing). The degree of detailing does not have to be uniform across the system. For example, critical system modes or functions may be analyzed to greater levels of detail.

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

3.1 Finding the customer profile

The customer profile is the complete set of customer groups with associated weighting factors, usually related to usage. If all customers will use the system in the same way, a customer profile is not necessary.

Note that the customer is the person, group, or institution that is acquiring the information system or information system service. A customer group is a set of customers that will use the system in the same way. For example, large pharmacies may use a telecommunications switching system in a manner very much like other large retail establishments and thus should be grouped with them.

Information on customers is typically obtained from marketing data on previous related systems, modified by marketing estimates that take the differences in the appeal of the new system into account. The expected customer base is usually documented as an integral part of the business case that has been developed for a proposed product. One software development project (Abramson et al., 1992) has customer model documents that describe current and expected use of their product. They are written by system engineers and are valuable source documents for developing operational profiles. In addition, they have been used in performance analysis, in defining cross-product test needs, and in requirements reviews.

The objective in choosing the weighting factor for each customer group is to use an easily obtained measure that best approximates relative usage. So that the operational profile that is determined will most closely match the one that actually occurs in the field. Probably the simplest approach is to use the proportion of total system deliveries expected to be made to that customer group, since this will ordinarily be proportional to use of the systems. In any case, it is important to use weighting factors that are consistent between test and field.

3.2 Establishing the user profile

The step of defining the user profile is necessary because users of a system are not necessarily identical to customers. The user profile is derived from the customer profile by looking at each customer group and determining what user groups may exist who will be using the system differently. If there are similar user groups among different customers, they should be combined.

A simple approach to establishing the weighting factors for a user group for a particular customer group is to use the proportion those users represent of the total users for the customer group. If an insurance company has 900 claims entry clerks and 20 actuaries out of 1000 users, the claims entry clerk user group would have a weighting factor of 0.90 and the actuary user group, 0.02. A much better approach would be to employ the proportion of total usage of the system by the customer group that the user group represents. In any event each user group weighting factor for a particular customer group should be multiplied by its customer group weighting factor to obtain the overall user group weighting factor. When user groups are combined across customer groups their overall user group weighting factors should be added to yield the total user group weighting factor.

Consider a simple example to clarify the foregoing points. We will follow this example throughout the paper. In this paper, examples will generally be selected to be realistic as to the principles they illustrate, but they will be smaller in size than real life examples for clarity and economy of space in presentation. Consider a telecommunications switching system that is sold to institutions for internal use, and of course connections to the outside. Assume that there are two customer groups, hospitals and large retail stores. Suppose that hospitals represent 40% of the
use of this product, and large retail stores, 60%.

Consider the hospitals customer group. System users include telecommunications users (the people making calls and sending data), attendants (the internal operators who answer the hospital's general telephone 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). The classes were chosen this way because they all use the system quite differently. The proportion of usage by each user group is noted in Table 1.

A system can switch between system modes so that only one system mode is in effect at a given time, or different system modes can exist simultaneously, sharing the same computer resources. A system mode can be defined as a weighted combination of other system modes.

An operational (and perhaps also functional) profile is determined for each system mode. Thus the definition of multiple system modes implies the determination of multiple operational and perhaps functional profiles. Such a determination is usually relatively easy to do, because someone can be found with the necessary expertise to know all of the functions or operations for that system mode. It is more difficult to find someone with enough knowledge to generate the complete functional or operational profile for the entire system. But the system mode profile can be determined and used to weight the separately established functional or operational profiles for each system mode, yielding an overall functional or operational profile.

Some different bases for characterizing system modes and associated examples are indicated in Table 2. Since some of these conditions vary with operating site or time, the latter variables are sometimes associated with system mode.

The use of different system modes is a particularly convenient way of accounting for operational profile changes as users become more experienced. Two different system modes can in most cases capture the extremes in experience. One system mode would characterize how novices use the system; the other, experts. Intermediate amounts of experience can be handled by adjusting the occurrence probabilities (which

---

Table 1. Establishing the user profile – example.

<table>
<thead>
<tr>
<th>User group</th>
<th>Overall user weighting factor for customer group</th>
<th>Overall user weighting factor for customer group</th>
<th>Overall user weighting factor for customer group</th>
<th>Overall user weighting factor for customer group</th>
<th>Total user weighting factor for customer group</th>
</tr>
</thead>
<tbody>
<tr>
<td>Telecommunications users</td>
<td>0.90</td>
<td>0.36</td>
<td>0.90</td>
<td>0.54</td>
<td>0.90</td>
</tr>
<tr>
<td>Attendants</td>
<td>0.05</td>
<td>0.02</td>
<td>0.07</td>
<td>0.042</td>
<td>0.062</td>
</tr>
<tr>
<td>System administrator</td>
<td>0.035</td>
<td>0.014</td>
<td>0.01</td>
<td>0.006</td>
<td>0.02</td>
</tr>
<tr>
<td>Maintenance personnel</td>
<td>0.015</td>
<td>0.006</td>
<td>0.02</td>
<td>0.012</td>
<td>0.018</td>
</tr>
</tbody>
</table>

An analysis of the large retail store customer group yields the same classes of users. However, we will assume that usage by attendants in the large retail store is higher because few customers calling in call directly to a department. On the other hand, usage by the system administrator in a hospital is higher because of the frequent changes needed as patients are admitted and discharged. The proportions of usage are again given in Table 1. Table 1 also gives the total user group weighting factors, computed by taking the customer-group-weighted sum of the user group weighting factors. For example, hospitals represent 0.4 of customer group usage and attendants represent 0.05 of that, so that hospital attendants represent 0.02 of overall use. Large retail stores represent 0.6 of customer group usage and attendants represent 0.07 of that, so that large retail store attendants represent 0.042 of overall use. The total use by attendants is 0.062, as indicated.

3.3 Defining system modes

A system mode is a set of functions or operations that are grouped together for convenience in analyzing system operational behavior.
must sum to 1) of the two modes.

Table 2. Bases for characterizing system modes.

<table>
<thead>
<tr>
<th>Basis</th>
<th>Examples</th>
</tr>
</thead>
<tbody>
<tr>
<td>User group</td>
<td>Telecommunications switching system with administration and maintenance modes</td>
</tr>
<tr>
<td>Significant environmental condition</td>
<td>Overload or normal traffic in any system</td>
</tr>
<tr>
<td></td>
<td>Prelaunch or flight in a space vehicle flight system</td>
</tr>
<tr>
<td></td>
<td>Initialization (startup or reboot for failure recovery) or continuous operation (includes warm start after an interruption)</td>
</tr>
<tr>
<td>Operational architectural structure</td>
<td>Transaction processing system for retail store with online sales mode and after hours billing mode</td>
</tr>
<tr>
<td>Criticality</td>
<td>Shutdown mode for nuclear power plant in trouble</td>
</tr>
<tr>
<td>User experience</td>
<td>Novice mode or experienced user mode</td>
</tr>
<tr>
<td>Hardware component</td>
<td>Distributed system with different hardware components performing different functions</td>
</tr>
</tbody>
</table>

Continuing the telecommunications switching system example we have been working with, we will construct a system mode profile from the following system modes:

1. Business use mode,
2. Personal use mode,
3. Attendant mode,
4. System administration mode,
5. Maintenance mode.

The latter three system modes represent user groups and are essentially disjoint in that they do not share functions or operations. The first two modes do share most of the functions or operations, and they could be combined into one. However, although the same kinds of use are expected in both, the relative patterns of use can be expected to be quite different. Since the statistics on these two user groups are readily available, division into these modes is a matter of considerable convenience. Assume that all of the large retail store use is business use, but that the hospital use is 60% business and 40% personal. Using Table 1, we can compute the business use probability as the probability of a large retail store telecommunications user (0.54) added to the probability of a hospital telecommunications user (0.36) multiplied by the proportion of business use (0.6). Personal use probability is computed in a similar fashion.

Table 3. Establishing the system mode profile – example.

<table>
<thead>
<tr>
<th>System Mode</th>
<th>Probability</th>
</tr>
</thead>
<tbody>
<tr>
<td>Business use</td>
<td>0.756</td>
</tr>
<tr>
<td>Personal use</td>
<td>0.144</td>
</tr>
<tr>
<td>Attendant</td>
<td>0.062</td>
</tr>
<tr>
<td>Administration</td>
<td>0.02</td>
</tr>
<tr>
<td>Maintenance</td>
<td>0.018</td>
</tr>
</tbody>
</table>

3.4 Determining the functional profile

The next step is to analyze each system mode in turn, breaking down the work to be done into functions and thus creating an initial function list. A choice must be made between developing an explicit or implicit operational profile. Environmental variables that will have a major effect on program development will then be identified. Next, a new function list will be created from the initial function list and the environmental variables. Finally, the probabilities of occurrence of the new functions will be established, using both measurement and estimation.

3.4.1 Selecting number of functions: There is no specific number of functions established for the functional profile of a software-based system, but it typically involves from 50 to several hundred. The number of functions generally increases with the size of the project, the number of system modes, the number of major environmental factors, and the degree of differentiation among functions. The key question in determining the degree of differentiation is, “Does this potential additional function have processing that is substantially different from what has already been covered, such that the development of the code to support it should be separately managed with respect to work priorities and resource allocations?”

As an example of differentiation, consider a command X with parameters A and B. Parameter A can have values A1 or A2 and parameter B can have values B1, B2, or B3. Suppose the setting of parameter A has more effect on the difference in code executed than parameter B. We could develop three different degrees of differentiation as noted:
Case 1: $X$ 1 function (all sets of parameter values considered as grouped under one function)

Case 2: $X\ A1\ B$ 2 functions (different values of parameter A considered as defining different functions, but not other parameters)

Case 3: $X\ A1\ B1$ 6 functions (each different set of parameter values considered as a different function)

In each case there are six possible ways in which the command can be executed, but they are grouped differently into functions. The degree of differentiation selected represents a balance between greater flexibility in allocating resources or sequencing work tasks and the greater cost involved in gathering and analyzing data on a more detailed basis and also managing on that basis.

The command activated and its parameters are input variables, since they exist external to the run being executed and are used by or affect the run. Those input variables that differentiate one function (or operation, as we will see later) from another are called key input variables. In Case 1 above, only the function name is a key input variable. In Case 2, the parameter A is also a key input variable. In Case 3, the function name and both parameters are key input variables.

In many cases, the different values of the input variable are actually ranges of values, which we call levels. The number and extent of the levels that we define should be determined by whether there are substantial differences in processing among them.

3.4.2 Choosing explicit or implicit operational profile: It is now necessary to decide whether to develop an explicit or implicit operational (and hence functional) profile or some combination of the two. The explicit profile consists of a set of completely enumerated (all values listed) functions and their associated probabilities of occurrence. If there are two key input variables A and B, each with three values, the set of completely enumerated functions is \{(A1,B1), (A1,B2), (A1,B3), (A2,B1), (A2,B2), (A2,B3), (A3,B1), (A3,B2), (A3,B3)\}. The implicit profile consists of the set of values of key input variables, each with the associated probabilities of occurrence. In the foregoing case, we would have the sets \{A1, A2, A3\} and \{B1, B2, B3\}. The implicit profile can only be used when the probabilities of occurrence of the key input variables are independent of each other (at least approximately). Otherwise, the explicit profile must be used, and its occurrence probabilities must be measured or estimated directly and not by multiplying the occurrence probabilities of levels of individual key input variables.

When testing with the explicit profile, runs are selected by randomly choosing among operations in accordance with their occurrence probabilities. When testing with the implicit profile, runs are selected by randomly choosing the level of each individual key input variable in accordance with its occurrence probabilities. The operation is determined by the naturally resulting conjunction of these values, and is thus selected implicitly.

The chief advantage of the implicit profile is that the number of elements that must be specified is equal to the sum of the numbers of levels of the key input variables. With the explicit profile, the number of elements can be as high as the product of the numbers of levels of the key input variables. In practical profiles, there is usually some but not complete interaction among key input variables in determining probability of occurrence. The number of elements will be somewhere between the sum and the product of the numbers of levels of the key input variables.

A practical application of the implicit operational profile has been made for a telecommunications switching system [Abramson et al., 1992 and Juhlin (1992)]. The implicit operational profile approach was used to determine occurrence probabilities of events in the associated telecommunications networks by using "call trees." It can be seen in Figure 1 that an event is characterized by a path through the network and that each branch has an associated probability. Instead of selecting test cases from a complete list of all possible paths with associated path probabilities, selections can be made from each set of branch alternates with the associated branch probabilities. The result is an implicit operational profile for calls based on a vector of key input variables. Each branch represents a value of a key input variable, and all the branches that diverge from a node represent that key input variable.

3.4.3 Creating the initial function list: The initial function list focuses on tasks. The best source of information for creating the initial list of functions is usually the system requirements. Other sources such as a rapid prototype, a user manual, or a previous version of the system can also be helpful, but one must be cautious, because they are often incomplete. The process of creating the function list should involve the users. It frequently serves to identify incomplete or fuzzily-defined requirements.
Let us return to our previous example of a telecommunications switching system and consider the system administration mode. We will take a greatly simplified example to illustrate the development of the part of the functional profile associated with this mode. We will develop an explicit profile.

Assume that the principal administration functions are adding a new telephone to the exchange, removing a telephone, relocating a telephone or changing the nature of service provided, and updating the online directory from the accumulated changes. Then the initial list of functions has 4 elements.

### 3.4.4 Identifying the environmental variables:

Next, you should identify the environmental variables and the different values or levels of these variables that will have major effects on processing (require major differential development efforts such as substantial new modules). Common environmental variables that could have a major effect on processing include computer and operating system employed and hardware configuration (types of devices your system interacts with and their network configuration). The best approach here is probably for several experienced system design engineers to brainstorm a list of environmental variables that might necessitate the program to respond in different ways, and then decide which of these would likely have major impact on the program.

In our example, the type of telephone will be an environmental variable with a major effect on processing. Assume that type of telephone can take on several values. However, only analog telephones A or digital telephones D have substantially different effects on processing. We then have two levels for the environmental state: A and D. Telephone type does not affect directory updating.

### 3.4.5 Creating the final function list:

Before creating the final function list, examine dependencies among the key environmental and functional input variables. If one variable is totally or almost totally dependent (correlation approaching 1) on another, the dependent variable can be eliminated, reducing the number of functions defined. If "partially" dependent (correlations between 0 and 1), then just as with independent variables all possible combinations of the levels of those variables must be represented in the final function list, in preparation for the explicit determination of occurrence probabilities.

The final function list for the telecommunications switching system is:

**Table 4. Final Function List**

<table>
<thead>
<tr>
<th>Function</th>
<th>Environmental State</th>
</tr>
</thead>
<tbody>
<tr>
<td>Relocation or Change</td>
<td>A</td>
</tr>
<tr>
<td>Addition</td>
<td>A</td>
</tr>
<tr>
<td>Removal</td>
<td>A</td>
</tr>
<tr>
<td>Directory Updating</td>
<td>A</td>
</tr>
</tbody>
</table>

### 3.4.6 Determining occurrence probabilities:

The best source of data for occurrence probabilities is measurements taken on the previous release or system, an analogous system, or the manual function that is being automated. These probabilities must then be adjusted to account for new functions and environments that are being added and known expected changes in probabilities due to other factors. Note that measurements are actually taken on operations. The measurements must be combined (usually by simple addition) to reflect the fact that functions commonly map to one or more operations. Measurement of operations will be covered in Section 3.5.4.

Most systems include a mixture of previously released functions for which measures can be taken plus new functions for which use must be estimated. Although estimates are less accurate than measures, the total proportion of use of new functions is usually small, perhaps 5 to 20%. With usage being measurable for most functions, the overall accuracy of the functional profile should be good.

In the rare event where a software-based system is completely new and the functions have never been executed before, even by an analogous system or by manual procedures, the functional profile could be considerably in error. However, it will still usually give a picture of customer usage that is much more accurate than anything else we have, and hence will be valuable in helping engineer and manage the project. Just the process of predicting the usage of the new functions is important because the interaction required with the customer(s) will highlight the relative values of the various functions. Dropping little-used functions can provide increases in reliability, faster delivery, and lower costs, resulting in a more competitive product.

In our example, assume that there are 80 additions, 70 removals, and 800 relocations or changes per month. Directory processing represents 5% of the total system administration mode functions. Recall from Table 3 that the probability of
occurrence of the administration mode is 0.02.

Thus the functions and their occurrences, without consideration of environmental factors, are:

<table>
<thead>
<tr>
<th>Function</th>
<th>Overall Occurrence</th>
<th>Probability</th>
</tr>
</thead>
<tbody>
<tr>
<td>Relocation or Change</td>
<td>0.80</td>
<td>0.016</td>
</tr>
<tr>
<td>Addition</td>
<td>0.08</td>
<td>0.0016</td>
</tr>
<tr>
<td>Removal</td>
<td>0.07</td>
<td>0.0014</td>
</tr>
<tr>
<td>Directory Updating</td>
<td>0.05</td>
<td>0.0001</td>
</tr>
</tbody>
</table>

Table 5. Initial Functional Profile

In our example, assume that 80% of the telephones are analog and 20%, digital. The environmental profile becomes:

<table>
<thead>
<tr>
<th>Telephone</th>
<th>Probability</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>0.8</td>
</tr>
<tr>
<td>D</td>
<td>0.2</td>
</tr>
</tbody>
</table>

Table 6. Environmental Profile

Assume that the probabilities of occurrence of the initial functions and the types of telephones are independent of each other. Multiply the values of the environmental profile by the values of the functional profile taken without respect to the environment to obtain the part of the overall final functional profile contributed by the system administration mode. We have:

<table>
<thead>
<tr>
<th>Function</th>
<th>Overall Occurrence</th>
<th>Probability</th>
</tr>
</thead>
<tbody>
<tr>
<td>Relocation or Change</td>
<td>0.0128</td>
<td>0.0032</td>
</tr>
<tr>
<td>Addition</td>
<td>0.00128</td>
<td>0.00032</td>
</tr>
<tr>
<td>Removal</td>
<td>0.00112</td>
<td>0.00028</td>
</tr>
<tr>
<td>Directory Updating</td>
<td>0.001</td>
<td></td>
</tr>
</tbody>
</table>

Table 7. Final Functional Profile

3.5 Determining the operational profile

Determination of the operational profile is generally performed by test planners, with extensive collaboration from system engineers and designers. There are two ways of obtaining the operational profile for use in testing:

1. recording the input states experienced in field operation,

2. developing the operational profile.

First, however, we must divide the execution of the program into runs, define the input space, and partition the input space into operations.

3.5.1 Dividing execution into runs: The division of the execution of a system into runs is somewhat arbitrary. The goal is to be able to relate to the environment. Usually a run is initiated by some environmental intervention or control input that is external to the program. It is usually terminated by the next intervention, unless termination occurs because of a failure. For example, a run might represent a command that an operator enters at a terminal, a transaction, a telephone call, or an alarm to be processed. In many cases, it may be useful to define a run as an end-to-end activity of a customer. There is no implication that a program necessarily stops or pauses between runs. “Initiation” and “termination” refer in most cases to distinct changes in control flow.

Runs are logical rather than machine entities. The execution of a run does not have to occur continuously in time, but can proceed in segments. For example, a run on a multiprogrammed system may exist as a set of time slices of execution.

3.5.2 Defining the input space: The input space (see Figure 2) consists of the set of all possible input states. The input state or set of values of input variables uniquely determines the particular instructions that will be executed and the values of their operands. Thus it establishes the path of control taken through the program. It also uniquely establishes the values of all intermediate variables computed by the program and all output variables (data items external to the program that are set by the program). Thus the output state or set of output variables is uniquely determined by the input state.

An input variable for a run is any data item that exists external to the run and is used by or affects the run. There doesn’t have to be a physical input process. The input variable may simply be located in memory or in a file, waiting to be accessed. “Used by” implies a deliberate design decision while “affects” also includes inadvertent influences on the program.

The significant issue in defining the input space is to develop a “practically” complete list of input variables. By “practically” complete we mean that all input variables are identified except for those that take on one value with very high probability. If variables that affect the operation of the program are left out, run types and hence failing input states can not be clearly identified. It will then not be possible to reproduce either failing or successful behavior unambiguously. For example, some input variables may be hidden and hence not considered...
as inputs, the fact that they influence the run not being initially apparent. They may show their effect only after interactions with other runs. Rather than wait for the interactions, a conscious effort should be made to identify the hidden input variables and add them to the input state. Since the identification process will never be perfect, other strategies that tend to diversify the environment in which a program is tested should also be employed. The amount of hidden input variable identification effort to be employed and the proper selection of additional strategies should be based on product reliability requirements, costs of the extra efforts, and any information you might have on the probability of occurrence of the interactions.

A failure is a departure of program operation from program requirements. The implication is that the program requirements are the operational behavior expected by the user and thus (at least eventually) the customer. Such an operational departure will clearly involve a departure of the output state from the requirements. Thus the occurrence or non-occurrence of a failure can (in theory) be determined by test for each input state or run type. We say "in theory" because the number of input states involved generally makes this determination impractical to pursue.

3.5.3 Partitioning the input space into operations: Practical profiles must be limited to several hundred (or at most, several thousand) elements, due to the cost of developing a profile (collecting data, making estimates, etc.). The idea of the "operation" and the "operational profile," based on grouping of input states or equivalently, a partitioning of input space into domains, was conceived to satisfy this necessity. Partitioning does not imply that only one test can be selected per operation; it simply provides the framework for sampling nonuniformly across the input space. If operations are selected randomly in accordance with the operational profile and then input states are selected within the domain of the operation, there will be nonuniform test selection that matches the operational profile.

Operations should consist of run types that share the same input variables. This makes it easier to view and understand possible interactive relationships among the input variables. Then one can take advantage of the relationships by setting up a common structure for the domain of the operation for more efficient testing.

The domain of the operation should be divided into subdomains, if possible, that represent run categories. The division is defined in terms of levels or ranges of values of the input variables. A run category is a group of run types that exhibit (as nearly as possible, recognizing perfection is impossible) homogeneous failure behavior. By homogeneous, we mean that each run type in the run category is representative of the failure behavior of all the run types in the run category. All run types in the run category are either nonfailing or have precisely the same failures (there can be multiple failures per run type).

Only one test is required per run category (one point in what can be a fairly large subdomain), greatly increasing testing efficiency. In practice, this may be two or several points to reduce the risk from the fact that you can approach homogeneity but very rarely achieve it.

In order to define run categories that approach homogeneity, look for run types that at least execute the same code path.

It is wasteful of testing resources to divide a truly homogeneous area of input space into more than one run category. However, it is desirable for run categories to have approximately equal probabilities of occurrence, because we will be selecting uniformly among them (or executing all of them, in which case, executing them different numbers of times would be too complex and time-consuming). These requirements are conflicting, but we suggest you emphasize equal probability. Since in practice, you can only achieve near homogeneity, the possible waste is not important. You will simply be reducing risk by dividing some areas of input space, and making them closer to homogeneous.

As an example of an operation, consider an airline reservation system with a reservation made for a single-leg flight. The input states or run types will differ, based on passenger name, flight number, originating city, terminating city, etc. A two-leg flight would represent a different operation. It will have a different set of input variables, because it will have an additional flight number and a connecting city.

When an important input variable is known to be nonuniformly distributed with respect to its values, it may be desirable to pick ranges of values and use these to define operations, based on the foregoing criteria.

Since the operational profile used in test may not match that experienced in the field, it is distinguished as the test operational profile. In this case, of course, an appropriate transformation will be required to convert failure intensity experienced under this profile to what would occur in the field.

3.5.4 Recording occurrence probabilities: The most accurate way of determining the operational profile is to measure it in the field for a current or previous release; modifications must be made for new systems as noted in Section 3.5.5. Some effort may be required to develop recording software, but it may be possible to develop a generic recording routine that simply has to be interfaced with the application software. The recording process will usually add some overhead to the application program. As long as this overhead is not excessive, it may be feasible to collect data from the entire community.
However, if the overhead is large, it will probably be necessary to randomly sample the user community. The same guidelines and statistical theory apply here as are used for polling and market surveys; a sample of 30 or perhaps even fewer users may be satisfactory for generating an operational profile with acceptable accuracy.

Recording the complete input states experienced in field operation and using this information directly to drive testing is an acceptable approach if you need usage data only for testing. You save effort and time because you don't have to partition the input space, explicitly determine the operational profile, or generate test cases, except in the case of new operations.

Recording just the usage of operations is much simpler than recording the complete input states. It involves a much smaller volume of data, but you must partition the input space and you must generate test cases. The system must be instrumented to extract sufficient input variable data to identify the operations that are being executed. A counter is maintained for each operation in a small data set, and incremented at each use. Remember that an operational profile can be recorded in either explicit or implicit form.

3.5.5 Estimating occurrence probabilities: The operational profile must be estimated for new systems or for those operations that are added to old systems. The operational profile is developed from the functional profile by successive refinement. Functions are refined to operations by considering the operational design of the system (How do the functions map into runs?) and the detailing of the functions that has occurred during development.

A convenient way to capture the detailing is to create a list of events that initiate runs of the program. If there is a command list, examine the commands and their parameters. Look for parameters whose values or levels cause significant differences in processing. If the system is data-driven, look for those variables that cause important differences in processing. Next, develop a list of environmental variables that will significantly influence processing. The foregoing variables are key input variables; the possible combinations of them will determine the operations to be defined.

Each operation is now considered and a probability of occurrence is estimated for it. This is usually best done by an experienced systems engineer who has a thorough understanding of the businesses and the requirements of expected users of the system and a good comprehension of how it is expected that users will take advantage of the new functions being provided. Review of these estimates with experienced users is vital.

3.5.6 Example:
Let us return to the administrative mode example for the telecommunications switching system. Assume that in implementing this system mode, it was decided to implement the following command set:

- reloc (old location) (new location)
- add -s {user type} (location)
- remove (location)
- dirup

Change is being handled by removing the old record and adding a new one. The four commands all have different input variable structures, so we will define at least four operations.

As we consider the different parameters, we note that location does not essentially affect the nature of the processing. However, type of user does in that the features provided are substantially different for staff, secretaries, and managers. We could define three run categories in one operation. However, since we have usage information and it is markedly nonuniform, we will refine the add command into three different operations, obtaining:

- reloc (old location) (new location)
- add -s staff (location)
- add -s secretary (location)
- add -s manager (location)
- remove (location)
- dirup

Let us consider the various transactions. All except dirup account for 0.019 of the occurrence probability; dirup accounts for 0.001. Suppose that the 80 additions of service per month expected break down into 70 staff, 5 secretaries, and 5 managers. There will be 780 relocations and 20 changes of service each month, the latter representing promotion to manager. There are 70 removals proper and 20 removals created as the result of "change" functions, yielding a total of 90 removals. The part of the operational profile for the administrative mode now becomes:

| Table 8. Operational profile based on functional information only. |
|------------------------|------------------|-----------------|
| Transactions per month | Occurrence probability |
| reloc                  | 780              | 0.0153          |
| remove                | 90               | 0.0017          |
| add -s staff          | 70               | 0.0014          |
| dirup                  |                   | 0.0010          |
| add -s manager        | 25               | 0.0005          |
| add -s secretary      | 5                | 0.0001          |

We proceed in this fashion until we have accounted for all the ways the system can be employed. Now we must consider the possible expansion of the operation list.
to account for environmental inputs that could change the processing (and thus result in different failure behavior). We will assume that the environmental and functional variables are independent of each other, in order to simplify our example.

Recall that this system is being developed to handle both analog and digital telephones. The operational profile segment developed above will thus expand under this environmental factor into 11 operations (directory update is not affected by phone types). Let us exclude directory update and consider the part for analog telephones A, which will have occurrence probabilities that are 80% of those for all configurations.

There will usually be additional environmental conditions that affect the processing and must now be considered for the operational profile, even though they were not sufficiently major to be included in the development of the functional profile.

Assume that system load is one of these; i.e., if administrative functions are performed when the system is in an overload condition due to heavy communication traffic, processing may be affected: administrative requests might be queued, for example. Assume that this occurs 0.1% of the time.

We can now generate the operational profile for the segment of operations we are considering. All values in Table 8 (except the directory update operation) are first multiplied by 0.8 to give the occurrence probabilities for analog telephones. Then they are multiplied by 0.999 to obtain the occurrence probabilities for normal load, or by 0.001 to obtain the occurrence probabilities for overload.

Table 9. Operational profile based on functional and environmental information.

<table>
<thead>
<tr>
<th>Operation</th>
<th>Occurrence Probability (×10⁻⁴)</th>
</tr>
</thead>
<tbody>
<tr>
<td>reloc</td>
<td>12228</td>
</tr>
<tr>
<td>remove</td>
<td>1359</td>
</tr>
<tr>
<td>add - s staff</td>
<td>1119</td>
</tr>
<tr>
<td>add - s manager</td>
<td>400</td>
</tr>
<tr>
<td>add - s secretary</td>
<td>79.9</td>
</tr>
<tr>
<td>reloc</td>
<td>12.24</td>
</tr>
<tr>
<td>remove</td>
<td>1.36</td>
</tr>
<tr>
<td>add - s staff</td>
<td>1.12</td>
</tr>
<tr>
<td>add - s manager</td>
<td>0.40</td>
</tr>
<tr>
<td>add - s secretary</td>
<td>0.08</td>
</tr>
</tbody>
</table>

Note that some of the operations occur quite infrequently. One should seriously question whether it is really necessary to test all of them: for example, consider eliminating the add - s secretary under overload conditions.

4. Test Execution

As has been noted previously, the operational profile or profiles (often weighted by criticality) are used to select runs in test. If different profiles occur at different times, separate tests should be conducted. However, if they occur simultaneously in the field, testing should occur concurrently. This is because there can be interactions between system modes running simultaneously.

It may also be desirable to test with several alternative operational profiles to investigate sensitivity, that is, to represent the variation in usage that can occur among different installations of the system and the resulting differences in reliability measurements.

Software reliability measurement and models currently are based on testing with replacement. This means that after an operation or run type has been selected it is "replaced" in the population and can hence be reselected. In testing without replacement, the operation or run type can be selected only once. Note that testing can also be performed with limited replacement, where replacement occurs some finite and specified number of times.

It is not wise to select operations without replacement. Operations can be associated with multiple faults. Hence there is a high risk that an operation that fails or succeeds at one selection may show different behavior at another. In any case, the explicit operational profile can be used to select with or without replacement, but the implicit operational profile can be used only to select with replacement, because you would be failing to replace an input variable rather than an input state.

It is desirable to select run types without replacement because repetition provides no new information about failure behavior and is therefore inefficient. In theory, repeating the same run type can have some value when the program has changed since the last execution of that run type. However, it is really a waste in this situation also. There is a good chance that a change that causes a particular run type to fail will cause other run types in the operation to fail as well. By selecting another run type in the operation, you will probably find the new fault introduced and you may find another fault that was associated with the operation but was previously uncovered.

If run types are selected from the true astronomically large set of input states (not an artificially limited set of test cases, which we very strongly discourage because of the risk of missing important failure behavior), then selecting with replacement is in practice identical to selecting without replacement (and requires no bookkeeping overhead) because the probability of repeating a test case is infinitesimal.
Test selection within operations is probably best done by random choice, so that extraneous factors do not affect failure intensity reduction or reliability evaluation.

If interactions between the input variables that define the input states of the operation with respect to failure behavior can be shown to be limited, the set of input states that must be tested in the operation can be reduced by the use of orthogonal arrays (or perhaps some other experimental design). This technique requires that we define levels or ranges of values for each input variable. There is an implicit assumption that the values within a level have the property of homogeneity with respect to failure behavior.

Thus the level is associated with a subdomain or run category. The orthogonal array describes a partitioning of the operation in which only certain subdomains will be sampled. This is possible when higher order interactions do not occur; sampling from part of the domain of the operation is sufficient to cover possible failure behaviors. If all the higher order interactions were present, all the subdomains would have to be sampled (this corresponds to a full factorial experiment in the terminology of experimental design). The situation is represented in Figure 3. Input variables A, B, and C each have two levels, 1 and 2. There are no interactions of higher than second degree among the variables. Hence we only have to cover each pair of variable levels to activate all possible failure behaviors. Note then that only half the operation domain needs to be covered.

Little is currently known about the degree of interaction with respect to failure behavior among input variables in an operation. There is a certain locality of function and hence usually of implementation. It isn’t clear whether locality tends to produce more or less interaction. This topic, and the applicability of orthogonal arrays to this situation, are subjects of current research.

Since run categories are near homogeneous, and thus one run type is about as representative of failure behavior as any other, it does not matter that run types within a run category might have very different occurrence probabilities. You can select them uniformly from within the run category.

5. Criticality of Operations

In all of this discussion, we have not considered the criticality of operations. Operations can be classified in accordance with the value (increased revenue, reduced operating cost, or increased profit) of having them execute properly. Conversely, they can be classified in accordance with the impact (severity) of having them fail. In some cases, there can be different kinds of failures for an operation, and these can have different severities. We will use the average of the severities, weighted by relative probability, as the criticality of the operation. The types of impact in most common use as criteria are risk to human life and financial effect.

When considering financial impact, costs include loss of revenue, costs of recovery, costs of failure resolution or workaround, and loss of sales. Old operations can be more critical than new operations because failure disrupts existing capabilities that many rely on; at least part of this effect may be captured by higher probability of occurrence of these operations. On the other hand, new operations may be critical to the success of a new product that features them. They will often not have a very high initial estimated probability of occurrence. It may be necessary to assign them high criticality to reflect their importance.

One way to handle criticality is to classify operations by category of criticality (perhaps into four categories separated by orders of magnitude of impact) and generate operational profiles for each category. This implies selecting a failure intensity objective for each operational profile (category of criticality). All must be met to insure satisfactory operation of the system. Note that the reliability error that can be tolerated for the critical operation category will be almost always smaller than for the other categories. Hence, the effects of environmental input variables such as traffic and human error rate on critical operations will often be material and necessitate more thorough coverage of the input space during test.

Another alternative is to weight the operational profile by criticality and to use the weighted operational profile to drive test planning and testing. If the measure of criticality is financial, then the weighted operational profile represents the proportions of expected life cycle failure costs attributable to the various operations.

Conclusions

The systematic approach presented here is being used in a number of AT&T projects. It organizes and makes more efficient the process of determining the operational profile and the use of the information in guiding software development. In one large software development organization, it is being fully integrated into the standard software development process.

Acknowledgments

The author is indebted to Ed Shaughnessy of Software Quality Tools Corporation for suggesting the concept of the customer profile. Bill Everett suggested viewing the operational profile as evolving during the life cycle. Beto Avritzer, Mary Donnelly, Willa Ehrlich, George Estes, Bill Everett, Mario Garzia, Bliss Jensen, Bruce Juhlin,
References


Figure 1. Example Call Tree
Figure 2. Input Space Concepts

Figure 3. Example of subdomains of operation domain sampled by orthogonal array.

A, B, C: input variables with levels 1, 2

Orthogonal array:

\[
\begin{array}{ccc}
A_1 & B_1 & C_1 \\
A_1 & B_2 & C_2 \\
A_2 & B_1 & C_2 \\
A_2 & B_2 & C_1 \\
\end{array}
\]