Chapter 4
Operation-Based Test-Case Selection Methods

J. C. Huang
Department of Computer Science
University of Houston

Background
The code-based and specification-based test-case selection methods are mostly suitable for use by the software developers in the process of unit and integration testing in which the primary goal is to discover errors.

Background (continued)
In practice, the need to do testing may arise in a user organization as well. The main purpose there is to assess the reliability of a software system in hand so that proper actions can be taken to deal with procurement or deployment issues.

Background (continued)
In that case, the code-based and specification-based methods may become difficult to use because, first of all, the source code is generally not available to users.

Background (continued)
Secondly, users may not have access to all documents needed to apply the specification-based methods.

Background (continued)
Lastly, a finished software product is usually so complex that users would not be in a position to do component test, even if all necessary information is available.
Operational testing

A possible alternative is to select test cases based on the operational characteristics of the software. For instance, test cases can be selected based on the probability distribution of possible inputs in production runs.

Operational testing (continued)

A test case with a high probability in input distribution is significant because, if the program works incorrectly for that test case, it will have a greater impact on the reliability of that program than those with a low probability.

Content of this chapter

This chapter includes an example from LeBlanc's work that illustrates how to construct the operational scenarios of a software system, and how to use it in conjunction with historical data and expert opinion to estimate its input distribution.

Content of this chapter (continued)

Using the operational scenarios of a software system, we may also be able to assess the economical impact of a failure. The ability to make this assessment is of practical importance because it will allow us to estimate the benefit of a test in economic terms. A method for this purpose, proposed by Sherer, is also included in this chapter.

How much testing should be done?

If we can assess the economical impact of a failure, we can assess the benefit of a test. When we start testing a software system, the cost and the benefit of the test will increase with time initially. At some point in time, however, the benefit will start to level off because errors become fewer and far in-between, while the rate of increase in cost will remain about the same throughout.

How much testing should be done?

Hence there will be a point in time at which the net benefit (i.e., the benefit minus the cost) peaks, and further down, there will be another point at which it diminishes completely. Either of these can serve as the time to stop testing.
The theory

What follows is a theory developed by Sherer that can be used to assess the risk of failure due to faults in a software module, and to estimate the cost and benefit of testing.

Module risk

It is assumed that the software in question consists of a number of functional modules.

Based on the software operational scenarios, we can estimate module risk, which is defined to be the expected loss during a specified period that results from (failures caused by) faults in a module.

Module risk (continued)

Module risk, denoted by \( R(T) \), can be expressed as

\[
R(T) = X(T) \times F(T)
\]

where \( X(T) \) is the module exposure, which is defined to be the economic consequence of failure during operational time \( T \), and \( F(T) \) is the failure likelihood, the expected number of failures during operational time \( T \) that may result from faults in the module.

Module-risk assessment

The process of module-risk assessment consists of three components:

– assessment of external risk,
– estimation of module exposure \( X(T) \),
– and calculation of module failure likelihood \( F(T) \).

External risk assessment

External-risk assessment is to be done by studying the operational environment of the software.

No detail knowledge about the software is required in this step because the consequence of failure does not depend on the nature of software used—it depends on how the software is used, and the environment in which it operates.
Module exposure

The result of external-risk assessment is then used to estimate module exposure by relating individual modules and their potential faults to the external-failure modes and their economical consequences.

This can be done by analyzing each module's expected use and the cause-effect relations it embodies.

Failure likelihood

To estimate failure likelihood, it is necessary to adopt a suitable reliability model.

Use experience with similar modules to estimate required parameters initially.

As the module is tested and used, calibrate the parameters to reflect the module's expected use during the operational period.

External risk assessment

It begins with external-risk identification.

Study the software operational environment to identify potential hazards, scenarios that can lead to these hazards, and failure modes that contribute to all events involved in the scenarios.

Estimate external exposure for each hazard, independent of the software itself.

External risk assessment (continued)

It is important that users, not developers, be the source of information because they have the external view.

The risk assessor's objective is to develop a preliminary list of potential hazards.

External risk assessment (continued)

Once the hazards have been identified, the risk assessor develops accident scenarios, again through user interviews.

An accident scenario is the possible sequences of events resulting from system use that can lead to a hazard.

External risk assessment (continued)

In this process it is useful to focus on determining the relation between an invalid input and an accident scenario.
External risk assessment (continued)

Next, the loss is estimated. The magnitude of loss that may result from inappropriate actions is a function of the environment and the context in which the system operates.

External risk assessment (continued)

Finally, the consequence, \( C_j(T) \), of hazard \( j \) during time \( T \) is computed by weighting the loss estimated for each accident scenario by the likelihood of the accident scenario by the likelihood of the accident scenario, conditional upon software failure.

Module exposure

Module exposure is estimated by relating a module to external hazards and their consequences.

This can be done by analyzing the specification to relate a module's function and its use to the accident scenarios and failure modes uncovered in the external-risk assessment.

Module exposure (continued)

Start this process by identifying potential software failure modes in the accident scenarios—ways in which the software can contribute to the invalid information involved in the scenario. The invalid (or missing) information produced by the software is the key to the analysis process. Trace through the software to determine all modules that use or update any data related to this invalid information.

Module exposure (continued)

Next, study each module's function to determine its effects on each failure mode. This process identifies modules that, although not directly involved in the processing of critical data, perform functions whose failure may affect the processing of the critical data.

Module exposure (continued)

The objective of this analysis is to determine all hazards that may result due to faults in the module. Again, this analysis may be incomplete if all failure modes and modules are not identified.
Module exposure (continued)

Estimation should also be made about how each module will actually be used, because the degree of exposure is a function of expected use.

Thus, to assess module exposure, we must identify the system's functional uses and assign a probability distribution to those functions that reflects their expected operational use.

This distribution can be calibrated as new information about system use becomes available.

Module exposure (continued)

The system is then partitioned into module sets invoked for each use to estimate $p(U_i)$, the probability that a module will be used in a certain way (or will perform function $i$).

Once a module is found to be related to hazard $j$ for use $i$, a probability distribution for all hazards that may result from a given use of that module can be established. The notation $p(H_j/U_i)$ denotes the probability that hazard $j$ could result if the module has use $i$.

While it is reasonable to assign a uniform distribution to all hazards for a given module and use, in the absence of formal methods, user may want to pay more attention to more critical functions. In the following we assume that the distribution is assigned based on the relative severity of the consequence of failure.

Module exposure can now be expressed as

$$X(T) = \sum U_i \sum p(H_j/U_i) C_j(T)$$

where $p(U_i)$ is the probability that the module has use $i$, $p(H_j/U_i)$ is the probability that hazard $j$ could occur when module has use $i$, and $C_j(T)$ is the consequence of hazard $j$. 
Module-failure likelihood

The expected number of failures within an operation period can be estimated with a statistical model of reliability growth that incorporates estimates of the number of faults in the module, the probability of failure per unit execution time per module fault, and the expected use of the module during the operation period.

Module failure likelihood (continued)

John Musa's model is used in this analysis. This model gives the expected number of failures during a specified operation period to be

\[ F(T) = \mu (1 - \exp(-\theta t_0(T))) \]

Module failure likelihood (continued)

where \( \mu \) is the mean number of faults in the module, \( \theta \) is the probability of failure per unit-execution time per fault in the module, and \( t_0(T) \) is the execution time of a module during operational time \( T \) (\( t_0 \) is measured in execution time and \( T \) is measured in calendar time).

Module failure likelihood (continued)

Values of \( \mu \), \( \theta \), and \( t_0(T) \), have to be estimated.

Information needed to estimate \( \mu \) and \( \theta \) can be obtained by analyzing the number of faults found in similar modules of similar size and operating characteristics developed by similar personnel. The execution time of the module during a calendar period can be estimated from an analysis of expected use.

Module risk

For each module, risk is initially assessed using these estimates of module exposure and the expected number of failures due to faults in the module within a period of operation time.

Module risk (continued)

Then, as additional information is gained over time, these assessments should be continually updated. As testing begins, the time of failure and the location of faults leading to these failures should be recorded.
Module risk (continued)

Use an appropriate statistical technique to adjust the initial estimates with failure information gained during testing. The updated parameters are then used to provide new estimates of software reliability, based on expected system use.

Cost and benefits

An expression can be derived to compute the optimum amount of testing, the amount of testing that maximizes the net benefit from testing.

The benefit of testing is measured as the reduction in risk due to additional testing. The cost of testing is a function of the time spent testing and the number of faults to be corrected.

An assumption

It is assumed here that testing will generally decrease the risk of failure, and that there will be no gain in releasing insufficiently tested software early. Otherwise, the benefit from testing will be reduced by an estimate of the economic gain of early release.

The net benefit

The net benefit, $NB(t_\gamma)$, resulting from testing a module for time $t_\gamma$, can be expressed as

$$NB(t_\gamma) = \Delta R(T, t_\gamma) - TC(t_\gamma)$$

where $t_\gamma$ is the test time measured in computer time, $\Delta R(T, t_\gamma)$ is the change in failure risk during operational period $T$ when the module is tested for time $t_\gamma$, and $TC(t_\gamma)$ is the total cost of testing for time $t_\gamma$.

How much testing?

When deciding how much testing is economically appropriate, one should consider each module's exposure level. Although it will generally be more beneficial to test modules with higher exposure, additional testing will generally not change the exposure. It will only reduce the likelihood of failure. That reduction in the expected number of failures due to testing can be measured.

How to allocate test time?

Equal allocation of test time to different modules may not produce equal reductions in the expected number of failures. First, the reduction in failure intensity, or rate of change in the number of failures per unit of execution time, is a function of the number of faults in the module, the probability that a fault will cause a failure, and the amount of prior testing of the module.
Effect of testing

However, the effect of testing on failure intensity is not the only factor affecting the reduction in the expected number of failures. The frequency of the module's operational use should also be considered in determining the resulting effect of a change in failure intensity on risk.

Effect of testing (continued)

Although an expenditure of test time can cause a substantial reduction in a module's failure intensity, the magnitude of the benefit is a function of the module's degree of use, which is a function of the magnitude of $t_0(T)$ for the operational period.

Reduction in risk

Although testing will generally decrease failure likelihood, to measure its economic benefit one must adjust the decrease in the expected number of failures by the magnitude of the exposure. The resulting reduction in risk due to testing can be expressed as

$$\Delta R(T, t_\gamma) = X(T) \mu (1 - \exp(-\theta t_\gamma(T))) + \exp(-\theta (t_\gamma + t_0(T)) - \exp(-\theta t_\gamma))$$

Cost of testing

The cost of testing includes that of both machine and personnel time to find and fix errors. This includes test planning and development as well as the costs incurred in running test cases and analyzing results. These costs are assumed to increase linearly with the amount of time spent in testing.

Cost of testing

In addition, the costs to prepare failure reports, find and remove faults, and retest, ensuring that faults no longer exist, are directly related to the number of failures.

It is assumed here that the cost of testing is a function only of the time spent testing and the number of failures detected and removed.

The total cost of testing

The total costs can thus be expressed as

$$TC(t_\gamma) = K_1 \mu (1 - \exp(-\theta t_\gamma)) + K_2 t_\gamma$$

where $K_1$ is the cost per failure, and $K_2$ is the cost per unit of machine time spent testing.
The net benefit

Therefore, the net benefit of testing becomes

\[ NB(t_γ) = \Delta R(T, t_γ) - TC(t_γ) = X(T) \mu (1 - \exp(-\theta t_0(T))) + \exp(-\theta t_γ) - \exp(-\theta t_γ)) - K_1 \mu (1 - \exp(-\theta t_0(T))) + K_2 t_γ \]

Optimal test time

While the cost of testing increases linearly with \( t_γ \) the increase in benefit from risk reduction levels off beyond a certain point in \( t_γ \). That means the net benefit from testing will initially increase with \( t_γ \), peak at \( t_γ^* \), and decrease beyond that. The value of \( t_γ^* \) can be derived from this equation by differentiation as

\[ t_γ^* = -\ln(K_2/(\theta \mu (X(T)(1 - \exp(-\theta t_0(T)))) - K_1))/\theta \]

The net-benefit curve

An example application

The software system that will be used as an illustrative example is a planning and scheduling system, known as the CPS, used by the NASA-JSC for the International Space Station (ISS) program.

The main operation of this system is to provide a valid schedule of daily on-orbit astronaut activities called the Short Term Plan (STP).

Description of the system

The CPS is a large-scale software system, so there are many modules used to build a schedule. However, the heart of the system is the automatic scheduling engine. The original engine used a five-rule-set model and could only accomplish forward path scheduling.

Description of the system (continued)

The latest version of the software incorporates an engine having forward, backward, middle and flexible scheduling and resource leveling methods that are used to automatically schedule activities and activity sequences according to certain constraints.

The number of possible combinations of scheduling methods and activity constraints are prohibitively large.