To the Graduate Council:

I am submitting herewith a thesis written by Jenny-Ann M. Morales entitled "Test Planning for Software reuse" I have examined the final copy of this thesis for form and content and recommend that it be accepted in partial fulfillment of the requirements for the degree of Master of Science, with a major in Computer Science.

Jesse Poore, Major Professor

We have read this thesis
and recommend its acceptance.

Accepted for the Council:

Associate Vice Chancellor and
Dean of the Graduate School

TEST PLANNING FOR SOFTWARE REUSE

A Thesis

Presented for the

Master of Science

The University of Tennessee, Knoxville

Jenny-Ann Marisol Morales

May 1997

Dedication

To Fernando for all his love and support.

To Andrés for all the happiness he brings to my life.

Acknowledgements

First I want to thank my advisor, Dr. Jesse Poore for suggesting the problem and for all the guidance and learning opportunities he has provided during the time I was part of his research group.

I am grateful to Dr. Michael Berry and Dr. David Straight for their service on my committee and for the review of this thesis.

I wish to thank Dr. Carmen Trammell and Dr. Jesse Poore for the opportunity to work in the Software Quality Research Laboratory. A special thanks goes to Dr. Trammell for her consideration and support.

I appreciate the encouragement I have received from all the members of the Software Quality Research Laboratory.

Thanks to Michael Muir who provided the usage model used in the simulation in this thesis.

Finally, I thank my family for their continuous encouragement and support. Dulita thanks for coming to help me with Andrés. Fernando thanks for your love and patience. Andrés thanks just for being here.

Abstract

This thesis describes a methodology to support the decision to reuse a software asset from a repository, using the information of previous testing to best advantage. The context of this work is statistical testing with usage models represented as Markov chains.

The methodology provides techniques to compare a new usage chain with other usage chains in the repository to determine the similarity between the prospective new use and the testing history. To determine if enough testing was performed, the method compares the prospective new use with the testing chains in the repository, and also analyzes the reliability and confidence measures when no failures are found.

The methodology also describes techniques to combine previous uses of the software in such a way as to have more information about the behavior of the software in general as well as for a particular environment. A technique to disregard the irrelevant (redundant) testing experience is also provided so that the stopping criteria in testing are met sooner than by testing exclusively with the new use, or by including all the testing history.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

A software component might be used in many different environments, therefore we need methods to assess its fitness for each different environment. Formal testing methods provide us with tools to evaluate the reliability and confidence of software in a given environment. Test budgets are usually restricted in time and money, therefore it is important to find the most efficient way to qualify a component.

Testing software is a hard problem because it is impossible to test every single input sequence applicable to the software. Testing methods should identify an appropriate subset of all possible input sequences and by analyzing test results on the subset infer the quality of the software. In other words testing methods should generate statistically valid data to infer the quality of the software.

Some testing methods in common use are partition testing, path testing and statistical testing. In partition testing the standard approach is to divide the input domain into classes and then to select test data from each class of the partition. The idea is that all the elements in a partition are equivalent for the purpose of testing. If the result of a single element in

the class is correct then all the elements in that class are assumed to be correct. On the other hand, if one element of a class exhibits a failure then so will all elements in that class [9].

In path testing the idea is to run as many test cases as necessary to test every path through the code. The problem with path testing is that it does not include loops and cycles.

Statistical software testing, in which the test cases are randomly generated according to a probability distribution, will find failures in generally the same order that a user would find them, if the distribution is an accurate characterization of field use. Also, such testing generates data that can be analyzed with standard statistical methods.

Several studies support the idea that statistical testing is the most efficient way to test software. Duran and Ntafos [3] showed that random testing is generally as effective as partition testing and it is much less expensive. Duran and Wiorkowski [4] argued that path testing is more expensive than statistical testing, and that it is very difficult to show that every single path has been covered. They also showed that path testing is not better at finding failures than random testing. Mills [8] indicated that testing with human-selected test cases produces only anecdotal evidence. If the goal in testing is to certify the reliability of the software, statistical testing is the most appropriate option because one can infer information from the actual testing performed. Research and practice have established that "apart form certain anecdotal tests that are justified on legal, moral, ethical or political grounds, the most effective way to spend the test budget is on statistical testing based on a usage model"[13].

This work presents a methodology to reduce the amount of statistical testing necessary to support a decision to reuse a software asset. In current practice, when a user wants to reuse a software component, testing is usually done without quantitatively considering results from previous testing experiences. This is not a very efficient way to spend the testing budget especially when the prospective new use of the software is similar to some previous usage.

We would like to consider previous testing experiences in a formal way to reduce the new testing necessary to satisfy quality requirements of a new user. In the case of statistical testing these requirements are usually expressed as reliability and confidence levels of the software, and the environment of use is described by a usage model.

1.1 Statistical Testing

The main elements in statistical testing are:

- Build the Markov chain usage model. The model can be constructed even before the code is written because it is based on functional specifications of the software.
- Generate random test sequences. Once the usage model is constructed we can generate as many sequences as we want.
- Run the test cases on the software under test and record the results in the testing chain.
- Record the entire testing record in the repository.

1.1.1 Usage Models

Software testing can be treated as a stochastic process, and the usage of the software can be modeled by a finite state, discrete parameter, time homogeneous, irreducible Markov chain [14,15]. The structure of the Markov chain reflects what the software can do, and the probability distribution associated with the Markov chain represents the expected use of the software. This Markov chain is called a usage model or usage chain.

A software usage model characterizes the intended use of the software in the intended environment [13]. Modeling the use of the software with Markov chains is a good choice because there is a well known body of theoretical results associated with Markov chains which allows a deep characterization of the software testing process. They are also mathematically tractable and they have proved to be effective in practice.

The transition probabilities among states can be expressed as constraints based on field data, informed assumptions about the expected use, uniformly where no information about the use of the software is available. Specific probability values are then determined from the system of constraints. Walton [12] proposed a method for the representation of usage models as constraints and the optimization of the transition probabilities relative to testing objectives.

The usage model can be used to plan the testing process even before the code is written because it is based on functional and usage specification. It also can be used to generate statistically correct samples of test cases.

1.1.2 Testing Sequences

The usage model is used to generate sequences (test cases) that are applied to the software in order to certify it. A test case is a realization of the usage chain that begins with the invocation state and ends with the first occurrence of the termination state. The sequences are generated by random walks (with a random number generator) on the usage model. Once the usage model is created, any number of test cases can be generated automatically.

1.1.3 Testing Chains

The testing chain records the result of applying the test cases to the software and it is used to determine software quality measures such as reliability and confidence, and also to support a decision to stop testing.

The testing chain is created by initializing a frequency model F with the same states and arcs of the usage model. The labels for the arcs in F are frequencies instead of probabilities, and are initially set to zero or to one. All frequencies are set to zero when arc coverage is expected soon in the testing process, otherwise all frequencies are set to one to permit computation of the Kullback discriminant (see Section 3.1.2). Every time a test event applied to the software traverses an arc, the frequency count is incremented by one. Failures are incorporated in the model by creating a new failure state for each failure. If a failure is observed when passing from state i in some test sequence, a new state f_i is cre-

ated with an arc from i to f_i with frequency count of one. The exit arc from f_i follows the execution of the software; that is, if the failure prevents continued testing of the sequence, the exit arc goes to termination, otherwise the exit arc goes to the next state in the model. It is assumed the failures are detected unambiguously. The testing chain is created by normalizing the frequency chain F.

1.2 Reuse

The repository is presumed to contain all the information about previous testing of the software, that is, the usage model, the sequences with which the software was tested, and the result of the testing experience. When a user wants to reuse a software component he should be able to consider all previous testing information available in order to reduce the amount of new testing necessary to certify the software for the new environment.

In the context of this work we use the term *reuse* to indicate changes only in the transition probabilities of the usage model. The software component will be taken from a repository, and tested as it is, with no changes in code.

1.3 Reuse Questions

The objective of this work is to investigate mathematical techniques to determine how much new testing (if any) is necessary to satisfy the requirements of reliability and confi-

dence expected for a new use of the software and at the same time reduce the amount of testing necessary. To reduce the amount of additional testing necessary we want to use all the previous testing experiences in order to avoid redundant testing. In this context we want to answer the followings questions:

- How can we compare one usage model with another, or with a testing model?
- What is the likelihood that a set of sequences was generated from a specific usage model?
- How can we combine testing records in order to get stronger conclusions about reliability and confidence of the software without performing extensive new testing?
- If we decide that new testing is necessary, what usage model is the most efficient to generate the new test cases, and how much testing is necessary?

1.4 Outline and Overview

This work presents a set of mathematical operations to be performed on usage models and testing chains in order to help decide the testing strategy when a user wants to reuse a software component from a repository. The idea is to use the information of previous testing experiences to best advantage.

The process begins with the selection of the usage model. To select this model we provide techniques to compare usage models against usage models and also usage models against testing chains. The comparison among usage models is used to determine the clos-

est model, and the comparison between usage models and testing chains is used to determine if enough testing was already performed.

We provide techniques to combine usage models in order to compile all the testing experience into a single model so that we can infer the cumulative reliability and confidence of the software taken in consideration of all the testing experiences.

In order to reduce the amount of testing required to certify a component for use in a new environment, we provide a technique to disregard irrelevant experience so that recorded experience plus the new tests will satisfy the stopping criteria sooner than if we test exclusively with the new model.

This document is organized as follows. Chapter 2 describes the mathematical operations and presents the properties required of each one. Chapter 3 presents candidates for each operation and delineates the methodology to support the reuse of an asset. Chapter 4 treats reliability and confidence measures. Chapter 5 presents simulation results for the technique proposed in Chapter 3. Chapter 6 provides a summary of the work and the conclusions.

Chapter 2

Basic Operations to Answer Reuse Questions

In this chapter we describe a set of operations over testing records and the required properties of each operation. These operations will be used to answer the reuse questions posed in Section 1.3.

2.1 Testing Records

A testing history is a collection of testing records (one or more), each of which contains the information related to the testing experience for a software item. The structure of a testing record is: $< U, S, F, T >$ where:

- U is the usage chain represented as a transition matrix. This matrix contains the structure of the model and the transition probabilities associated with a particular usage environment of the software.
- S is the set of sequences or test cases actually used to test the software.

- F is the failure log of all failures found during testing.
- T is the Testing chain produced from U, S and F.

The reliability, confidence, MTTF (mean-time-to failure), state coverage, arc coverage, and other measures, can be generated with computations over the elements of the testing record. When the history contains more than one testing record, they will be named and indexed.

2.2 Operations Over Testing Records

In order to outline a method that supports the decision to reuse an asset from a repository we need to define a set of basic operations to be performed with testing records. In this section we will define the following operations:

- Comparison of stochastic matrices
- Likelihood that a sequence would be generated by a given model
- Combining testing records
- Assessing the difference between testing records

2.2.1 Comparison of Stochastic Matrices

The first step in the process of deciding to reuse an asset is to compare the usage model of the proposed usage with those in the repository to assess the similarity to prior testing.

If there is one similar to the propose usage, we will take that model either as a reference for reliability and confidence or as starting point in further testing. After this initial comparison we must be able to determine if enough testing was already performed or not. To determine if enough testing was performed we will compare the intended use with all the testing chains available. A schema of this process if shown in Figure 2.1.

The comparison will be defined as a function $C(A,B)$, which is the distance from record A to record B, and must have the following properties:

i. $C(A,B) = 0$ iff $A = B$

ii. $C(A,B) = C(B,A)$ (symmetrical)

iii. $C(A,C) \leq C(A,B) + C(B,C)$ (triangle inequality)

2.2.2 Sequences and Models

We need to compute the likelihood that a set of sequences would be generated by a specific usage model, that is P(SequencelUsageModel). This information can be used to determine the relevance of a sequence to a new model, and it will be useful to determine if enough testing was already performed. We can use this information in two ways:

- To see if some of the sequences generated for previous testing would be likely generated by the new model. In this case we can select the sequences relevant to the new usage.

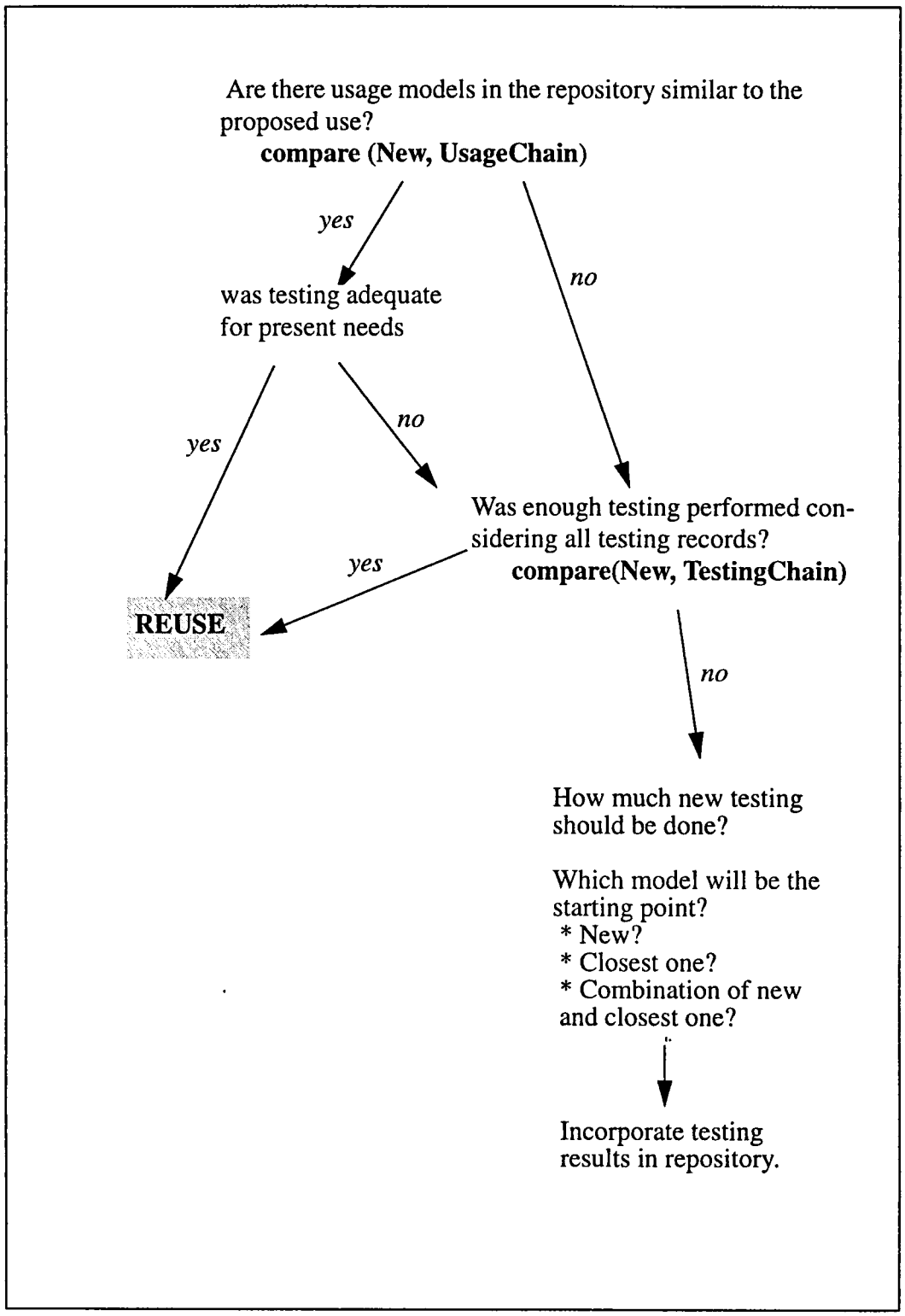

Figure 2.1: Comparison of Testing Records

- To see if the sequences generated for the new model would be likely generated for some of the previous models (the generation of statistical test cases is easy and inexpensive, and for this experiment we don't need to run the test cases).

2.2.3 Combining Testing Records

We need to combine testing records in such a way that we have more information about the reliability and confidence in the software, and also more information about some specific testing situation.

Our purpose is to reduce testing. We want to combine testing records in the following circumstances:

- When no single previous testing is similar to the new use of the software, we want to check if some combination of two or more of them is close to our proposed usage. In that way, testing for a new environment will use as a starting point the closest possible model and reach requirements of reliability and confidence as inexpensively as possible.
- With the combination of all the testing records we can estimate reliability of the software with greater confidence. In this way we have more information about the behavior of the software in general as well as the behavior for a particular environment.

Let A and B be two testing records with the following structure: $A = < U_a, S_a, F_a, T_a>$ and $B = <U_b, S_b, F_b, T_b>$. We want to define an operator $\oplus$ that combines testing records

to produce a new (synthetic) testing record, while respecting certain properties. We will analyze the information shown in Figure 2.2.

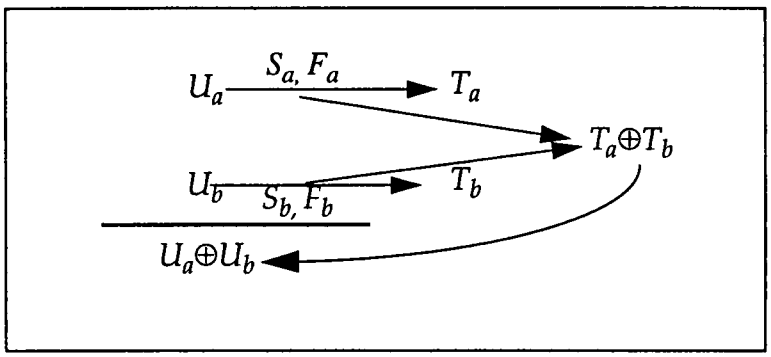

Figure 2.2: Combining Testing Records

The operator ⊕ should combine the elements in the testing records point by point, so that for testing records $A = < U_a, S_a, F_a, T_a>$ and $B =<U_b, S_b, F_b, T_b>$

$$A \oplus B = \begin{cases} U_a \oplus U_b \\ S_a \oplus S_b \\ F_a \oplus F_b \\ T_a \oplus T_b \end{cases}$$

The combination should have at least the following properties:

- $A \oplus A = A$, Idempotent
- $A \oplus B = B \oplus A$, Commutative
- $A \oplus (B \oplus C) = (A \oplus B) \oplus C$, Associative

$U_a \oplus U_b$ is a combination of the usage models. It will be checked for similarity with proposed new usages of the software.

$T_a \oplus T_b$ is a new testing chain constructed from both sets of sequences and failure logs. This chain will be used to represent the combined testing experience of the software. It summarizes the information of all testing of the software.

The arrow from $T_a \oplus T_b$ to $U_a \oplus U_b$ means that we can get a usage chain for the cumulative testing experience by transforming the frequency testing chain into a Markov chain.

2.2.4 Usage Model Difference

Let *A* be a testing record in the repository and let *B* represent a prospective use of the software. If we consider the experience obtained with the testing of *A*, we will usually have some irrelevant experience, and some experience directly relevant to situation *B*. Some aspects of *B* might have been under addressed by *A* (Figure 2.3).

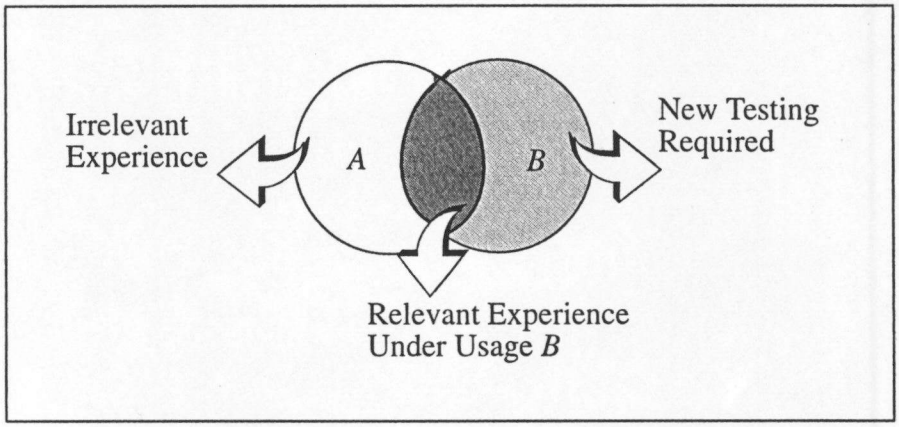

Figure 2.3: Relevancy of Previous Testing Experiences

If analysis of the previous testing experiences shows that more testing is necessary we need to determine which Markov chain is more effective in generating the additional test

cases. The method to find the most appropriate Markov chain to generate the additional test cases will depend on the similarity between the previous testing experience and the new usage of the software. It also depends on the amount of testing already performed.

We need to consider two issues:

- How to ignore irrelevant experience
- How to compensate for missing experience

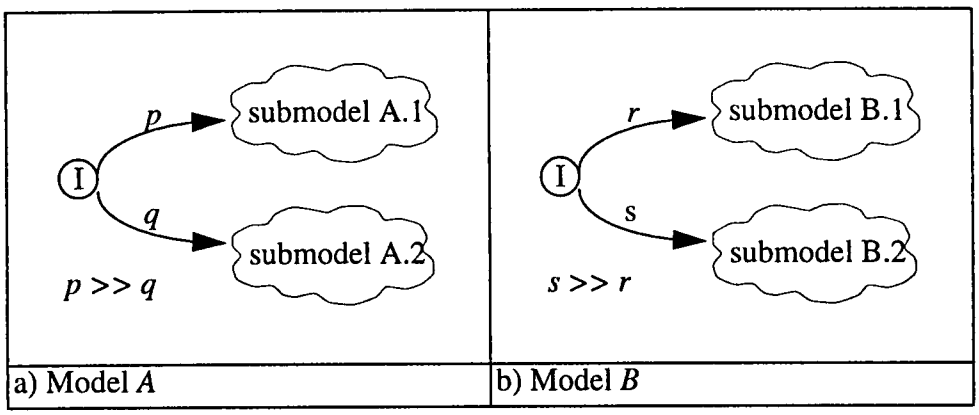

Figure 2.4: Examples of Testing Situations

Figure 2.4 shows two testing situations. Let us say that model *A* was already tested and the results are in the repository. Model B is the proposed new usage of the software. Given that *p* is much bigger than *q*, submodel A.1 was heavily tested and that experience should not count against us in the evaluation of reliability and confidence for usage *B*, therefore we need to find a method to "ignore" the redundant experience for usage *B* when comparing B to A. On the other hand, given that *s* is much bigger than *r* submodel B.2 needs to be tested more than submodel B.1. Therefore we need to compensate for the missing experience. In addition to that, we want to use previous testing experience in such a way that the

previous experience plus the new tests will satisfy stopping criteria sooner than if we test exclusively with model *B* (Figure 2.5).

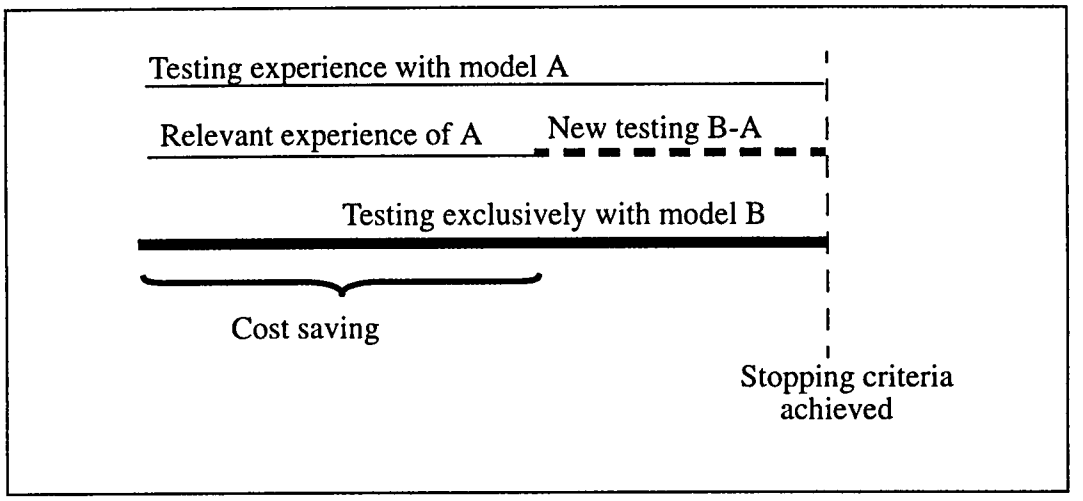

Figure 2.5: Cost Saving in Testing

2.3 Outline of A Method to Support Reuse

A new user of an asset must decide if the testing already performed by other users satisfies his own requirements. The user will have all the individual testing records and the combinations of all records to evaluate. The first step is to find out if some of the previous experience is similar to the proposed one.

The next step in the process is to determine if enough testing was already performed, either in an individual testing situation or some combination of the experiences. If the user decides to carry out more testing all the relevant testing already done would be used. To do that the user must select the sequences that are applicable and supplement them with new

sequences. In this way the user will achieve the stopping criteria sooner than by testing exclusively with test cases generated with the new usage model. The results of the new testing should be included in the repository as an individual testing record.

Chapter 3

Candidates for Basic Operations

In this chapter we define a set of candidate operations on testing records that have the desired properties as explained in Chapter 2, and describe some circumstances in which they can be used. The best operator depends on the specific situation and all of them allow us to decide in an informed way about the possibility of reuse of an asset from a repository. Experience in real applications will show us the range of values appropriate for each operator.

3.1 Comparison of Stochastic Matrices

Let A and B be two stochastic (transition) matrices (or two probability distribution vectors) of the same structure. Some comparison methods for stochastic matrices are:

- Euclidean distance
- Kullback discriminant

- Matrix norms
- Bhattacharyya distance

3.1.1 Euclidean Distance

The Euclidean distance is used as a primary source to determine the differences between probability distributions. Larger numbers for this measure reflect greater differences between the distributions.

The Euclidean distance is computed as follows:

$$d(A,B) = \sqrt{\sum_i \sum_j (A_{ij} - B_{ij})^2}$$

This distance is zero only when the distributions are equal. This measure satisfies all the properties of a distance function.

3.1.2 Kullback Discriminant

The discriminant is a measure of the likelihood that two usage profiles will generate a similar set of typical sequences. The smaller the value of $D(A,B)$ the greater the likelihood of generating similar sequences. If two usage profiles generate similar sequences then we expect similar outcomes using the testing information of either one. The discriminant is computed by

$$\mathrm{D(A,B)} = \sum_{i,j} \pi_i A_{ij} \log \frac{A_{ij}}{B_{ij}}$$

where π is the stationary distribution vector.

When A_{ij} is zero, we just skip that value from the computation because in that case a transition from state i to state j is not allowed, therefore it does not provide information. If $A_{ij} \neq 0$ and $B_{ij} = 0$, the Kullback discriminant is not defined. The Kullback discriminant has a valid value only when the chains have the same structure. When we compare a usage chain against a testing chain we get a valid value only when arc coverage is achieved in the testing chain. For some models it is necessary to run lots of test cases (more than you can afford to run) before achieving arc coverage. In this work we use a frequency chain initialized to ones instead of zeros. In this way we always have a valid value for the Kullback discriminant.

Notice that the Kullback discriminant is not symmetrical. The symmetrized version of the discriminant is the average between *D(A,B)* and *D(B,A)*, which measures the difficulty of discriminating between *A* and *B* [6].

3.1.3 Matrix Norms

Matrix norms provide a measure of the distance on the space of matrices [5].

For $A,B \in \Re^{m \times n}$ and $\alpha \in \Re$, $f:\Re^{m \times n} \rightarrow \Re$ is a matrix norm if and only if

i. $f(A) \geq 0$ ($f(A) = 0$ iff $A = 0$)

ii. $f(A+B) \leq f(A) + f(B)$

iii. $f(\alpha A) = |\alpha| f(A)$

The norm of matrix A is denoted $\|A\|$. The most frequently used matrix norms are the Frobenius Norm and the p-norm, specially when $p = 2$. The Frobenius Norm is defined as

$$\|A\|_F = \sqrt{\sum_{i=1}^{m}\sum_{j=1}^{n}|a_{ij}|^2} \text{ or } \|A\|_F = \sqrt{\sum \text{Diag}\,(A^T A)}$$

The 2-Norm is the square root of the largest eigenvalue of $A^T A$.

Given that we are interested in differences between matrices we will compute the norms for A-B and in that case the Frobenius Norm is the same than Euclidean Distance. The norms of *(A-B)* will be zero if and only if $A = B$.

3.1.4 Bhattacharyya Distance

The Bhattacharyya Distance [11] is a measure of the separability between density functions. It is defined as:

$$B = -\ln\left(\sum\sum\sqrt{p_1(y)\,p_2(y)}\right)$$

where $p_1(y)$ and $p_2(y)$ are probability density functions.

This measure can be applied to the stationary distribution and in this case we have a measure of the similarity of the chains in the long run.

3.2 Sequences and Models

Let testing record $A = < U_a, S_a, F_a, T_a >$ be the testing experience, and D be the new prospective use. Therefore, we have $D = < U_d, S_d, -, -> $ since no testing has been done with S_d. We need to determine the relationship between models and sequences.

3.2.1 New Models versus Test Experience

To decide how much of the previous testing is relevant to the new environment we need to compute the relationship between the testing history and the new model. Let A be the testing record and D be the new model. We need to compute the relationship between $P(D|S_a)$ and $P(A|S_a)$. Notice that A can be a combination of previous experiences or an individual testing record.

Applying Bayes's Theorem ($P(X|Y) = \frac{P(Y|X)\,P(X)}{P(Y)}$) we have

$$P(D|S_a) = \frac{P(S_a|D)\,P(D)}{P(S_a)} \quad \text{and} \quad P(A|Sa) = \frac{P(S_a|A)\,P(A)}{P(S_a)}$$

$P(S_a)$ is a (non zero) scaling factor, therefore it can be removed from consideration.

P(Sequence|Model) can be computed by multiplying the probabilities of each event in the sequence using the stochastic matrix associated with the respective model. Notice that as a sequence grows longer, its probability of being generated becomes lower.

P(Model) represents the probability of choosing that model as starting point in the testing experience for *D*. Since this is totally under the analyst's control we can say they are equally likely and remove this factor from the comparison.

If $P(S_a|D) \geq P(S_a|A)$ then we have $P(D|S_a) \geq P(A|S_a)$ and can say that the testing experience generated for *A* will be relevant for the new use.

3.2.2 New Sequences versus Previous Models

Another way to assess the similarity between the prospective use and the testing history is to compute the probability that sequences generated by the new model could also be generated by some of the previous uses. In this case we need to find out the relationship between $P(S_d|D)$ and $P(S_d|A)$. If $P(S_d|A) \geq P(S_d|D)$ then S_d is at least as likely to come from environments *A* as *D*.

3.3 Combining Testing Records

We need to combine each element of the testing record, but the only point of interest is the usage chain because sequences and failure logs can be easily combined with the union (with repetition) of their elements and the generation of the testing chain is accomplished with the incorporation of all sequences and failures logs in a single chain. We will focus now on the combination of usage chains.

3.3.1 Combining Usage Models by Mean Values

The simplest way to combine usage chains, and Markov chains in general, is with the mean of the values, that is:

$$U_a \oplus U_b = \frac{U_a(i,j) + U_b(i,j)}{2} \text{ for all } i \text{ and } j$$

Clearly this way of combining usage chains has the expected properties.

3.3.2 Combining Usage Models by Entrywise Multiplication

Another way to combine usage chains is with the following algorithm:

Step 1 : Compute matrix C such that $C(i,j) = U_a(i,j) * U_b(i,j)$. Notice that this matrix is not row stochastic and must be normalized.

Step 2 : Compute the square root of each entry in matrix C *(i.e., $Q = sqrt(C)$).*

Step 3 : Get the Perron-Frobenious eigenvector V, associated with the maximum eigenvalue Y.

Step 4 : Construct a diagonal matrix X with element os the Perron-Frobenious eigenvector, i.e., $X_{ii} = V_i$

Step 5 : Compute the normalization of the matrix with $U_a \oplus U_b = \frac{1}{Y} \times \text{inv}(X \times Q \times X)$.

For the idempotent property ($A \oplus A = A$), in step 1 of the algorithm we multiply each element by itself and in step 2 we get the square root, therefore we are again working with

the original matrix. This matrix is normalized, therefore steps 3 to 5 should have no effect on it.

The commutative property ($A \oplus B = B \oplus A$) is kept because multiplication is commutative in step 1, and all the computations depend on the result of step 1.The associative property ($A \oplus (B \oplus C) = (A \oplus B) \oplus C$) is also preserved.

3.3.3 Cumulative Experience

The usage chain of the cumulative experience can be obtain by updating the frequency testing chain with all the sequences available and transforming frequencies to probability estimates.

This way of combining is the best starting point for the rest of the process because it has all the testing experience summarized in a single chain.

The idempotent property is kept. Every arc in the frequency matrix will have twice as many values and we divide by the sum of them to obtain the original probability value. Commutativity and associativity are also maintained since the order in which the sequences are incorporated to the frequency chain does not matter.

3.4 Usage Model Difference

Once we get the closest possible model as starting point for testing, we need to solve two problems:

- How to ignore irrelevant experience from distance considerations
- How to compensate for missing experience

Let us say that A is the closest model with testing experience in the repository and D is the new intended use. Notice that A can be a particular testing situation or the combination of previous experiences.

3.4.1 Ignoring Irrelevant (Redundant) Experience

To remove irrelevant (redundant) experience we will use a schema of picking and choosing sequences associated with some testing experience.

One way to remove irrelevant experience is to consider only those sequences that have a greater or equal probability of being generated by the new model than by the model in the repository. Let S_{ai} be the ith sequence generated by model A. We will consider S_{ai} iff $P(S_{ai}|D) \geq P(S_{ai}|A)$. The problem with this method is that we cannot assure arc coverage even if in model A we had arc coverage, because we might remove sequences critical to coverage.

A more efficient way to remove irrelevant experience is to consider any sequence that incorporates new information to the testing chain, that is any sequence that traverse an arc

not already traversed, and for those sequences that do not introduce new information we use the previous scheme, and incorporate it if the probability in the new model is greater or equal to the probability in the old model.

If we use a testing chain initialized to ones instead of zeros the former approach is better because we already have arc coverage and in this case we consider only test cases that contribute more to the new environment. In addition to that, this approach requires less computation which could be a significant saving for large models with lots of test cases.

3.4.2 Compensating for Missing Experience

After eliminating the irrelevant (redundant) experience with the method explained above, we are in the closest possible situation between the prospective use and the testing chain generated considering the previous experience. The direct method to compensate for missing experiences and preserve all statistical properties is to simply generate and run new random test cases from the prospective new model. More efficient methods may be found in the future.

3.5 A Method to Support Reuse

Let us say that we have three testing records $A = < U_a, S_a, F_a, T_a>$, $B =<U_b, S_b, F_b, T_b>$ and $C = < U_c, S_c, F_c, T_c>$ in the repository, $X$is the new intended use, therefore we have $X = < U_x, S_x, -, ->$.

We propose the following method to support the decision for reusing an asset from a repository.

Step 1 : Decide which is the closest usage model to be used as starting point. To do this we need to create Table 3.1 and analyze its information

Table 3.1: Evaluation for Model X

	D(X,•)	*D(•,X)*	*mean*	*d(X,•)*	*N2(X,•)*	*B(X,•)*
A						
B						
C						
A⊕B						
A⊕C						
B⊕C						
A⊕B⊕C						

where:

• is the model in the repository

D(X, •) and *D(•, X)* are the Kullback discriminants

mean is the mean value between *D(X, •)* and *D(•, X)* (symmetrized version of the Kullback discriminant).

$d(X, \bullet)$ is the Euclidean distance (which is equal to the Frobenius Norm)

$N2(X,\bullet)$ is the norm-2

$B(X, \bullet)$ is the Bhattacharyya distance

All the computations follow the same trend. The most accurate is the Kullback Discriminant because it give us a difference of ensemble characteristics of the chains and its ability to generate a similar set of test cases, but it requires significantly more computation.

After analyzing the values in Table 3.1 we should be able to indicate which model is the closest one to the intended use. If no model is close enough we should choose the combination of all experiences ($A \oplus B \oplus C$).

We must also keep in mind that the size of the model is very important in the comparison of values. That is, a value that is bad for a small model could be good enough for a larger model.

Step 2 : After choosing the closest model we need to determine if enough testing was already performed. To do that we will compute $D(X,T_m)$, that is the Kullback discriminant between the prospective use and the testing chain for the closest model in the repository. We will also compute Kullback discriminate for $D(X,T_a \oplus T_b \oplus T_c)$ to determine if enough testing was performed considering all the testing experience available.

Step 3 : Analyze the information of reliability and confidence achieved with previous testing but using the prospective use as the probability distribution in the computations.

Step 4 : If not enough testing was performed or if we want to perform independent testing we will use the model with a better value for the Kullback discriminant as the starting point. That is, either the closest model or the combined model is used.

Step 5 : Eliminate the irrelevant experience from the chosen model in step 4 with the method explained in Section 3.4.1.

Step 6 : Generate a set of test cases with the intended model.

Step 7 : Run the test cases and evaluate the results. If the results are good enough then we should reuse the asset.

With this methodology we will reduce the amount of testing necessary in order to decide to reuse an asset.

Chapter 4

Reliability and Confidence

Reliability and confidence are two fundamental measures in testing software. Usually these measures define the stopping criteria of the testing process.

Software reliability is the probability that the software will perform correctly in a given operational environment. Confidence is a measure of the certainty that a given value of the reliability lies in a certain interval. Reliability and confidence are closely related to each other, therefore we use terms like the reliability of this software is 0.90 with 95% confidence, i.e., we have 95% confidence in our estimate that the reliability is in the interval [0.9, 1.0].

In terms of Markov chain usage models, reliability is the probability of a randomly generated test case running from invocation to termination without encountering a failure state. When the testing chain contains no failure state, this definition results in a value of 1.0, which provides no information. If interpreted to mean that every single fault was removed from the software, and it will produce the right result 100% of the time, would be an unrealistic affirmation. In this work we focus on situations in which testing reveals no failures. If test-

ing reveals failures, we use the Markovian reliability. The following formulae are based on [7].

4.1 Probability of Failure

Probability of failure is the probability that an input selected at random according to the input distribution and run on the system under test will give an incorrect output.

Let θ be the unknown true proportion of failures in the software and $\widehat{\theta}$ be an estimate of θ. We assume that the test selection does not prohibit repetition. This is equivalent to picking balls from an urn with replacement. Intuitively, as we run more test cases without failure, we become more confident that the proportion of failures is small. We cannot be sure that θ is zero because we are selecting test cases with replacement. To estimate θ we must quantify the previous intuition.

4.1.1 The Overall Approach

The Bayesian estimate for θ is

$$\widehat{\theta} = \frac{a}{t+a+b}$$

where t is the number of test cases run and a,b are the parameters of *Beta(a,b)* chosen consistently with the prior knowledge about θ [7]. In this case the prior assumptions are

appropriate to the entire input domain of a program, and sampling is from the entire domain.

If no prior knowledge about θ is available we must set a and b to 1. This assumption embodies the believe that the software is equally like to have any possible value of θ in the interval [0,1]. In this work we will use $a = 1$ and $b = 1$.

4.1.2 Binning the Domain

A more conservative alternative to the overall approach is to partition the input domain into bins and compute θ_i for each bin i.

The tester can choose the way in which to define the bins as long as they form a mathematical partition, that is, all the elements of the input domain are represented and no element appears in more that one bin. In addition to the partition requirement, each element in a bin must have equal probability of being selected. Each bin has a probability p_i of being selected.

Following [7] compute $\widehat{\theta_i}$ using

$$\widehat{\theta_i} = \frac{a_i}{t_i + a_i + b_i}$$

and the estimate for θ is

$$\widehat{\theta} = \sum_i \widehat{\theta_i} p_i$$

where a_i, b_i denote the prior Bayesian knowledge about θ for bin i, and t_i is the number of test cases run in bin i.

In this approach, the distribution of $\widehat{\theta}$ is a probability weighted consideration of the estimators of each bin. The binning approach allows us to select a prior distribution for each bin, which means that each bin can be analyzed independently. For all bins, a_i and b_i will be set to 1.

Repetitions of test cases are not forbidden for this method, however we will not consider them. If a test case is repeated it will not impact the computations of $\widehat{\theta}_i$ since no new information is contibuted.Three possible ways to define the bins are now presented.

First, we consider a separate bin for each test case run, and one bin for all the test cases not run. In this approach the probability of each bin will be defined by *P(Sequence|Model)*. Table 4.1 shows the binning process for n test cases. This is the most conservative scenario.

Table 4.1: Binning for n+1 Bins (one for each test case run)

bin 1	bin2	.....	bin n	bin n+1
test case 1	test case 2	.....	test case n	Input Domain - {test case 1, test case 2,..., test case n}
$t_1 = 1$	$t_2 = 1$	.....	$t_n = 1$	$t_{n+1} = 0$
$a_1 = 1$ $b_1 = 1$	$a_2 = 1$ $b_2 = 1$	.....	$a_n = 1$ $b_n = 1$	$a_{n+1} = 1$ $b_{n+1} = 1$
$p_1 = P(t_1 \mid Model)$	$p_2 = P(t_2 \mid Model)$	.....	$p_n = P(t_n \mid Model)$	$p_{n+1} = 1 - \sum_i^n P_i$

Second, we consider one bin for all the sampling and one for the bin not sampled. The issue here is how to assign the probability to the bins. It is necessary to estimate the proportion of the population from which those run were drawn, p, and the proportion not participating in the sample, 1-p. Table 4.2 shows some example values for the probabilities in the two-bin model.

Table 4.2: Probabilities for Two Bins

Test Cases Run	Test Cases Not Run
0.8	0.2
0.9	0.1
0.95	0.05
0.99	0.01

Third, we consider an intermediate selection of bins between two and n+1 bins. As before, we assign one bin for the partition from which no samples were drawn. We propose two ways to assign the probability mass to the bins. Let n be the number of bins (n-1 bins for the partitions sampled and one for the partition not sampled). The first method is to use uniform probabilities for the bins sampled, that is, $p_i = \frac{p}{n-1}$ $i = 1,..,n\text{-}1$, and 1-p for the bin not sampled, where p is the proportion of the population from which the test cases were drawn.

The second way to assign the probabilities to n bins will consider the proportion of the population from which the test cases run were drawn (p) and the distribution of test cases and their probabilities among bins.

The first step is to generate a large number of test cases (t) and compute

$$q_k = \sum_{i=1}^{t_k} P(t_i|Model) \text{ for } k = 1,..., n-1$$

and $Q = \sum_{i=1}^{n-1} q_i$

where $t = \sum_i t_i$ and t_i is the number of test cases assigned to bin i.

Using q_k, Q, and p we will compute the probability mass of each bin for test cases run by

$$p_k = \frac{q_k}{Q} \times p \text{ for } k = 1,..., n-1$$

The probability for the bin with test cases not run is $p_n = 1 - p$.

The estimates of probabilities of failure $\widehat{\theta}_i$ and $\widehat{\theta}$ will be computed as above using the real number of test cases run.

As an example, Table 4.3 summarizes these two methods for the binning process using bins. Rows 1 and 2 are common for both methods and are the parameters for the estimate of the probability of failure of each bin $\widehat{\theta}_i$. Row 3 shows a method in which uniform probabilities are assigned to the bins for test cases run, and row 4 shows the second way to assign the probability mass.

Table 4.3: Binning for Six Bins

rows	$1 \geq P_i \geq 0.1$	$0.1 > P_i \geq 0.01$	$0.01 > P_i \geq 0.001$	$0.001 > P_i \geq 0.0001$	$0.0001 \geq P_i \geq 0$	Bin not Sampled
1	t_1	t_2	t_3	t_4	t_5	t_6
2	$a_1 = 1$ $b_1 = 1$	$a_2 = 1$ $b_2 = 1$	$a_3 = 1$ $b_3 = 1$	$a_4 = 1$ $b_4 = 1$	$a_5 = 1$ $b_5 = 1$	$a_6 = 1$ $b_6 = 1$
3	$p_1 = p/5$	$p_2 = p/5$	$p_3 = p/5$	$p_4 = p/5$	$p_5 = p/5$	$p_6 = 1 - p$
4	$p_k = \frac{q_k}{Q} \times p$ For k = 1, 2 ....5 where $q_k = \sum_{i=1}^{t_k} P(t_i \mid Model)$ for $k = 1,..,5$ and a large number of test cases $Q = \sum_{i=1}^{n-1} q_i$ t_k is the number of test cases run in bin k.					$p_6 = 1 - p$

Figure 4.1 shows the behavior of the reliability for the overall method and the different binning methods. The method with six bins uses $p = 0.9$. Limits in reliability are the overall approach (most aggressive) and the binning with $n+1$ bins (most conservative). For two bins we can see that they are closer to the overall approach than to the $n+1$ bins. The higher the probability for the bin for test cases run the closer the reliability will be to the overall approach. When we increase the number of bins we get an intermediate value for the reliability and also we get a curve with a more gradual slope, which means that the asymptotic behavior is reached after a larger number of test cases. In conclusion, we can say that if we want a curve with gradual slope we need to increase the number of bins, and a higher value of reliability will result from test cases sampled from the higher probability

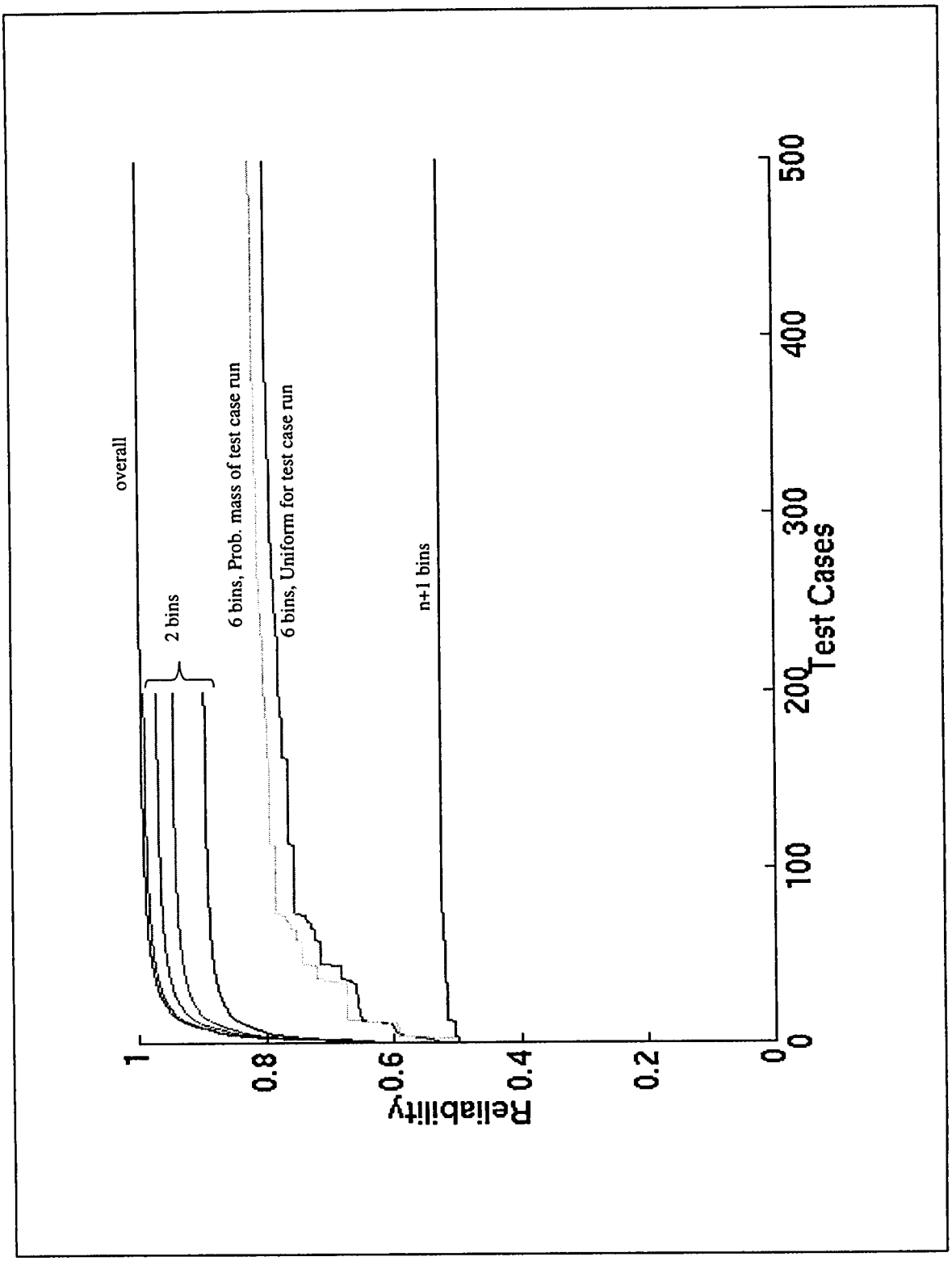

Figure 4.1: Behavior of Reliability

bins. Further work must be done to determine in a formal way the appropriate number of bins and how to assign the probability mass to each bin, but that is out of the scope of this work.

4.2 Confidence

Let Θ be the posterior distribution of $\widehat{\theta}$. We can define a confidence level c which defines a critical value θ_c such that $P\{\Theta \leq \theta_c | x\} = c$, where x is the number of failures. By solving this equation for θ_c, we can determine an interval $0 \leq \Theta \leq \theta_c$ in which Θ lies with confidence c.

For $a = 1$, $b = 1$, and $x = 0$ we have that the confidence is

$$c = 1 - (1 - \theta_c)^{t+1}$$

We will compute the confidence level c for each estimate of θ. That is we will use $\widehat{\theta}$ (computed for both the overall method and the binning method) as the critical value in the previous equation to get a value of confidence that the unknown value θ lies in the interval $[0, \widehat{\theta}]$.

Table 4.4 shows the number of test cases required to achieve a certain level of reliability and confidence assuming zero failures and no previous knowledge of the true proportion of failures ($a = 1$, $b = 1$ in prior Beta distribution). We can see that to achieve

reliability of 0.999 with a confidence of 90% we need to run 2301 test cases without failure, which for manual testing will usually be more than the tester can afford to run.

Table 4.4: Number of Test Cases for Certain Reliability and Confidence Levels

Reliability	Confidence%	Test Cases
0.9	90.00	21
	95.00	28
	99.00	43
	99.90	65
0.95	90.00	44
	95.00	58
	99.00	89
	99.90	134
0.99	90.00	229
	95.00	298
	99.00	457
	99.90	687
0.999	90.00	2301
	95.00	2994
	99.00	4602
	99.90	6904

Figure 4.2 shows the relationship between Reliability and Confidence for both the overall method and the method of n bins. We clearly can see that the reliability with bins is more conservative than the overall reliability. The confidence is very high for the method with bins. Therefore we can see that for the method with bins we get a very conservative reliability with a high confidence. On other hand the overall method give us a higher value for the reliability, but a lower confidence.

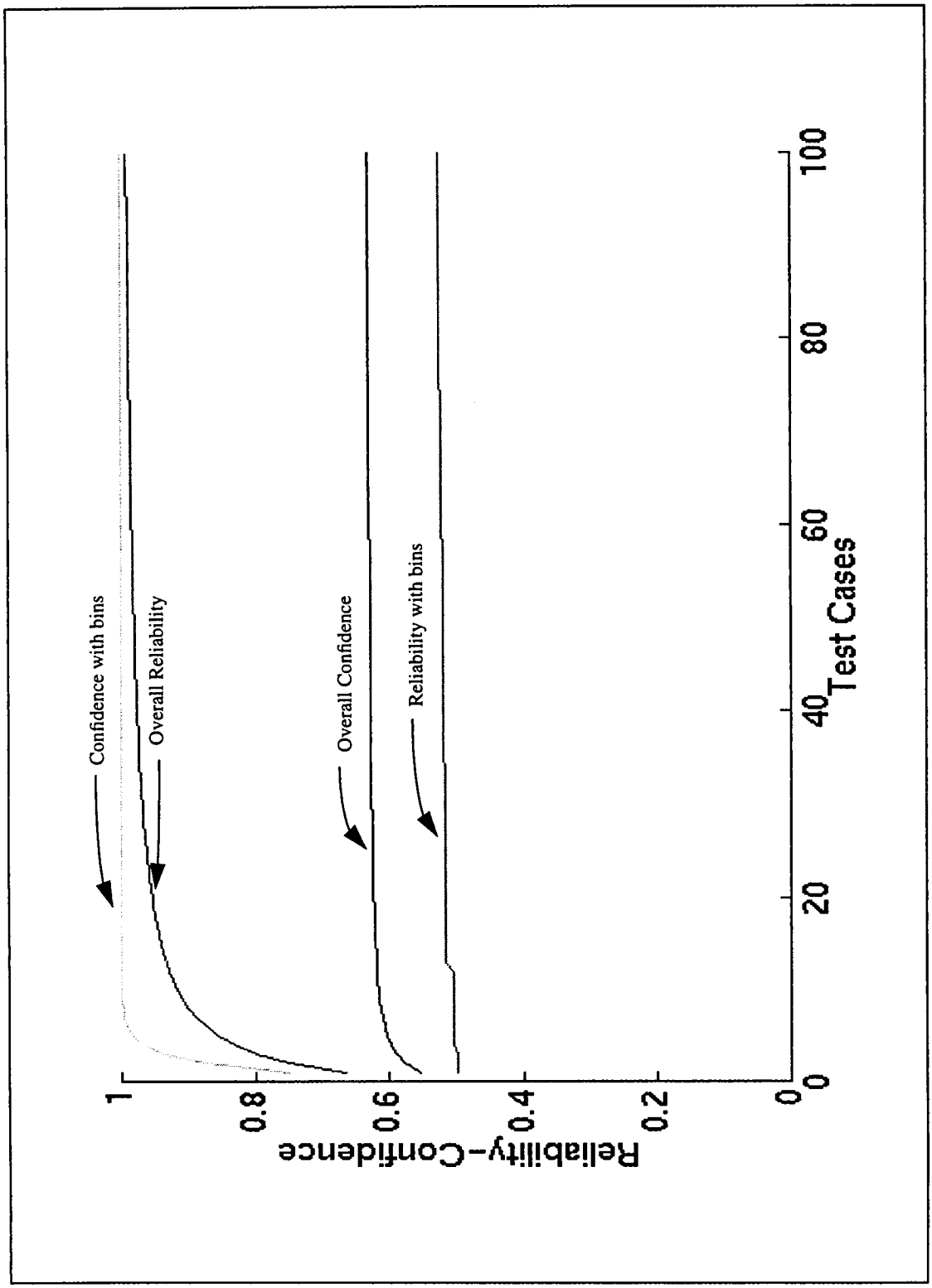

Figure 4.2: Relationship between Reliability and Confidence

Figure 4.3 shows the behavior of the confidence for the overall method and the different binning methods. We get about the same level of confidence for 6 bins (with both ways of assigning probabilities) and for $n+1$ bins, that is very high confidence with just a few test cases (we approach 100% confidence with 21 test cases for the $n+1$ bins method and 100% for the six bins method with about 40 test cases). For the two bins method we get close to the overall confidence when we assign a higher probability to the bins with test cases run. We can also see that the slopes of the curves with lower probabilities are greater than the ones with higher probabilities. This means that for low probabilities, the confidence reaches the asymptotic behavior faster.

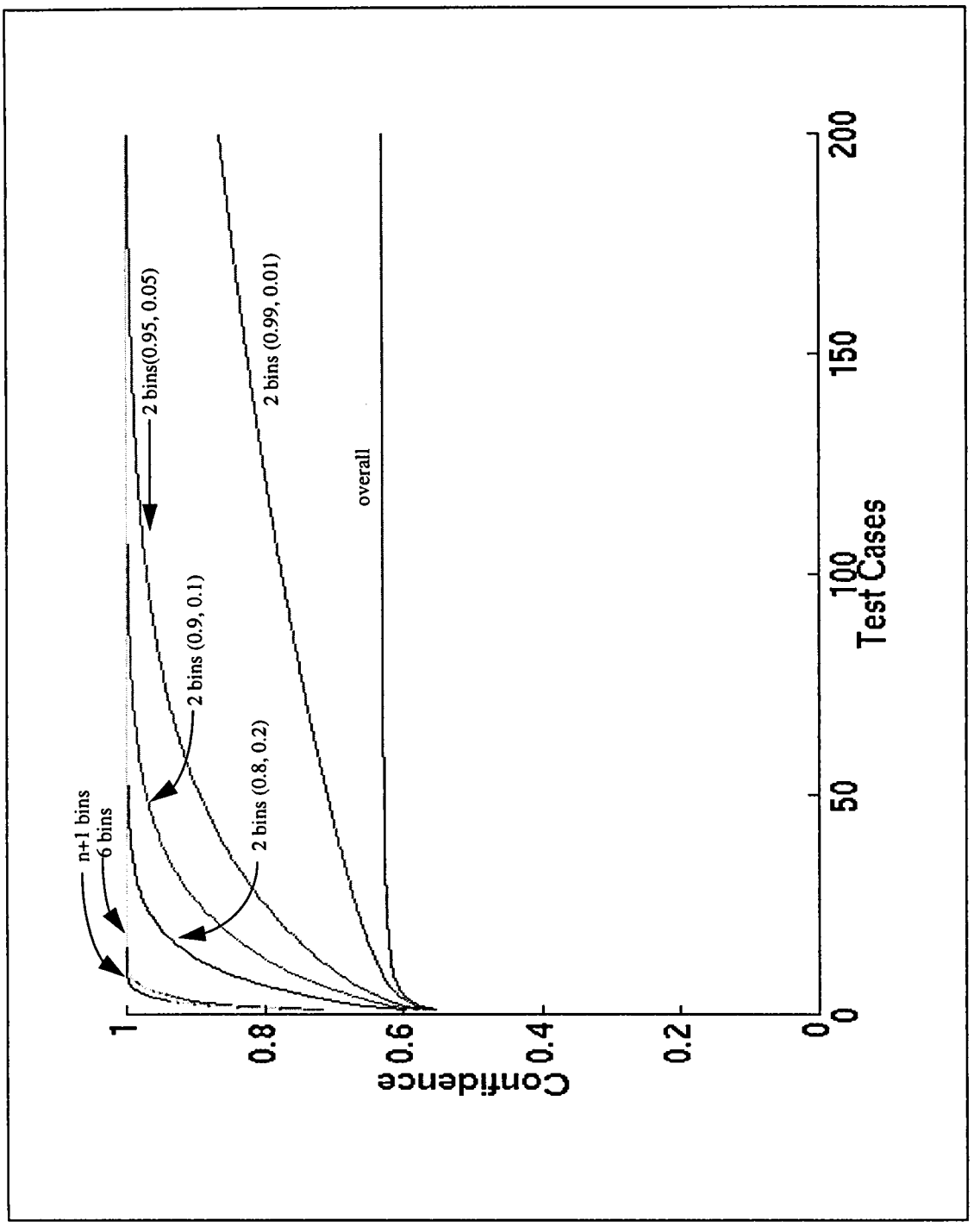

Figure 4.3: Behavior of Confidence

Chapter 5

Examples

The examples and simulations presented here are based on the usage model of a real product with 130 states and 566 arcs.

We assume that no errors were found during testing. In all cases, 200 random test cases were generated for each environment. For this specific model, coverage of arcs was not achieved at that point, consequently we initialized the frequency testing chain with ones instead of zeros, to permit computation for the Kullback discriminant for every test case run. This is reasonable practice if the arc-coverage sequences are tested prior to random testing.

We will show two situations. In the first one, Testing Records A, B and A⊕B are available for the comparisons and C is the prospective new use of the software. Environment A was provided by the original builder of the model. Environment B uses uniform probabilities. Environment C is a variation on use B.

In the second situation we have environments A, B, C, and all the combinations. Model D is the new use of the software. A, B, C are the same as in situation one. D is a variation on use A.

The first step in the method is to compare the prospective use with the testing history. Table 5.1 shows the values for the different distance measures proposed above for situation one. The dot (•) represents the first column of the matrix.

Table 5.1: Distances from Usage Model C to Usage Models in the Repository

	D(C,•)	D(•,C)	mean	d(C,•)	N2(C,•)	B(C,•)
A	0.049239	0.138626	0.093932	3.131496	1.064059	0.204616
B	0.000892	0.000950	0.000921	0.793828	0.424264	0.000469
A⊕B	0.027342	0.031828	0.029585	2.370636	0.712345	0.109151

For this situation the closest model is B.

Now we need to determine whether or not enough testing has been already performed. To do that we compute the distances between the new use (C) and all the testing chains available. Notice that in this case we do not need the mean value for the Kullback discriminant. We only need to compute D(C,•). Table 5.2 shows these distances.

Table 5.2: Distances from Usage C to Testing Chains in the Repository

	D(C,•)	d(C,•)	N2(C,•)	B(C,•)
Ta	0.125316	3.017741	1.008647	0.198698
Tb	0.005819	1.484031	0.505076	0.003523
A⊕B	0.031828	2.370636	0.712345	0.109151

The value of greater interest in our analysis is the distance for the model close to the prospective use, in this case T_b. If that value is good enough for our purposes we can decide to reuse the software element without further testing. On other hand, if we decide to perform more testing we must choose the starting model.

Another aspect that we must consider is the reliability and confidence achieved with previous testing but using probability distribution C to compute the values. In that way we have the reliability and confidence under the new use. Table 5.3 shows these values. We can see that the best reliability (with 6 or n+1 bins) is for the closest model, in this case B, but we have only a slight difference on the reliability between B and B without the irrelevant experience. Therefore the we can say that the irrelevant experience does not contribute much to the convergence of the model nor to the reliability or confidence achieved. For the overall approach the best value is for model A⊕B which can be understood because the computation is based on the number of test cases, although the differences are small.

In the simulation that follows, we will plot the information for the following situations:

I. Testing exclusively with the prospective new use. In this case the testing chain is initialized either with all zeros or all ones. In this situation, no previous testing information is considered.

II. Add test cases generated by the new model to previous testing. This situation considers all the previous testing history for the closest model.

III. Considering only test cases in the history that have a greater or equal probability of being generated by the prospective usage model.

Table 5.3: Reliability and Confidence with Use C

Model	Test Cases	Overall	2 bins (0.9, 0.1)	6 bins (0.9 0.1)	n+1 bins
A	166	R: 0.994048 C: 63.10 %	R: 0.944643 C: 99.99 %	R: 0.699970 C: 100 %	R: 0.55592 C: 100 %
B	135	R: 0.992701 C: 63.08 %	R: 0.943431 C: 99.96 %	R: 0.775334 C: 100 %	R: 0.609705 C: 100 %
A⊕B	293	R: 0.996610 C: 63.15 %	R: 0.946949 C: 100 %	R: 0.765499 C: 100 %	R: 0.576682 C: 100 %
B-irrelevant 1	120	R: 0.991803 C: 63.06 %	R: 0.942623 C: 99.92 %	R: 0.775331 C: 100 %	R: 0.609704 C: 100 %
B-irrelevant 2	127	R: 0.992248 C: 63.07 %	R: 0.943023 C: 99.95 %	R: 0.775333 C: 100 %	R: 0.609705 C: 100 %

B-irrelevant 1: Only test cases with a greater probability in the prospective new use are considered
B-irrelevant 2: test cases with a greater probability in the prospective new model are considered or those that contribute information to the coverage of the model.

IV. Considering only test cases in the history that have a greater or equal probability of being generated by the prospective usage model, or test cases that produce new information relative to achieving arc coverage of the model.

Figure 5.1 shows the results of the Kullback Discriminant for testing simulation starting with use B and applying test cases generated with C. The top line (I) correspond to testing only with environment C, without considering any of the previous experiences. For this example, we clearly got quicker convergence with either method rather than with C alone. We can see that methods II and III are practically indistinguishable. This happens because both models are very similar. For this example there is not a big difference among methods II, III or IV, although method IV is slightly better than methods II and III. When two environments are very similar one can simply add the test cases generated with the

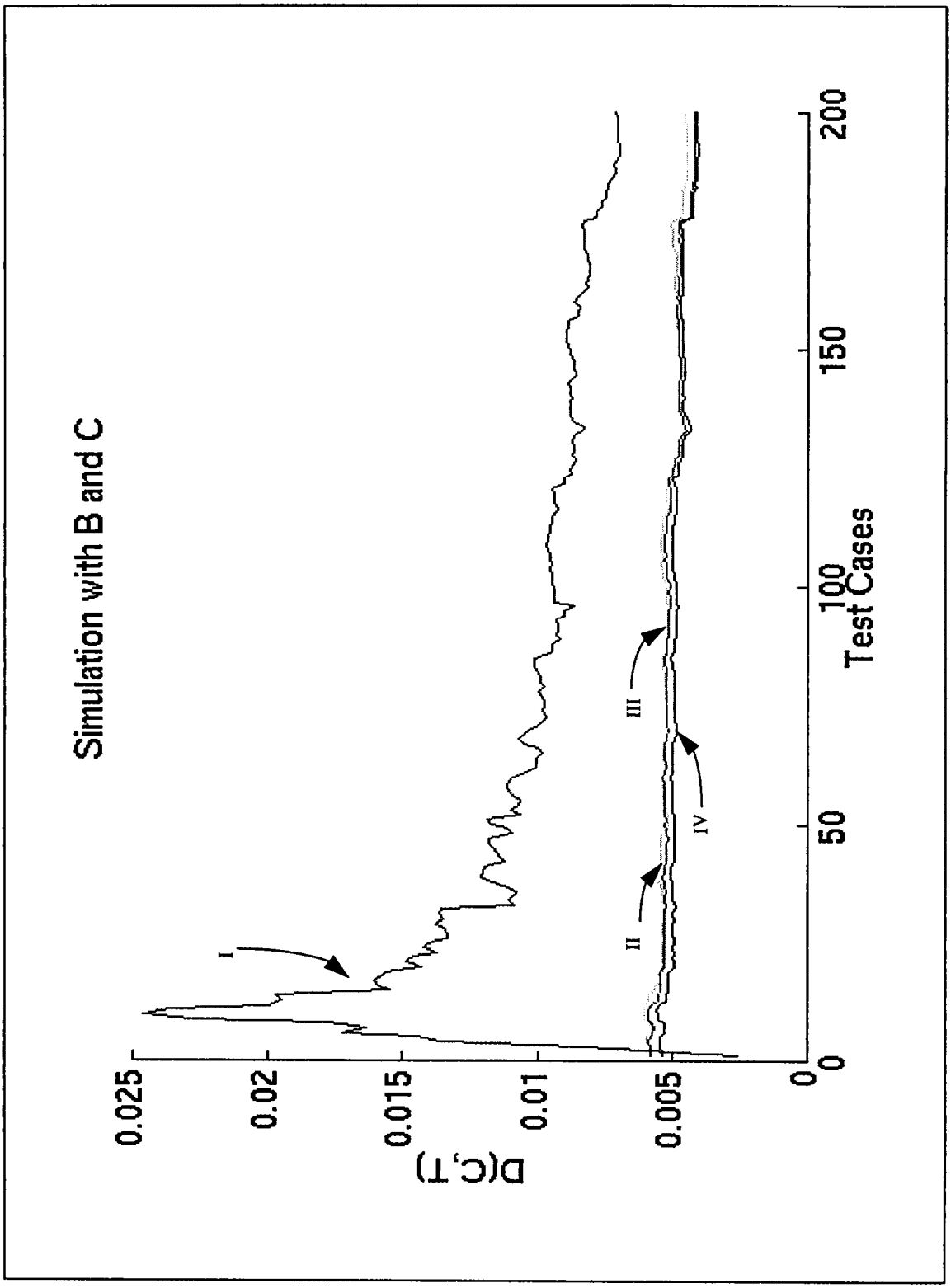

Figure 5.1: Kullback Discriminant for Simulation with B and C

new usage model to the previous testing experiences. We can notice that when we test only with C we never get values as good as those obtained with the other methods, and we need around 100 test cases to get convergence. That is about a 50% saving in the testing if we use previous experienceIn the second situation, we will show what happens when the models are not very similar. Table 5.4 shows the distance from use D and all the testing record available. In this case the best value is the one with use A⊕C although there is no big difference if we use all the testing history (A⊕B⊕C). The worst value is given for the use B⊕C.

Table 5.4: Distances from Usage Model D to Usage Models in the Repository

	D(C,•)	D(•,C)	mean	d(C,•)	N2(C,•)	B(C,•)
A	0.016089	0.033986	0.025038	1.987489	0.880057	0.032050
B	0.110713	0.039376	0.075045	2.318271	1.064059	0.128366
C	0.109790	0.039010	0.074400	2.351291	1.064059	0.122003
A⊕B	0.026198	0.024068	0.025133	1.959053	0.736965	0.020468
A⊕C	0.024404	0.023966	0.024185	1.945051	0.733213	0.021238
B⊕C	0.108388	0.041756	0.075072	2.446171	1.076049	0.123541
A⊕B⊕C	0.034881	0.023962	0.029422	1.918573	0.809156	0.022931

Table 5.5 shows the distance between D and all the testing chains. In this case A⊕B⊕C is slightly better than A⊕C. We can expect to get similar results if we use either one as starting point. In this example we will use A⊕C.

Table 5.5: Distances from Usage D to Testing Chains in the Repository

	D(C,•)	d(C,•)	N2	B(C,•)
Ta	0.033631	2.069465	0.855750	0.127369
Tb	0.047365	2.553247	1.108259	0.127369
Tc	0.046956	2.735245	1.048036	0.125127
A⊕B	0.024068	1.959053	0.736965	0.020468
A⊕C	0.023966	1.945051	0.733213	0.021238
B⊕C	0.041756	2.446171	1.076049	0.123541
A⊕B⊕C	0.023962	1.918573	0.809156	0.022931

Table 5.6 shows the values for reliability and confidence for model D. The tester must use this information to determine if enough testing was already performed to achieve the reliability and confidence goals under the prospective new use.

Figure 5.2 shows the results of Kullback Discriminant for testing simulation starting with use A⊕C and applying Test Cases generated with D. In this case we can see that just adding new test cases to the testing history is not a good choice. When the models are not so similar, the best option is to eliminate the irrelevant experience first and then run the test cases generated with the new model. When the initial frequency testing chain is set to ones instead of zeros we can use either method to eliminate the irrelevant experience, but when the testing chain is initialized in zeros it is better to use the method that includes test cases which incorporate new information to the testing chain. In this example, we need about 50 test cases with the new model to reach the same point as if we used previous experience. If we start with the worst possible model (the most different) we still save some testing, but of course in this case it is less than if we chose the closest model.

Table 5.6: Reliability and Confidence with Use D

Model	Test Cases	Overall	2 bins (0.9, 0.1)	6 bins (0.9 0.1)	n+1 bins
A	166	R: 0.994048 C: 63.10 %	R: 0.944643 C: 99.99 %	R: 0.785213 C: 100 %	R:0.522431 C: 100 %
B	135	R: 0.992701 C: 63.08 %	R: 0.943431 C: 99.96 %	R: 0.808042 C: 100 %	R: 0.527803 C: 100 %
C	135	R: 0.992701 C: 63.08 %	R: 0.943431 C: 99.96 %	R: 0.808042 C: 100 %	R: 0.527803 C: 100 %
A⊕B	293	R: 0.996610 C: .63.14%	R: 0.946875 C: 100 %	R: 0.800270 C: 100 %	R: 0.524840 C: 100 %
A⊕C	286	R: 0.996528 C: 63.14 %	R: 0.943023 C: 99.95 %	R: 0.792116 C: 100 %	R: 0.523505 C: 100 %
B⊕C	241	R: 0.995885 C: 63.14 %	R: 0.946296 C: 99.99 %	R: 0.809103 C: 100 %	R: 0.528005 C: 100 %
A⊕B⊕C	399	R: 0.997506 C: 63.14 %	R: 0.947756 C: 100 %	R: 0.801487 C: 100 %	R: 0.525042 C:100 %
A⊕C-irrelevant 1	67	R: 0.985507 C: 62.94 %	R: 0.936957 C: 98.80 %	R: 0.694256 C: 100 %	R: 0.513502 C: 100 %
A⊕C-irrelevant 2	129	R: 0.992366 C: 63.07 %	R: 0.943130 C: 99.95 %	R: 0.785095 C: 100 %	R: 0.522347 C: 100 %

A⊕C-irrelevant 1: Only test cases with a greater probability in the prospective new use are considered
A⊕C-irrelevant 2: Test cases with a greater probability in the prospective new model are considered or those that contribute information to the coverage of the model.

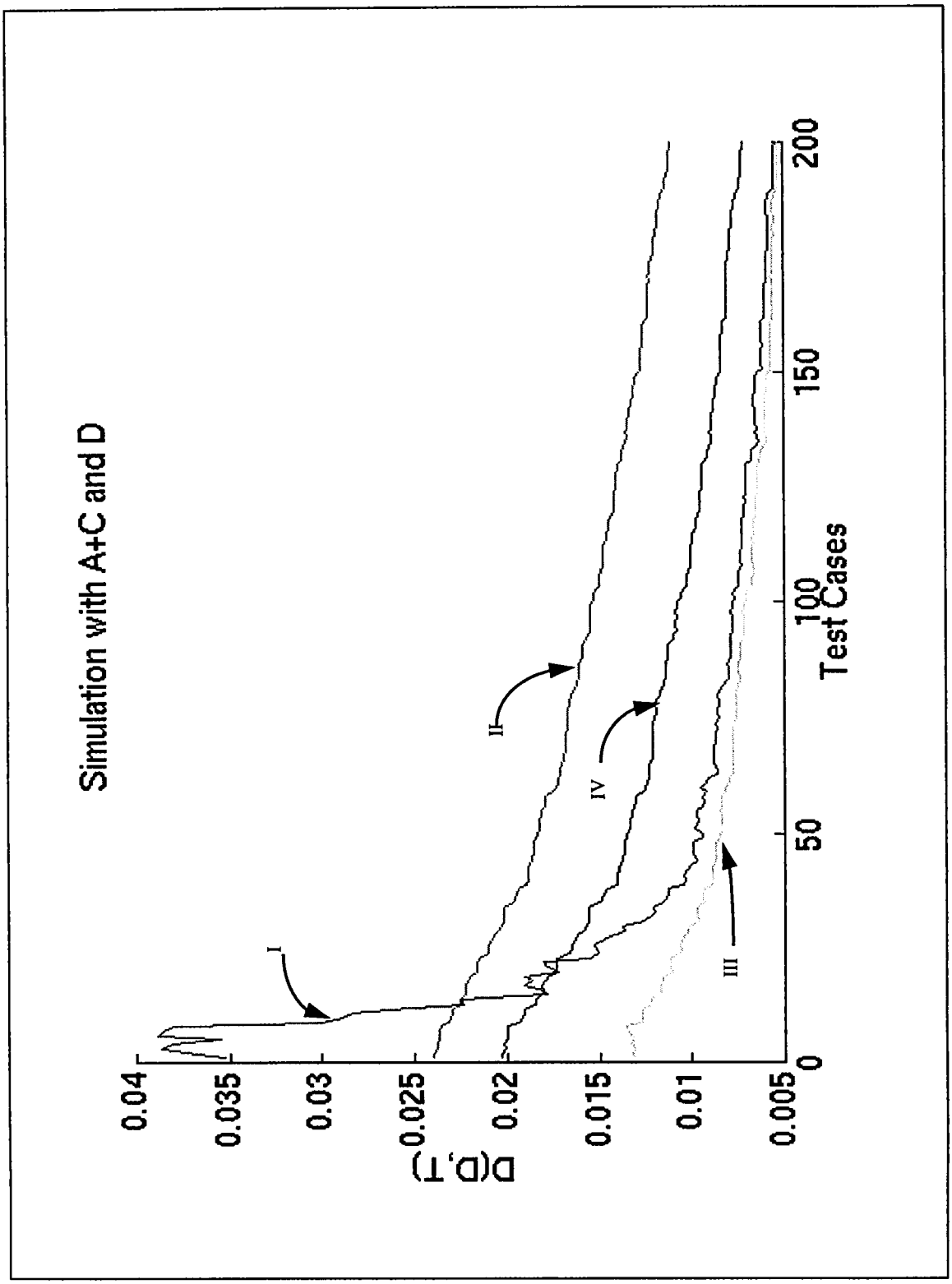

Figure 5.2: Kullback Discriminant for Simulation with A+C and D

Chapter 6

Conclusions

This work presents a methodology for using previous testing information in order to reduce additional new testing. We provide several candidate operators in order to answer the reuse questions posed in Section 1.3. Practice will show which candidate operator to answer each question is the most appropriate depending on the testing situation. Practice will also show which values are appropriates in the comparisons.

The method basically consists of the following steps:

Step 1 : Determine which of the previous testing records is closest to the new prospective use.

Step 2 : Determine whether enough testing was already performed. To do this, compare the prospective use with previous testing chains and also analyze the reliability and confidence information.

Step 3 : If it is necessary to carry out more testing, assess the difference between the prospective use and the previous experience. If they are very similar, add the test cases generated with the prospective use to the testing already performed. If the

uses are not very similar, the best alternative is to eliminate the irrelevant experience from the closest model and add test cases generated with the new model.

Step 4 : Incorporate results of new testing to the repository.

Table 6.1 shows the alternative methods proposed to answer each of the reuse questions.

Table 6.1: Summary of Candidates to Answer Reuse Question

Question	Candidates	Comments
How to compare one usage model with another?	1. Euclidean distance 2. Kullback discriminant (symmetrical version) 3. Matrix norms 4. Bhattacharyya distance	In general, we prefer the symmetrical version of the Kullback discriminant. Kullback Discriminant requires more computation that all the other candidates.
How to compare a usage model with a testing mode?	Kullback discriminant	Kullback discriminant is the right measure because it indicates if two chain will generate similar test cases.
What is the likelihood that a sequence was generated from a specific usage?	1. New model versus Test experience 2. New Sequences versus Previous models	We compute the probability that a sequence generated for previous testing experience is also generate for the new model with a greater or equal probability than in the previous model. We use this computation to select the relevant experience.
How to combine testing records?	1. Means Values 2. Entrywise Multiplication 3. Cumulative experience	In this work we use the cumulative experience method.
What is the most efficient model to generate the new testing?	1. Add new test to previous experience 2. Eliminate irrelevant experience	For very similar models is better to add testing to previous experience. For models which are not very similar it is better to eliminate irrelevant experience first.

Extensions of this work will be to analyze the values generated for each of the candidates in order to answer the reuse question. Further work must also be done in the compu-

tation of reliability with bins to determine in a formal way the appropriate number of bins and how to assign the probability mass to each one.

In this work we assumed that a software element is tested and used as it is, with no change in code. Under this assumption the structure of the models are identical. Future work must be done when the structure of the models is similar but not identical. Another area to explore is porting software to different platforms.

REFERENCES

References

[1] A.O. Allen. *Probability, Statistics and Queueing Theory with Computer Science Applications (Second Edition).* Academic Press, Inc. 1990.

[2] A.B. Clarke, and R. L. Disney. *Probability and Random Processes: A First Course with Applications.* John Wiley & Sons. 1985.

[3] J.W. Duran, and S.C. Ntafos. "An Evaluation of Random Testing". *IEEE Transactions on Software Engineering,* Vol 10, No 4, July 1984. pp 1418-1421.

[4] J.W. Duran, and J.J. Wiorkowski. "Quantifying Software Validity by Sampling". *IEEE transactions on Reliability,* Vol. R-29, No 2, June 1980. pp 141-144.

[5] G.H. Golub, and C.F. Van Loan. *Matrix Computations (Second Edition).* The Johns Hopkins University Press. 1989.

[6] B.H. Juang, and L.R. Rabiner. "A Probabilistic Distance Measure for Hidden Markov Models". *AT&T Technical Journal,* Vol 64, No 2, February 1985. pp 391-408.

[7] K.W. Miller, L.J. Morell, R. E. Noonan, S.K. Park, D.M Nicol, B.W. Murriel, and J.M Voas. "Estimating the Probability of Failure When Testing Reveals No Fail-

ure". IEEE Transaction of Software Engineering, Vol 18, No 1, January 1992. pp 33-44.

[8] H.D. Mills. "Certifying the Correctness of Software". *Proceedings of the Hawaii International Conference on System Sciences, Vol II, Software Technology,* 1992. pp. 373-381.

[9] T.J. Ostrand, and M.J. Balcer. "The Category-Partition Method for specifying and generating functional tests". Communications of ACM, Vol 31 No 6, June 1988. pp 676-686,

[10] J.H. Poore, H.D. Mills, and D. Mutchler. "Planning and Certifying Software System Reliability". *IEEE Software,* Vol 10 No 1, January 1993. pp 88-99.

[11] C.W. Therrien. *Decision, Estimation and Classification: An introduction to Pattern recognition and Related Topics.* John Wiley & Sons, Inc. 1989.

[12] G.H. Walton. *Generating Transition Probabilities for Markov Chain Usage Models*, Ph.D. Dissertation, Department of Computer Science, University of Tennessee, Knoxville, 1995.

[13] G.H. Walton, J.H. Poore, and C.J. Trammell. "Statistical testing of Software based on a Usage Model". *Software Practice and Experience,* Vol 25, No 1, January 1995. pp 97-108.

[14] J.A. Whittaker, and J.H. Poore. "Markov Analysis of software Specifications". *ACM Transactions on Software Engineering and Methodology,* Vol 2 No 1, January 1993. pp 93-106.

[15] J.A Whittaker, and M.G. Thomason. "A Markov Chain Model for Statistical Software Testing". *IEEE Transactions on Software Engineering,* Vol 30, No 10 October 1994. pp 812-824.

Vita

Jenny-Ann Morales was born in Puerto Natales, Chile, on July 7, 1964. She finished her high school at Colegio de Religiosas Carmelitas, San Felipe, Chile, in December 1981. In March 1982, she entered the Engineering School at Universidad de Santiago de Chile (a six years degree). In the fifth year at USACH, she received the degree of Licenciado en Ciencias de la Ingeniería, and in August 1988 she received the Ingeniero Civil en Informática degree.

From September 1988 to July 1989 Ms. Morales worked as an instructor in the Departamento de Ingeniería en Informática at Universidad de Santiago de Chile. After that she worked as system analyst at the bank of A. Edwards, Santiago Chile until March 1992 when she moved to Knoxville, Tennessee.

In January 1994, she entered the Graduate School at The University of Tennessee, Knoxville. From August 1994 to December 1996 she worked as either Teaching Assistant of the Department of Computer Science, or Graduate Research Assistant in the Software Quality Research Laboratory of the University of Tennessee.

She completed the requirements for the Master of Science with major in Computer Science, in December 1996.

She is married to Fernando Rodrigo Rannou and they have a wonderful son, Andrés Alberto.