1996

Optimal developmental test programs for one-shot systems

Michael James Moon
Iowa State University

Optimal developmental test programs for one-shot systems

by

Michael James Moon

A Dissertation Submitted to the
Graduate Faculty in Partial Fulfillment of the
Requirements for the Degree of
DOCTOR OF PHILOSOPHY

Department: Industrial and Manufacturing Systems Engineering
Major: Industrial Engineering

Approved:

Signature was redacted for privacy.

In Charge of Major Work

Signature was redacted for privacy.

For the Major Department

Signature was redacted for privacy.

For the Graduate College

Members of the Committee:

Signature was redacted for privacy.

Signature was redacted for privacy.

Signature was redacted for privacy.

Signature was redacted for privacy.

Iowa State University
Ames, Iowa
1996

UMI Number: 9626057

UMI
300 North Zeeb Road
Ann Arbor, MI 48103

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

NOTATION

The following symbols will be used throughout this manuscript.

N	Initial budget of devices	Page 1
n	An arbitrary budget of devices	6, 11
p_j	Probability of failure in state j	5, 44
s_0	Initial probability of design having state 2 failure probability	5
s_i	Posterior probability design is in state 2 after i developmental steps	5
u	Probability of a successful redesign	5
Π_i	Design failure probability after i step development process	6
s_i^*	Smallest s_i for which no testing is recommended	6
x_i	$(0-1)$ binary value of ith test, $x_i = 1$ if and only if ith test fails	6
$r(s)$	Probability that a system of the current design is effective if the probability that the design state is 2 is s	7

$\eta_x(s)$	Function updating the probability that the design, with prior state 2 probability of s, is in state 2 after a test which gave result x	7, 10
$V_n(s)$	Optimal expected number of good systems from an n-system budget which has prior state 2 design probability of s	8, 11
d	Rational design cost, $d = p/q$ devices	9, 19
$\delta(s)$	Probability the design reliability is in state 2 after a redesign having prior state 2 probability of s	10
$f_{n-1}(s)$	$r(s)V_{n-1}(\eta_0(s)) + (1 - r(s))V_{n-1}(\eta_1(s))$, "test" function candidate in $V_n(s)$	11
G_n	$V_n(s_0)/r(s_0)$, growth in stockpile due to optimal development policy	26
C_n	DevCost$/n$, cost of optimal policy	26
u_j	Probability that a potentially harmful redesign performed in state j results in state 2 design reliability	31
$\delta_2(s)$	Probability design reliability is in state 2 after a potentially regressive redesign having prior state 2 probability of s	35
$\underline{p}$	Vector containing multiple-state failure probabilities	44
$\underline{s}$	Vector containing probabilities that design has reliability in state i	44

$\underline{a}$	Vector of positive weights – initial distribution of multiple-state design reliability	45
(X,Y)	Development process state vector	45
$s_i(X,Y)$	Probability design reliability is in state i, given development vector (X,Y)	46
$r(X,Y)$	Multiple-state design reliability, given development vector (X,Y)	46
$h_0(X,Y)$ $h_1(X,Y)$	Redesign update functions for X and Y components of development vector, respectively	46
$V_n(X,Y)$	Optimal expected number of good systems from an n-system budget which has development state vector (X,Y)	47
$W_n^j(X,Y)$	Maximum expected number of effective systems in a potentially n-device development program allowing at most j decisions, with development state (X,Y), a "look ahead" rule	55
$\mathbf{S}_n$, $\mathbf{D}_n$	Number of look ahead steps from "Step" and "Device" look ahead rules, respectively	56

ABSTRACT

This dissertation considers two developmental testing models for *one-shot* systems, non-repairable devices that are destroyed by testing or first normal use (for example, rocket engines) when there is the potential for *reliability growth* through redesign. Given a limited budget and fixed cost per redesign, we wish to determine the sequence of redesigns and tests so that the expected number of effective systems in an ultimate stockpile of systems of the final design is maximized. We develop mathematical models and analyses which describe the optimal testing policies for this *sequential decision* problem. Further, we present properties of the models which, in some cases, make the computation of solutions feasible. We begin with the analysis of a two-state reliability model. Besides determining the effect of redesign costs on the optimal strategy, we show how the possibility of harmful redesigns can be incorporated into the model. We explore how this model behaves when even a "poor" reliability design is highly reliable but extremely high reliability is desired. A primary contribution of the thesis is an analysis of the general multiple-state (k-state) reliability model and, most interestingly, the presentation of a 2-variable formulation by which some k-state models can be analyzed. Under some restrictions, we analyze and compute solutions to this latter model on a lattice.

CHAPTER 1. INTRODUCTION

This dissertation presents two models for the analysis of developmental test programs for *one-shot* systems when there is potential for *reliability growth* through system redesign. One-shot systems are non-repairable and are destroyed upon first use; testing constitutes use. One example of such a device is a rocket engine.

We suppose there is a budget sufficient to construct N one-shot systems. Considering the budget size and current estimate of design reliability, we would like to determine if reliability growth can be achieved through a system redesign. Any *activity* (redesigning, testing, or building) exacts a price from the current budget. We seek a plan which will yield high device reliability and will simultaneously make the final number of devices in the stockpile as large as possible. More precisely, we wish to design and develop the systems so that the expected number of effective systems built at the end of the development process is maximized. Roughly speaking, the development period of testing/redesign ceases and we "build" using the remaining budget when the expected number of acceptable systems using the current reliability estimates is, by some measure, large.

Given the budget, we build using the present design when it appears that device effectiveness (reliability) cannot be improved through further Research and Development.

Our objective is to identify good policies dictating which development activity should occur next. The policies will consider the current budget, the current estimate of the system reliability, the likelihood of improving reliability through redesign or improving the estimate of the reliability via testing, and the costs associated with any such efforts. We seek methods for computing the optimal strategies for this *sequential decision problem.*

Such policies are necessary for efficient development of complex systems under a fixed budget. The policies are given in partial response to the question posed by Pentagon Science Advisor Dr. Ernest Seglie [21], "How Much Testing is Enough?"

We consider two models where reliability growth may occur after purchasing a system redesign. In both models, after each step in the development process, one chooses to redesign, test a device, or cease the development process and "build" — manufacture devices according to the current design.

The first model we consider generalizes one used by Huang in her Ph.D. dissertation [12]. The Huang model supposes that the (unknown) design reliability is in one of two possible states and that testing gives a binary (success/failure) response. Reliability growth is achieved *only* if a *failed* test triggers a redesign that proves successful. Once the "good" reliability state is reached, it cannot be left.

The two-state model was a necessary first step in developing useful testing strategies. However, the model is unrealistic in its simplicity: the redesign process is assumed to be free or of negligible cost; redesigns can not harm the current reliability; redesigns are allowed only after failed tests; and the expected reliability can be quite different from the actual reliability. Further, in a long, complex development process it seems reasonable that design reliability would change in steps or stages and not simply from "bad" to "good." Finally, given the extremely high reliability demanded of today's rocket motors and military weapons systems, waiting to witness a test failure is often not practically feasible. Ekstrom and Allred [10] point out that approximately 2300 successful tests, in a row, would be required to demonstrate reliability of 0.999 at a 90% confidence level.

The research described here addresses these issues. We modify Huang's model to reflect four additional features of typical development processes. First, we incorporate design costs into the model. Second, we remove the stipulation that redesigns can only follow a failed test. Next, by separating the testing and design activities, testing is given its proper role of providing information on current reliability while redesign fills its role of changing reliability. Finally, we model the possibility of reliability degrading redesigns, i.e., redesigns which may harm the current design reliability are considered.

We then propose a new model to describe a development process where the design reliability is in one of k states. The model is very general and the work described here is only a beginning in understanding its utility and power.

The next chapter is a literature review. Chapter 3 discusses the modifications to Huang's binary model we have studied. Chapter 4 is an introduction to the multiple-state reliability model and Chapter 5 contains the solutions we have developed for the multiple-state model. The last chapter gives conclusions and outlines questions for further study. The Appendix gives pseudo-code descriptions of the computer programs used to produce the numeric examples contained in this dissertation.

CHAPTER 2. LITERATURE REVIEW

While reliability growth has received substantial attention, most of the models in the literature describe how design reliability typically changes with time (presumably in response to some unspecified design improvement activities occurring behind the scenes according to some unspecified process) and not how it changes specifically in response to the redesign/testing development process. For example, Pollock [18], Bell [2], Duane [8], Dwyer [9], Fard and Dietrich [11], and Martz and Waller [17] have all discussed such models. A model for reliability growth leading to maximum likelihood estimates for failure rates is given by Barlow and Scheuer [1]. Ekstrom and Allred [10] discuss a design and demonstration process which seeks to ensure a prescribed reliability at minimum cost. Their S.A.F.E.R.^{sm} (**S**tatistical **A**pproach **F**or **E**ngineering **R**eliability) methodology is a six step, systemic approach to verifying reliability at a specified confidence level. The existing discussions of reliability growth provide little guidance for choosing developmental strategies.

The developmental model that is currently used by some developers and suppliers to the United States military was proposed by Lloyd and Lipow [16]. They analyze a model in which a device, when operated, either succeeds or fails and, if it fails, it does so in only one possible way. They assume that a redesign effort, if successful, permanently and completely removes the defect — that resultant devices will never fail. If the redesign doesn't succeed, the failure rate is unaffected. If the device fails a test, it is redesigned and this redesign has constant probability, less than 1, of removing the failure mode. Their analysis leads to an exponential reliability growth model of the form

$$R_n = 1 - Ae^{-C(n-1)},$$

where R_n is the reliability of the device before trial n is made and A and C are parameters of the model which need to be estimated.

Most recently, the question of when to stop a developmental testing process was the subject of Huang's Ph.D. thesis [12], and of Huang, McBeth and Vardeman [13]. Huang et al. developed a model for reliability growth with two reliability states similar to that given by Lloyd and Lipow. Huang's analysis is the starting point for this discussion, and so will next be discussed in some detail.

A fixed budget, sufficient to build N one-shot systems, is assumed. The development process is a sequence of tests and accompanying redesigns. The process ceases when the reliability is judged to be acceptable or the remaining number of devices becomes small. Binary test information indicates if a tested system "passes." A free (or negligible cost) redesign is ordered only after a failed test. There are two possible states, $j = 1, 2$, for design reliability. When in state j, a tested system fails with probability p_j. (Assume $p_1 > p_2$, so that state 2 represents the "good" state, that of better reliability.) The design reliability is initially in state 2 with probability s_0, and s_i is the posterior probability that the system is in state 2 after observing i tests and performing any prescribed redesigns. The system reliability improves only after an effective redesign (allowed only after a failed test). The redesign either improves reliability (moves it to state 2) with fixed probability u, or leaves it unchanged (the re-

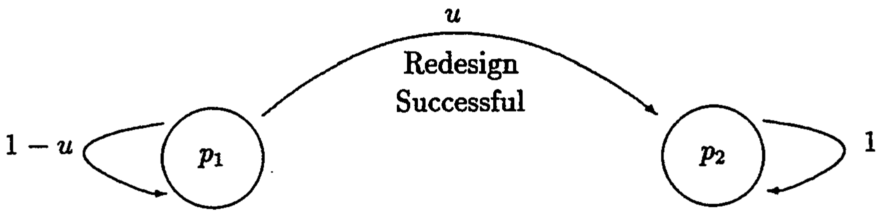

Figure 2.1: Possible change in reliability due to a redesign

liability cannot degrade). Figure 2.1 depicts the movement between design reliability states graphically.

Let n be the cost, in terms of potential systems, of development and Π_n be the resultant system failure probability. Using dynamic programming for a developmental phase "costing" n systems, Huang determines a stopping rule n^*, so that the final mean number of effective systems,

$$E[(N-n^*)(1-\Pi_{n^*})], \tag{2.1}$$

is as large as possible.

For a testing program potentially involving n systems, let s_n^* denote the smallest value of s_n at which no testing is recommended. Huang identifies the form of n^* and shows that it is the first i for which

$$s_i \geq s_{N-i}^*.$$

In this model, s is updated depending on whether the most recent test produced a success or a failure. If the test is failed, a free redesign (or one of negligible cost), which might improve reliability, is performed. The mathematical model is as follows.

Let $x_i \in \{0,1\}$ specify the outcome of test i. If x_i is 1, the ith test is failed and a redesign is performed before any further testing occurs. Otherwise, a successful test gives $x_i = 0$. Since the (conditional) probability of being in state 2 *before* test $i+1$ is s_i, the probability that the test is a success, i.e., the current reliability, is

$$\begin{aligned} r(s_i) &= s_i(1-p_2)+(1-s_i)(1-p_1) \qquad (2.2) \\ &= 1-p_1+s_i(p_1-p_2). \end{aligned}$$

To track s_i through the development process, let $\eta_0(s_i)$ and $\eta_1(s_i)$ give s_{i+1} depending on whether test $i+1$ is a success or failure, respectively. More precisely, η_0 and η_1 give the posterior probability of being in state 2 after a test with prior state 2 probability s_i, and the test produces a success or failure, respectively. Then, when $x_{i+1} = 0$, s_{i+1} is computed as the conditional or posterior probability of a successful test, given prior state 2 probability s_i. When a test produces a failure ($x_{i+1} = 1$), the posterior probability of being in state 2 is calculated by computing the sum of the probability of a successful redesign after failure in state 1 and the probability of failure in state 2, and dividing by the probability of failure. So, when expressed in terms of the $\eta_x(s_i)$, we see that s_{i+1} is

$$s_{i+1} = \eta_{x_{i+1}}(s_i) = \begin{cases} \dfrac{(1-p_2)s_i}{r(s_i)}, & \text{if } x_{i+1} = 0 \\ \\ \dfrac{p_1(1-s_i)u+p_2s_i}{1-r(s_i)}, & \text{if } x_{i+1} = 1. \end{cases} \qquad (2.3)$$

Let $V_n(s)$ be the optimal expected number of good systems from an n-system budget given prior probability s of being in state 2. Then, as discussed in Bellman [3], V_n is the maximum of the number of effective systems if the development process immediately ceases and "building" commences ($nr(s)$) and the expected maximum number of effective systems if at least one additional test is performed and the development process proceeds optimally after that. Then $V_1(s) = r(s)$ ("build" if budget is 1) and

$$V_n(s) = \max\{nr(s),\ r(s)V_{n-1}(\eta_0(s)) + (1 - r(s))V_{n-1}(\eta_1(s))\}. \qquad (2.4)$$

For small values of n, $V_n(s)$ can be computed using (2.4). Huang characterizes the optimal stopping rule and compares its performance against that of several heuristics.

The research presented in this paper shows how to incorporate redesign costs and model regressive (harmful) redesigns in the above framework. We study the utility of this model when developing extremely highly reliable one-shot systems (those with reliability of 0.999x). Finally and more generally, we propose a new model for reliability growth involving more than 2 reliability states.

CHAPTER 3. BINARY RELIABILITY MODEL WITH REDESIGN COSTS

3.1 Fixed Cost Designs

It is important to note that in Huang's model it is *necessary* to allow designs *only* after failed tests. Otherwise, a prescription to achieve good reliability is to simply perform (free) redesigns until s is essentially 1. We improve this model by incorporating design costs and compute an optimal policy when each redesign has the same fixed cost. We accomplish this by "uncoupling" the design process from testing and allowing it to be a distinct developmental choice. In comparison with Huang's model where the developmental action was strictly prescribed at each step, this uncoupling allows choices "redesign" and "test" at any time before finally recommending "build." Huang allowed redesigns only after a test failed and her model presented two choices at each step during design development: "test" or "build." We select from the three activities "redesign," "test," or "build," at each developmental step and update s_i accordingly.

We represent the redesign cost as d devices (d a positive real) and, as before, a test continues to cost 1 device (it's destroyed). (The test cost need not be 1 device, see Section 3.3.2.) Redesigns have the same dynamics as before, a constant probability u of improving the design reliability when in the poor reliability state (state 1). Let

$\delta(s)$ be the probability that the design reliability is in state 2 after a redesign, given prior probability s of being there. Then, by relegating "redesign" (and so reliability growth) to a separate activity, the update of s after a test, $\eta_x(s)$, becomes

$$\eta_x(s) = \begin{cases} \dfrac{(1-p_2)s}{r(s)}, & \text{if } x = 0 \\ \\ \dfrac{p_2 s}{1-r(s)}, & \text{if } x = 1, \end{cases} \tag{3.1}$$

(compare with (2.3)) where $r(s)$ is as given in (2.2) and

$$\delta(s) = u(1-s) + s = u + s(1-u). \tag{3.2}$$

3.2 Analysis of Design Cost Model

The free-redesigns model, given goal (2.1), has functional equation (2.4) given on page 8. The functional equation for the new model can be seen to be like (2.4) if we understand each piece of that equation. In Huang's analysis, if the current budget is sufficient to build n devices, the expected number of effective devices with no further development will be n multiplied by the current expected reliability, $r(s)$. Thus, if we "build" with the current budget, the expected effective yield will be $nr(s)$, the first term in (2.4). On the other hand, should a test (and accompanying redesign when relevant) improve the reliability enough that the expected number of effective devices remaining is greater than the number given by building, the expected yield is the value function, V_{n-1}, evaluated depending on whether the test succeeds ($\eta_0(s)$) or fails ($\eta_1(s)$), multiplied by the current probability of test success or failure, respectively. Thus, through "test," we have the second term in (2.4). At each step we seek to maximize the yield; hence $V_n(s)$ is as given by (2.4).

Keeping the same goal, (2.1), now consider the situation where redesigns are no longer free. The current budget of n devices is decreased by d if a redesign is performed. The redesign will leave a budget of $(n-d)$ devices. After a redesign, the probability that state 2, the desirable reliability state, has been reached is $\delta(s)$. Thus the expected effective yield after a redesign is given by $V_{n-d}(\delta(s))$. In this scenario, tests still cost one device but the probability of a device possessing good reliability is now given by the ηs of (3.1). As before, if the development process were to cease ("building" to commence), the expected yield would be $nr(s)$. So the new functional equation for this model, replacing (2.4), is to choose the maximum of the expected payoffs: "build," "test," or "redesign" at each step, that is

$$V_n(s) = \max\{nr(s),\ r(s)V_{n-1}(\eta_0(s)) + (1-r(s))V_{n-1}(\eta_1(s)),\ V_{n-d}(\delta(s))\}, \qquad (3.3)$$

where η_0, η_1, and δ are given by (3.1) and (3.2), respectively, and $V_1(s) = r(s)$. For notational convenience, label the second term of (3.3) $f_{n-1}(s)$,

$$f_{n-1}(s) = r(s)V_{n-1}(\eta_0(s)) + (1-r(s))V_{n-1}(\eta_1(s)). \qquad (3.4)$$

Given a set of parameters $(p_1, p_2, u, N$ and $d)$, to determine the optimal development policy for a design requires the input of s_0, the probability the design is initially in state 2. Using (3.3), one computes $V_N(s_0)$ (exactly *how* this is accomplished is the topic of what follows) and the optimal initial action is "build," "test," or "redesign" as the first, second or third term, respectively, of (3.3) gives the function value. Let n be the budget remaining after the first development step (here n is either $N-1$ or $N-d$). Suppose that the first action is *not* to build. Then, the next development action is determined using s_1, which is updated using (3.1) (test) or (3.2) (redesign), as appropriate, to compute $V_n(s_1)$. The second action is recommended, as before, according to which term in (3.3) gives the function value. This process is repeated until "build" is recommended (and, why "build" will eventually be optimal and why the development process ceases when this happens, will be discussed below). In what follows, we use n to represent the budget at any point in the development process and we use N when it is important to emphasize that the discussion pertains to the *initial* budget.

Given the discussion of the previous paragraph, it is clear that $V_n(s)$ can be computed using the recursive definition given in equation (3.3) for only very small n. (On a dedicated *DEC Alpha* workstation (100 MHz, 32 Meg RAM), a reasonable graph (s ranges from 0 to 1) of V_{30} will take *days* to produce.) The computational complexity of the problem as stated grows like $\Theta(3^n)$. So it is infeasible to compute solutions using (3.1), (3.2), and (3.3) directly even if n is only on the order of one hundred. Happily, the model has properties which enable the computation of a solution. We present them now.

Lemma 3.1 *For every $s \in [0,1]$,*

$$r(s)r(\eta_0(s)) + (1 - r(s))r(\eta_1(s)) = r(s).$$

Proof. Since $r(\eta(s)) = 1 - p_1 + \eta(s)(p_1 - p_2)$, we have

$$\begin{aligned} r(s)r(\eta_0(s)) &= r(s)[1 - p_1 + \frac{(1-p_2)s}{r(s)}(p_1 - p_2)] \\ &= r(s)(1 - p_1) + (1 - p_2)s(p_1 - p_2), \end{aligned} \tag{3.5}$$

and

$$(1 - r(s))r(\eta_1(s)) = (1 - r(s))(1 - p_1 + \frac{p_2 s}{1 - r(s)}(p_1 - p_2)$$

$$= (1 - r(s))(1 - p_1) + p_2 s(p_1 - p_2). \qquad (3.6)$$

Adding (3.5) and (3.6) gives $r(s)$. ■

Lemma 3.1 says the expected value of the reliability after the next test is the current reliability. In other words, nothing is gained, reliability-wise, through testing alone. It is the redesign activity that has the potential to improve reliability. Testing only provides insight into the current design reliability. This is another improvement from the earlier model where the reliability growth after a test (that can potentially result in a failure and therefore a redesign) is given by

$$p_1 u(1 - s)(p_1 - p_2).$$

Proposition 3.1 *If $n \leq d$, then $V_n(s) = nr(s)$.*

Proof: If $n \leq d$ a redesign cannot be performed so $V_n(s)$ is given by (2.4) on page 8 and (3.1). When $n = 1$, $V_1(s) = r(s)$. Now, suppose $n < d$ and $V_n(s) = nr(s)$. Thus, $(n + 1) \leq d$ and

$$\begin{aligned} V_{n+1}(s) &= \max\{(n+1)r(s), r(s)V_n(\eta_0(s)) + (1 - r(s))V_n(\eta_1(s))\} \\ &= \max\{(n+1)r(s), r(s)(nr(\eta_0(s))) + (1 - r(s))(nr(\eta_1(s)))\} \qquad (3.7) \\ &= \max\{(n+1)r(s), nr(s)\} \\ &= (n+1)r(s), \end{aligned}$$

by induction. ■

An important consequence of Proposition 3.1 is that testing will not be beneficial if a design cannot be purchased to improve the reliability. It also assures us that eventually (in terms of n) optimal plans "build."

Proposition 3.2 *$V_n(s)$ is piecewise linear.*

Proof: By Proposition 3.1, when $n \leq d$,

$$V_n(s) = nr(s) = n\{(1-p_1) + s(p_1 - p_2)\} = as + b, \tag{3.8}$$

a, b real numbers. When $n = d+1$,

$$\begin{aligned} V_{d+1}(s) &= \begin{cases} (d+1)r(s), & s > s_c \\ r(\delta(s)), & s \leq s_c \end{cases} \\ &= \begin{cases} l_0(s), & s > s_c \\ l_1(s), & s \leq s_c \end{cases} \end{aligned}$$

where $l_i(s) = a_i s + b_i$, for real numbers a_i and b_i. The point s_c is the value of s where the optimal activity changes. It is given by equation (3.11) on page 16. Now, suppose $V_n(s)$ is piecewise linear for $n = 1, 2, \ldots, d+1, \ldots, k-1$. To complete the proof, we show that V_k is piecewise linear.

$$V_k(s) = \max \left\{ \begin{array}{c} kr(s) \\ f_{k-1}(s) \\ V_{k-d}(\delta(s)) \end{array} \right\}$$

The first and third terms of V_k are piecewise linear by (3.8) and the induction hypothesis, respectively. Using the induction hypothesis again we have

$$\begin{aligned} f_{k-1}(s) &= r(s)V_{k-1}(\eta_0(s)) + (1-r(s))V_{k-1}(\eta_1(s)) \\ &= r(s)l_j(\eta_0(s)) + (1-r(s))l_m(\eta_1(s)), \end{aligned} \tag{3.9}$$

where l_j and l_m are linear components of V_{k-1}. Simplifying, $l_p(\eta_i(s)) = a_p\eta_i(s) + b_p$, for some reals a_p, b_p. Using this in (3.9) we have

$$r(s)(a_j\eta_0(s) + b_j) + (1-r(s))(a_m\eta_1(s) + b_m)$$

$$
\begin{aligned}
&= a_j(1-p_2)s + b_j r(s) + a_m p_2 s + b_m(1-r(s)) \\
&= a_q s + b_q,
\end{aligned}
$$

for a_q, b_q real constants, since $r(s)$ is linear. ■

Lemma 3.2 allows that we can compute V_n using linear interpolation and get exact values except at "corners."

The following lemmas are used to prove Proposition 3.3.

Lemma 3.2 *For each* $s \in (0,1)$, $\eta_1(s) < s < \eta_0(s)$.

Proof: Using (3.1), this is easily seen by computing

$$s - \eta_1(s) = \frac{s(1-s)(p_1-p_2)}{1-r(s)} > 0$$

and

$$\eta_0(s) - s = \frac{s(1-s)(p_1-p_2)}{r(s)} > 0. \blacksquare$$

Note that Lemma 3.2 proves that s will improve after a test succeeds and decrease if a test fails. In the earlier model, this was not true. The value of s could increase or decrease after a failed test, depending on the parameters of a problem. Using (2.3), we see that if

$$s > \frac{p_1 u}{(p_1-p_2)},$$

then a failed test will improve s. (This phenomenon comes about because in Huang's model, a failed test immediately results in a redesign, which can have substantial probability of improving reliability.)

Differentiating in (3.1) it is easy to see

Lemma 3.3 *The functions* $\eta_0(s)$ *and* $\eta_1(s)$ *are monotone increasing in* s.

Proposition 3.3 *For each n, $V_n(s)$ is monotone increasing in s.*

Proof: If $0 \le s_0 \le s_1 \le 1$, we want to show that $V_n(s_0) \le V_n(s_1)$. By Proposition 3.1, when $n \le d$, $V_n(s) = nr(s) = n(1 - p_1 + s(p_1 - p_2))$ and, since $p_1 > p_2$, V_n is monotonically increasing in s. When $n = d+1$, Proposition 3.1 gives

$$V_{d+1}(s) = \max \begin{cases} (d+1)r(s), & s > s_c \\ r(\delta(s)), & s \le s_c, \end{cases} \tag{3.10}$$

where the cut-off point, s_c is

$$s_c = \frac{u(p_1 - p_2) - d(1 - p_1)}{(p_1 - p_2)(d+u)}, \tag{3.11}$$

the point where the two parts of (3.10) are equal. Now V_{d+1} can be seen to be monotonically increasing since the slope of each line segment is positive (the first has slope $(d+1)(p_1 - p_2)$ and the segment corresponding to an argument of $r(\delta(s))$ has slope $(1-u)(p_1 - p_2)$).

Now suppose that for $n = 1, 2, \ldots, d+1, \ldots, k-1$, V_n is monotone increasing in s. Since $nr(s)$ is linear with positive derivative and $V_{n-d}(\delta(s))$ is monotone increasing by virtue of the induction hypothesis, we will be done if we show that $f_{k-1}(s)$ (from (3.4) on page 11) is monotone increasing. Let $0 \le \Delta \le 1 - s$. We wish to show that $f_{k-1}(s + \Delta) \ge f_{k-1}(s)$. Computing the difference

$$\begin{aligned} & f_{k-1}(s+\Delta) - f_{k-1}(s) \\ = \; & [r(s+\Delta)V_{k-1}(\eta_0(s+\Delta)) + (1 - r(s+\Delta))V_{k-1}(\eta_1(s+\Delta))] \\ & -[r(s)V_{k-1}(\eta_0(s)) + (1 - r(s))V_{k-1}(\eta_1(s))] \\ = \; & r(s)[V_{k-1}(\eta_0(s+\Delta)) - V_{k-1}(\eta_0(s))] \\ & +(1 - r(s))[V_{k-1}(\eta_1(s+\Delta)) - V_{k-1}(\eta_1(s))] \qquad (3.12) \\ & +\Delta(p_1 - p_2)[V_{k-1}(\eta_0(s+\Delta)) - V_{k-1}(\eta_1(s+\Delta))], \end{aligned}$$

we see that by virtue of the induction hypothesis and Lemma 3.3, the first two terms in (3.12) are positive. The last term is seen to be positive by using the induction hypothesis along with Lemma 3.2. ▪

Proposition 3.4 *For each $n, V_n(s)$ is convex.*

Proof: There are only a finite number of possible developmental strategies, at most 3^ρ ($\rho = N \times max\{p,q\}$, where $d = p/q$) in the N device problem, and $V_N(s)$ is the maximum of that finite number of linear functions of s. ▪

Using convexity and the fact that $V_n(1) = nr(1) = n(1-p_2)$, we have

Proposition 3.5 *If $s \in [0,1]$ is such that $V_n(s) = nr(s)$, then for all $s \leq t \leq 1$,*

$$V_n(t) = nr(t).$$

Proposition 3.5 and the fact that $V_n(1) = nr(1) = n(1-p_2)$ imply that the set of s where $V_n(s) = nr(s)$ is a nonempty interval containing 1. (That is, for every n we "build" for large enough s.) Let s_n^* be the first point at which $V_n(s) = nr(s)$. We now show that the sequence of cut-off points is monotone nondecreasing.

Proposition 3.6 *The sequence of cutoff points, $s_1^*, s_2^*, \ldots$, is such that*

$$s_1^* \leq s_2^* \leq s_3^* \leq \ldots$$

Proof: The proof here is adapted (for this model) from one of a similar result for Huang's model given in [13].

It is sufficient to show that if $s \leq s_n^*$, then it is also true that $s \leq s_{n+1}^*$.

Suppose that the budget is sufficient to build $(n+d+1)$ devices. Then we may proceed by building one device, setting it aside, and consider the developmental program arising from a budget of $n+d$. Thus we have

$$V_{n+d+1}(s) \geq V_{n+d}(s) + r(s). \tag{3.13}$$

If $s < s_{n+d}^*$, either a test or a design will be performed. First, suppose making a test is the initial optimal action, given the current s. Using (3.4) combined with (3.13) we have

$$\begin{aligned} f_{n+d}(s) &= r(s)V_{n+d}(\eta_0(s)) + (1-r(s))V_{n+d}(\eta_1(s)) \\ &\geq r(s)[V_{n+d-1}(\eta_0(s)) + r(\eta_0(s))] + (1-r(s))[V_{n+d-1}(\eta_1(s)) + r(\eta_1(s))] \\ &= f_{n+d-1}(s) + r(s), \end{aligned}$$

using Lemma 3.1.

Since $s < s_{n+d}^*$, and a test is the optimal first step, $f_{n+d-1}(s) \geq (n+d)r(s)$. Using this with the calculation just completed

$$\begin{aligned} f_{n+d}(s) &\geq f_{n+d-1}(s) + r(s) \\ &\geq (n+d)r(s) + r(s) \\ &= (n+d+1)r(s), \end{aligned}$$

which implies that $s \leq s_{n+d+1}^*$.

Now, suppose that a design is the optimal initial activity at the current s value. Then (3.13) can be written as

$$V_{n+d+1}(s) \geq V_n(\delta(s)) + r(s).$$

Since $s < s^*_{n+d}$ and a design is the prescribed first activity, we have that

$$V_n(\delta(s)) \;\geq\; (n+d)r(s)$$

and so

$$V_{n+d+1}(s) \geq (n+d)r(s) + r(s) = (n+d+1)r(s),$$

which again implies that $s \leq s^*_{n+d+1}$. ■

Note that Proposition 3.1 says that s^*_n is zero when $n \leq d$ and that s^*_{d+1} is given by s_c of (3.11). For larger n, we determine s^*_n and the developmental activities for $s < s^*_n$ using the methods of the next section.

3.3 Calculation of $V_n(s)$

3.3.1 Method

As mentioned earlier, the recursive definition (3.3) is directly useful for computing V_n only for small values of n, values less, certainly, than 30. But the theory of the previous section allows accurate and fast computation of V even for very large n. Our approach is as follows.

Given a set of parameters — p_1, p_2, u, N, and $d = p/q$ — we compute and store the values of $V_1(s) = r(s)$, for s on a grid of 100,001 equally spaced points across $[0, 1]$. Using (3.3) on page 11 and linear interpolation, we compute and store $V_2, V_3, \ldots, V_N$. The method requires that we save only the last $j = [\max\{p, q\} + 1]$ arrayed V_n. The rational design cost can be looked upon as changing the size of the initial budget from N devices to Nq "pieces," where $d = p/q$. Actually, since we require at least 1 device, the budget has $(N-1)q + 1$ pieces. Let m be the number of budget pieces currently under consideration. We will compute V_m for m ranging from q to Nq. Using (3.3) to compute V_m, the "testing" function (f_{n-1}) will require linear interpolation in V_{m-q}, while the "redesign" function ($V_{n-d}(\delta(s))$) will be computed using linear interpolation in the values of V_{m-p}. Thus, at each step we will need to store at most the last j functions V. A pseudo-code computer program to compute $V_n(s)$ using this method is given in Appendix.1.

Some examples will clarify. For the first example, suppose the initial budget is $N = 4$ devices and a redesign costs $d = 3/2$ devices. We consider a total budget of $(4-1)2+1 = 7$ pieces. Let m be the number of budget pieces currently under consideration so that m ranges from 2 (the minimum budget of 1 device) to 8 (the initial budget of 4 devices). Table 3.1 shows how we compute $V_8(s)$ and which subproblems are stored at each step. When $m-p<q$ (here: $m-3<2$), a redesign is not permitted since less than 1 device will result from considering these cases. When $m-q<q$ (here: $m-2<2$), a test is not permitted for the same reason. The subscript on the "test" function f from (3.4) on page 11 is changed to reflect that 1 device is now q "pieces." Similarly, the subscript on the "redesign" function $V(\delta(s))$ is written to correspond to looking p "pieces" back in the computed and stored V function values.

Table 3.1 is to be read as follows. The column titled "Array" gives the numbers of

Table 3.1: Computing V_4, $d = 3/2$

Array	m	$V_m(s)$	m	$V_m(s)$
1	2	$\max\{r(s)\}$	6	$\max\{3r(s), f_4(s), V_3(\delta(s))\}$
2	3	$\max\{r(s)\}$	7	$\max\{3r(s), f_5(s), V_4(\delta(s))\}$
3	4	$\max\{2r(s), f_2(s)\}$	8	$\max\{4r(s), f_6(s), V_5(\delta(s))\}$
4	5	$\max\{2r(s), f_3(s), V_2(\delta(s))\}$		

the stored arrays needed to compute V_m. For example, to compute V_5 one selects the largest of the values $\lfloor 5/2 \rfloor r(s) = 2r(s)$, $f_{5-2}(s) = f_3(s)$, and $V_{5-3}(\delta(s)) = V_2(\delta(s))$. ($\lfloor x \rfloor$ = Integer part of x.) The $2r(s)$ term is calculated using (2.2) while the $f_{m-q}(s)$, and $V_{m-p}(\delta(s))$ terms are calculated using linear interpolation between the stored values in the Array containing the appropriate V values. Thus, V_5 will require linear interpolation in the values of V_3 (stored in Array 2) and V_2 (stored in Array 1). Once the values stored in an array have been used, they can be overwritten. Hence, V_6 (which will interpolate in V_3 and V_4) will use Arrays 2 and 3 and can be stored in Array 1.

The second example is provided to illustrate that the number of stored arrays must be one greater than maximum of the numerator and denominator of d, the design cost. Again, we consider an initial budget of $N = 4$ devices but now choose $d = 2/3$. The arrays needed to compute $V_{N_q} = V_{12}$ are depicted in Table 3.2. In this example the number of budget pieces m ranges from 3 to 12 and now the testing function (f_{n-1}) will look back 3 steps while the redesign function $(V_{n-d}(\delta(s)))$ looks back only 2 steps. Note that V_{11} and V_{12} are stored in Arrays 1 and 2, respectively.

Table 3.2: Computing V_4, $d = 2/3$

Array	m	$V_m(s)$	m	$V_m(s)$
1	3	$\max\{r(s)\}$	7	$\max\{2r(s), f_4(s), V_5(\delta(s))\}$
2	4	$\max\{r(s)\}$	8	$\max\{2r(s), f_5(s), V_6(\delta(s))\}$
3	5	$\max\{r(s), r(\delta(s))\}$	9	$\max\{3r(s), f_6(s), V_7(\delta(s))\}$
4	6	$\max\{r(s), f_3(s), V_4(\delta(s))\}$	10	$\max\{3r(s), f_7(s), V_8(\delta(s))\}$
1			11	$\max\{3r(s), f_8(s), V_9(\delta(s))\}$
2			12	$\max\{4r(s), f_9(s), V_{10}(\delta(s))\}$

3.3.2 General Fixed Test Costs

As mentioned in Section 3.1, we can compute $V_N(s)$ with any test cost. We use the method of Section 3.3.1.

Suppose now that a test costs $t = p_t'/q_t$ devices and a redesign costs $d = p_d'/q_d$ devices (the primes will be dropped). We assume that both fractions are in lowest terms (i.e., in each one, the numerator and denominator do not share any common factors) and express the test and redesign costs in terms their least common denominator.

Let q be the least common multiple of $\{q_t, q_d\}$ so that the test and redesign costs can be expressed as $t = p_t/q$ and $d = p_d/q$, respectively. Proceeding as above (Section 3.3.1), the initial budget of N devices becomes Nq "pieces."

We compute

$$V_n(s) = \max\{\lfloor n/q \rfloor r(s),\ f_{n-p_t}(s),\ V_{n-p_d}(\delta(s))\}, \tag{3.14}$$

($\lfloor x \rfloor$ is the integer part of x) where n ranges from q (1 device) to Nq and, to start, $V_q(s) = r(s)$.

If $n - p_t > q$, then (a test can be bought)

$$f_{n-p_t}(s) = r(s)V_{n-p_t}(\eta_0(s)) + (1 - r(s))V_{n-p_t}(\eta_1(s)) \tag{3.15}$$

and otherwise f is zero. A redesign *cannot* be purchased if $n - p_d < q_d$, and so then $V_{n-p_d}(\delta(s)) = 0$.

To calculate V_n on a computer will require storage of the previous $\max\{p_t, p_d\} + 1$ computed values of V for interpolation. The program outlined in Appendix.1 is easily modified to compute V when a test costs t devices. The necessary modifications are given at the end of Appendix.1.

3.3.3 Behavior of V

3.3.3.1 Qualitative Analysis of the Effect of Considering Redesign Costs To explore how design costs affect the model, we simulated the development process for a factorial set of the parameters n, p_1, p_2, u, d (as a percent of n), and s_0. The values of p_2 included were chosen by setting $m = p_2/p_1$. The values we considered are given in Table 3.3. This gives a total of 972 ($= 3^5 \times 4$) different problems. Use of an optimal strategy was simulated 5,000 times for each combination of parameters. A description of the computer program and a discussion of the random number generator used to perform these simulations is given in Appendix.2.

Table 3.3: Values of Parameters Used in Factorial Study

Parameter	Values
n	100, 500, 1000
p_1	0.2, 0.5, 0.8
m	0.1, 0.3, 0.5
u	0.01, 0.1, 0.5
d	0.01, 0.02, 0.05
s	0.05, 0.25, 0.5, 0.75

Huang [12] demonstrated that the optimal development process can significantly improve a final stockpile — in some cases, the process is expected to more than triple (indeed, almost *quadruple*) the expected number of effective systems. In the current model, with two activities to choose from, we would like to know what mix of activities achieves the expected optimal yield, $V_n(s)$. An example will help make this clear and Table 3.4 contains the summary output from four of the problems simulated, the ones with parameters $(p_1, p_2, u, n, d) = (0.8, 0.4, 0.5, 1000, 5\%)$ and

$s_0 \in \{0.05, 0.25, 0.50, 0.75\}$.

Each of the four sections of the table contains the following. The first line in each section describes the initial values of the problem summarized in that section: the starting point (s_0), the value of equation (3.3) ($V(s_0)$) at this point, the initial "build" value ($Nr(s_0) = 1000r(s_0)$), and the 1st Action — which activity ("redesign," "test," or "build") is the optimal first step given the current s_0.

The **Measure** column gives the names of the quantities for which we computed the minimum, maximum, mean, standard deviation, and standard error during the 5,000 simulated development processes. The quantities recorded are as follows.

Given the initial value s_0, "Final s" is the value to which it grew during the development process (the mean reliability growth attained in each problem can be computed as the difference, $r(\bar{s}) - r(s_0)$). (Here, $\bar{x}$ refers to the mean value of x.) "Yld(D)" is the number of *devices* ultimately "built" (stockpiled/delivered at the end of the development process) and is $N-$DevCost (development cost) (in Table 3.4, $N = 1000$). The expected number of *effective devices* is given by "Yld(E)"which is, on average, $V(s_0)$, and is approximately $\overline{\text{Yld(D)}}r(\bar{s})$. The last three rows of each section of Table 3.4 summarize the number of tests, redesigns, and the total development cost for these actions, in the rows named "nTest," "nDesign," and "DevCost," respectively.

As one would expect, as the initial probability of being in state 2, s_0, increases, so does the expected yield (V) and the likelihood of achieving good reliability, both at lesser cost as well.

The improvement in the effective yield, the effect of the development process, decreases as s_0 increases. This can be measured, roughly, as the ratio of $V_n(s_0)/nr(s_0)$

Table 3.4: Example of simulation results of 5,000 runs with parameters: $(p_1, p_2, u, n, d) = (0.8, 0.4, 0.5, 1000, 5\%)$

Measure	Min	Max	Mean	S.Dev.	S.Err.
$(s_0, V(s_0), 1000r(s_0),$ 1st Action$) = (0.05, 525.57, 220.00,$ Redesign$)$					
Final s	0.5198	0.9971	0.9939	0.0072	0.0001
Yld(D)	112.00	945.00	878.66	82.0095	1.1598
Yld(E)	45.69	565.60	525.14	49.3682	0.6982
nTest	5.00	138.00	20.7056	14.3728	0.2033
nDesign	1.00	15.00	2.0126	1.4057	0.0199
DevCost	55.00	888.00	121.3356	82.0095	1.1598
$(s_0, V(s_0), 1000r(s_0),$ 1st Action$) = (0.25, 536.44, 300.00,$ Test$)$					
Final s	0.9751	0.9973	0.9942	0.0021	0.0000
Yld(D)	343.00	994.00	897.45	85.2290	1.2053
Yld(E)	202.38	594.77	536.42	51.2316	0.7245
nTest	6.00	121.00	22.8642	14.5702	0.2061
nDesign	0.00	11.00	1.5938	1.4762	0.0209
DevCost	6.00	657.00	102.5542	85.2290	1.2053
$(s_0, V(s_0), 1000r(s_0),$ 1st Action$) = (0.50, 553.63, 400.00,$ Test$)$					
Final s	0.9686	0.9973	0.9945	0.0020	0.0000
Yld(D)	347.00	995.00	926.39	80.0052	1.1314
Yld(E)	204.32	595.37	553.81	48.1135	0.6804
nTest	5.00	103.00	21.2308	14.0163	0.1982
nDesign	0.00	11.00	1.0476	1.3735	0.0194
DevCost	5.00	653.00	73.6108	80.0052	1.1314
$(s_0, V(s_0), 1000r(s_0),$ 1st Action$) = (0.75, 572.65, 500.00,$ Test$)$					
Final s	0.9114	0.9973	0.9948	0.0022	0.0000
Yld(D)	232.00	996.00	956.61	71.3897	1.0096
Yld(E)	130.98	595.97	572.00	42.9535	0.6075
nTest	4.00	123.00	16.4408	13.7489	0.1944
nDesign	0.00	13.00	0.5390	1.1956	0.0169
DevCost	4.00	768.00	43.3908	71.3897	1.0096

and will be discussed further, below. In Table 3.4, the value of this ratio is 2.39 when $s_0 = 0.05$ and when $s_0 = 0.75$, the ratio has decreased to 1.14.

Our interest in these simulations, the question regarding how the optimal developmental process transpires, is answered, in summary fashion, by the remainder of the table.

Specifically, consider the third section of Table 3.4, where $s_0 = 0.50$. The expected effective yield is $V(0.50) = 553.63$, while initially "building" would give an expected effective yield of $1000 \times r(0.50) = 400$ $(= 1000(.5(1-.8)+.5(1-.4)))$. The initial activity is to perform a test. During the 5,000 runs, on average, s_0 grew from 0.50 to 0.9945, and the design reliability grew from $r(0.50) = 0.4$ to $r(0.9945) = 0.5978$. (The best achievable reliability in this problem is $1 - p_2 = 0.60$.) This growth is accomplished by a process which, on average, consists of about 21.2 tests and 1.048 redesigns (in this problem, redesigns cost $0.05(1000) = 50$ devices each) for a (mean) total development cost of $21.2 + (1.048(50)) = 73.6$ devices.

We can use the output from all the simulated development processes to give a rough description of how the parameters affect V_n. Proceeding as in Huang [12] and Huang, McBeth, and Vardeman [13], we define

$$G_n = \frac{V_n(s)}{nr(s)},$$

to make a simple tool for investigating the changes in the effectiveness of the final stockpile due to the development process. Further, we examine the costs involved in the development processes as a proportion of the initial budget, and so define

$$C_n = \frac{\text{DevCost}}{n}.$$

Tables 3.5 and 3.6 give frequency counts for these two measures for each of the initial

Table 3.5: Frequencies of G_n, by n

	n		
Interval	1000	500	100
[3.5, 4.0)	2	2	2
[3.0, 3.5)	5	5	3
[2.5, 3.0)	5	5	5
[2.0, 2.5)	13	9	10
[1.5, 2.0)	22	24	16
[1.4, 1.5)	12	11	10
[1.3, 1.4)	9	11	13
[1.2, 1.3)	22	18	13
[1.1, 1.2)	30	31	32
(1.0, 1.1)	77	73	69
[1.0, 1.0]	127	135	151

Table 3.6: Frequencies of C_n, by n

	n		
Interval	1000	500	100
[0.400, 0.500)	4	4	2
[0.350, 0.400)	2	2	1
[0.300, 0.350)	6	5	4
[0.250, 0.300)	1	3	5
[0.200, 0.250)	12	12	9
[0.150, 0.200)	10	7	16
[0.125, 0.150)	9	14	14
[0.100, 0.125)	25	23	16
[0.075, 0.100)	15	12	19
[0.050, 0.075)	30	34	40
[0.250, 0.050)	56	57	36
(0.000, 0.025)	27	17	12
[0.000, 0.000]	127	134	150

budgets $n \in \{100, 500, 1000\}$.

Comparing the last two rows in Tables 3.5 and 3.6, we see that there were two problems (one with $n = 500$, the other with $n = 100$) in which no reliability growth (or, more precisely, no growth in effectiveness of the yield) occurred *but* developmental costs were required! The simulation results for the two problems in which this occurred are given in Table 3.7 and are summarized below.

In the first problem in Table 3.7, $s_0 = 0.50$ is less than $s^*_{100} = 0.50051$ (s^* is not in the table and was computed elsewhere) and we see that s_0 is in a "redesign" region. A single redesign gives $s_1 = 0.55$ which is compared to $s^*_{99} = 0.4959$ (the resulting problem's cut-off point). Since s_1 exceeds s^*_{99}, the development process ceases.

For the second problem presented in Table 3.7, the development process can be considerably more involved. In this problem, the mean developmental cost is 23.05 devices which are spent, on average, to buy 4.69 tests and 3.673 redesigns

Table 3.7: Simulation results of two problems with positive developmental costs and no reliability growth

Measure	Min	Max	Mean	S.Dev.	S.Err.
$(p_1, p_2, u, n, d) = (0.20, 0.02, 0.10, 100, 1\%)$					
$(s_0, V(s_0), 100r(s_0),$ 1st Action$) = (0.50, 89.00, 89.00,$ Redesign$)$					
Final s	0.5500	0.5500	0.5500	0.0000	0.0000
Yld(D)	99.00	99.00	99.00	0.0000	0.0000
Yld(E)	89.00	89.00	89.00	0.0000	0.0000
nTest	0.00	0.00	0.0000	0.0000	0.0000
nDesign	1.00	1.00	1.0000	0.0000	0.0000
DevCost	1.00	1.00	1.0000	0.0000	0.0000
$(p_1, p_2, u, n, d) = (0.80, 0.40, 0.01, 500, 1\%)$					
$(s_0, V(s_0), 500r(s_0),$ 1st Action$) = (0.75, 250.00, 250.00,$ Test$)$					
Final s	0.0722	0.9000	0.7754	0.2351	0.0033
Yld(D)	311.00	499.00	476.95	52.8151	0.7469
Yld(E)	71.64	279.44	248.20	60.4912	0.8555
nTest	1.00	48.00	4.6872	6.7237	0.0951
nDesign	0.00	34.00	3.6730	9.3889	0.1328
DevCost	1.00	189.00	23.0522	52.8151	0.7469

(here, redesigns cost 5 devices each). At the end of the development process, s has improved slightly from its initial value of 0.75 to 0.7754 (mean value). The first action is a test. If this test is a success, the development process stops. However, if this test is failure, the process could prove to be very expensive — the worst case observed results in a development process that consumes 189 devices! This problem is an example of a trend we observed when studying the whole set of simulation results graphically using *XGobi*, a computer package developed for the purpose of enabling visualization of high dimensional data. We now summarize those observations.

For all three budget sizes, n, the best growth (above 3.5) occurs when, not surprisingly, $(p_1, p_2, u, s_0) = (0.8, 0.08, 0.5, 0.05)$ and $d \in \{1\%, 2\%\}$. These are the failure and redesign probabilities and redesign costs that allow the greatest potential for reliability growth. These problems have the cheapest optimal policies as well. The poorest performances (in terms of "low" G and "high" C) come from problems which also have $s_0 = 0.05$, but with probabilities $p_1 = 0.8$, $p_2 \in \{0.08, 0.4\}$, and $u \in \{0.01, 0.1\}$. These problems are the most expensive and give only mediocre growth, in the interval $[1.30, 2.21]$. The problems which have 1% and 5% redesign costs, in general, have more expensive development programs than those problems in which the design cost is 2%.

We tried to characterize the optimal development process by way of simple *rules.* By a developmental rule we mean that we will perform the activities prescribed, in the order given, and then "build." Using "T" and "R," for "perform a test," and "perform a redesign," respectively, we ran a large number of problems and compared the expected effective yields from the five rules: TR, RT, TTR, TRT, and RTT to V_n. Additionally, we looked at a number of problems using the rule, "*perform k tests* *in a row, then a redesign, and then* "build"", for $k \in \{1, 2, \ldots, 10\}$. Our efforts to characterize effective development processes by way of these rules did not reveal any interesting observations or trends.

Given a set of problem parameters, the optimal developmental policy according to goal (2.1) can be computed by the methods of this chapter quickly, in linear time. The lesson of the analysis given here, that is, compute the policy for any problem in which one is interested, is reinforced in an interesting, and rather surprising way, in the next section.

3.3.3.2 Surprising Behavior It seems natural to expect that, for a fixed budget n, as s increases across $[0, 1]$, the recommended action would change from "redesign" to "test" to "build." That is, if one thinks the chances are poor that acceptable reliability has been achieved, perform a redesign; if one is uncertain about the reliability, another test will help to estimate it; finally, if one is confident that the reliability is good or cannot be improved — build.

For many sets of parameter mixes, this is indeed the course of action prescribed. One can devise parameter vectors where the optimal behavior is always "build," or to "redesign and then build," or to "test and then build." (As is assured by Proposition 3.5 on page 17, the recommended behavior will be "build" for large enough s.) But the conjecture that is intuitively appealing, namely that "usually" we "redesign for small s," "build for large s," and "test for s in between" is, however, false! The "test" and "redesign" regions may be interleaved as demonstrated by Figure 3.1. The splitting of the redesign region in this problem occurs in V_6 where, for a short interval, the optimal move is to test. At the redesign/test change-over point, the slope increases from 2.4576 to 3.7839. The difference in final yield, if a redesign and not a test was performed for s in the test interval, is inconsequential as can be seen from the graph of V_6 in Figure 3.2.

The conjecture that this behavior occurs only because both redesign and test have unit cost (1 device) is also false. We have found an instance where there are repeated changes between "redesign" and "test" in V_{61} with redesign cost of $d = 5$* (devices) and $(p_1, p_2, u) = (0.8, 0.4, 0.1)$. We have computed in this problem using one hundred thousand, five hundred thousand, and 1 million points s and as the density of our grid grows, the number of crossovers does too. For $s \in (0.1749, 0.2969)$, the optimal developmental behavior is chaotic. The recommended action changes repeatedly between "test" and "redesign." A graph of V_{61} and a blow-up of its behavior for $s \in [0.1749, 0.1800]$ is given in Figure 3.3.

The graph in Figure 3.4 shows the values of s at which the developmental activity changes. For s less than 0.17497 the recommended strategy is "redesign." On the graph, the point $(0.17497, 1)$ signals "test." Now the recommended action changes repeatedly. On the interval $[0.17497, 0.17499)$ — test. When $s \in [0.17499, 0.1751)$ — redesign, for $s \in [0.1751, 0.17511)$ — test, $s \in [0.17511, 0.17524)$ — redesign, and so on. This phenomenon is very interesting. It is counterintuitive that the corner points of the sub-problems would be so numerous.

3.3.4 Regressive Redesigns

In our non-regressive model, the posterior probability of being in state 2 after a redesign was given by (3.2) on page 10, $\delta(s)$.

To model the possibility of regressive designs, let u_j be the probability that a

$$V_1(s) = r(s) = \frac{48s+1}{50}$$

$$V_2(s) \;=\; \begin{cases} \dfrac{192s+53}{250}, & s \leq \dfrac{43}{288} \quad \text{(Design)} \\[2ex] \dfrac{48s+1}{25}, & \dfrac{43}{288} < s \quad \text{(Build)} \end{cases}$$

$$V_3(s) \;=\; \begin{cases} \dfrac{192s+53}{125}, & s \leq \dfrac{13}{48} \quad \text{(Design)} \\[2ex] \dfrac{3(48s+1)}{50}, & \dfrac{13}{48} < s \quad \text{(Build)} \end{cases}$$

$$V_4(s) \;=\; \begin{cases} \dfrac{768s+457}{625}, & s \leq \dfrac{17}{192} \quad \text{(Design)} \\[2ex] \dfrac{3(192s+53)}{250}, & \dfrac{17}{192} < s \leq \dfrac{139}{384} \quad \text{(Design)} \\[2ex] \dfrac{2(48s+1)}{25}, & \dfrac{139}{384} < s \quad \text{(Build)} \end{cases}$$

$$V_5(s) \;=\; \begin{cases} \dfrac{3(768s+457)}{1250}, & s \leq \dfrac{311}{1536} \quad \text{(Design)} \\[2ex] \dfrac{2(192s+53)}{125}, & \dfrac{311}{1536} < s \leq \dfrac{187}{432} \quad \text{(Design)} \\[2ex] \dfrac{48s+1}{10}, & \dfrac{187}{432} < s \quad \text{(Build)} \end{cases}$$

Figure 3.1: Explicit form of $V_6(s)$ which shows "splitting" in redesign region.
$(p_1, p_2, u, N, d) = (.98, .02, .2, 6, 1)$

$$V_6(s) = \begin{cases} \dfrac{3(3072s+3053)}{6250}, & s \le \dfrac{19}{6144} \quad \text{(Design)} \\ \dfrac{2(768s+457)}{625}, & \dfrac{19}{6144} < s \le \dfrac{502}{1727} \quad \text{(Design)} \\ \dfrac{2(29562s+8413)}{15625}, & \dfrac{502}{1727} < s \le \dfrac{527}{1752} \quad \text{(Test)} \\ \dfrac{192s+53}{50}, & \dfrac{527}{1752} < s \le \dfrac{47}{96} \quad \text{(Design)} \\ \dfrac{3(48s+1)}{25}, & \dfrac{47}{96} < s \quad \text{(Build)} \end{cases}$$

Figure 3.1: (continued) Split V_6 – with "test" between two "redesign" regions

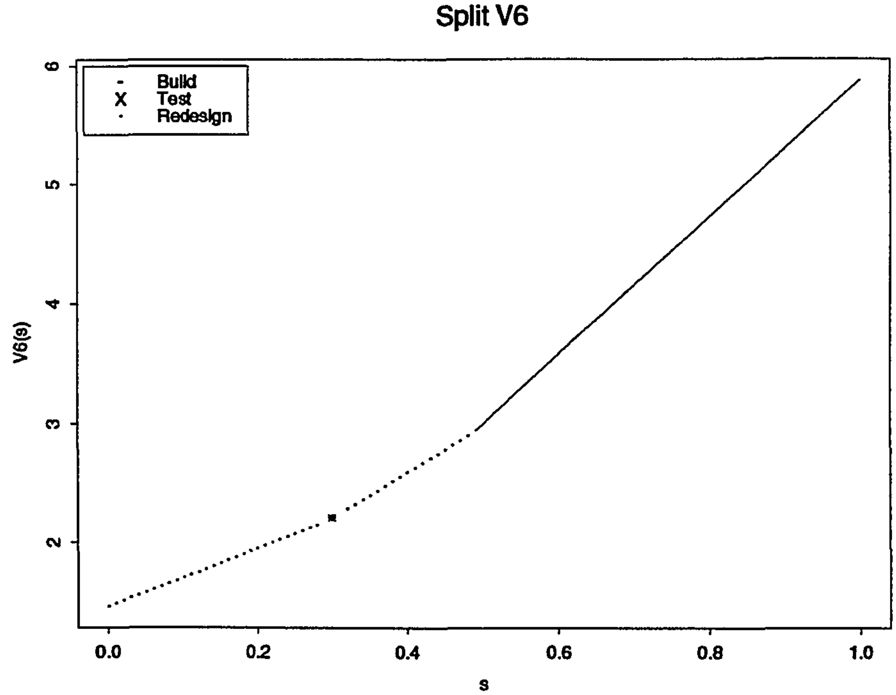

Figure 3.2: Graph of $V_6(s)$ computed in Figure 3.1. Notice the "test" region at $s \approx 0.30$.

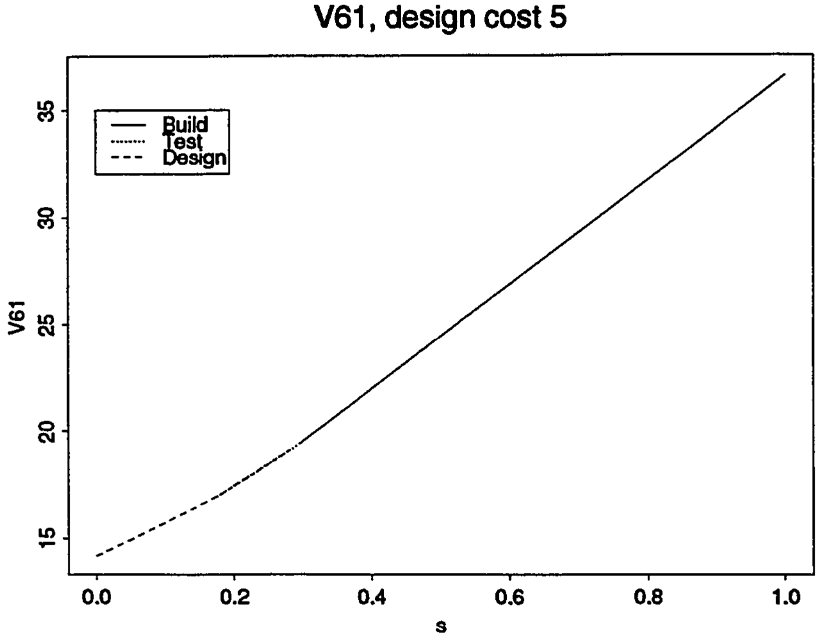

Figure 3.3: Graph of V_{61}. Test and redesign region interleaved

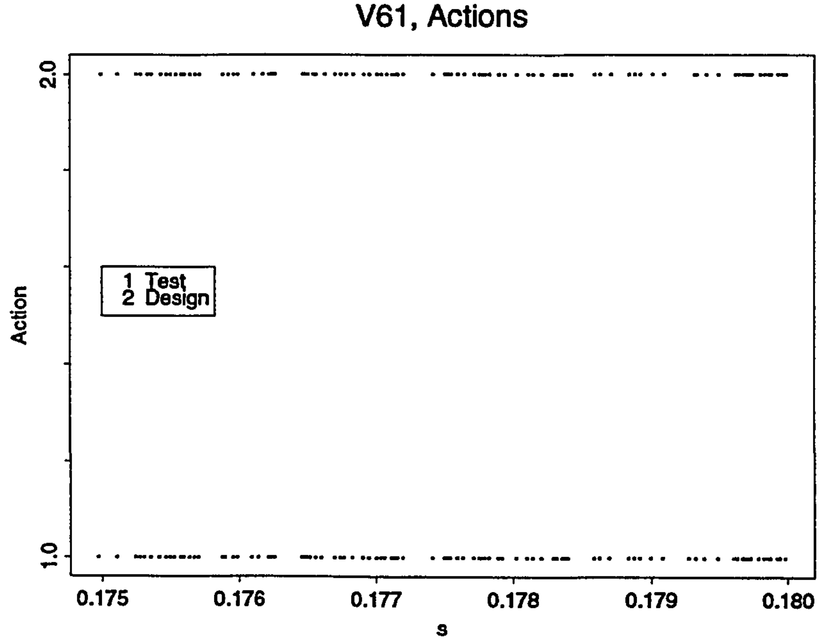

Figure 3.4: Example of chaotic behavior in V_{61}. The behavior occurs for $s \in [0.1749, 0.2969]$.

redesign performed when reliability is in state j results in good design reliability. The situation is illustrated in Figure 3.5. As always, s is the probability of being in the good reliability state. After a redesign is completed, the design may have attained good reliability in one of two ways: if the reliability was poor before, a successful redesign improved it (probability of $u_1(1-s)$) or, an unnecessary redesign is performed but it doesn't harm the reliability (probability of $u_2 s$). So, replacing (3.2), the update of s after a potentially harmful design is

$$\delta_2(s) = u_1(1-s) + u_2 s = u_1 + s(u_2 - u_1). \tag{3.16}$$

If we pick $u_1 = u$ (from (3.2)) and, for reliability growth, require $u_2 > u_1$, the effect, as expected, is to slow the rate at which the reliability improves.

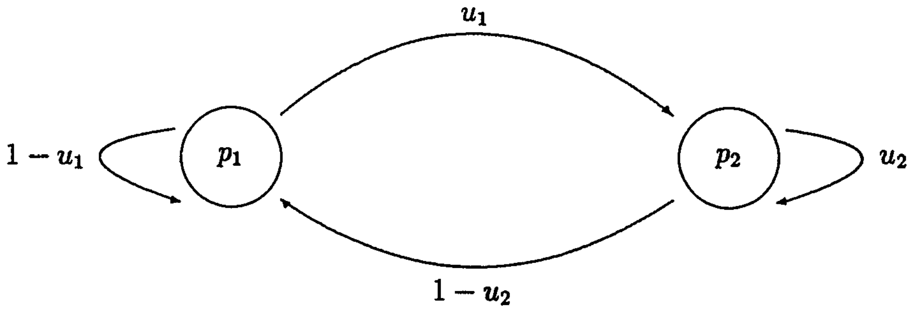

Figure 3.5: Two state model with u_i giving the probability that a redesign performed in state i will yield good reliability.

We don't present the details of the theory here as the arguments used thus far in this chapter don't change significantly. That is, the theory and analysis from Section 3.2 carry over to this model with very few adjustments. The slope of $r(\delta(s))$ in (3.10) becomes $(u_1 - u_2)(p_1 - p_2)$ and the redesign/build cut-off point for a remaining budget of $n = d + 1$ given in (3.11) becomes

$$s_c = \frac{u_1(p_1 - p_2) - d(1 - p_1)}{(p_1 - p_2)((d+1) - (u_2 - u_1))}.$$

The results of a number of calculations comparing the non-regressive with the regressive redesigns model are shown in Tables 3.8 and 3.9. For several combinations of (p_1, p_2, u, d), the function values (V_{500}) and their associated cut-off points, s^* (the smallest value of s at which "build" is recommended, i.e., $V_{500}(s^*) = 500r(s^*)$), are presented in Table 3.8. We include a number of different u values because it is not evident before comparing with the values of s^* and V in Table 3.9 which comparison is most appropriate. The u values given in Table 3.8 correspond to to the u_1 and $(u_2 - u_1)$ values considered in Table 3.9. In each case, the closest comparison between tables occurs when u_1 (from Table 3.9) is compared with u (in Table 3.8). Some very loose conclusions can be drawn.

As expected, the cut-off points and resultant yields are, in general, smaller in the regressive model. The most extreme difference occurs when $(p_1, p_2, d, u_1, u_2) = (0.5, 0.25, 10, 0.1, 0.7)$. In this case the yield from the regressive model is about 87% of that given from the non-harmful one. From the table we see that, unsurprisingly, the larger the value of u_2, the smaller the difference in yields between the two models. The larger the cost of the design or the larger the possibility for reliability growth (the difference between p_1 and p_2), the smaller the difference in yield between the two models.

3.3.5 High Reliability

Ekstrom and Allred [10] explain that new solid rocket motor systems must have *verified* (not simply predicted) reliabilities of $0.999x$. Table 3.10 examines the pos-

Table 3.8: Non-regressive cut-off points and values

p_1	p_2	d	u	s^*	V_{500}
0.5	0.25	5	0.1	0.8583	357.28
			0.4	0.9343	366.78
			0.5	0.9452	368.16
			0.6	0.9544	369.30
			0.8	0.9658	370.72
		10	0.1	0.6687	333.59
			0.4	0.9136	364.20
			0.5	0.9229	365.36
			0.6	0.9308	366.35
			0.8	0.9469	368.36
		25	0.1	0.0	250.00
			0.4	0.8510	356.37
			0.5	0.8806	360.08
			0.6	0.8976	362.20
			0.8	0.9186	364.83
0.8	0.08	5	0.1	0.9951	458.23
			0.4	0.99571	458.46
			0.5	0.99575	458.47
			0.6	0.99578	458.48
			0.8	0.99582	458.50
		10	0.1	0.9941	457.89
			0.4	0.9955	458.39
			0.5	0.99562	458.42
			0.6	0.99567	458.44
			0.8	0.9957	458.47
		25	0.1	0.9899	456.35
			0.4	0.9950	458.19
			0.5	0.9952	458.27
			0.6	0.9953	458.32
			0.8	0.9955	458.38

Table 3.9: Regressive cut-off points and values: values computed using (3.16) in (3.3). Compare the V_{500} values given here with those in Table 3.8.

p_1	p_2	d	u_1	u_2	s^*	V_{500}
0.5	0.25	5	0.1	0.7	0.6848	335.60
				0.9	0.7974	349.67
			0.5	0.9	0.9280	366.00
		10	0.1	0.7	0.3254	290.67
				0.9	0.5207	315.08
			0.5	0.9	0.9154	364.42
		25	0.1	0.7 0.9	0.0	250.00
			0.5	0.9	0.8733	359.16
0.8	0.08	5	0.1	0.7	0.99497	458.19
				0.9	0.99502	458.21
			0.5	0.9	0.9957	458.47
		10	0.1	0.7	0.9940	457.84
				0.9	0.9941	457.86
			0.5	0.9	0.9956	458.42
		25	0.1	0.7	0.9896	456.24
				0.9	0.9898	456.31
			0.5	0.9	0.9952	458.27

sibility of perceiving reliability changes in model (3.3) as highly reliable systems are developed to become those possessing "extremely" high reliability. The model, perhaps surprisingly, offers advice – to perform a redesign if s is low. However, since s is a probability, the model cannot *verify* conformance to reliability specifications, but rather, predicts compliance with them.

Table 3.10 demonstrates three interesting features of the model. All the problems examined had only two activities – the optimal behavior was always to redesign to the left of s^*. This is encouraging, since the possibility of a system failure is unlikely. Another way of considering these outcomes points to some of the artificiality of the model: if no testing is allowed, why redesign (potentially, repeatedly)? Models need to be developed in which the probability of a successful redesign is not fixed but is a function of the development process history.

Further, the larger the probability of a successful design, the later the model advises stopping the development process. This is especially evident in the last two rows of the table where $n = 2000$.

And finally, since the optimal process never recommends testing, when the design cost d and the budget n are in the same proportion, the problems are identical. This is apparent if, for example, with $(p_1, p_2) = (0.05, 0.01)$, one compares $n = 100, d = 1$ with $n = 500, d = 5$ $(d/n = 0.01)$, to see that the cut-off points are identical! How does this happen? The design costs and so the step sizes between subproblems are in proportion to each other when testing is not part of the optimal solution.

Table 3.10: Extremely High Reliability: In all problems examined, s^* is the point at which the recommended activity changes from design to build. Testing was never called for. Notice that if d and n are in the same proportion, then the cut-off point is unchanged as well.

p_1	p_2	n	d	u	s^*
0.05	0.01	100	1	0.25	0.0388
				0.5	0.5099
				0.75	0.6711
				1.0	0.7525
			2	0.25	0.0
				0.5	0.0294
				0.75	0.3444
				1.0	0.5050
			5	1.0	0.0
		500	1	0.25	0.8032
				0.5	0.9012
				0.75	0.9340
				1.0	0.9505
			2	0.25	0.6087
				0.5	0.8028
				0.75	0.8682
				1.0	0.9010
			5	0.25	0.0388
				0.5	0.5099
				0.75	0.6711
				1.0	0.7525
			10	0.25	0.0
				0.5	0.0294
				0.75	0.3444
				1.0	0.5050
			25	1.0	0.0

p_1	p_2	n	d	u	s^*
0.05	0.01	1000	1	0.25	0.9013
				0.5	0.9506
				0.75	0.9670
				1.0	0.9753
			2	0.25	0.8032
				0.5	0.9012
				0.75	0.9340
				1.0	0.9505
			5	0.25	0.5123
				0.5	0.7537
				0.75	0.8353
				1.0	0.8763
			10	0.25	0.0388
				0.5	0.5099
				0.75	0.6711
				1.0	0.7525
			25	0.25	0.0
				0.5	0.0
				0.75	0.1818
				1.0	0.3813

Table 3.10: (continued)

p_1	p_2	n	d	u	s^*
0.01	0.001	100	1	1.0	0.0
		500	1	0.25	0.1173
				0.5	0.5569
				0.75	0.7042
				1.0	0.7780
			2	0.25	0.0
				0.5	0.1155
				0.75	0.4088
				1.0	0.5560
			5	1.0	0.0
			10	1.0	0.0
			25	1.0	0.0
		1000	1	0.25	0.5573
				0.5	0.7782
				0.75	0.8520
				1.0	0.8890
			2	0.25	0.1173
				0.5	0.5569
				0.75	0.7042
				1.0	0.7780
			5	0.5	0.0
				0.75	0.2612
				1.0	0.4450
			10	1.0	0.0
			25	1.0	0.0
0.001	0.0005	500	1	1.0	0.0
		1000	1	1.0	0.0
		2000	1	1.0	0.0005
0.001	0.0001	2000	1	1.0	0.4445

CHAPTER 4. MULTIPLE–STATE RELIABILITY MODELS

4.1 The Need for a New Model

We have thus far assumed that the design has two reliability states: “good” and “bad.” While this is unrealistic for a development process which may improve system reliability in steps, it has several advantages. The model is an obvious and relatively easy first step in understanding the sequential nature of the decision problem; it has mathematical properties (piecewise linearity, convexity, etc.) which can be exploited to compute solutions quickly and accurately; and the current reliability is cheap to compute after each developmental step. Of these characteristics, the first and last were contemplated when the model was constructed. The mathematical properties were fortuitous.

The case where the *testing* response is not binary is not considered here, though it is certainly worthy of further study. Huang [12] began to examine this problem under the assumption that the test response was a continuous variable.

The expense of calculating $V_N(s)$ for more than two states appears, initially, to quickly grow astronomical. To see this, note that modeling a development process when the reliability may reside in more than two states requires another component of “$\underline{s}$” for each additional state. For example: to model a process with three possible reliability states we could let s_1 and s_2 give the probabilities that the design reliability was in state 1 or 2, respectively, and then $(1-s_1-s_2)$ would give the probability that this reliability currently resided in state 3. Ignoring the details of how the reliability $r(\underline{s})$ is updated, the recursion would require 6 test update functions (three states with two possible outcomes each) versus the current 2 (the ηs) and, depending on one's approach, at least 3 design update functions versus the present 1 (δ). Under the kind of analysis introduced in the previous section, V_n would have the form

$$V_n(\underline{s}) = \max\left\{\begin{array}{c} nr(\underline{s}), \\ \\ r(\underline{s})[V_{n-1}(\eta_{10}(s_1)) + V_{n-1}(\eta_{20}(s_2)) + V_{n-1}(\eta_{30}(s_3))] \\ \quad +(1-r(\underline{s}))[V_{n-1}(\eta_{11}(s_1)) + V_{n-1}(\eta_{21}(s_2)) + V_{n-1}(\eta_{31}(s_3))], \\ \\ s_1[V_{n-d}(\delta_{12}(s_1)) + V_{n-d}(\delta_{13}(s_2))] \\ +s_2 V_{n-d}(\delta_{23}(s_2)) \end{array}\right\},$$

where the η_{ij} compute the state updates when in state i and test response j is observed while the δ_{ij} give the probabilities the state moves from i to j. (Since we are not going to pursue this vein of computation, this kind of admittedly unpleasant notation will be of no further consequence.)

Continuing, *four* reliability states would require three $\underline{s}$ dimensions, 8 ηs and at least 6 δs. It seems that as the number of states grows the accompanying computational expense and complexity quickly grows beyond the realm of what can be handled.

To circumvent this vexing intractability, we propose a model which, ultimately requires only **two** variables to describe the k-state reliability growth problem.

4.2 Multiple-State Reliability Models

4.2.1 A General Multiple-State Model

A general model for the multiple-state reliability problem can be given as follows. Suppose the design reliability can be in one of k states. Let $\underline{p} = (p_1, p_2, \ldots, p_k)$ give the failure probabilities so that p_i = Pr(device failure when in state i). Let $\underline{s} = (s_1, s_2, \ldots, s_k) > \underline{0}$ contain the probabilities that the design currently has reliability in state i. As before, the development process consists of a sequence of decisions, at each step choosing from "redesign," or "test," until finally "build" is chosen. After a successful test, $\underline{s}$ is updated as

$$s_i^+ = \frac{s_i q_i}{\sum_{j=1}^{k} s_j q_j},$$

where $\underline{q} = \underline{1} - \underline{p}$. If the test produces a failure, the update is

$$s_i^+ = \frac{s_i p_i}{\sum_{j=1}^{k} s_j p_j}.$$

A redesign changes each element of $\underline{s}$ according to

$$s_i^+ = \sum_{j=1}^{k} s_j h_{ji}(s_j),$$

where $H = [h_{ji}(s_j)]$ is a $k \times k$ matrix of redesign functions appropriate for the problem.

The overall reliability at each step is given by

$$r(\underline{s}) = \sum_{j=1}^{k} s_j q_j.$$

To analyze a particular design development process using this model, we would proceed as discussed in Section 4.1. We have claimed to be able to do better than this and now, we show how to do so.

4.2.2 An Intrinsically Two-Variable Multiple-State Model

In Chapter 3 we described a design with 2 reliability states and s_n gave the probability of "good" reliability after a development process (tests and redesigns) costing n units. We now present a specialized version of the model presented in Section 4.2.1 which has a convenient transition structure. This new model will model reliability growth when the design can be in any one of k reliability states. While the model requires at least $(2k-1)$ *parameters*, it requires only two *variables* to describe the evolution of the development process — one that may be thought of as the total cost of the development process to date and another that may be thought of as representing the portion of this expense which increases device reliability (the difference of these two giving a measure of the cost for unsuccessful attempts to increase reliability).

Beginning with usual problem elements — an initial budget of N systems, test cost of 1 system, redesign expense of d systems, and current total development cost of n systems — we proceed as follows.

For $i = 1, 2, \ldots, k$, let p_i = Pr(system failure | state i), and for definiteness, $1 \geq p_1 \geq p_2 \geq \cdots \geq p_k \geq 0$. Let $0 < a_i$ (positive weights) and, as usual, $q_i = 1 - p_i$. In this discussion, we assume that the a_i give the initial probability distribution of the design reliability. In this regard, the requirement that $\underline{a}$ be strictly positive is essential. If any $a_i = 0$, then state i is effectively removed from the model. This is because if a_i is zero, then s_i in (4.1) (below) is too.

Given a development process state vector (X, Y), X and Y real numbers, we assume that the probability that the design reliability is in state i is

$$s_i(X,Y) = \frac{a_i q_i^X p_i^Y}{\sum_{j=1}^{k} a_j q_j^X p_j^Y}. \tag{4.1}$$

The design reliability at this point is given by

$$\begin{aligned} r(X,Y) &= \sum_{j=1}^{k} s_j(X,Y) q_j \\ &= \frac{\sum a_j q_j^{X+1} p_j^Y}{\sum a_j q_j^X p_j^Y}, \end{aligned} \tag{4.2}$$

which is the probability the design reliability resides in state i times the success rate in state i. As before, our goal is to determine n^*, the development cost, so that

$$E[(N - n^*) \cdot r(X_{n^*}, Y_{n^*})], \tag{4.3}$$

the final mean number of effective systems, is as large as possible.

The vector (X,Y) contains the information necessary to compute the probability that the device reliability is in state k at each step in the development process. Thus, we can consider this vector to be the "development state." From state (X,Y), a test moves one to $(X+1,Y)$ with probability $r(X,Y)$ (success) otherwise, to $(X,Y+1)$ with probability $1 - r(X,Y)$. When a redesign is completed, the state becomes $(X + h_0(X,Y),\ Y + h_1(X,Y))$. And, unlike the situation in the binary model of Chapter 3, a redesign may now increase, decrease, or leave the reliability unaffected by appropriate choice of functions h_0 and h_1. Figure 4.1 gives a graphical representation of the potential state changes.

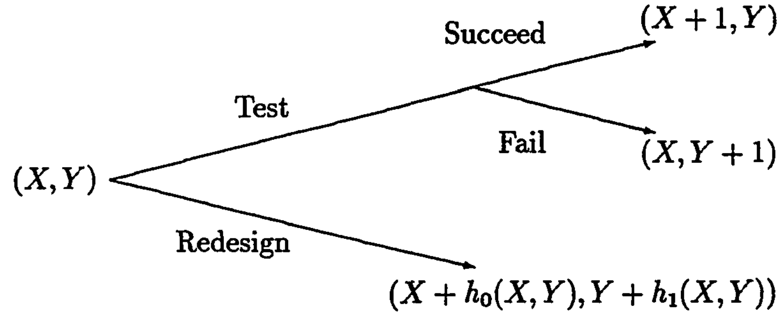

Figure 4.1: Development state changes in a two-variable, multiple-state reliability model

4.3 The Functional Equation For Optimal Developmental Testing in the Two-Variable Multiple-State Reliability Model

In Section 3.2 we produced functional equation (3.3) by considering the effect of each developmental activity upon the current budget. Using exactly the same kind of reasoning, given goal (4.3) and the updating on (X, Y) after a developmental action summarized in Figure 4.1, the dynamic programming equation becomes

$$V_n(X,Y) = \max \left\{ \begin{array}{c} nr(X,Y), \\ r(X,Y)V_{n-1}(X+1,Y) + (1-r(X,Y))V_{n-1}(X,Y+1), \\ V_{n-d}(X+h_0(X,Y),Y+h_1(X,Y)) \end{array} \right\}, \tag{4.4}$$

where $V_1(X,Y) = r(X,Y)$, is computed using (4.2).

4.4 The Binary Model of Chapter 3 as a Special Case of the Two-Variable Multiple-State Model

In light of the effort expended in Chapter 3, our first goal is to understand how that model might be realized in this new environment.

In the two-state model of Chapter 3, the p_i are the failure probabilities in state i and s_j is the posterior probability of being in state 2 after a j-step development program. During the development period (the sequence of redesigns and tests before building is recommended), the probability of residing in state 2 is updated according to (3.1) if step $j+1$ is a test and via (3.2) if this action is a redesign.

To demonstrate that the model of Chapter 3 is a special case of the model given in Section 4.2.2, we need to show that $s_i(X,Y)$ in (4.1) is equivalent to (3.1) and (3.2) after testing and redesign, respectively, and that the same value of s is given after each step.

To reproduce the Chapter 3 model in the terms of Section 4.2.2, choose the same p_i and let $(X_0, Y_0) = (0,0)$. We will discuss u, the probability of a successful redesign, below. Using (4.1), the initial probability of being in state 2, the "good" state, is

$$s_2(0,0) = \frac{a_2}{a_1 + a_2},$$

which corresponds to s_0 of Chapter 3, the initial probability that the design has good reliability. Corresponding to $(1 - s_0)$ we have

$$s_1(0,0) = 1 - s_2(0,0) = \frac{a_1}{a_1 + a_2}.$$

For convenience, suppress the subscripts and use $s(X,Y)$ and $1 - s(X,Y)$.

First we show that in terms of Section 4.2.2, relationship (2.2) holds. Equation

(4.2) says

$$\begin{aligned} r(X,Y) &= \frac{a_1 q_1^{X+1} p_1^Y + a_2 q_2^{X+1} p_2^Y}{a_1 q_1^X p_1^Y + a_2 q_2^X p_2^Y} \\ &= q_1(1 - s(X,Y)) + q_2 s(X,Y), \end{aligned}$$

using (4.1). But this is exactly relationship (2.2).

To demonstrate that the conditional probabilities after testing are equal, start by supposing that the first step is a test. If it is a success, then

$$s(1,0) = \frac{a_2 q_2}{a_1 q_1 + a_2 q_2},$$

which is $\eta_0(s)$ from (3.1), and if a failure, then

$$s(0,1) = \frac{a_2 p_2}{a_1 p_1 + a_2 p_2},$$

which is $\eta_1(s)$ from (3.1). Continuing by induction, the update on $s(X_i, Y_i)$ after a test is

$$\begin{aligned} s(X_{i+1}, Y_{i+1}) &= \begin{cases} s(X_i, Y_i) q_i, & \text{if the test produces a success} \\ s(X_i, Y_i) p_i, & \text{if the test produces a failure} \end{cases} \\ &= \begin{cases} \dfrac{q_2 s(X_i, Y_i)}{q_1(1 - s(X_i, Y_i)) + q_2 s(X_i, Y_i)} \\ \dfrac{p_2 s(X_i, Y_i)}{p_1(1 - s(X_i, Y_i)) + p_2 s(X_i, Y_i)} \end{cases}, \end{aligned}$$

which is $\eta_x(s_i)$ from (3.1) and the correspondence through testing is demonstrated.

Next we need to show how $\delta(s)$ from (3.2) can be incorporated into the update of $s(X,Y)$ after a redesign. We have

$$s(X,Y) = \frac{a_2 q_2^{X+h_0(X,Y)} p_2^{Y+h_1(X,Y)}}{a_1 q_1^{X+h_0(X,Y)} p_1^{Y+h_1(X,Y)} + a_2 q_2^{X+h_0(X,Y)} p_2^{Y+h_1(X,Y)}}$$

following a redesign. Let $h_1(\cdot) = 0$. We wish to determine the form of $h_0(\cdot)$ so that the redesign update on $s(X,Y)$ (see (3.2) on page 10) satisfies

$$\delta(s) = u + s(1-u) = \frac{a_2 q_2^{X+h_0(X,Y)} p_2^Y}{a_1 q_1^{X+h_0(X,Y)} p_1^Y + a_2 q_2^{X+h_0(X,Y)} p_2^Y}.$$

Solving for $h_0(\cdot)$, we have

$$h_0(X,Y) = \frac{ln\left(\dfrac{(1-u)q^X}{1+uaq^Xp^Y}\right)}{ln(q)} - X,$$

where $a = a_1/a_2$, $q = q_1/q_2$, $p = p_1/p_2$, and u is the probability of a successful redesign used in $\delta(s)$ from (3.2) on page 10 .

4.5 Choice of Design Functions

In the formulation of Section 4.2.2, we have included two design functions, $h_0(\cdot)$ and $h_1(\cdot)$, to allow the redesign portion of the developmental testing model to be as general as possible (under the assumption that each response is "success" or "failure"). The purpose of these functions is to compute the impact of a redesign on the vector (X,Y). In Section 4.4, we set $h_1(X,Y) = 0$ and derived a redesign function $h_0(X,Y)$ so that (4.4) and (3.3) were equivalent models. That is, for the model of Chapter 3, there is no need for two redesign functions. This demonstration raises at least two issues: 1) Does the model need two redesign functions? and 2) What types of redesign functions are reasonable? Considering the first of these questions, we proceed to show that the answer is "*No*, when $k = 2$, but *Yes*, for $k > 2$."

Lemma 4.1 *When $k = 2$, then one redesign function suffices in the reliability model (4.1) - (4.4).*

Proof: Let $k = 2$. Recall that equation (4.1) gives the probability that the reliability resides in state i when the development state is (X, Y). Suppose that there exists a function $\tilde{h}(X,Y)$ such that

$$s_i(X + \tilde{h}(X,Y), Y) = s_i(X + h_0(X,Y), Y + h_1(X,Y)).$$

The form of $\tilde{h}(X,Y)$ can be determined by solving

$$\frac{s_1(X + \tilde{h}(X,Y), Y)}{s_2(X + \tilde{h}(X,Y), Y)} = \frac{s_1(X + h_0(X,Y), Y + h_1(X,Y))}{s_2(X + h_0(X,Y), Y + h_1(X,Y))} \tag{4.5}$$

for $\tilde{h}(X,Y)$. Again let $q = q_1/q_2$ and $p = p_1/p_2$, substitute (4.1) into (4.5), and simplify to obtain

$$q^{X+\tilde{h}} p^{Y} = q^{X+h_0} p^{Y+h_1},$$

which yields

$$\begin{aligned} \tilde{h}(X,Y) &= h_0(X,Y) + h_1(X,Y) \left(\frac{\ln p}{\ln q}\right) \\ &= h_0(X,Y) + h_1(X,Y) \left(\frac{\ln p_1 - \ln p_2}{\ln q_1 - \ln q_2}\right). \end{aligned} \tag{4.6}$$

That $\tilde{h}$ is the unique solution to (4.5) when $k = 2$ follows from the fact that $\ln(1/x) = -\ln x$. ∎

Lemma 4.2 *If $k > 2$ and $h_0(X,Y) \neq 0$ and $h_1(X,Y) \neq 0$, then it is not possible to reduce a 2 redesign function model to a single redesign function model.*

Proof: Since the p_i are unique by definition, when $k > 2$ the ratio s_1/s_k will not give the function $\tilde{h}$ obtained in (4.6). ∎

CHAPTER 5. NUMERICAL SOLUTIONS OF THE TWO-VARIABLE MULTIPLE–STATE PROBLEM

Computing solutions to (4.4) using (4.1) and (4.2) is a much larger and harder problem than computing solutions to (3.3). Even if linear interpolation were to be used to compute solutions to (4.4) in a fashion analogous to the method used to compute solutions to (3.3), there are further difficulties. We don't know *a priori* what the range on X and Y is going to be. Of course, even without this issue, straight-away computation of (4.4) is more expensive than computing (3.3) since we must compute across a portion of the XY-plane, rather than just an interval.

The efforts described here are a first step to compute (4.4). The need for further research will be discussed later.

5.1 Computing $V_n(X, Y)$

The definition (4.4) can of course be used to compute V_n and identify optimal policies, but only for very small values of n ($n \leq 30$). Because of the recursion, the size of the calculation grows so rapidly that other methods must be developed to compute the optimal plan for larger budget n. We have developed two methods. The first, given appropriate assumptions, will compute V_n across a lattice, and the second is a myopic heuristic similar to one used by Huang [12].

5.1.1 Computing V_n on a Lattice

Suppose we have the usual set of problem parameters: an initial budget of N devices, k-vectors $\underline{a}$ and $\underline{p}$, and initial state (X_0, Y_0). We can compute $V_N(X,Y)$ when $h_1(X,Y) \equiv 0$, $h_0(X,Y)$ is a fixed positive integer and d, the design cost, is equal to p/q, for positive integers p and q.

The calculation proceeds as follows. Given initial state (X_0, Y_0) and budget N, the development process can spend at most $N-1$ devices. One can compute back from $V_N(X_0, Y_0)$ to $V_1(X,Y)$ and form a lattice of all possible states (X,Y) which will be needed. At each stage m $(m \geq q)$, the number of possible redesigns (say j) and tests (t) must satisfy the three equations:

$$\begin{aligned} tq + jp &= Nq - m, & (5.1) \\ t + jh_0 &= X + Y - (X_0 + Y_0), & (5.2) \\ X &\geq X_0 + jh_0. & (5.3) \end{aligned}$$

Equation (5.1) says that the amount of the budget spent, $Nq - m$, will equal a mix of tests and redesigns. Equation (5.2) relates the steps of the development process to the set of possible states. Relation (5.3) is included to make it clear that any design activity will be reflected in X.

An example will make this clear. Suppose that $N = 4$, $d = 3/2$, and (X_0, Y_0) and positive integer h_0, are all given. The lattice tables for this problem are given in Table 5.1 (each sub-budget m has its own table, we've put all the tables into one for convenience). We compute V_m for m from 2 to 8. The last rows $(m = 2)$ of Table 5.1 are computed using (4.2) and ascending rows are then computed using the values contained in the lower ones, as needed, by using (4.4) on page 47.

Table 5.1: Lattice points needed to compute $V_4(X_0, Y_0)$ when $d = 3/2$. The stage, m, is the number of budget pieces left, n is the number of devices remaining. The number of tests and designs required to be at stage m are given by T and D, respectively.

m	n	T	D	**Possible States**
8	4	0	0	(X_0, Y_0)
7	7/2	0	0	none
6	3	1	0	(X_0+1, Y_0), (X_0, Y_0+1)
5	5/2	0	1	(X_0+h_0, Y_0)
4	2	2	0	(X_0+2, Y_0), (X_0+1, Y_0+1), (X_0, Y_0+2)
3	3/2	1	1	(X_0+h_0+1, Y_0) , (X_0+h_0, Y_0+1)
2	1	3	0	(X_0+3, Y_0), (X_0+2, Y_0+1), (X_0+1, Y_0+2), (X_0, Y_0+3)
		0	2	(X_0+2h_0, Y_0)

Computing solutions to a k-state reliability problem using (4.4) is possible only for small budget n; the computation grows exponentially like 3^n. For fixed d and h_0, the time to compute $V_n(X, Y)$ with the lattice technique grows polynomially. (The total number of lattice points grows like n^3.) Using the *DEC Alpha* workstation, we would say that the lattice technique is useful for n less than 200. (Depending on the problem parameters, it can take more ***hours*** to compute a single value of a V_{200}. For example, it took approximately 70 hours to compute a single point of a 4-state $V_{500}(0, 0)$ with $d = h_0 = 2$ on a *Silicon Graphics Indy* workstation (100 MHz, 32 Meg RAM, a slightly faster computer than the *DEC Alpha* mentioned earlier).)

A pseudo-code computer program to compute $V_n(X, Y)$ subject to the restrictions discussed in this section is given in Appendix.3.

Solutions to large problems can always be quickly approximated, and in some cases, given exactly, using the look ahead methods of the next section.

5.1.2 Look Ahead Rules

We now examine two suboptimal stopping rules which result from the "myopic" or look ahead principle discussed in [12] and [13]. That is, for purposes of making the current decision, we will temporarily entertain further development costs of at most j "steps" (to be defined below) and subject to this constraint, initially proceed optimally with testing and redesigns.

Suppose we measure "steps" as the *number of devices* "spent" or used in testing and redesign. Let $W_n^j(X,Y)$ represent the maximum expected number of effective systems resulting from an n system program, given developmental state (X,Y), when development costs of at most j will be allowed. As with $V_n(X,Y)$, and arrived at by entirely the same reasoning (with the additional stipulation that the process stops in j steps), there is a recursive functional equation for W_n^j,

$$W_n^j(X,Y) = \max \left\{ \begin{array}{c} nr(X,Y), \\ r(X,Y)W_{n-1}^{j-1}(X+1,Y) + (1-r(X,Y))W_{n-1}^{j-1}(X,Y+1), \\ W_{n-d}^{j-d}(X+h_0(X,Y), Y+h_1(X,Y)), \quad j \geq d \end{array} \right\}, \tag{5.4}$$

where $W_n^0(X,Y) = nr(X,Y)$ and $W_n^n(X,Y) = V_n(X,Y)$, as given in [13].

If, on the other hand, we measure "steps" as the ***number of developmental decisions*** made, and not the number of devices spent, we get (5.4) with one change; the third term changes from $W_{n-d}^{j-d}(\cdot)$ to become $W_{n-d}^{j-1}(\cdot)$.

For convenience, we will refer to the look ahead rules as "Device" and "Step" rules. Equation (5.4) is the "Device" rule and by the "Step" rule we mean replacing the third term of (5.4) with $W_{n-d}^{j-1}(\cdot)$.

Using a j-step look ahead rule, one stops the developmental phase at the first i

for which

$$W^j_{n-i}(X_i, Y_i) = (n-i)r(X_i, Y_i).$$

5.2 Numerical Examples and Comparison of Methods

To understand how different parameter values affect the model, we computed the values of (4.4) and (5.4) (for both Device and Step look aheads) on a small set of problems.

Given initial budget N, and redesign cost d, completely parameterizing a problem requires that one must give values for: k, the number of states the problem is to have; $\underline{a}$, the initial probability distribution of the initial reliability; $\underline{p}$, the failure probabilities for each state; h_0, the redesign value; and (X_0, Y_0), the initial development state. The values of these parameters used to compute the V and W given in Table 5.3 are given in Table 5.2.

For the two different values of k, the $\underline{p}$ (we refer to them as $\underline{p_k}$, not to be confused with the subscripts on the $\underline{a_i}$ s) vectors have the same "best" and "worst" failure probabilities. The $\underline{a_1}$ vector gives a uniform distribution of initial reliability states. That is, the design reliability is equally likely to be in any one of the k states. The vector $\underline{a_2}$ gives a distribution which is weighted so that the probability the reliability is initially very good is "high," while $\underline{a_3}$ was chosen to make this probability "poor." Later, we will present the results of some calculations made with non-integer d.

The V_n values were computed using (4.4), "**S**$_n$," and "**D**$_n$" are the number of look ahead steps needed to compute V_n from the Step and Device look ahead rules of equation (5.4), respectively. Some simple observations may be made.

As one would expect, the value of V decreases when the initial probability of

Table 5.2: Values of Parameters Used in k-State Calculations, Integer d

Parameter	Values	
n	100, 200	
d	1, 2, 3, 4, 5	
h_0	5	
(X_0, Y_0)	(0,0)	
k	4	10
$\underline{p}$	$(.99, .30, .09, .01)$	$(.99, .8, .7, .6, .5, .35, .2, .05, .01)$
$\underline{a_1}$	$(1,1,1,1)$	$(1,1,1,1,1,1,1,1,1,1)$
$\underline{a_2}$	$(1,9,25,49)$	$(1,9,25,\ldots,17^2,19^2)$
$\underline{a_3}$	$(49,25,9,1)$	$(19^2,17^2,\ldots,9,1)$

having reliability in state k (the "best" state, that of highest reliability) is low. This value decreases as the cost of a redesign rises. Given cost d and the "same" $\underline{a_i}$, each 4-state V is larger than the 10-state V, but not by any appreciable amount. The mean failure probability given $\underline{p_4}$ is 0.3475, while with $\underline{p_{10}}$ this mean failure probability is 0.42, making it slightly easier to achieve better reliability in the 4-state versus the 10-state model.

Turning now to the look ahead data. In all but two cases the number of Device look aheads is equal to the number of Step look aheads multiplied by the redesign cost, d. The two exceptions occur when $k = 4$, the "$\underline{a}$" value is $\underline{a_2}$, $d = 4$ or 5, and $n = 100$ (V_{100}). Why? We suspect this means that the optimal policy is "redesign" and then "test," in both cases.

This demonstrates the utility of both rules used together — in some simple cases, the optimal strategy can be deduced. If one was to pick a single rule, and this seems unlikely, then it is premature to conclude that the Step rule is to be preferred over

Table 5.3: Example of k-State Reliability, Integer d, $n = 100$ and 200

k	$\underline{a}$	d	V_{100}	$\mathbf{S}_{100}$	$\mathbf{D}_{100}$	V_{200}	$\mathbf{S}_{200}$	$\mathbf{D}_{200}$
4	$\underline{a_1}$	1	94.21	3	3	191.54	4	4
		2	92.18	2	4	188.41	3	6
		3	90.54	1	3	186.29	2	6
		4	89.61	1	4	184.37	2	4
		5	88.67	1	5	182.44	2	10
	$\underline{a_2}$	1	95.48	2	2	193.00	3	3
		2	94.43	1	2	190.96	2	4
		3	93.46	1	3	189.81	1	3
		4	92.63	2	5	188.85	1	4
		5	92.57	2	6	187.89	1	5
	$\underline{a_3}$	1	90.03	4	4	186.51	9	9
		2	86.84	3	6	180.06	5	10
		3	84.14	2	6	176.46	3	9
		4	82.35	2	8	173.69	3	12
		5	80.56	2	10	170.91	3	15
10	$\underline{a_1}$	1	94.19	3	3	191.30	4	4
		2	92.12	2	4	188.38	3	6
		3	90.20	2	6	186.15	2	6
		4	88.28	2	8	184.23	4	8
		5	87.35	1	5	182.32	2	10
	$\underline{a_2}$	1	94.72	2	2	192.00	3	3
		2	92.79	2	4	189.44	2	4
		3	91.35	1	3	187.51	2	6
		4	90.41	1	4	185.57	2	8
		5	89.47	1	5	183.65	1	5
	$\underline{a_3}$	1	90.91	4	4	186.22	5	5
		2	87.39	3	6	181.82	4	8
		3	84.60	3	9	178.03	4	12
		4	81.81	3	12	174.78	3	12
		5	79.88	2	10	171.99	3	15

its Device counterpart. Because the calculations are recursive, they can be finished in reasonable time only for a small number of look ahead steps (on our equipment, $j \leq 15$ or 16). For a fixed number of look ahead steps, when $d > 1$, the recursive calculation for the Device look ahead rule will not look as far ahead in the optimal strategy as the Step rule.

Recognizing *when* a look ahead rule has returned the optimal value of V is another issue entirely. For example, earlier we alluded to the fact that it took about 70 hours, using the lattice method of (5.1)–(5.3), to compute $V_{500}(0,0)$, for a particular instance of a 4-state problem . The Step rule gives this value, on the same equipment, in under one second, requiring a look ahead of 6 steps, and the Device rule gives it in 12 steps, taking under 10 seconds. Had we not known the value of V_{500}, how we would we deduce the optimal value, given a sequence of look ahead values? In many cases, simply put: we couldn't. Since the problem complexity grows exponentially as a function of the budget, n, the naive approach of, "computing a few more look ahead steps" is, as we've pointed out repeatedly, not an option.

To demonstrate the symmetric role that the two look ahead rules can play, we look at another set of calculations, these performed with non-integer design costs. The parameter values used to compute Table 5.5 are given in Table 5.4.

As was hinted earlier, for $d < 1$, the Device rule computes the optimal value, with fewer look aheads than the Step rule. Again, this time with three exceptions (the entries in rows 4, 6, and 8 of Table 5.5), the depth the Device rule has to look to compute the optimal value is equal to number of Step look aheads multiplied by d. All our previous comments, with respect to how the function values change with the parameters, still hold. Two additional observations can be made. As expected,

Table 5.4: Values of Parameters Used in k-State Calculations, Non-Integer d

Parameter	Values
k	4
n	100
d	1/2, 3/4, 3/2, 5/2
h_0	2, 5
$\underline{a}$	$\underline{a_1}$, $\underline{a_2}$
$\underline{p}$	(.99, .30, .09, .01)
(X_0, Y_0)	(0, 0)

Table 5.5: Example of k-State Reliability, Non-Integer d, $n = 100$

$\underline{a}$	h_0	d	V_{100}	$\mathbf{S}_{100}$	$\mathbf{D}_{100}$
$\underline{a_2}$	2	1/2	95.16	4	2
		3/4	94.19	4	3
		3/2	93.11	2	3
		5/2	92.48	3	6
	5	1/2	96.46	2	1
		3/4	95.48	2	2
		3/2	94.51	2	3
		5/2	93.46	1	3
$\underline{a_3}$	2	1/2	89.09	10	5
		3/4	87.17	8	6
		3/2	82.77	6	9
		5/2	78.78	4	10
	5	1/2	93.22	8	4
		3/4	91.28	8	6
		3/2	88.15	4	6
		5/2	85.04	2	5

given two values of h_0, the larger value gives a greater yield, V, and consequently, a shorter development process than is required by the smaller h_0 value. In some cases the cheaper designs made no difference. By comparing rows 6 and 8 of Table 5.3 to the same rows in Table 5.5, we see that, in some cases, the cheaper designs do not increase the V value. When we compare rows 11–15 of Table 5.3 to rows 13–16 of Table 5.5, we see that the less expensive designs can have an effect on the value of V. In this example, the vector $\underline{a_3}$ gives poor initial probability of good design reliability.

CHAPTER 6. CONCLUSIONS

6.1 Conclusions

We have extended the analysis of the optimal development of one-shot systems when reliability growth is possible due to system redesign. Huang posed four questions in her Ph.D. thesis and we have answered three of them here. Naturally however, in doing so, we have raised more new questions than we were able to answer.

We have improved the best existing binary model by incorporating design costs and we have proposed a new multiple-state model.

Incorporating design costs into the Huang 2-state model is in itself important, but this change improves the model in some additional ways:

1. The stipulation that redesigns can only occur following a failed test is removed.

2. Repeated redesigns are allowed. This is a mixed change.

    - There are no restrictions on the course of action the development program should follow.

    - The sequence of redesigns may not allow for a testing period. In general, performing redesigns and not verifying their outcome may be unreasonable.

3. Reliability growth does not occur through repeated testing. Testing only provides information on the current reliability state.

4. The reliability changes in accordance to the success/failure of the most recent test.

5. The value function, $V_n(s)$, is now monotone increasing.

6. Regressive designs can be included in this model in a straightforward way.

7. The redesigns can be used to predict compliance with extremely high reliability specifications.

For the most part, the model gives developmental procedures which progress as one would intuitively expect: redesign if the estimated reliability is very low, test if the current reliability state is unclear, and build when the reliability estimate is satisfactorily high. But there are parameter mixes where the behavior is strange, unexplained ... chaotic.

The proposed model for the analysis of multiple-state reliability growth has a form which is intrinsically a 2-variable model and this model contains the original binary model of Chapter 3. Realizing the original 2-state model is an instance of the k-state model is an exercise which makes an important point — much work is needed to develop appropriate/realistic redesign functions.

We show how to solve for optimal strategies in the new model, given some restrictions, on a lattice. Finally, we show how to apply more easily computed but suboptimal "look ahead" heuristics, of which there are two types: we can look ahead according to the development budget we will entertain, or according to how many further developmental decisions will be allowed. We make comparisons, in a number of cases, between the two types of myopic rules.

6.2 Further Research

The following topics and questions need additional study:

1. Can mathematical models provide any further guidance in the development of extremely high reliability devices?

2. Models in which u, the probability of a successful redesign, is a function of the development history are needed. Does a model in which u is a function of the redesign cost contribute any further insight? Can the redesign vary according to the previous test results?

3. Is it unreasonable to allow repeated, consecutive redesigns?

4. What are the distributions of $r(s_n^*)$ and $r(X, Y)$?

5. Functions to approximate the s_n^*, the cut-off values are needed. Attempts to characterize the optimal developmental plan are still inconclusive. Given the chaotic behavior of some problems, this seems likely to remain the case. Along this line of inquiry, is there a way by which the policies dictated by a problem's parameters can be recognized? For example, is there a way to recognize the "no test" problems we saw when studying the extremely high reliability problems in Section 3.3.5?

6. The multiple-state models need realistic redesign functions.

7. What role, other than serving as initial probability state vector, can the weights $\underline{a}$ of the multiple-state model serve?

8. Further work modeling non-binary test responses is needed.

9. Develop other goals, in comparison to (2.1) and (4.3), which are useful for studying these problems. Given other goals, what techniques are required to compute solutions?

10. Develop more complete, and hopefully, faster methods by which to compute $V_n(X,Y)$. Does the model have properties, similar to those possessed by the original binary model, that can be exploited to speed up the computation of solutions?

BIBLIOGRAPHY

[1] R.E. Barlow and E.M. Scheuer. Reliability growth during a development testing program. *Technometrics*, 8 No. 1:53–60, 1966.

[2] R. Bell. Army reliability growth policy. In *Proceedings Annual Reliability and Maintainability Symposium*, pages 210–214, 1986.

[3] Richard Bellman. *Dynamic Programming.* Princeton University Press, Princeton, New Jersey, 1957.

[4] W.G. Cochran and G.M. Cox. *Experimental Designs.* John Wiley, New York, 2nd edition, 1957.

[5] L.H. Crow. Methods for assessing reliability growth potential. In *Proceedings Annual Reliability and Maintainability Symposium*, pages 484–489, 1984.

[6] L.H. Crow. Reliability values in the development process. In *Proceedings Annual Reliability and Maintainability Symposium*, pages 383–388, 1985.

[7] L.H. Crow. On the initial system reliability. In *Proceedings Annual Reliability and Maintainability Symposium*, pages 115–119, 1986.

[8] J.T. Duane. Learning curve approach to reliability monitoring. *IEEE Transactions on Aerospace*, 2 No. 2:563–566, 1964.

[9] D.J. Dwyer. Reliability test planning for one-shot systems. In *Proceedings Annual Reliability and Maintainability Symposium*, pages 412–415, 1987.

[10] J.L. Ekstrom and A.G. Allred. Verifying reliability of solid rocket motors (srms) at minimum cost. In *Proceedings Annual Reliability and Maintainability Symposium*, pages 502–508, 1991.

[11] N.S. Fard and D.L. Dietrich. A Bayes reliability growth model for a development testing program. *IEEE Transactions on Reliability*, 36 No. 5:568–572, 1987.

[12] M.Y. Huang. *Design of developmental test programs for one-shot systems with two state reliability.* Ph.D. dissertation, Iowa State University, 1995.

[13] M.Y. Huang, D.W. McBeth, and S.B. Vardeman. Developmental test programs for one-shot systems: Two state reliability and binary developmental test results. *IEEE Transactions on Reliability*, forthcoming, 1996.

[14] B.W. Kernighan and D.M. Ritchie. *The C Programming Language.* Prentice Hall, Englewood Cliffs, New Jersey, 07632, 2nd edition, 1988.

[15] D.E. Knuth. *The Art of Computer Programming, Volume 2 / Seminumerical Algorithms.* Addison-Wesley, Reading, Massachusetts, 2nd edition, 1981.

[16] D.K. Lloyd and M. Lipow. *Reliability: Management, Methods, and Mathematics.* Redondo Beach, California, Published by the Authors, 1977.

[17] H.F. Martz and R.A. Waller. *Bayesian Reliabilty Analysis.* Krieger Publishing Company, Malabar, Florida 32950, reprint edition, 1991.

[18] S.M. Pollock. A Bayesian reliability growth model. *IEEE Transactions on Reliability*, 17 No. 4:187–198, 1968.

[19] B. Ray, I. Bhandari, and R. Chillarege. Reliability growth for typed defects. In *Proceedings Annual Reliability and Maintainability Symposium*, pages 327–332, 1992.

[20] D. Robinson and D.L. Dietrich. A nonparametric-Bayes reliability growth model. *IEEE Transactions on Reliability*, 38 No. 5:591–598, 1989.

[21] E. Seglie. How much testing is enough? In *Workshop on Statisitical Issues in Defense Analysis and Testing*. National Research Council, Washington, D.C., 1992.

[22] G.W. Snedecor and W.G. Cochran. *Statistical Methods*. Iowa State University Press, Ames, Iowa, 50011, 8th edition, 1989.

APPENDIX PSEUDO-CODE

The author will be happy to supply interested persons the source code used to create the numerical examples in this dissertation.

The following conventions are used in the descriptions which follow:

- Variables and arrays are printed in *italic.*
- Array indices are indicated individually in square brackets so that, for example, in an $n \times m$ array (matrix) A, the element $A(i, j)$ (row, column) is written as $A[i][j]$. We exploit (abuse?) this property in Appendix.2 below. Finally, note that array indices start at 0.
- The names of procedures are written in SMALL CAPITAL LETTERS

.1 Computing $V_n(s)$

The following is a pseudo-code description of the program used to compute $V_n(s)$.

1. Initialization

   (a) Input

      i. K = number of grid points s for which $V_n(s)$ will be computed

      ii. Problem parameters = $\{N, p_1, p_2, u, d = p/q\}$

```
    (b) Storage
        i. $probmod = \max\{p, q\} + 1$
        ii. Establish $probmod$ arrays $v$, each having size $K + 2$
        iii. Establish 1 $action$ array of size $K + 1$
2. Computation
    (a) Compute $V_1$:
        i. For $j = 0$; $j \leq K$; increment $j$ by 1
           Compute: $v[0][j] = r(j/K)$
        ii. $v[0][K + 1] = v[0][K]$
        iii. $intbudget = 1$
    (b) Compute $V_{1+1/q}$ to $V_{Nq}$
        i. For $i = 1$; $i \leq (N - 1)q$; increment $i$ by 1
            A. $itest = (i - q) \bmod probmod$
            B. $idesign = (i - p) \bmod probmod$
            C. $inow = i \bmod probmod$
            D. If $inow = 0$, then increment $intbudget$ by 1
            E. For $j = 0$; $j \leq K$; increment $j$ by 1
               $v[inow][j] = \text{VALUE}(intbudget, itest, idesign, j)$
            F. $v[inow][K + 1] = v[inow][K]$
        ii. Procedure: $\text{VALUE}(b, t, dg, j)$
            A. Compute $r(s)$, $\eta_0(s)$, $\eta_1(s)$, and $\delta(s)$ using (2.2), (3.1), and (3.2),
               respectively, where $s = j/K$
            B. $build = b \cdot r(s)$
            C. If $t \geq 0$, $test = r(s)\text{INTERP}(t, \eta_0(s)) + (1 - r(s))\text{INTERP}(t, \eta_1(s))$;
                  else, $test = 0$
            D. If $dg \geq 0$, $design = \text{INTERP}(dg, \delta(s))$;
                  else, $design = 0$
            E. $value = \max\{build, test, design\}$
            F. $action[j]$ = "Build," "Test," or "Redesign," as appropriate, depending on which term gave $value$ in previous step
        iii. Procedure: $\text{INTERP}(index, pt)$
                (Note: $\lfloor x \rfloor$ = Integer part of $x$ )
            A. $up = K \cdot pt - \lfloor K \cdot pt \rfloor$
            B. $down = 1 - up$
            C. Return: $((down \cdot v[index][\lfloor K \cdot pt \rfloor]) + (up \cdot v[index][K \cdot pt + 1]))$

3. Output
    (a) $wantv = (N - 1) \bmod probmod$
    (b) Print $v[wantv][0]$, $action[0]$
    (c) $build.flag = 0$
    (d) While $build.flag = 0$
        i. For $j = 1$; $j \leq K$; increment $j$ by 1
              If $v[wantv][j - 1] \neq v[wantv][j]$
                  Print $v[wantv][j]$, $action[j]$
```

ii. If *action*$[j]$ = "Build"

build.flag $= 1$

4. Do another problem or exit program

As mentioned in Section 3.3.2, this program can be modified to compute V when tests cost t devices. We assume the test and design costs are input in three integers: $\{p_t, p_d, q\}$, where the variable names are as given in Section 3.3.2. Now V is computed by changing the indices in point 2(b)i., above, as follows.

A. $probmod = \max\{p_t, p_d\} + 1$

B. $itest = (i - q - p_t) \bmod probmod$

C. $idesign = (i - q - p_d) \bmod probmod$

D. $inow = (i - q) \bmod probmod$

The value of $V_N(s)$ will be in $v[wantv]$, where $wantv = (N-1)q \bmod probmod$ (in Section 3 above, Output).

.2 Simulated Optimal Development Process

The simulation program was written in the C programming language and the simulations were run on a *Silicon Graphics Indy* workstation (100 MHz, 32 Meg RAM). When a test is performed a random number is drawn. The number is compared to $r(s)$ (the current design reliability) and, if the random number is less than

$r(s)$, the test is determined to be successful while otherwise the test is deemed a failure.

The C library function *random()* was used to produce the random numbers used in the "test" part of these simulations. The function *random()* was seeded from the computer system clock by a call to the C library function *srandom()*. Interested readers may consult Kernighan and Ritchie [14] as well as the *Unix* system *man* pages for more specific implementation details.

On the *SGI Indy*, the function *random()* uses a non-linear additive feedback random number generator to return pseudo-random numbers in the range from 0 to $2^{31} - 1$. The period of this random number generator is approximately $16(2^{31} - 1)$ (approximately 34 billion cycles). In an attempt to check the randomness of the function *random()*, we used it to produce a number of sequences of length 5,000 and 10,000. For each sequence generated, we examined a histogram of the (pseudo-) random set of numbers, a point plot of the successive pairs of the numbers in the sequence (plot (X_i, X_{i-1}), where $X_i = i$th number drawn), and performed a Chi-square test (for a uniform distribution of the numbers in the sequence over their range) as discussed in Knuth [15], pages 39–45. All these (admittedly simple) tests indicate that *random()* is a satisfactory random number generator on the *Indy* platform.

The following is a pseudo-code description of the program written to simulate the optimal development process discussed in Section 3.3.3.1.

```
1. Initialization
    (a) Input
        i. $K$ = number of grid points $s$ for which $V_n(s)$ will be computed
```

ii. $SIM.RUNS$ = number of times development process will be simulated with current set of problem parameters

iii. Problem parameters = $\{N, p_1, p_2, u, d = p/q\}$

iv. s_0 = Probability initial design is in state 2

(b) Storage

i. *probmod* $= \max\{p, q\} + 1$

ii. Establish *probmod* arrays v, each having size $K + 2$

iii. Establish 1 *action* array of size $K + 1$

iv. *vrecord* = array of $((N-1)q+1)$ records with each record, potentially, of varying length. For $i = 0$ to Nq, and for $j = 0$ to l (l may vary with each subproblem), *vrecord*$[i][j]$ will contain s_j, $V_i(s_j)$, and *action*$[s_j]$, where s_j is the jth (developmental) *action* change in V_i. We will denote the specific element of record j by *vrecord*$[i][j]$[*element*], where *element* $\in \{s_j, V_i(s_j), action[s_j]\}$. (In context, it is not a problem to keep the initial input value s_0 separate.)

2. For each V_n, compute and store the following information: s_j, the jth *action* change point; as well as $V_n(s_j)$ and *action*$[s_j]$

(a) Use the program from Appendix.1 to compute V_1 to V_{Nq}; but now, immediately after computing each $v[inow][j]$, do the following ...

(b) When computing V_1, store *vrecord*$[0][0] = 0, 0$, "Build"

(c) For $V_{1+1/q}$ to V_{Nq} (i.e., For $i = 1$; $i \leq (N-1)q$; increment i by 1)

i. Store: *vrecord*$[i][0] = 0, v[i][0], action[0]$

```
        ii. For $j > 0$; $j \leq K$; increment $j$ by 1
            If $action[j-1] \neq action[j]$, then
             A. Check: is there storage space for additional $vrecord[i][l]$?
                  If not, allocate additional storage
             B. Store: $vrecord[i][l] = s_j, v[i][j], action[j]$
                  (Note: we've just finished computing$v[i][j]$ and at this point
                  a developmental activity change has occurred. This activity
                  change corresponds to the $l$th change in the developmental
                  action.)
3. Simulate Development Process
   (a) Seed random number generator
   (b) If $vrecord[Nq][s_0][action] \neq$ "Build"
        i. For $i = 1$; $i \leq SIM.RUNS$; increment $i$ by 1
           dosim($s_0, Nq, vrecord[Nq][s_0][action]$)
       ii. Report statistics of $SIM.RUNS$ developmental processes
      iii. Procedure: DOSIM($s, n, act$)
             A. Case $act =$ "Build"
                  • Update development statistics
                  • Do next simulation run
             B. Case $act =$ "Test"
                  • $n = n - q$
                  • $s =$ TEST($s$)
        • Update test statistics
        • Goto NEXT.BUDGET
    C. Case $act$ = "Redesign"
        • $n = n - p$
        • $s = \delta(s)$
        • Update redesign statistics
        • Goto NEXT.BUDGET
    D. Procedure: TEST($s$)
        • $outcome$ = RANDOM / (Maximum Random Number)
        • If $outcome \leq r(s)$, return: $\eta_0(s)$ (success);
            else, return: $\eta_1(s)$ (failure)
    E. Procedure: NEXT.BUDGET
        • $act = vrecord[n][s][action]$
        • DOSIM($s, n, act$)
    F. Procedure: RANDOM
        Function call to random number generator, discussed above
4. Do simulation of another problem (new set of input parameters) or exit
```

.3 Computing $V_n(X, Y)$ on a Lattice

The (recombinant) lattice program uses modular arithmetic in some storage arrays like the original $V_n(s)$ program (Appendix .1). The program computes the value of $V_m(X,Y)$ for every possible state vector (X,Y) when the budget is m, for m ranging from q to Nq.

The following is a pseudo-code description of the lattice technique introduced in Section 5.1.1. The program was used to compute the values given in Tables 5.3 and 5.5.

```
1. Initialization
   (a) Problem parameters $= \{N, \underline{p}, \underline{a}, (X_0, Y_0), d = p/q, h_0(\text{integer})\}$
   (b) Storage
       i. $probmod = \max\{p,q\} + 1$
       ii. Establish arrays ...
              $maxtest$, $maxdsgn$, and $maxposition$
           to store the maximum number of tests, redesigns and number of combinations of tests and redesigns, respectively, realized for the last $probmod$ subproblems
       iii. Establish array $difference$ to track the size of the developmental budget for the last $probmod$ subproblems
       iv. Establish $probmod$ arrays $v$, each of length $(Nq)(Nq+1)/2$ (More conservative memory usage is possible and in the program actually used, we allocated memory later, "on the fly," when we knew exactly the number of elements $v[m]$ would possess.)
2. Compute $V_N(X_0, Y_0)$
   (a) $(X,Y) = (X_0, Y_0)$, $intbudget = 0$
   (b) For $m = q$; $m \le Nq$; increment $m$ by 1
```

i. $currpos = (m - q) \bmod probmod$

ii. $testpos = (m - 2q) \bmod probmod$

iii. $dsgnpos = (m - p - q) \bmod probmod$

iv. $difference[currpos] = Nq - m$

v. $maxtext[currpos] = difference[currpos]/q$,
the maximum number of tests **possible**

vi. $maxdsgn[currpos] = difference[currpos]/p$,
the maximum number of designs **possible**

vii. If $m \bmod q = 0$, then $intbudget$ is increased by 1

viii. Determine the maximum number of tests and designs which can be **realized** with the current budget (m) and the number of developmental states (X, Y) possible

A. $pos = maxt = maxd = 0$

B. Using equations (5.1)-(5.3) of Section 5.1.1, page 53, look for integers t (j) in the range of 0 to $maxtest(maxdsgn)[currpos]$ so that

$$t\,q + j\,p = difference[currpos] \tag{A.1}$$

Do this as follows:

For $t = maxtest[currpos]$; $t \geq 0$; decrement t by 1

For $j = 0$; $j \leq maxdsgn[currpos]$; increment j by 1

If (t, j) satisfy (A.1), then

- $pos = pos + t + 1$
- $maxt = \max\{t, maxt\}$

```
        • $maxd = \max\{j, maxd\}$
      C. $maxposition[currpos] = pos - 1$
      D. $maxtest[currpos] = maxt$
      E. $maxdsgn[currpos] = maxd$
ix. Now fill the state table for this budget – compute $V_m(X, Y)$ for all
    $(X, Y)$ possible with budget $m$
    For $i = 0$; $i \leq maxposition[currpos]$; increment $i$ by 1
      A. $\textsc{posgivexy}(currpos, i, X, Y)$
      B. $rXY = \mathrm{r}(X + X_0, Y + Y_0)$ (Use (4.2) to compute $rXY$)
      C. $build = intbudget \cdot rXY$
      D. If $testpos \geq 0$, then
           $test = (rXY)\, v[testpos][\textsc{xygivepos}(testpos, X + 1, Y)]$
                $+(1 - rXY)\, v[testpos][\textsc{xygivepos}(testpos, X, Y + 1)]$;
           else, $test = 0$
      E. If $dsgnpos \geq 0$, then
           $design = v[dsgnpos][\textsc{xygivepos}(dsgnpos, X + h_0, Y)]$;
           else, $design = 0$
      F. $v[currpos][\textsc{xygivepos}(currpos, X, Y)] = \max\{build, test, design\}$
```

x. Procedure POSGIVEXY$(tbl, index, x, y)$

This procedure will return the development state vector (X, Y) given some measure of the budget (tbl) currently under consideration, table position ($index$) and maximum number of tests allowed for the current budget ($maxtest[tbl]$). Knowing these numbers, POSGIVEXY determines which table row *index* is contained in and thus (since h_1 is 0) the Y of the vector (X, Y). Knowing Y, a brute force search for the correct X values ensues.

A. Determine Y (Knowing *tbl* and *index*, we know how many elements are in the current table and how many elements are in each row.)

- $a = q$
- $b = 2(maxtest[tbl] + 1) + 3q$
- $c = 2(maxtest[tbl] + q + index + 1)$
- $root = \lfloor (b - sqrt(b^2 - 4ac))/2a \rfloor$

  root, the row which contains *index*, and the use of the quadratic formula to compute *root*, comes from counting how many items are in the current table and determining which row number *index* is contained in, since we want an integer row number we use $\lfloor x \rfloor$ – keeping only the integer part.
- $Y = index{-}\mathrm{NUM.B.4}(root, maxtest[tbl])$

B. Now determine X:

- $found = 0$
- While $found = 0$

```
For $j = maxtest[tbl] - (root - 1) \cdot p$; $j \geq 0$; decrement $j$ by 1
  For $k = 0$; $k \leq maxdsgn[tbl]$; increment $k$ by 1
    If $jq + kp = difference[tbl]$, then
```

```
            $nrowpos = j,\ krowpos = k,\ found = 1$

    • $X = nrowpos - Y + (krowpos \cdot h_0)$

  C. Return: $(X, Y)$
```

xi. Procedure XYGIVEPOS(tbl, x, y)

This procedure is the "inverse" of POSGIVEXY. Given the current budget (tbl) and the current developmental state (X, Y), it is relatively straight-forward (in comparison to POSGIVEXY) to determine the associated table position. XYGIVEPOS uses (5.1) and (5.2) to determine in which row of the current budget table (X, Y) resides. The position is then the total number of elements in the table before this row, plus Y.

```
  A. $detrmnt = (q \cdot h_0) - p$
  B. If $detrmnt = 0$, return: $Y$
  C. $(detrmnt \neq 0)$
    • $numberX = \lfloor((h_0 \cdot difference[tbl]) - p(X + Y))/detrmnt\rfloor$
    • For $j = 0$; $j \leq maxtest[tbl]$; increment $j$ by 1
          If $maxtest[tbl] - j \cdot p = numberX$, then $rowX = j + 1$
    • Return: $Y+$ NUM.B.4$(rowX, maxtest[tbl])$
```

xii. Procedure NUM.B.4$(row, ntest)$

This procedure computes the number of elements in a table before row when the table has a maximum of $ntest$ tests.

```
  Return: $((row - 1) \cdot (2(ntest + 1 + p) - p \cdot row)/2)$
```

3. Output

    (a) $wantv = (N-1)q \bmod probmod$

    (b) POSGIVEXY($wantv, 0, X, Y$)

    (c) Print $v[wantv]$[XYGIVEPOS($wantv, X, Y$)]

4. Do another problem or exit program