Calhoun
Institutional Archive of the Naval Postgraduate School

Calhoun: The NPS Institutional Archive

DSpace Repository

Theses and Dissertations | 1. Thesis and Dissertation Collection, all items

1990-03

A prolog implementation of pattern search to optimize software quality assurance

Buzzard, Raymond Karl

Monterey, California. Naval Postgraduate School

https://hdl.handle.net/10945/30680

This publication is a work of the U.S. Government as defined in Title 17, United States Code, Section 101. Copyright protection is not available for this work in the United States.

Downloaded from NPS Archive: Calhoun

DUDLEY KNOX LIBRARY

http://www.nps.edu/library

Calhoun is the Naval Postgraduate School's public access digital repository for research materials and institutional publications created by the NPS community. Calhoun is named for Professor of Mathematics Guy K. Calhoun, NPS's first appointed -- and published -- scholarly author.

Dudley Knox Library / Naval Postgraduate School
411 Dyer Road / 1 University Circle
Monterey, California USA 93943

DTIC FILE COPY

NAVAL POSTGRADUATE SCHOOL
Monterey, California

AD-A225 435

THESIS

DTIC
ELECTE
AUG 20 1990

A PROLOG IMPLEMENTATION OF PATTERN SEARCH TO OPTIMIZE SOFTWARE QUALITY ASSURANCE

by

Raymond Karl Buzzard

March 1990

Thesis Advisor: Tarek Abdel-Hamid

Approved for public release; distribution is unlimited

90 08 [illegible] 013

UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE

REPORT DOCUMENTATION PAGE

1a. REPORT SECURITY CLASSIFICATION UNCLASSIFIED	1b RESTRICTIVE MARKINGS
2a. SECURITY CLASSIFICATION AUTHORITY	3. DISTRIBUTION / AVAILABILITY OF REPORT Approved for public release; distribution is unlimited
2b. DECLASSIFICATION / DOWNGRADING SCHEDULE	
4. PERFORMING ORGANIZATION REPORT NUMBER(S)	5. MONITORING ORGANIZATION REPORT NUMBER(S)
6a. NAME OF PERFORMING ORGANIZATION Naval Postgraduate School 6b. OFFICE SYMBOL (If applicable)	7a. NAME OF MONITORING ORGANIZATION Naval Postgraduate School
6c. ADDRESS (City, State, and ZIP Code) Monterey, California 93943-5000	7b. ADDRESS (City, State, and ZIP Code) Monterey, California 93943-5000
8a. NAME OF FUNDING / SPONSORING ORGANIZATION 8b. OFFICE SYMBOL (If applicable)	9. PROCUREMENT INSTRUMENT IDENTIFICATION NUMBER
8c. ADDRESS (City, State, and ZIP Code)	10. SOURCE OF FUNDING NUMBERS PROGRAM ELEMENT NO. / PROJECT NO. / TASK NO. / WORK UNIT ACCESSION NO.

11. TITLE (Include Security Classification)
A PROLOG IMPLEMENTATION OF PATTERN SEARCH TO OPTIMIZE SOFTWARE QUALITY ASSURANCE

12. PERSONAL AUTHOR(S)
Buzzard, Raymond K.

13a. TYPE OF REPORT	13b. TIME COVERED	14. DATE OF REPORT (Year, Month, Day)	15. PAGE COUNT
Master's Thesis	FROM _____ TO _____	1990, March	159

16. SUPPLEMENTARY NOTATION
The views expressed in this thesis are those of the author and do not reflect the official policy or position of the Department of Defense or the U.S. Government.

17. COSATI CODES			18. SUBJECT TERMS (Continue on reverse if necessary and identify by block number)
FIELD	GROUP	SUB-GROUP	Expert System; Simulation Model

19. ABSTRACT (Continue on reverse if necessary and identify by block number)

Quality Assurance (QA) is a critical factor in the development of successful software systems. Through the use of various QA tools, project managers can ensure that a desired level of performance and reliability is built into the system. However, these tools are not without cost. Project managers must weigh all QA costs and benefits for each development environment before establishing an allocation strategy.

The development of a system dynamics model has provided project managers with an automated tool that accurately replicates a project's dynamic behavior. This model can be used to determine the optimal quality assurance distribution pattern over a given project's life cycle.

The objective of this thesis was to enhance a prototype expert system module that interacts with the system dynamics model for determining QA effort

20. DISTRIBUTION / AVAILABILITY OF ABSTRACT	21. ABSTRACT SECURITY CLASSIFICATION
☒ UNCLASSIFIED/UNLIMITED ☐ SAME AS RPT ☐ DTIC USERS	Unclassified
22a NAME OF RESPONSIBLE INDIVIDUAL Prof. Tarek Abdel-Hamid	22b TELEPHONE (Include Area Code) (408) 646-2686
	22c. OFFICE SYMBOL Code AS/Ah

DD FORM 1473, 84 MAR 83 APR edition may be used until exhausted
All other editions are obsolete

SECURITY CLASSIFICATION OF THIS PAGE

☆ U.S. Government Printing Office: 1986—606-24

UNCLASSIFIED

#19 - ABSTRACT - (CONTINUED)

allocation schemes. The new module uses a pattern search algorithm to derive an optimal distribution scheme from a given set of project parameters. This system not only resolves all limitations discovered in the prototype model but also achieved significant reductions in total project cost.

Approved for public release; distribution is unlimited.

A Prolog Implementation of Pattern Search to
Optimize Software Quality Assurance

by

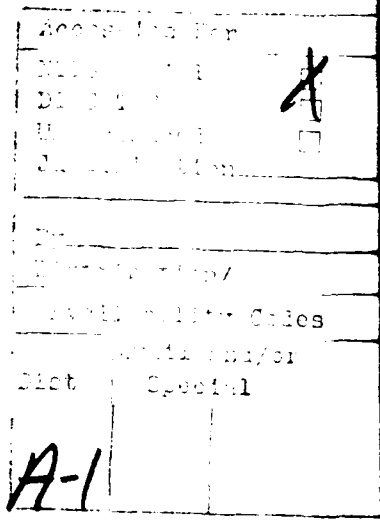
Raymond Karl Buzzard
Lieutenant, United States Navy
B.S., University of Wisconsin-Milwaukee 1982

Submitted in partial fulfillment
of the requirements for the degree of

MASTER OF SCIENCE IN INFORMATION SYSTEMS

from the

NAVAL POSTGRADUATE SCHOOL
March, 1990

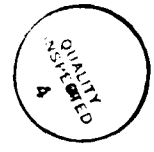


Author: [redacted]
Raymond Karl Buzzard

Approved by: [redacted]
Tarek Abdel-Hamid, Thesis Advisor

[redacted]
Tung X. Bui, Second Reader

[redacted]
David R. Whipple, Chairman
Department of Administrative Sciences



ABSTRACT

Quality Assurance (QA) is a critical factor in the development of successful software systems. Through the use of various QA tools, project managers can ensure that a desired level of performance and reliability is built into the system. However, these tools are not without cost. Project managers must weigh all QA costs and benefits for each development environment before establishing an allocation strategy.

The development of a system dynamics model has provided project managers with an automated tool that accurately replicates a project's dynamic behavior. This model can be used to determine the optimal quality assurance distribution pattern over a given project's life cycle.

The objective of this thesis was to enhance a prototype expert system module that interacts with the system dynamics model for determining QA effort allocation schemes. The new module uses a pattern search algorithm to derive an optimal distribution scheme from a given set of project parameters. This system not only resolves all limitations discovered in the prototype model, but also achieved significant reductions in total project cost.

TABLE OF CONTENTS

I. INTRODUCTION

A. BACKGROUND

1. Quality Assurance in Project Management

The technological advancements in the computer industry have penetrated all elements of our modern day society. With increased public acceptance and proliferation of the computer, we have seen the emergence of a more sophisticated class of end-user. This emergence has lead to increased demand on the development, maintenance, and modification of software systems to satisfy their insatiable appetite for new functionality. [Ref. 1:p. 100]

Hardware technology has progressed at a faster rate than our ability to develop comparable software systems. The result is an industry-wide backlog of software projects. Since software is a very critical and complex component of most major systems, the expanding hardware/software gap is becoming an increasing concern to system developers. The software development problem can be directly attributed to the lack of evolving software management techniques to keep pace with technological advances. [Ref. 2:pp. 6-7]

To compound this situation, the supply of qualified programmers and systems analysts is not keeping up with the software industry's demands. This adds to the growing trend of overdue and over budget projects, that when delivered don't

perform as required. Many clients find that functional requirements have been down scaled to cut further cost increases or reduce time delays. These shortcuts often lead to an increase of undetected errors in the final delivered system. [Ref. 2:pp. 6-7]

In resolving the software development dilemma we must focus our attention on the managerial issues present today. Many project managers have failed to learn or admit past mistakes and are doomed to repeat them on future projects. One example is the practice of attempting to introduce quality into the developed system at the final testing phase. This is usually insufficient in resolving many of the hidden errors committed during the early development phases of the project. Software productivity directly corresponds to the development practices that have been implemented by management. Only through the implementation of sound project management tools and techniques can we begin to reverse the current climate surrounding systems development. [Refs. 3:pp. 3-5; 2:pp. 6-7]

Quality assurance (QA) is one technique that positively impacts software productivity. This technique introduces into project controlled systematic development processes, which ensure quality is built into the system from the beginning. Walkthroughs, inspections, code readings are a few of the quality checks that are scheduled at predetermined points, to continually review and test the

system. Of course these techniques are not without costs. The introduction of quality assurance into the software development process will increase total project costs (man-days). Therefore, the project manager must utilize cost-benefit analysis techniques to derive an economical QA distribution scheme (over the design and coding phases). [Ref. 4:pp. 1-2]

> Throughout software development, the management of software quality must be an overriding concern of all project personnel. Quality must be planned into the project structure, constantly evaluated, and corrections applied when deficiencies are identified. [Ref. 2:p. 8]

2. Expert Simulators

The computing demands of business and industry in the last decade have contributed to the emergence and expansion of new artificial intelligence (expert systems) and operational research (simulation) disciplines. These advances, coupled with the increasing complexity in the business world, has lead to growing interest in merging expert systems and simulation technologies into a single management system. "Proponents of AI have stated [expert systems] will revolutionize managements' use of computing, and have profound effect on the art and science of simulation." [Ref. 5:p. 723]

Expert systems and simulation models are very similar technologies. Both attempt to model reality, however use drastically different methods and tools to accomplish similar end result. Expert Systems focus on capturing the problem

solving techniques of the human expert, often in the form of a heuristic.

These systems are generally composed of three basic subsystems:

- User Interface--enables the system to interact with the outside environment.
- Inference Engine--uses deductive reasoning to drive the system towards a conclusion.
- Knowledge Base--collection of rules that model the expert's knowledge in the form of heuristics or "rules-of-thumb."

A key feature that distinguishes an expert system is its ability to justify a solution, in much the same manner as a human expert. [Ref. 6:pp. 12-18]

Simulation is a process of developing a model of reality, and using it to carry out experimentations. This technique is math intensive, using algorithms in a repetitive manner to derive a solution. The primary function of a simulator is to model how a system behaves over a period of time. Beginning with a "current" state it searches to determine if certain preconditions have been met, and if so moves to a "future" state. [Ref. 7;pp. 701-702]

The similarities between expert systems and simulation can be exploited in one general taxonomy where both share a common data base and cooperate in accomplishing a task. Parallel combination of an expert system and a simulator is one of the more common development methods used in joining these systems. (See Figure 1-1)

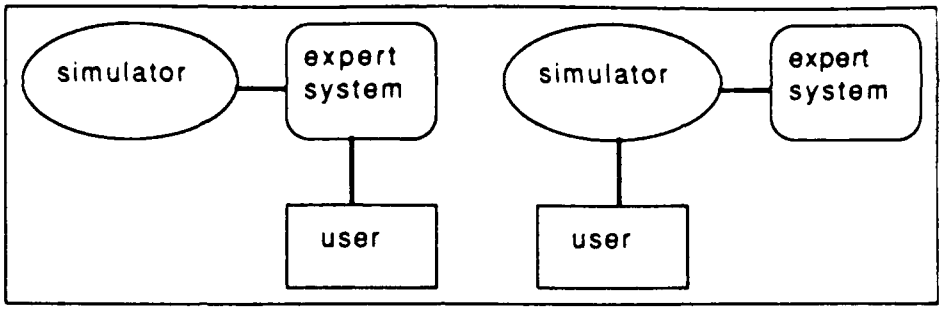


Figure 1-1 Parallel Expert System and Simulator

Both systems are developed separately and special interfaces are created to facilitate their interaction. Figure 1-1 displays two methods of integration, where either the expert system or the simulator acts as the front-end user interface for the model. [Ref. 8:pp. 10-12]

B. THESIS OBJECTIVES

The emphasis of this thesis is the continued development of an existing expert system simulation model (see Reference 11). Both the expert system and simulation model were designed, developed, and implemented as separate software, in parallel. The purpose of this combined system is to provide a project manager with an optimal quality assurance distribution scheme, throughout a software project's life cycle (design and coding phases).

An experimental systems dynamics model for software project management will be utilized as the simulator within this combined system. This model accepts the quality

assurance schemes from the expert system, performs project simulations, and provides cost information for evaluation.

The primary control vehicle of the expert system simulation model is the expert system module. Acting as an interface between the user and the simulator, it accepts input from both and controls the continuous simulation test process. The expert system uses an optimization heuristic for deriving and evaluating quality assurance distribution schemes.

This thesis will focus on identifying shortcomings that exist in the current expert system portion of this model. New search techniques will be implemented to improve the model overall performance. This enhanced expert systems simulation model will more accurately and efficiently identify the most optimal quality assurance distribution.

II. SYSTEMS DYNAMICS MODEL OF SOFTWARE PROJECT MANAGEMENT

A. OVERVIEW

In the last two decades, the technological achievements of the computer industry has caused a deluge of highly capable, reliable, and relatively inexpensive hardware. Accompanying this "technical revolution" has been the growing demand for more sophisticated software applications. As stated in Chapter I, the software development industry has not been able to effectively keep pace with these demands. "Software systems development has been plagued by cost overruns, late deliveries, poor reliability, and user dissatisfaction." [Ref. 9:p. 1426]

Recently, the software industry has made some technical advancements in the software production process. The creation of effective program development techniques such as structured system design, reverse engineering, and case tools have all lead to improvements in the production of software. Yet, the research community credited with these advancements have given very little attention to the managerial aspects involved with software development. This lack of attention may account for the persisting difficulties encountered when producing software systems. [Ref. 9:p. 1426]

Effective management of software production is not well understood. Based on this fact, research was conducted to

capture the dynamic properties of the software development process. Interviews with project managers of five larger development organizations were combined with the data from exhaustive literature searches to create a developmental base. A system dynamics model was developed from this research that provides management with a vehicle for making predictions about the software development process. This model integrates the multiple functions of software development with management functions (i.e., planning, controlling, staffing) and production activities (i.e., design, coding, and testing). Another feature of the model is the use of feedback principles. Feedback provides clarity to the "complex conglomerate of independent variables that are interrelated in various nonlinear fashions" within the model. The purpose of this feedback system is to assist in evaluating the complex circular (cause-effect) relationships that exist in the development world. [Ref. 4:p. 4]

The system dynamics model is comprised of four subsystems: (1) the human resource management subsystem; (2) the software production subsystem; (3) the controlling subsystem; and (4) the planning subsystem. Figure 2-1 illustrates a simplified view of how these subsystems interrelate. [Ref. 9:p. 1429]

The human resource subsystem captures the hiring, training, assimilation, and transfer of the project staff. It segments these resources into two categories, "newly hired" and "experienced." This segregation allows the system to keep

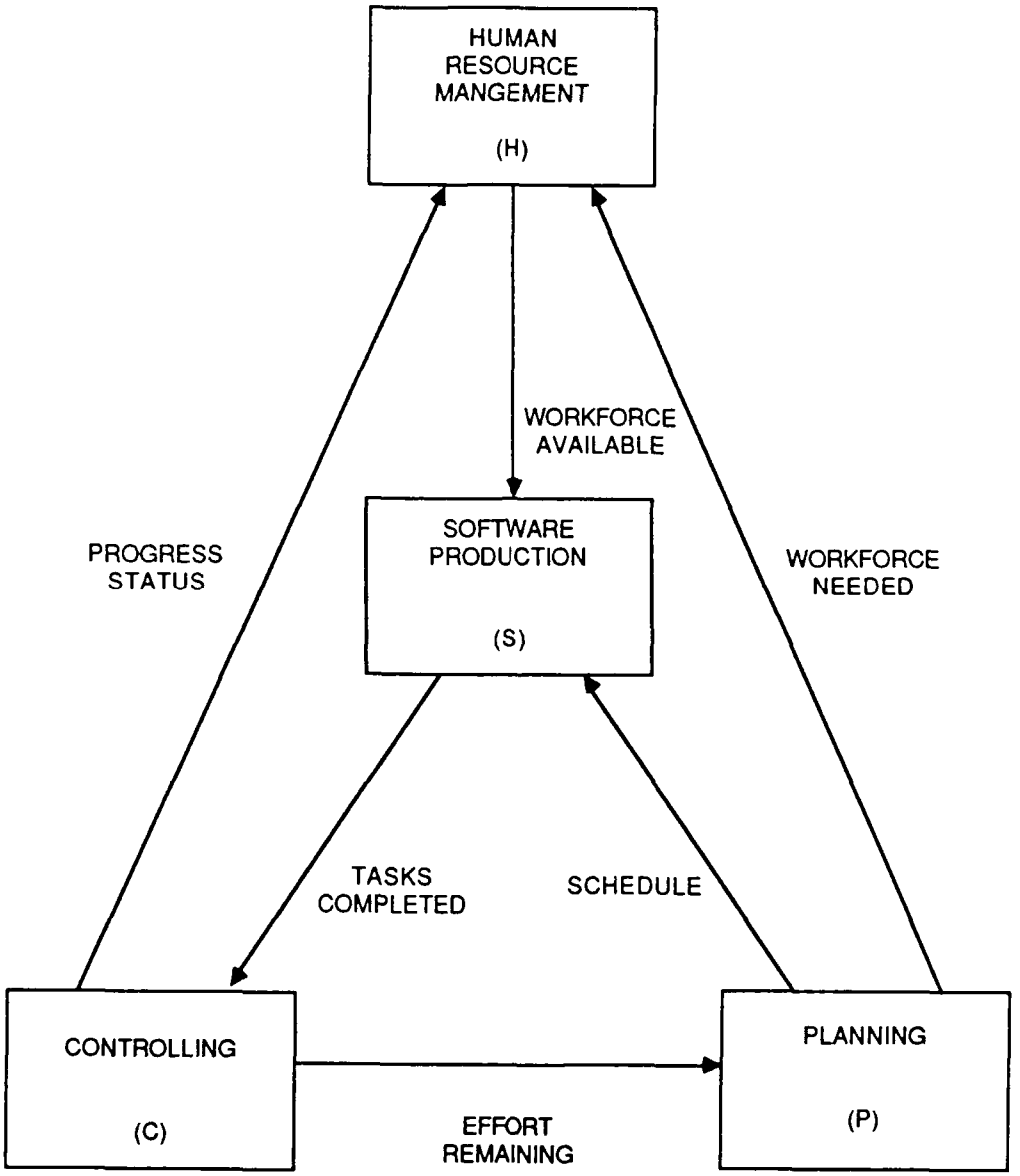


Figure 2-1 System Dynamics Model

track of differing productivity levels for each member of the project team. Secondly, it provides a means of monitoring the experienced staffs training of new personnel. The subsystem accepts inputs from the model, such as WORK FORCE NEEDED, PROGRESS STATUS, and processes them with other factors to determine the appropriate WORK FORCE AVAILABLE. [Ref. 10:p. 102]

The software production subsystem models the development process through the design, coding, and testing phase of the software life cycle. The requirements, operations, and maintenance phases were not included because they fall outside the boundaries of the actual software development process and are not under the direct control of the development team. The focus of this subsystem is enhancing productivity. Through the use of quality assurance activities it detects project errors as soon as possible in the life cycle process. These "detected" errors are then passed to the rework sector for correction. Any "undetected" errors, that escape QA and rework, filter through to the testing sector for final detection and correcting. The underlying intent is to detect and correct errors early in the life cycle process. Design and code errors grow exponentially as the project progresses. Early corrective action minimizes the costs associated with there removal. This subsystem notifies the control subsystem of TASKS COMPLETED. [Ref. 10:pp. 102-103]

The control subsystem makes distinctions between actual and perceived model variables to estimate project progress. Often, human estimates are very inaccurate or over-inflated. This subsystem evaluates many variables built into the system to determine what the actual projects status is. This subsystem supplies the planning subsystem with EFFORT REMAINING. [Ref. 10:p. 103]

The final subsystem is planning. It provides the system with initial project estimates, and continually updates them as the project progresses. Evaluating scheduled times, work force stability, and training requirements, this subsystem supplies WORK FORCE NEEDED and SCHEDULE data to the other subsystems. [Ref. 10:p. 103]

B. QUALITY ASSURANCE

The quality assurance and rework sector is one of four major activities that comprise the software production subsystem. Its primary objective is the detection and subsequent correction of generated software errors. Receiving newly generated code from the software development sector, the QA activity uses accepted techniques, such code readings and periodic group walkthroughs to identify errors. Discovered errors are corrected by the rework portion of this activity, with all "undetected" errors passed to the testing sector. Figure 2-2 is a simplified view of the software production subsystem process. [Ref. 4:p. 6]

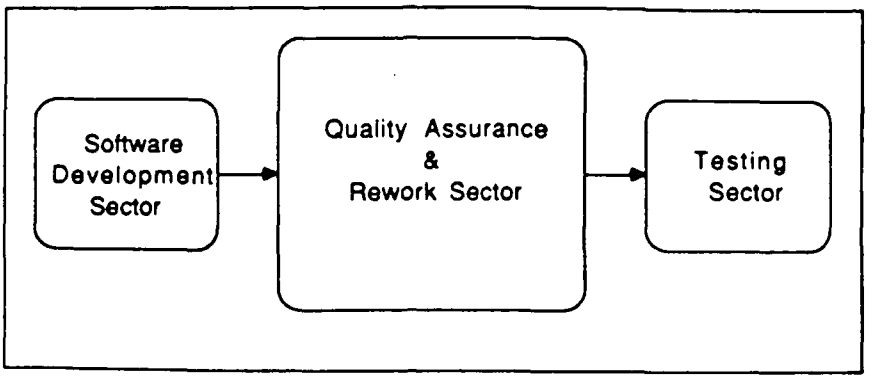


Figure 2-2 Software Production Subsystem

The quality assurance effort is assigned as a fixed portion of the total man-days (weekly) available for a given project. This is distributed over ten life cycle points through the project's design and coding phases (i.e., 10%, 20%,30%,...,100%). These values can be entered by the project manager using a table variable called PLANNED FRACTION OF MANPOWER FOR QUALITY ASSURANCE (TPFMQA). The actual values assigned to TPFMQA are percentages of the total man-days available that are desired for QA effort. For example, a value of 0.15 at the 10% life cycle completion point represents an allocation of 15% of total effort dedicated to QA. [Ref. 4:p. 6]

A large segment of the of the software production industry is akin to allocating an even QA percentage throughout the entire project life cycle. This practice obviously does not take cost effectiveness into consideration. (See Figure 2-3.)

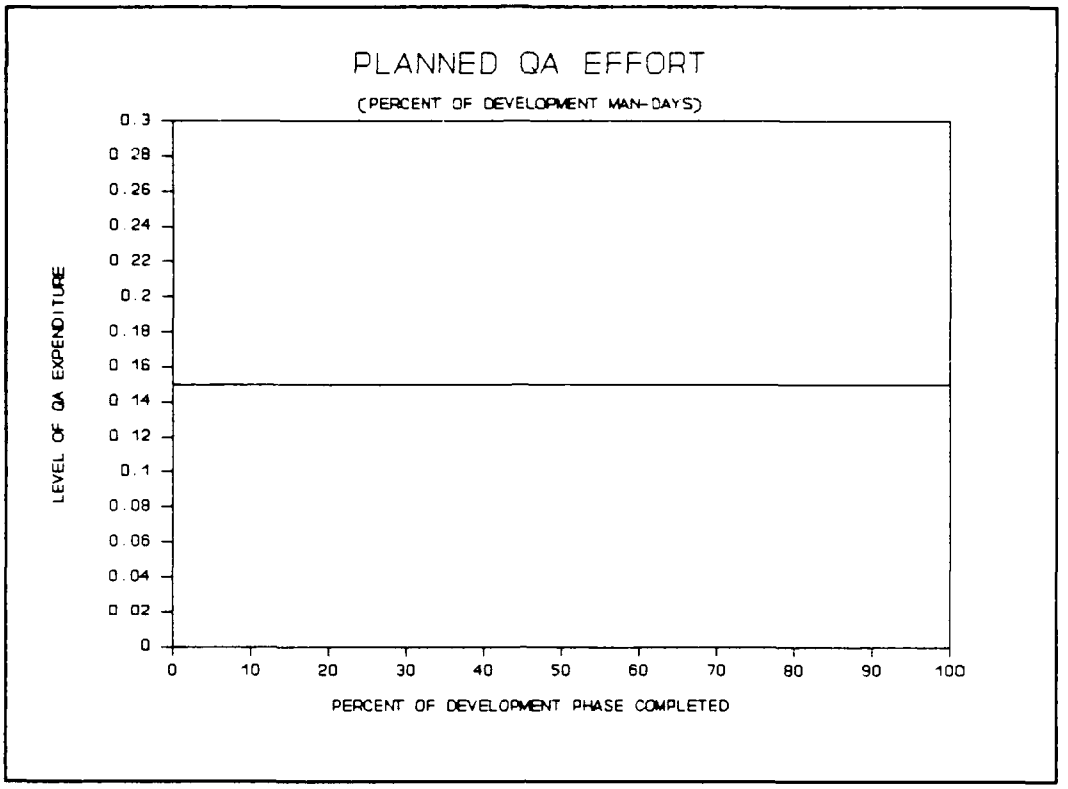


Figure 2-3 Uniformly Distributed QA Effort

The number of errors detected in a software project directly correlates to the amount of QA effort allocated. The greater the QA effort during the design and coding phases means the fewer the errors that will be left to be corrected when testing the system. However, quality assurance is a very expensive activity. Excessive QA will lead to unnecessary project costs. As errors are detected and corrected it becomes increasingly more difficult, time consuming, and costly to uncover the few remaining errors. It becomes more economical to allow the elusive errors to be handled during the system testing phase. [Ref. 4:pp. 7-8]

C. OPTIMIZATION SCHEME FOR QA EFFORT (TPFMQA)

The model's QA table variable (TPFMQA) allows the project manager to experiment with various distributions to find the most economic solution without sacrificing quality. A three-step process can be performed at each of the ten life cycle points.

First, we apply a predetermined negative pulse to a life cycle (l.c.) point leaving all other points unaltered (i.e., 15% pulse = l.c. point - (l.c. point * .15), see Figure 2-4). The new value is entered into the TPFMQA table and a simulation run is conducted. If the results of the simulation

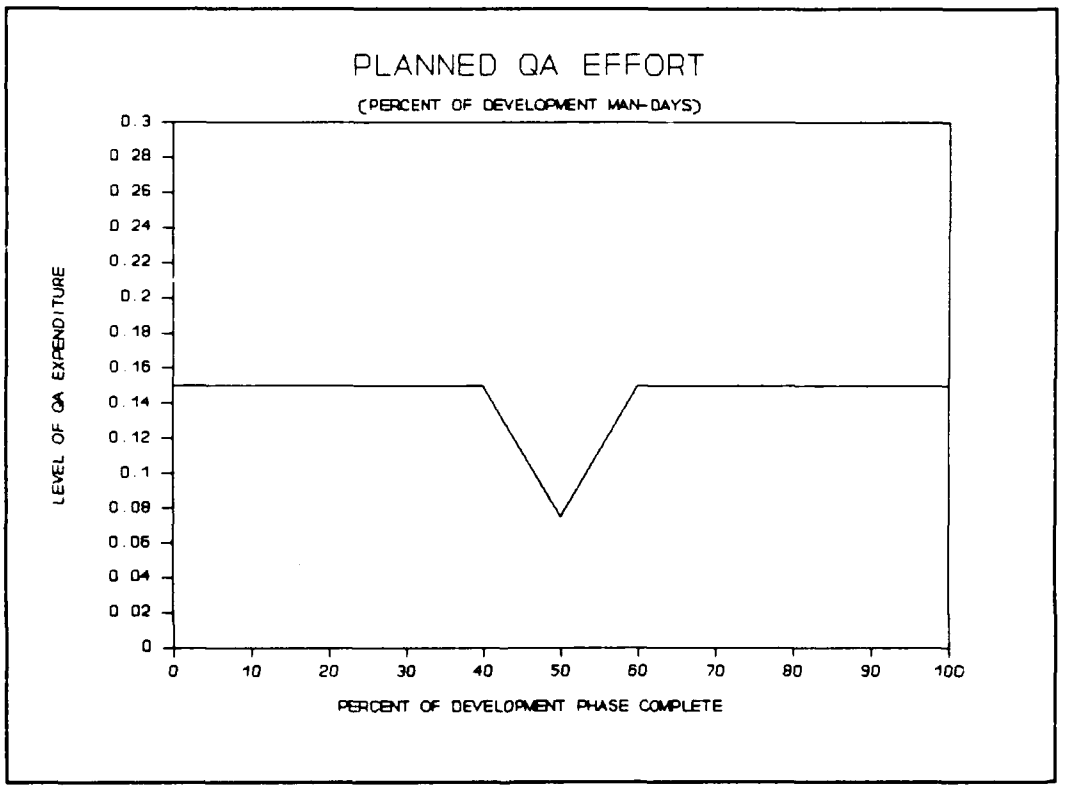


Figure 2-4 Example of a Negative Pulse

lower the total project cost the new value is adopted, and we proceed to the next life cycle point. If the results are worse we apply a positive pulse to the original value and conduct another simulation run. If neither perturbation causes improvement, the original value is retained. Perturbation tests on all ten life cycle point constitutes one complete cycle.

Figure 2-5 displays the results for an actual experiment using the system dynamics model. The project was initiated with a uniform 15% QA distribution (cost was 1656.71 man-days). After only one cycle the project manager was able to significantly reduce total project cost to 1537.99 man-days. This experiment support the theory that by emphasizing QA early in the design process, the effort required at the later portion of the project could be drastically reduced.

D. CASE STUDY

1. DE-A Project

To experiment with the cost-reducing capabilities of the system dynamics model, an actual NASA software project was used as a test platform. The Fortran-based program was developed for processing telemetry data, altitude determination, and control of a DE-A satellite. [Ref. 4:pp. 2-3]

The estimated and actual project results were as follows:

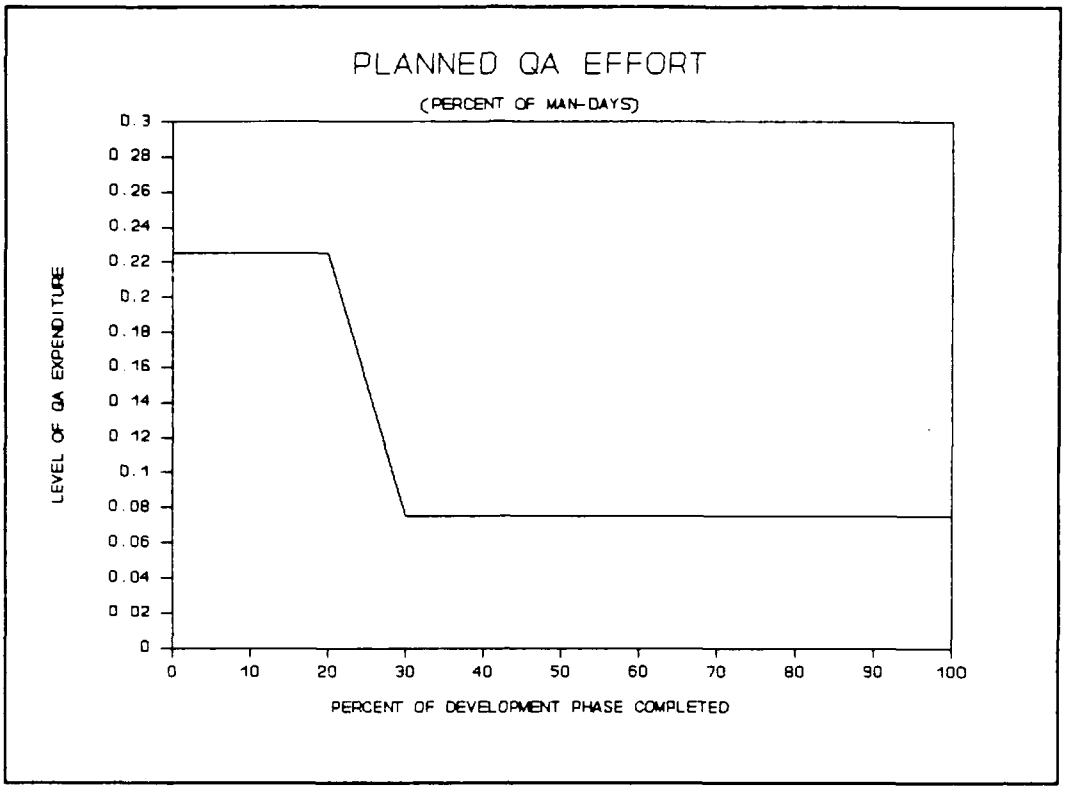


Figure 2-5 QA Distribution After One Cycle

	estimated	actual	
project size	16,000	24,000	DSI
development cost	1,100	2,200	Man-days
completion time	320	387	Days

These results indicate that the project was not a complete success, even though the system's performance was rated extremely reliable. Development costs were twice what was predicted, delivered source instructions (DSI) were one and a half times the estimate, and the project fell behind schedule. A possible contributor to these overruns and delays was the 36% (on average) allocation of the total available

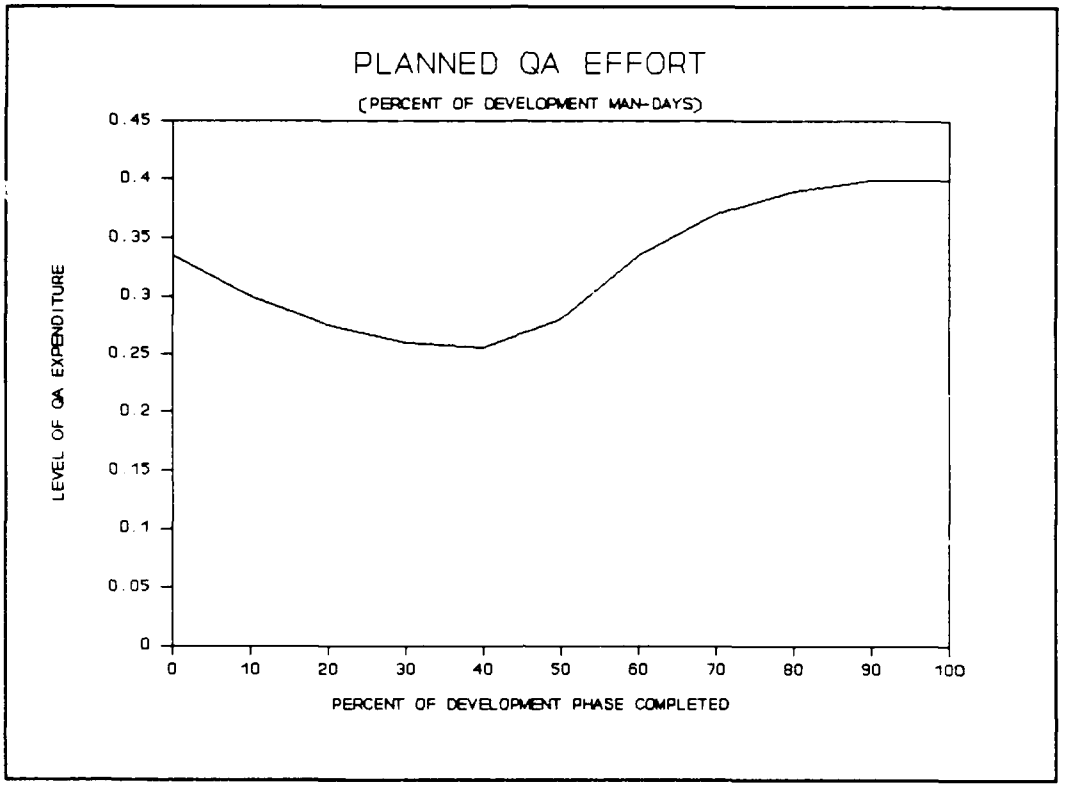


Figure 2-6 Actual DE-A Project QA Distribution

resources to quality assurance. Excessive QA was planned into the project to detect the relatively few elusive errors that may exist. This higher than industry norm distribution is show on Figure 2-6. [Ref. 4:p. 3]

2. Manual Experiment

Manual manipulations of the TPFMQA values were conducted to determine if a more cost effective QA distribution for the DE-A project could be discovered. The initial starting point for this experiment was a uniform 15%

allocation of resources for QA across the ten life cycle points. These life cycle points indicate the percentage of the project completed in increments of ten. Perturbing each point separately, negative and positive pulses at a magnitude of 50% were applied. If a negative pulse successfully lowered the total project cost it was adopted, and then experiment progressed to the next life cycle point. However, if the negative pulse was not successful, then a positive pulse was applied. When a perturbation failed in reducing costs the original value was restored. This process was continued until the improvements in costs between consecutive cycles became nominal. [Ref. 12:p. 43]

Figure 2-7 displays the improved QA distribution curve. The new allocation scheme resulted in significantly lowering development costs to 1524.5 man-days total. Quality assurance effort was reduced from 524 man-days (original 30% distribution) to approximately 162 man-days. The simulator successfully lowered overall costs while maintaining the desired level of quality in the final product. [Ref. 12:p. 43]

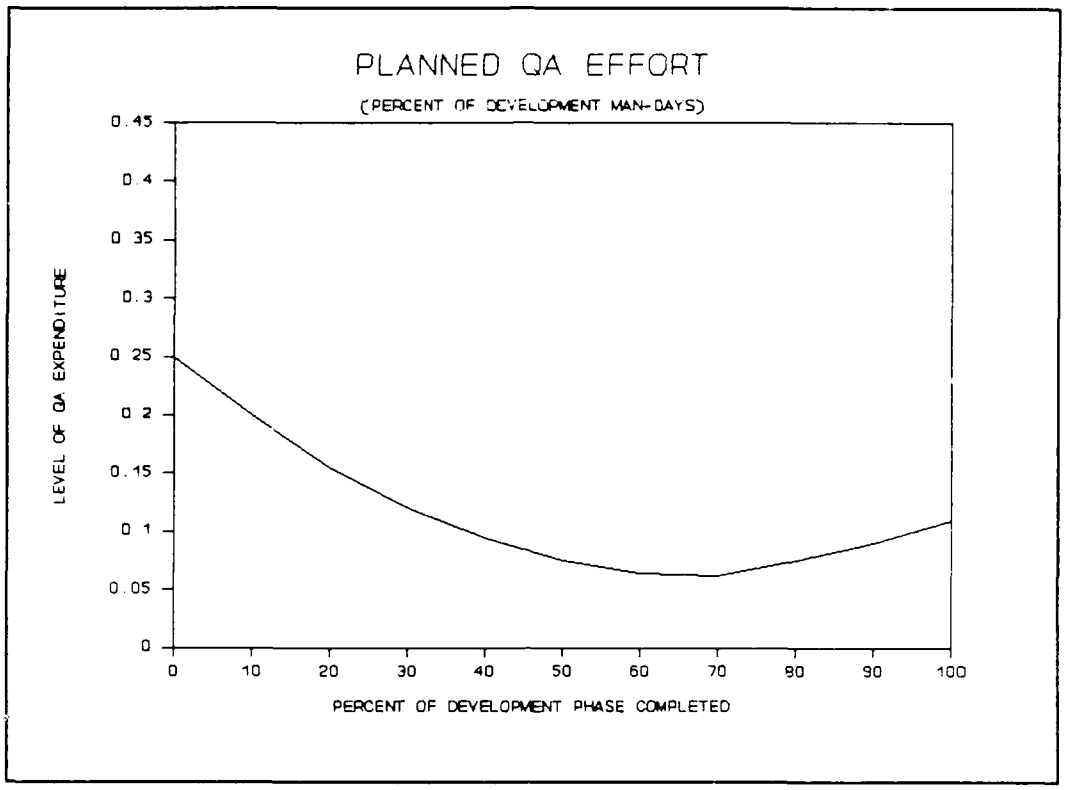


Figure 2-7 Manually Derived QA Distribution

III. PRIOR DEVELOPMENT IN EXPERT SYSTEM SIMULATION MODELS

A. EXISTING EXPERT SYSTEM FOR QA ALLOCATION

1. The Expert System Module

Prior research (Reference 11) has yielded a prototype expert system simulation (ESS) model that is decomposable into three separate subsystems: An expert system module, the system dynamics model, and a set of interfacing files, which are displayed below.

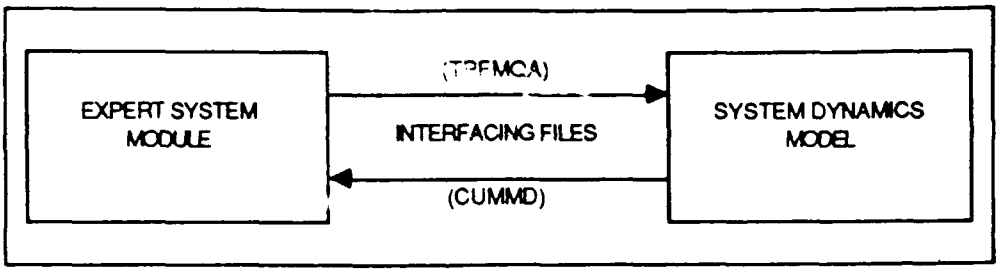


Figure 3-1 The ESS model

The expert module was developed in Prolog and is comprised of 15 rules. This module performs two functions, overall system control and derivation of QA effort allocation schemes. The derived QA patterns are transferred to the systems dynamics model, and a simulation is conducted. When the simulation is complete, the results are returned to the expert system for evaluation and further action. [Ref. 4:pp. 8-9]

The ESS model is initialized by the user through a generated series of expert module queries. Once initialized, the system operates in an automatic state until encountering a user-determined termination condition. Of the five required user inputs, two are critically important: (1) a desired pulse size factor (PSF), and (2) a base-line QA effort scheme for ten life cycle points (usually a uniform distribution across all points).

Pulse size factor is applied as a percentage of the current QA value at a given life cycle point. It mathematically alters (perturbs) each point for future evaluation in the simulation model. Each of the ten life cycle points are perturbed in sequential order, constituting a cycle when the system returns to the first point.

This perturbation process begins by applying a negative pulse to the current QA value (QA - PSF(QA)), sending the new distribution scheme (TPFMQA) to the simulation model, and adopting the perturbed QA value if the simulation results indicate a reduction in total project costs (CUMMD). If the negative pulse causes an increase in total costs, a positive pulse is applied (QA + PSF(QA)) and another simulation conducted. If this results in a cost reduction, the positive perturbation is adopted. However, if both pulses increase total project cost then the original QA effort value is restored. After the lowest cost alternative (negative pulse, positive pulse, or no pulse) is determined and recorded, the

system moves on to the next life cycle point for perturbation. [Ref. 12:pp. 17-40]

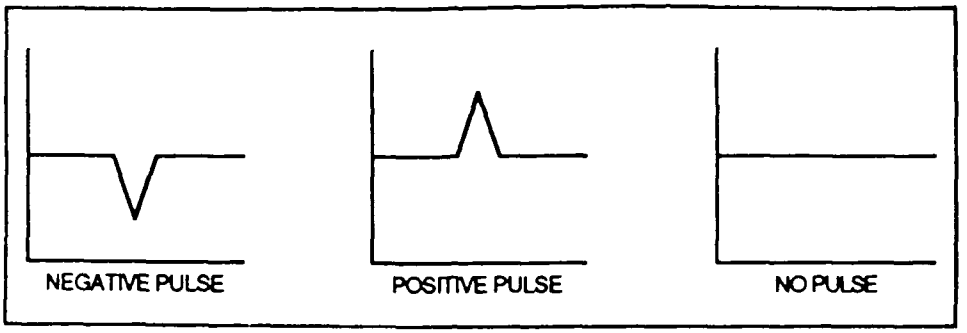


Figure 3-2 Perturbations

Other user inputs include two termination conditions: (1) the maximum cycle limit, and (2) minimum threshold level for cost reduction. The cycle limit terminates the system when the cycle counter surpasses a user-established maximum. The minimum threshold level terminates the system if the cost reduction at the end of a cycle is not greater than a set amount over the cost at the beginning of the cycle (e.g., a 1% improvement minimum). Finally, the user can input minimum QA value. This keeps the system from going below a possible company established minimum QA policy or at very least above zero. [Ref. 4:p. 10]

2. Interface Mechanisms

The two major components of the ESS model were developed in different programming languages. The expert system module is in Arity Prolog, and the systems dynamics

model is in Professional Dynamo Plus (PD+). To allow these separate systems to communicate, a group of interfacing DOS files were created as a transfer vehicle (see Figure 3-3). [Ref. 4:p. 10-11]

To communicate with the system dynamics model (project.dyn), the expert system (pqa.ari) requires a means of transferring the most currently derived QA distribution scheme. First, the expert system transforms the data into a format the simulator can understand (TPFMQA). Next, the data are copied into an ascii holding file (project.dnx). System control is then passed from the expert system to a DOS batch file (project.bat), which moves the TPFMQA data into the simulator and starts the model. Upon completion of the simulation, the systems dynamics model's report generator records the total project costs (CUMMD) to an internal file (report.exe). The batch file then extracts the CUMMD value and stores it in an ascii file (project.out), in a format understandable to the expert system. Control is then returned to the expert system which reads in the CUMMD value, evaluates performance, and determines the next course of action. The above process is repeated for each subsequent life cycle point. [Ref. 4:p. 11]

The summary file (summary.dat), attached to the expert system in Figure 3-3, records the results of each interface exchange. This file provides the user with a chronological listing of every QA distribution scheme attempted and the

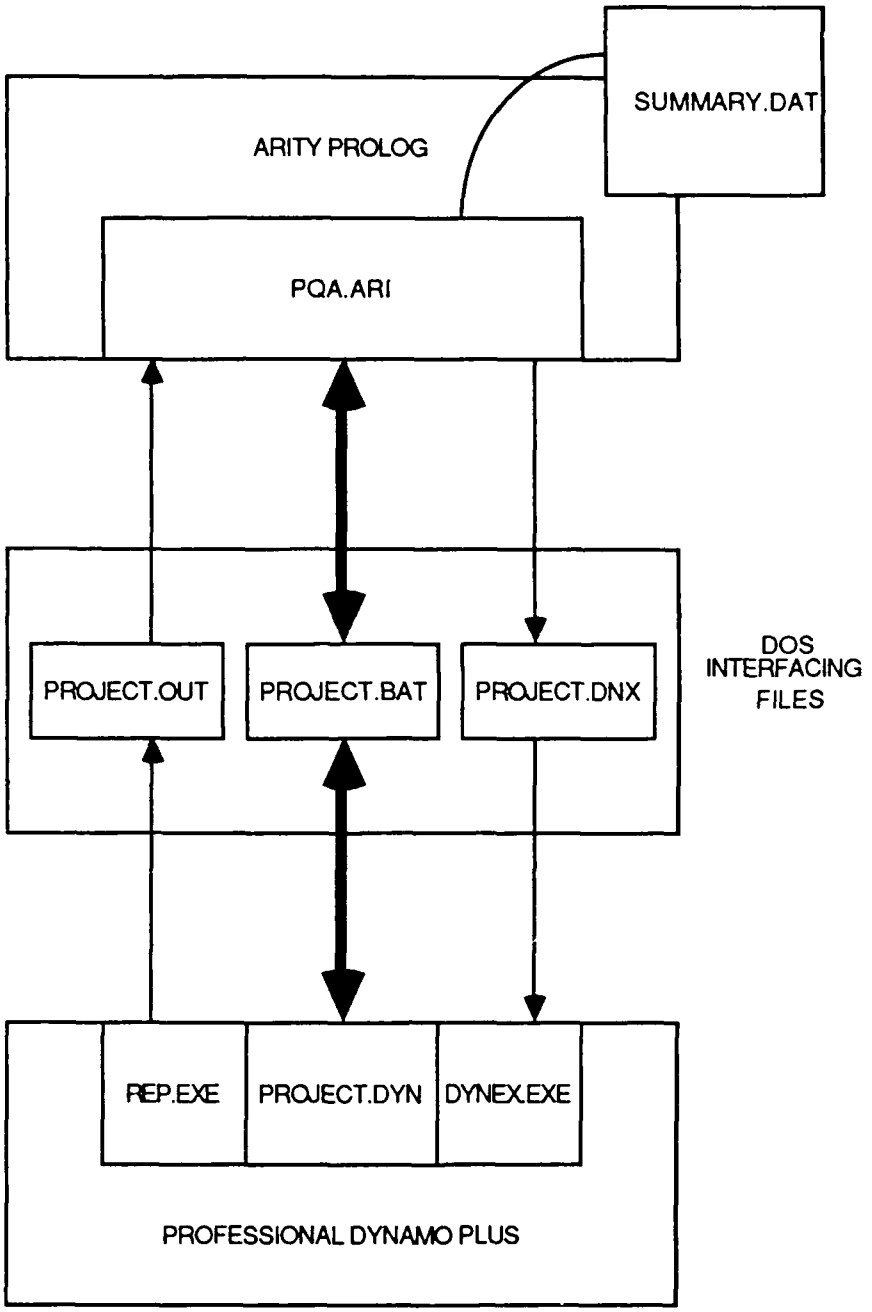


Figure 3-3 ESS Model File Structure

associated total project cost. When a final termination condition is met, the system records the best CUMMD discovered into the file.

3. DE-A Project Revisited

The QA allocation performance of the ESS model was evaluated using the NASA DE-A software project. The system was initialized using a pulse size (PSF) of 15%, minimum threshold for cost reduction of .00001%, maximum number of cycles at 30, minimum QA value of 3%, and a uniform base-line QA effort of 15% effort.

The results of this experiment are compared below:

	QA costs (man-days)	Total costs (man-days)
Actual DE-A	524	2,200
Manually derived	161.9	1,524.5
Expert Simulator	170.04	1,521.07

Figure 3-4 displays a composite of the QA distribution patterns for the three experiments listed above. Notice that the expert simulator places more emphasis on QA in the beginning of the design phase. This capitalizes on the cost savings associated with detection and correction of errors early in the life cycle process. The expert simulator's results are a considerable improvement over the actual QA effort distribution used by NASA in developing the project. Slight cost reduction over the manual manipulation of the system dynamics module was achieved. However, the real benefit of the prototype model was the elimination of the

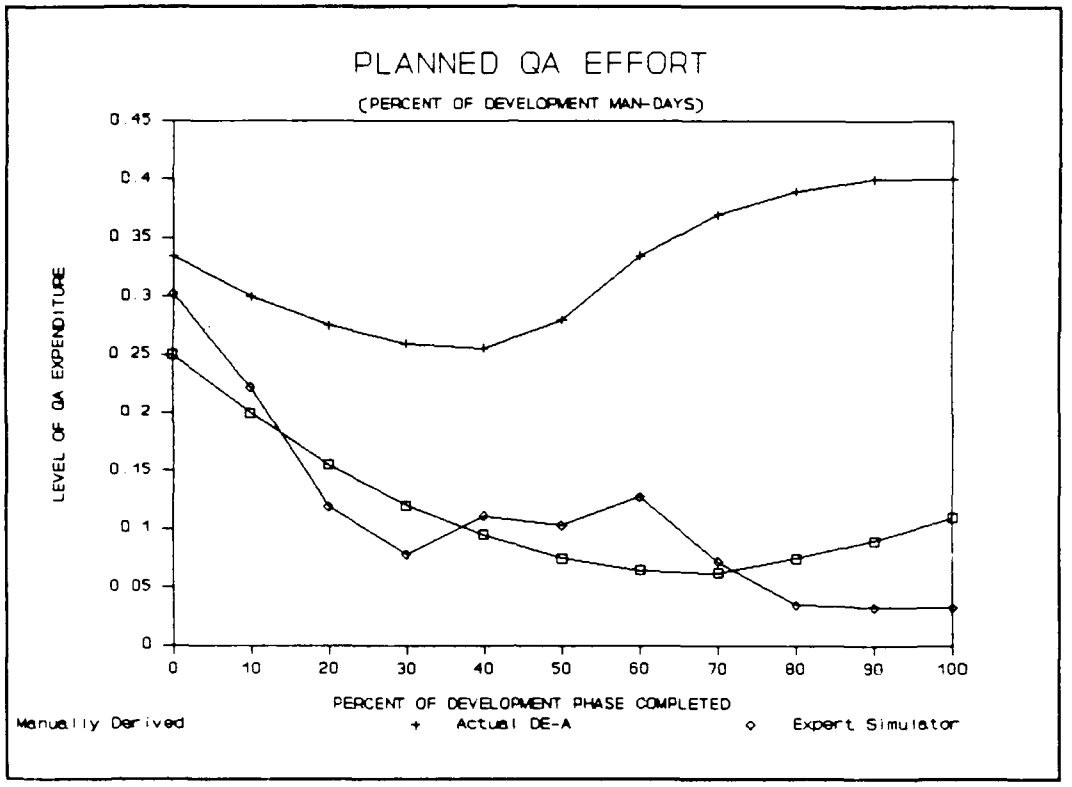


Figure 3-4 Composite of QA Distributions

extensive human interaction time needed when using the simulator separately. The prototype ESS model proved to be effective as an automated tool for distributing QA effort. [Ref. 4:pp. 11-12]

B. SENSITIVITY ANALYSIS OF THE MODEL

1. Introduction

The current ESS model has demonstrated a capability to produce distribution schemes that reduce project cost, when compared to the practice of uniform allocation. However, the ability of the model to derive the "most optimal" QA scheme is suspect. Multiple experiments were conducted in which the

initial five input variables were slightly altered. The results of these experiments produced significantly different QA allocation schemes and final project costs. The following model sensitivity questions have arisen for these experiments:

- Is the expert system simulator model sensitive to pulse size factor? (The prototype system used a static pulse size for every cycle.)
- Is it the system capable of achieving one global optimal QA distribution pattern irrespective of where the initial base-line is established? (The previous experiment used a 15% initial effort scheme across all life cycle points.)
- Is it possible for multiple local optimal solutions (at each life cycle point) to exist? If so, can this deceive the simulator into believing it has discovered the optimal for that life cycle point? (This phenomenon would cause multiple global optimization schemes to be produced by the existing model. See Figure 3-5 below.)

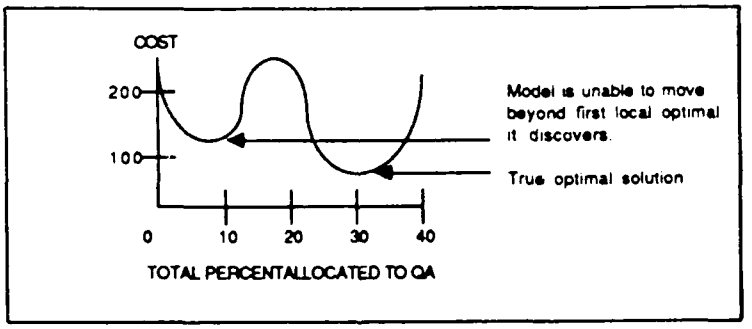


Figure 3-5 Bi-modal Problem

2. The Test Process

A sensitivity experiment was conducted and divided into two separate sub-experiments, each using different QA base-line values. The "low" base-line sub-experiment was established at 3% of resources allocated to QA and a "high"

base-line at 50%. These extreme starting points were used to examine the possible disparities in the model's global (final QA allocation pattern) and local (individual life cycle point) "optimal" allocation solutions.

The two sub-experiments were further divided into a set of three test runs each (for a total of six experiments). These test runs utilized a 15%, 05%, and 01% pulse size factor respectively. The variable pulse (PSF) was used to measure the sensitivity of the model to dynamic pulse size.

Each of the six test runs was limited to a maximum of 20 cycles with the minimum threshold level for cost reduction set at zero (effectively eliminating it as a model termination vehicle). The results of the previous higher PSF experiment were used as the base-line values to initialize the next run. For example, the results of the "high" base-line's first 20 cycle run at a 15% pulse size factor were: .293, .137, .137, .099, .116, .153, .137, .071, .071, .133. These values were then used as the starting points for the next 20 cycle run at 5% pulse size factor. Although the simulator did not reach an optimal solution for any of the six runs, the 20 cycle limit was chosen for the following reasons:

- To reduce the excessive time require to run the ESS model.
- The degree of change in the QA distribution pattern, with a static PSF, becomes insignificant after approximately 15 cycles.
- With a minimum threshold level of cost reduction set at zero, the model could theoretically run on indefinitely making minute changes in the allocation pattern.

3. Experiment Results

Both sub-experiments were initialized with a pulse size factor of .15. The resulting QA effort distributions for the first set of high/low runs are shown in Table 3-1 and Figure 3-6.

TABLE 3-1

RUN SET #1 AT 15% PSF

FIRST SET OF HIGH/LOW RUNS AT 15% PSF										
Base	10%	20%	30%	40%	50%	60%	70%	80%	90%	100%
50%	.293	.137	.137	.099	.116	.153	.137	.071	.071	.133
3%	.2[illegible]5	.283	.105	.070	.123	.123	.070	.069	.046	.060

Observing the two distribution patterns indicates that they have not only met but have over-lapped at various points. This indicates a bi-modal situation does exist for many of the life cycle points. Figure 3-6 also proves that the ESS model is not capable of effectively handling a bi-modal environment. The model tends to accept the first mode it encounters. Therefore, if a better local solution lies outside of the currently established pulse factor range, it will be overlooked. By being "trapped" into a local optimal value, the model creates inconsistent global distribution schemes dependent upon where the expert system is started.

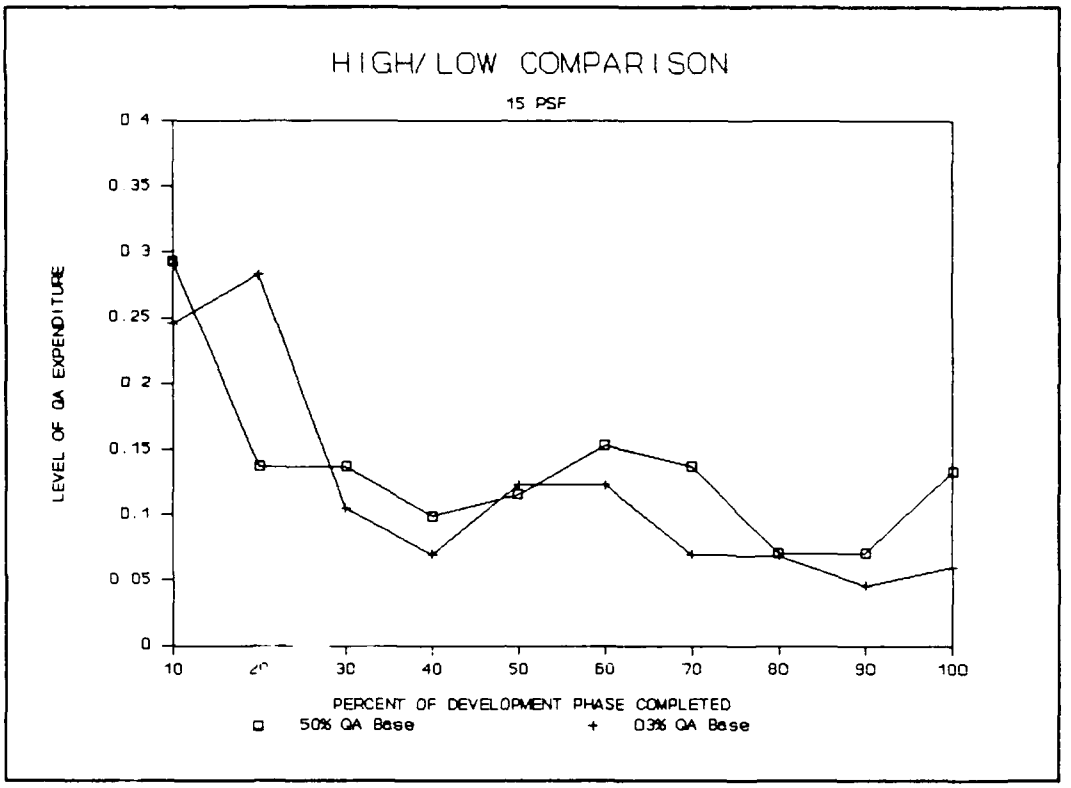


Figure 3-6 15% PSF High/Low Comparison

The distribution adjustment process for the first set of high/low runs (every four cycles) has been recorded in Table 3-2, Figures 3-7 and 3-8.

An interesting observation is that the expert system developed a definite allocation pattern within a relatively few cycles. With only minor refinements, these patterns resemble the final high/low distribution schemes from the third set of runs.

TABLE 3-2

HIGH/LOW DISTRIBUTION COMPARISON FOR FIRST 20 CYCLES

HIGH/LOW DISTRIBUTION ADJUSTMENTS										
3% BASE-LINE MOVEMENTS										
CYCLE	10%	20%	30%	40%	50%	60%	70%	80%	90%	100%
0	.030	.030	.030	.030	.030	.030	.030	.030	.030	.030
4	.053	.053	.030	.030	.035	.035	.035	.040	.040	.053
8	.093	.093	.053	.035	.053	.053	.030	.040	.040	.070
12	.162	.162	.081	.053	.081	.081	.046	.045	.040	.045
16	.246	.283	.105	.070	.123	.123	.076	.069	.046	.060
20	.246	.283	.105	.070	.123	.123	.076	.069	.046	.060
50% BASE-LINE MOVEMENTS										
CYCLE	10%	20%	30%	40%	50%	60%	70%	80%	90%	100%
0	.500	.500	.500	.500	.500	.500	.500	.500	.500	.500
4	.261	.261	.261	.261	.261	.261	.261	.261	.261	.261
8	.222	.137	.137	.137	.137	.137	.137	.137	.137	.137
12	.293	.137	.137	.099	.116	.153	.137	.071	.071	.133
16	.293	.137	.137	.099	.116	.153	.137	.071	.071	.133
20	.293	.137	.137	.099	.116	.153	.137	.071	.071	.133

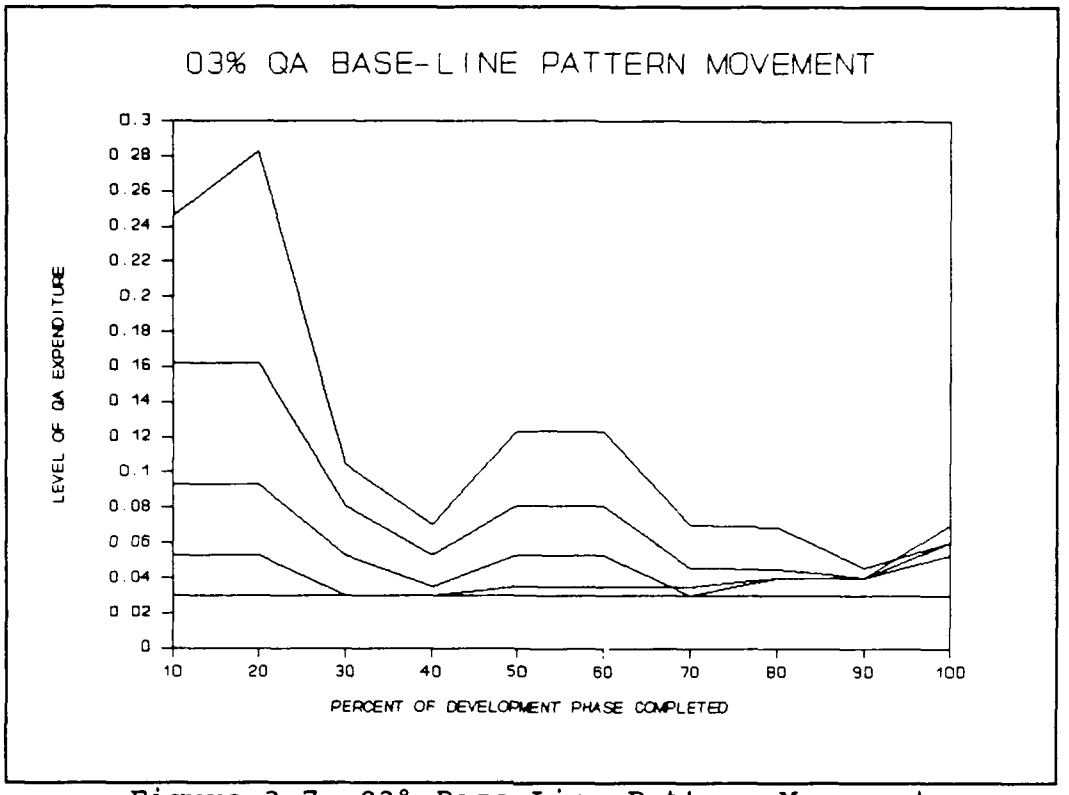


Figure 3-7 03% Base-Line Pattern Movement

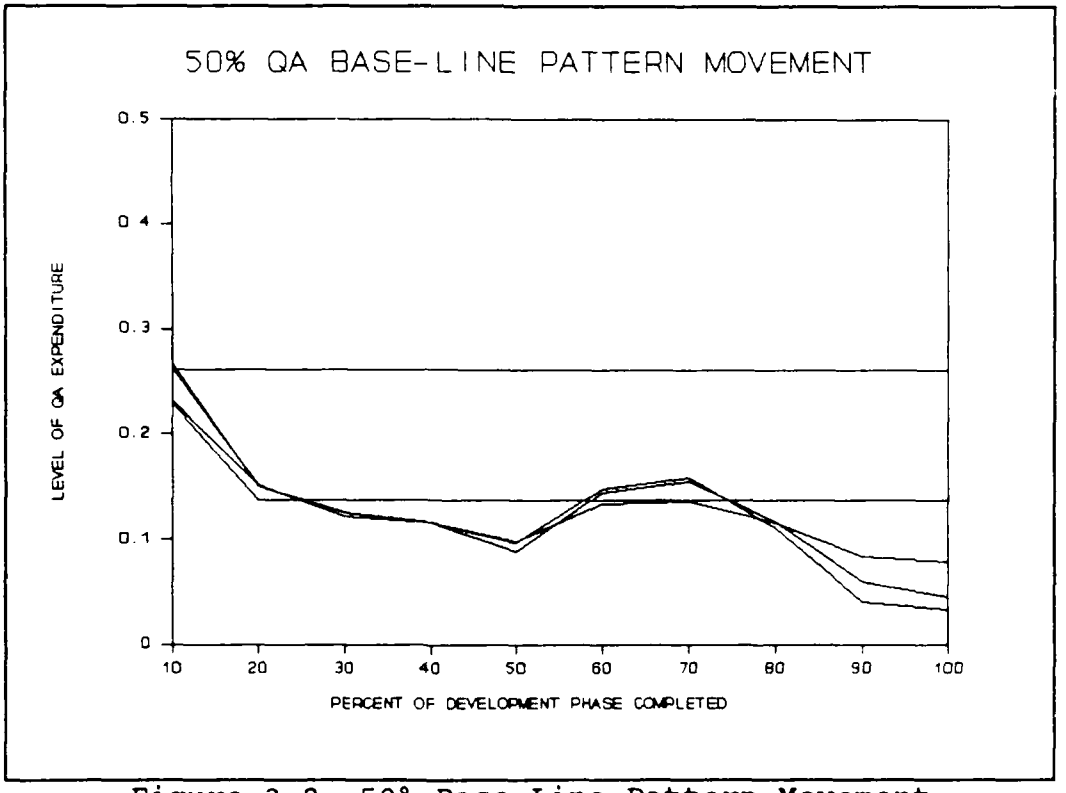


Figure 3-8 50% Base-Line Pattern Movement

The final distribution schemes from the first runs were used to initialize the model for a second set of high/low runs. A reduced pulse size factor of .05 was used for another 20 cycles. Table 3-3 and Figure 3-9 are composites of the second run's final distribution schemes.

TABLE 3-3

RUN SET #2 AT 5% PSF

SECOND SET OF HIGH/LOW RUNS AT 5% PSF										
Base	10%	20%	30%	40%	50%	60%	70%	80%	90%	100%
50%	349	.156	.150	.066	.129	.164	.106	.041	.046	.105
3%	.263	.283	.105	.073	.123	.141	.046	.060	.040	.059

The two patterns show a definite merging trend in over 50% of the project life cycle. This observation supports the theory that the model is sensitive to pulse size factor. Lowering the PSF allows the model to fine tune its global solution.

A final set of runs was conducted using the results from the second run and a .01 pulse size factor. The distributions, after 20 cycles, are shown in Table 3-4 and Figure 3-10.

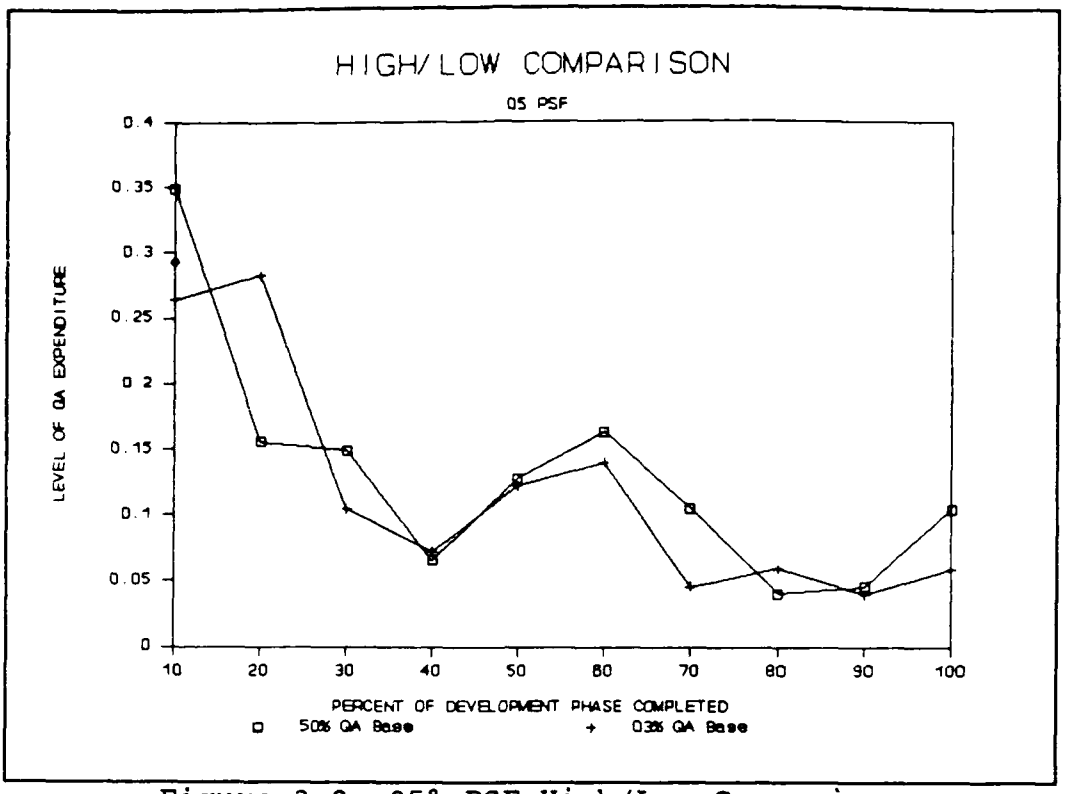


Figure 3-9 05% PSF High/Low Comparison

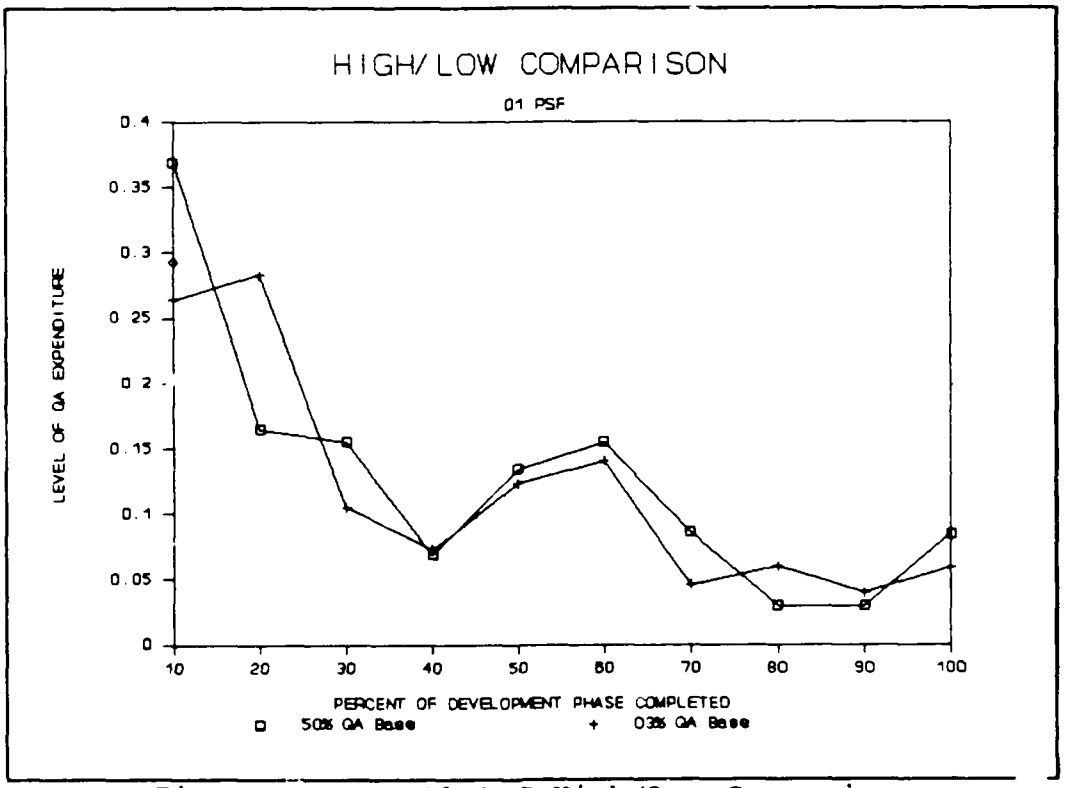


Figure 3-10 01% PSF High/Low Comparison

TABLE 3-4

RUN SET #3 AT 1% PSF

THIRD SET OF HIGH/LOW RUNS AT 1% PSF										
Base	10%	20%	30%	40%	50%	60%	70%	80%	90%	100%
50%	.369	.165	.155	.069	.134	.155	.086	.030	.030	.085
3%	.264	.283	.105	.073	.123	.141	.046	.060	.040	.059

This run provides more proof that the static pulse size of the expert system degrades overall model performance. Also, the increasing disparity between effort recommendations of the first three life cycle points further supports the model's inability to deal with complex environments.

C. REDESIGN CONSIDERATIONS

The existing expert system can be used as the basis for redevelopment. This will require minor modifications to the current 15 Prolog rules, and additions to resolve the identified limitations of the model. The interfacing files will remain unaltered, along with the technique of perturbing each life cycle point sequentially.

New rule additions will include a mechanism for reducing pulse size factor. The expert system will evaluate when the current PSF is no longer effective in processing change, and reduce its value automatically (not to go below a user established minimum pulse size). The model needs to

capitalize on the early emergence of a distribution pattern. This can be used to significantly reduce the number of cycles the expert system simulation model must go through in deriving solution. Secondly, the pattern can be used to "look beyond" local optimums, thus reducing the negative effects of bi-modal life cycle points.

With these additions, the need for inputting a maximum number of cycles can be eliminated. The system will be able to determine a optimal global solution in fewer cycles than currently required and subsequently run to completion.

IV. PATTERN SEARCH

A. METHODOLOGY

The limitations identified in the prototype ESS model can be rectified through the incorporation of an existing pattern search algorithm. The chosen optimization pattern search technique is the result of extensive academic research conducted by Robert Hooke and Thomas Jeeves [Ref. 13]. This technique is based on a simple movement principle that, "a successful earlier move in a particular direction is worth attempting again." [Ref. 13:p. 145]

This search algorithm identifies a pattern early, and cautiously makes further excursions in the directions of improvement. If these excursions prove to be successful, the algorithm adopts them as a base for further excursions, with an increasing step size per subsequent success. When an excursions fails to produce an improvement, the technique performs local explorations, from the unsuccessful location, as a last attempt for success. If improvement is achieved, the algorithm adopts these explorations as a base and continues the excursion process. However, if these local explorations are unsuccessful, the pattern is "destroyed," the base is reestablished at the last successful position, and the excursion step size (PSF) is reduced. This three-element technique accomplishes two tasks. First, it enables an

optimal pattern to be discovered quickly. Secondly, it allows fine tuning of the final solution when in the vicinity of a true minimum. [Ref. 13:p. 145]

This search technique begins with a set of base values (b0), which may be chosen arbitrarily (user input QA base-line values). A step size (PSF) is also chosen, which will be used for perturbations around the base values. The algorithm begins by applying a negative perturbation (b0 - PSF). If this new value causes an improvement, it is adopted as a new base (b1) and the algorithm moves to the next life cycle point for perturbation. However, if no improvement occurs, a positive perturbation is applied to the old base (b0 + PSF). If this new value causes improvement, it is adopted as the new base (b1); otherwise the original value at b0 also becomes the new base (b1) value. At this point the algorithm moves to the next life cycle point for perturbations. In summary, when attempting to minimize the objective function (y), the perturbation results are:

- $b1 = b0 - PSF$ if $y(b0 - PSF) < y(b0)$.
- $b1 = b0 + PSF$ if $y(b0 + PSF) < y(b0) < (b0 - PSF)$.
- $b1 = b0$ if $y(b0) < \min[y(b0 - PSF), y(b0 + PSF)]$.

Figure 4-1 displays the perturbation process over the first three project life cycle points.

When all perturbations have been performed, the algorithm moves into the pattern search portion of the technique. The old base values (b0) and the newly determined base values (b1)

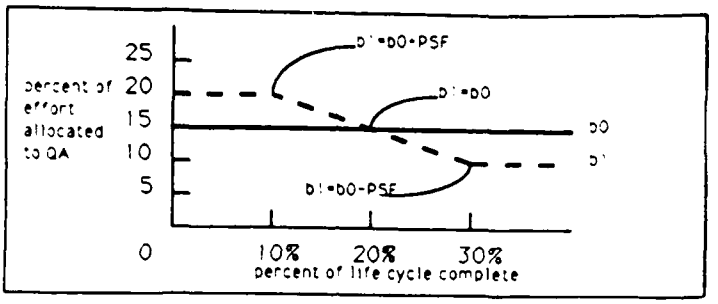


Figure 4-1 Perturbation Process

together establish the pattern search step size. Extending an arrow from b0 to b1 and doubling its length determine where the new temporary head (T) will fall. These temporary values are mathematically derived by the following equation:

$$T = b0 + 2(b1 + b0)$$

which is demonstrated in Figure 4-2 below.

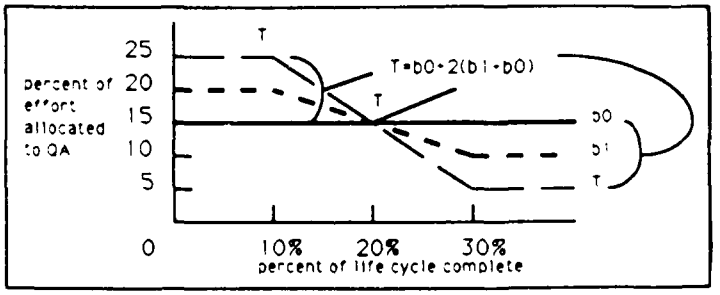


Figure 4-2 Pattern Jump

If the temporary head pattern (T) is an improvement (lowers total project costs) over the new base's (b1) distribution, the temporary values are retained by the

algorithm. Next, the new base values (b1) are redesignated as the b0 values, and perturbations are conducted around the temporary head values to determine the next set of b1 values (see Figure 4-3). [Ref. 13:p. 148]

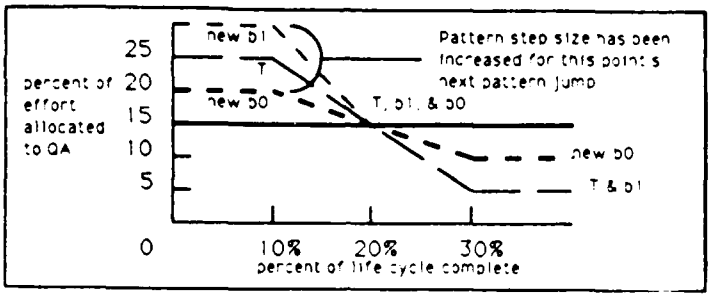


Figure 4-3 Perturbations/Pattern Jump Successful

As portrayed in the figure above, after all perturbations have occurred, the distance between all b0 points to b1 points (the ten life cycle values) are doubled constituting the "pattern search" movement to the next set of temporary heads.

This process continues until a temporary head distribution pattern fails to provide improvement over the new base (b1) distribution. When this occurs the algorithm conducts "local explorations" about the temporary head values (T). These local explorations are actually perturbations, except the automatic redesignation of b1 values to b0 values is temporarily placed on hold. If the local explorations succeed in producing an improved distribution pattern, they are adopted as the new base values (b1) and the redesignation of

the b0 values occurs (the old b1 values are redesignated the b0 values). The system continues as before (see Figure 4-4).

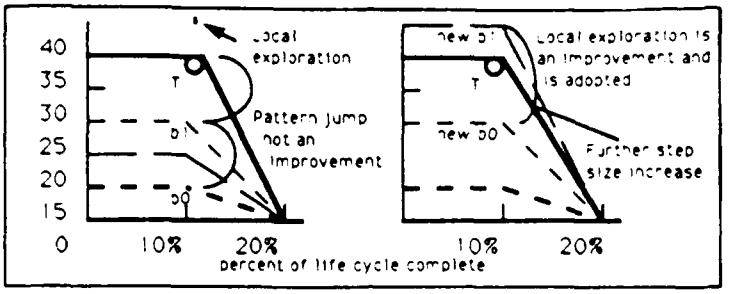


Figure 4-4 Local Explorations/Pattern Jump Unsuccessful

Figure 4-4 displays the increasing pattern size that can occur with repeated success in the same direction. [Ref. 13:pp. 148-149]

However, if the local explorations do not produce an improved distribution scheme, then the pattern is "destroyed" (see Figure 4-5). Next, the redesignation of the b0 values occurs and the step size (PSF) is reduced. A new set of perturbations are then applied to derive the new base values (b1) and the algorithm proceeds on as before. The search

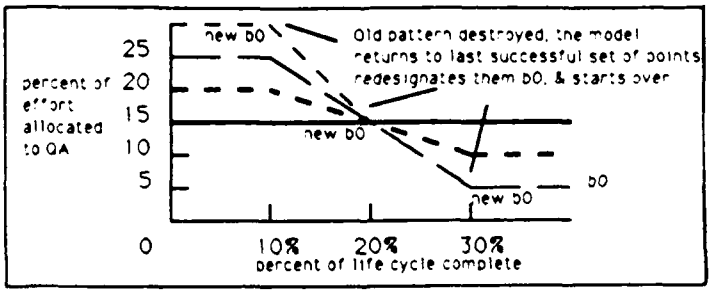


Figure 4-5 Pattern Destroyed

continues until step size falls below a preselected minimum (an input from the user). [Ref. 13:pp. 148-150]

This technique quickly and smoothly finds a pattern and capitalizes upon it to discover an optimal solution. When the technique is in close proximity to the global optimal scheme, it reduces step size to refine its final answer. The use of local explorations, when a pattern jump is unsuccessful, enables the algorithm to "look beyond" local optimal values. The combination of these characteristics gives the pattern search techniques a better likelihood of discovering the true optimal QA distribution than using just a perturbation process alone.

B. THE PATTERN SEARCH EXPERT SYSTEM ARCHITECTURE

The pattern search expert system module (pattern.ari) is a refinement of an existing prototype system. However, the underline purpose of the module remains the same; to derive the optimum quality assurance distribution for a given software project.

The pattern search technique was developed separately as a Arity Prolog rule-based subsystem. This seven-rule subsystem was then added to the existing 15-ruled prototype. Minor modifications to the original rules were required to facilitate this addition. All interfacing files and rules, used to communicate with the system dynamics model, were left intact.

The original 15-rule prototype system performed important administrative control functions for the ESS model. Interfacing with the user, running the simulation model, creating and maintaining files, and monitoring progress are just a few of its duties. This functionality will be carried over to the new expert system module. However, the prototype's most significant contribution to further development is its life cycle point perturbation technique. With minor modifications this technique will be combined with the three elements of the pattern search algorithm (pattern jumps, local explorations, and pulse size reduction) to enhance the expert system's performance.

The new expert system will at the end of each cycle (all ten life cycle points perturbed), transfer control to the pattern search algorithm. One of three possible event then ensue:

- Global adjustments will be made to the QA distribution in the form of a pattern jump. If successful in reducing total project costs, then the system continues on with the perturbation process, again.

- Global adjustments will be made to the QA distribution in the form of a pattern jump. If unsuccessful in reducing total project costs, then the system returns to the perturbation process to perform local explorations about the temporary head (actually the perturbation process with a flag set on to indicate the last pattern was unsuccessful).

- If the local explorations occurred and were unsuccessful in reducing total project costs then the pattern is destroyed, the pulse size factor is reduced, and the perturbation process is restarted from the last successful set of base values.

The rest of this section is dedicated to displaying and explaining each of PATTERN.ARI's 22 Arity Prolog rules. To differentiate the unaltered, modified and new program code the following will be typed after each subsection heading:

- (Original)--rules that were not altered by the redevelopment.
- (Modified)--rules that were altered by the redevelopment. If the rule name was also changed, the indicator will appear as (Modified - old name:_____).
- (Pattern Search)--rules that were added to the system.

A complete code listing of the pattern search expert system module is provided in Appendix A. For description of the prototype system the reader is directed to Reference 12, Chapter III. The following subsections will be presented in the same manner as the earlier work for continuity.

1. Rule--pqa (Modified)

This rule is used only once, in conjunction with the initial_run rule, to initialize the system. Four global variables are established to monitor all previous actions taken by the expert system module. The "newcycle" and "number" predicates keep track of the current cycle and iteration (life cycle point) the system is at, respectively. The "calc" predicate monitors whether the last applied pulse was negative or positive. Lastly, the "flag" predicate records if the last pattern jump was successful or not. These four global variables play a key role in determining the appropriate sequence of events within the system.

Next, this rule queries the user for five parameters: pulse size factor, minimum QA effort value, maximum QA effort value, minimum desired pulse size (used as the exit condition), and the initial ten base-line QA values. After the user has entered the tenth base-line value, the rule records these initial parameters in the SUMMARY.DAT file for future reference. The dopqa rule is called, whicn takes control and operates the system in an automatic mode until an optimal QA distribution pattern has been discovered.

```
pqa:-
/* sets initial global values */
   asserta(number(1)),
   asserta(calc(0)),
   asserta(flag(0)),
   asserta(newcycle(1)),
/* user input queries */
   write('What is your desired pulse size factor? '),
   read(PU),
   asserta(size(PU)),
   write('What is the minimum QA value? '),
   read(MN),
   asserta(min(MN)),
   write('What is the maximum QA value? '),
   read(AX),
   asserta(max(AX)),
   write('What is your minimum desired pulse size? '),
   read(MP),
   asserta(minpsf(MP)),
```

```
   write('Enter the initial QA distribution. Point 1  '),
   read(QA1),
   write('                              Point 2  '),
   read(QA2),
   write('                              Point 3  '),
   read(QA3),
   write('                              Point 4  '),
   read(QA4),
   write('                              Point 5  '),
   read(QA5),
   write('                              Point 6  '),
   read(QA6),
   write('                              Point 7  '),
   read(QA7),
   write('                              Point 8  '),
   read(QA8),
   write('                              Point 9  '),
   read(QA9),
   write('                              Point 10 '),
   read(QA10),
/* prints starting values into the summary output */
   print_head(PU,MN,AX,MP),
/* initializes the system with user inputs */
   initial_run(QA1,QA2,QA3,QA4,QA5,QA6,QA7,QA8,QA9,QA10),
   dopqa.
```

2. Rule--dopqa (Original)

This rule is the repeat/fail loop which automates the pattern search ESS model. To accomplish this, it makes use of the built-in back-tracking capabilities of the Prolog

language. The repeat statement always succeeds when encountered by the interpreter. This tells the interpreter to perform all statements that follow it. The main rule is then invoked, and when finished moves to the fail statement. The fail statement causes the interpreter to be "tricked" into believing it needs more data to make this statement succeed, and automatically back tracks in search of these data. It encounters the main rule and invokes it. After the main rule has finished, the interpreter further back tracks to the repeat statement and the cycle continues as before.

The rule continues to run indefinitely. The system runs until it encounters a termination condition elsewhere in the program.

```
dopqa:-
  repeat,
  main,
  fail.
```

3. Rule--main (Original)

This rule is the central control point for the perturbation process. It passes essential system status data, such as the current life cycle point for perturbations (number(ITER)) and last pulse direction applied (calc(TYPE)), to the pulse application rules. The rule recalls the previously derived CUMMD from the system's built-in recording

system (cummdold). It then compares the previous CUMMD with the most currently derived CUMMD (read_cummd), to determine if a negative (calc_less) or a positive (calc_more) pulse should be applied to the life cycle point. Finally, the rule outputs statistics and current life cycle undergoing perturbations to the screen for performance monitoring by the user.

```
main:-
/* gets the x value for the pulse */
   call(number(ITER)),
   call(calc(TYPE)),
/* gets the previous man days */
   PREV is [[[ITER + 8] mod 10] + 1],
   call(cummdold(PREV,CHECK)),
   call(cummdold(ITER,OLD)),
/* gets the man days from the last QA numbers */
   read_cummd,
   call(cummd(NEW)),
/* calculates the new y value (QA) for the current x value */
   case([NEW =< CHECK -> calc_less(ITER,NEW,OLD,TYPE),
        NEW > CHECK -> calc_more(ITER,NEW,OLD,CHECK,TYPE)]),
/* prints to screen module statistics for monitoring */
   statistics,
   write(' Iteration = '),write(ITER).
```

4. Rule--end_cycle (Modified--old name: quit_test)

This rule starts the pattern search algorithm after the perturbation cycle has been completed. It also advances

the cycle count (newcycle(NEXT)) and outputs the current cycle to the screen for monitoring.

```
end_cycle(NEW,ITER):-
   call(newcycle(NOW)),
/* initializes the "Pattern Search" algorithm */
   ifthenelse(NOW > 1,pattern_search(NEW,ITER),
            output_cummd(NEW,ITER)),
/* advances cycle number */
   NEXT is NOW + 1,
   write('Cycle number = '),write(NOW),
   retract(newcycle(NOW)),
   asserta(newcycle(NEXT)).
```

5. Rule--pattern_search (Pattern Search)

This rule uses the flag predicate value to determine a course of action. If TYPE = 0 it indicates that the last pattern jump was successful and to attempt another search (temp_base). However, if TYPE = 1 it indicates that the last pattern jump was unsuccessful and that local explorations were conducted. The rule calls the reduction_test rule to evaluate if the local explorations were successful.

```
pattern_search(NEW,ITER):-
   call(flag(TYPE)),
   case([TYPE =:= 0 -> temp_base(NEW,ITER),
        TYPE =:= 1 -> reduction_test(NEW,ITER)]),!.
```

6. Rule---reduction_test (Pattern Search)

This rule determines if the local explorations, following an unsuccessful pattern jump, have produced an improvement in total project costs. The rule calls on the last successful perturbation base value's CUMMD (b1_cummd(BASE)) and compares that value with the local exploration's CUMMD. If the local explorations produce a reduction in total costs the new temporary values are adopted as the base for the next pattern search. However, if the local explorations are not an improvement, then the reduce_pulse rule is invoked.

```
reduction_test(NEW,ITER):-
  call(b1_cummd(BASE)),
  ifthenelse(NEW < BASE,temp_base(NEW,ITER),
                    reduce_pulse(NEW,ITER)),!.
```

7. Rule--temp_base (Pattern Search)

This rule actually performs the pattern jump on the ten life cycle points. First, it outputs the final perturbation cycle CUMMD to the SUMMARY.DAT file (output_cummd). Then it resets all base CUMMD recording files with the most current values (b0_cummd and b1_cummd). Next, each life cycle point is adjusted mathematically creating the new pattern search distribution. The rule then calls new_base rule to redesignate the old base (b0) and the temporary head

(T) values. A pattern search header and listing of the new temporary values are recorded in the SUMMARY.DAT file. Next, the simulation model is called (shell(model)), and the total project costs for the pattern search distribution is determined. The simulation derived CUMMD is then called back into the rule (read_cummd) and recorded into the SUMMARY.DAT file. A comparison between the pattern search distribution CUMMD (HEAD) and the last perturbation distribution CUMMD (NEW) is conducted. If the pattern search fails to lower total project costs the local_explorations rule is called. Finally, the system updates the cummdold recording system and erases the temporary head (T) values for the next pattern search run (the T values were redesignated as TPFMQA values by the new_base rule).

```
temp_base(NEW,ITER):-
   output_cummd(NEW,ITER),
/* resets "flag" to record the last action taken */
   abolish(flag/1),
   asserta(flag(0)),
/* resets both base CUMMD value holders */
   call(b1_cummd(OLD)),
   abolish(b0_cummd/1),
   asserta(b0_cummd(OLD)),
   abolish(b1_cummd/1),
   asserta(b1_cummd(NEW)),
/* pattern search algorithm */
/* point one */
```

```
    call(b0(1,Q1)),
    call(tpfmqa(1,QA1)),
    A1 is Q1+[2*[QA1-Q1]],
    qa_test(A1,T1),
    asserta(t(1,T1)),
  /* point two */
    call(b0(2,Q2)),
    call(tpfmqa(2,QA2)),
    A2 is Q2+[2*[QA2-Q2]],
    qa_test(A2,T2),
    asserta(t(2,T2)),
  /* point three */
    call(b0(3,Q3)),
    call(tpfmqa(3,QA3)),
    A3 is Q3+[2*[QA3-Q3]],
    qa_test(A3,T3),
    asserta(t(3,T3)),
  /* point four */
    call(b0(4,Q4)),
    call(tpfmqa(4,QA4)),
    A4 is Q4+[2*[QA4-Q4]],
    qa_test(A4,T4),
    asserta(t(4,T4)),
  /* point five */
    call(b0(5,Q5)),
    call(tpfmqa(5,QA5)),
    A5 is Q5+[2*[QA5-Q5]],
    qa_test(A5,T5),
    asserta(t(5,T5)),
  /* point six */
    call(b0(6,Q6)),
    call(tpfmqa(6,QA6)),
```

```
   A6 is Q6+[2*[QA6-Q6]],
   qa_test(A6,T6),
   asserta(t(6,T6)),
/* point seven */
   call(b0(7,Q7)),
   call(tpfmqa(7,QA7)),
   A7 is Q7+[2*[QA7-Q7]],
   qa_test(A7,T7),
   asserta(t(7,T7)),
/* point eight */
   call(b0(8,Q8)),
   call(tpfmqa(8,QA8)),
   A8 is Q8+[2*[QA8-Q8]],
   qa_test(A8,T8),
   asserta(t(8,T8)),
/* point nine */
   call(b0(9,Q9)),
   call(tpfmqa(9,QA9)),
   A9 is Q9+[2*[QA9-Q9]],
   qa_test(A9,T9),
   asserta(t(9,T9)),
/* point ten */
   call(b0(10,Q10)),
   call(tpfmqa(10,QA10)),
   A10 is Q10+[2*[QA10-Q10]],
   qa_test(A10,T10),
   asserta(t(10,T10)),
/* establishes a new temporary base */
   new_base,
/* notification in output that a pattern search occurred */
   open(S,'summary.dat',a),
   nl(S),write(S,
```

```
   '****************** Pattern Search ******************'),
   nl(S),nl(S),close(S),
/* runs the system dynamics model for pattern search CUMMD */
   output_tpfmqa,
   shell(model),
   read_cummd,
   call(cummd(HEAD)),
   output_cummd(HEAD,ITER),
/* if CUMMD not an improvement, local explorations occur */
   ifthen(HEAD >= NEW,local_explorations),
/* resets CUMMD recording system */
   retract(cummdold(ITER,NEW)),
   asserta(cummdold(ITER,HEAD)),
/* erases temporary head (T) values */
   abolish(t/2),!.
```

8. Rule--qa_test (Pattern Search)

This rule is used by the temp_base rule to keep from pattern jumping below the user established minimum QA level. The post pattern jump value is compared with the allowed minimum value (min(MINQA)) and the greater value is adopted.

```
qa_test(IN,OUT):-
   call(min(MINQA)),
   ifthenelse(IN < MINQA,OUT is MINQA,OUT is IN).
```

9. Rule--new_base (Pattern Search)

This rule performs two functions. First, it redesignates the last perturbation derived distribution as the

old base distribution (b0). Secondly, it redesignates the pattern search derived temporary head (T) distribution as the temporary base (TPFMQA's) to conduct the next set of perturbations from.

```
new_base:-
/* calls last perturbation and pattern search values */
   call(tpfmqa(1,QA1)),
   call(tpfmqa(2,QA2)),
   call(tpfmqa(3,QA3)),
   call(tpfmqa(4,QA4)),
   call(tpfmqa(5,QA5)),
   call(tpfmqa(6,QA6)),
   call(tpfmqa(7,QA7)),
   call(tpfmqa(8,QA8)),
   call(tpfmqa(9,QA9)),
   call(tpfmqa(10,QA10)),
   call(t(1,A1)),
   call(t(2,A2)),
   call(t(3,A3)),
   call(t(4,A4)),
   call(t(5,A5)),
   call(t(6,A6)),
   call(t(7,A7)),
   call(t(8,A8)),
   call(t(9,A9)),
   call(t(10,A10)),
/* resets old base values */
   abolish(b0/2),
   asserta(b0(1,QA1)),
```

```
   asserta(b0(2,QA2)),
   asserta(b0(3,QA3)),
   asserta(b0(4,QA4)),
   asserta(b0(5,QA5)),
   asserta(b0(6,QA6)),
   asserta(b0(7,QA7)),
   asserta(b0(8,QA8)),
   asserta(b0(9,QA9)),
   asserta(b0(10,QA10)),
/* resets a new temporary base values */
   abolish(tpfmqa/2),
   asserta(tpfmqa(1,A1)),
   asserta(tpfmqa(2,A2)),
   asserta(tpfmqa(3,A3)),
   asserta(tpfmqa(4,A4)),
   asserta(tpfmqa(5,A5)),
   asserta(tpfmqa(6,A6)),
   asserta(tpfmqa(7,A7)),
   asserta(tpfmqa(8,A8)),
   asserta(tpfmqa(9,A9)),
   asserta(tpfmqa(10,A10)),!.
```

10. Rule--local_explorations (Pattern Search)

This rule is invoked by the temp_base rule each time the pattern search distribution fails to lower total project costs. This rule sets the flag predicate to 1. On the next cycle this flag is detected by the pattern_search rule notifying it that the new perturbation distribution is to be treated as local explorations. Further evaluation of the results are required before the system can adopt these values.

The rule also outputs a local exploration header in the SUMMARY.DAT file to record the occurrence.

```
local_explorations:-
/* resets "flag" to record last action taken */
   abolish(flag/1),
   asserta(flag(1)),
/* notification in output that local exploration occurred */
   open(S,'summary.dat',a),
   nl(S),write(S,
   '**************** Local Explorations ****************'),
   nl(S),nl(S),close(S),!.
```

11. Rule--reduce_pulse (Pattern Search)

This rule is invoked by the reduction_test rule when local explorations are unsuccessful in lowering total project costs. The current pulse size factor (size (PULSE)) is brought into the rule and halved. This event is then recorded with a pulse reduction header in the SUMMARY.DAT file along with the new pulse size value. The pattern is then destroyed by resetting the distribution values back to the last successful perturbation values (b0). Next, the simulation model is run (shell(model)) with the reset values to derive CUMMD. Both the new CUMMD and reset distribution values (TPFMQA's) are recorded in the SUMMARY.DAT file, and the cummdold recording system is updated. Finally, the rule checks to see if the pulse reduction value is less than the

user input minimum PSF. If it falls below the minimum (minpsf(MPSF)), then the run_stop rule is invoked for system termination. Otherwise the flag is reset to 0 and control is transferred back to the perturbation process.

```
reduce_pulse(NEW,ITER):-
   output_cummd(NEW,ITER),
   call(size(PULSE)),
   call(minpsf(MPSF)),
   call(b0_cummd(OLD)),
/* reduces current pulse size (PSF) by half */
   REDUCED is round(PULSE / 2,3),
/* notification in output that pulse reduction occurred */
   open(S,'summary.dat',a),
   nl(S),write(S,
   '**********      Pulse reduction      **********'),
   nl(S),write(S,'Pulse size is: '),write(S,REDUCED),
   nl(S),nl(S),
   close(S),
/* resets old base TPFMQA values to those prior  */
/* to the pattern search and local explorations */
   call(b0(1,QA1)),
   call(b0(2,QA2)),
   call(b0(3,QA3)),
   call(b0(4,QA4)),
   call(b0(5,QA5)),
   call(b0(6,QA6)),
   call(b0(7,QA7)),
   call(b0(8,QA8)),
   call(b0(9,QA9)),
```

```
    call(b0(10,QA10)),
    abolish(tpfmqa/2),
    asserta(tpfmqa(1,QA1)),
    asserta(tpfmqa(2,QA2)),
    asserta(tpfmqa(3,QA3)),
    asserta(tpfmqa(4,QA4)),
    asserta(tpfmqa(5,QA5)),
    asserta(tpfmqa(6,QA6)),
    asserta(tpfmqa(7,QA7)),
    asserta(tpfmqa(8,QA8)),
    asserta(tpfmqa(9,QA9)),
    asserta(tpfmqa(10,QA10)),
    output_tpfmqa,
    shell(model),
    read_cummd,
    call(cummd(AFTER)),
    output_cummd(AFTER,ITER),
/* resets CUMMD value holders */
    abolish(b1_cummd/1),
    asserta(b1_cummd(OLD)),
    retract(cummdold(10,BEFORE)),
    asserta(cummdold(10,AFTER)),
    retract(cummdold(ITER,NEW)),
    asserta(cummdold(ITER,AFTER)),
/* terminates the model if pulse is below minimum */
    retract(size(PULSE)),
    ifthenelse(REDUCED >= MPSF,asserta(size(REDUCED)),
            run_stop(OLD)),
/* resets "flag" to record last action taken */
    abolish(flag/1),
    asserta(flag(0)),!.
```

12. Rule--print_head (Modified--old name: calc_zero)

This rule creates and records into the SUMMARY.DAT file all expert system derived QA distributions (TPFMQA) that are sent to the system dynamics model, and the total project costs (CUMMD) that are returned. It also records the user input parameters as a header to this listing.

```
print_head(PU,MN,AX,MP):-
  create(S,'summary.dat'),
  write(S,'     Pulse size factor  = '),write(S,PU),nl(S),
  write(S,'     Minimum QA value   = '),write(S,MN),nl(S),
  write(S,'     Maximum QA value   = '),write(S,AX),nl(S),
  write(S,'     Minimum pulse size = '),write(S,MP),nl(S),
  nl(S),close(S).
```

13. Rule--calc_less (Modified)

This rule applies the negative pulse during the perturbation process. It is always the first pulse applied to each of the ten life cycle points. Because of this fact it is tasked with the administrative duties of the perturbation process. These duties include: updating the cummdold recording system, outputting previously derived CUMMDs to the SUMMARY.DAT file, determining when the pattern search process should be invoked, and keeping a track of the original "pre-perturbation" QA value for a life cycle point (holder(QA)). After applying the negative pulse, the rule compares the new QA value (NEWQA) to the user established minimum QA value

(min(MINQA)). The greater of the two values is adopted. Next, the rule outputs the new QA distribution (output_tpfmqa) to the SUMMARY.DAT file and runs the simulation model with these values. Finally, it advances the iteration count (number(NEWITER)) and records the fact that a negative pulse was applied (calc(0)).

```
calc_less(ITER,NEW,OLD,TYPE):-
/* records man-days for this cycle */
   retract(cummdold(ITER,OLD)),
   asserta(cummdold(ITER,NEW)),
   ifthenelse(ITER =:= 1,end_cycle(NEW,ITER),
            output_cummd(NEW,ITER)),
/* reads the current QA values */
   call(tpfmqa(ITER,QA)),
/* establishes an "initial" QA holder for perturbation use */
   abolish(holder/1),
   asserta(holder(QA)),
   call(size(PULSE)),
/* calculates a new QA value (NEWQA) */
   NEWQA is round(QA-PULSE,3),
/* checks if the new QA is less than the minimum (MINQA) */
/* add the new QA value to the database */
   call(min(MINQA)),
   retract(tpfmqa(ITER,QA)),
   ifthenelse(NEWQA < MINQA,asserta(tpfmqa(ITER,MINQA)),
            asserta(tpfmqa(ITER,NEWQA))),
/* moves to the next life cycle position */
   retract(number(ITER)),
   NEWITER is [ITER mod 10] + 1,
```

```
   asserta(number(NEWITER)),
   ifthen(ITER =:= 1,output_break),
   output_tpfmqa,
   shell(model),
/* records that last pulse was negative */
   retract(calc(TYPE)),
   asserta(calc(0)).
```

14. Rule--calc_more (Modified)

This rule applies to both the positive pulse and no pulse (return to the original value) during the perturbation process. It is only invoked by the main rule when the negative pulse was unsuccessful at reducing total project costs. First, the rule internally resets the iteration count back one, to allow the rule to reference the current life cycle point data (the prior negative pulse advanced the iteration count!). When the last TYPE value is 0 (last pulse was negative), the rule applies a positive pulse and calls the calc_up rule. Likewise, if the TYPE value is 1 (last pulse was positive), the original QA value before any perturbations were applied is restored and the calc_orig rule is called. The element of this rule is the maximum QA value limiting code. The derived QA value from the positive pulse is compared with the maximum QA value input by the user. The minimum of the two values is adopted.

```
calc_more(ITER,NEW,OLD,CHECK,TYPE):-
/* resets the life cycle position back 1 */
   NEWITER is [[[ITER + 8] mod 10] + 1],
   call(tpfmqa(NEWITER,QA)),
/* calls the "initial" QA value for perturbation use */
   call(holder(VALUE)),
   call(size(PULSE)),
/* computes the new QA value depending on whether the last */
/* pulse was negative (TYPE = 0) or positive (TYPE = 1) */
   case([TYPE =:= 0 -> NEWQA is round(VALUE+PULSE,3),
         TYPE =:= 1 -> NEWQA is VALUE]),
/* checks if new QA is greater than the maximum (MAXQA) */
   call(max(MAXQA)),
   retract(tpfmqa(NEWITER,QA)),
   ifthenelse(NEWQA > MAXQA,asserta(tpfmqa(NEWITER,MAXQA)),
              asserta(tpfmqa(NEWITER,NEWQA))),
   retract(calc(TYPE)),
/* resets the type of calculation */
   case([TYPE =:= 0 -> calc_up(NEW,NEWITER),
         TYPE =:= 1 -> calc_orig(NEWITER,ITER,NEW,OLD,CHECK)]).
```

15. Rule--calc_up (Original)

This rule follows the application of a positive pulse by the calc_more rule. It records the last simulation derived CUMMD and QA distribution (TPFMQA) intc the SUMMARY.DAT file. Secondly, it records the fact that a positive pulse was applied (calc(1)). And finally, runs the simulation model with the current QA distribution scheme.

```
calc_up(NEW,NEWITER):-
   asserta(calc(1)),
   output_cummd(NEW,NEWITER),
   output_tpfmqa,
   shell(model).
```

16. Rule--calc_orig (Original)

This rule follows the application of no pulse by the calc_more rule. It records the last simulation derived CUMMD and QA distribution (TPFMQA) into the SUMMARY.DAT file. This rule resets TYPE to 0 and returns directly to the calc_less rule for perturbations around the next life cycle point.

```
calc_orig(NEWITER,ITER,NEW,OLD,CHECK):-
   asserta(calc(0)),
   output_cummd(NEW,NEWITER),
   output_tpfmqa,
   calc_less(ITER,CHECK,OLD,0).
```

17. Rule--run_stop (Modified--old name: no_calc)

This rule is the only means of terminating the expert system simulation model. It is invoked by the reduce_pulse rule when pulse size is reduced below a user established minimum. The best CUMMD value is then assessed and recorded in the SUMMARY.DAT file.

```
run_stop(NEW):-
   call(cummdold(10,BEST)),
   open(S,'summary.dat',a),
   nl(S),nl(S),
   write(S,'The best CUMMD is: '),
   ifthenelse(BEST < NEW,write(S,BEST),write(S,NEW)),
   nl(S),
   close(S),
   halt.
```

18. Rule--output_tpfmqa (Original)

This rule outputs a current QA distribution scheme to two different files. First, the distribution is recorded in the SUMMARY.DAT file for post run process evaluation by the user. Secondly, the distribution is translated and stored in the PROJECT.DNX file for use by the system dynamics simulation model.

```
output_tpfmqa:-
   call(tpfmqa(1,QA1)),
   call(tpfmqa(2,QA2)),
   call(tpfmqa(3,QA3)),
   call(tpfmqa(4,QA4)),
   call(tpfmqa(5,QA5)),
   call(tpfmqa(6,QA6)),
   call(tpfmqa(7,QA7)),
   call(tpfmqa(8,QA8)),
   call(tpfmqa(9,QA9)),
   call(tpfmqa(10,QA10)),
```

```
create(D,'project.dnx'),
write(D,'T TPFMQA='),
write(D,QA1),write(D,' '),
write(D,QA2),write(D,' '),
write(D,QA3),write(D,' '),
write(D,QA4),write(D,' '),
write(D,QA5),write(D,' '),
write(D,QA6),write(D,' '),
write(D,QA7),write(D,' '),
write(D,QA8),write(D,' '),
write(D,QA9),write(D,' '),
write(D,QA10),nl(D),
close(D),
open(S,'summary.dat',a),
write(S,'TPFMQA='),
write(S,QA1),write(S,'/'),
write(S,QA2),write(S,'/'),
write(S,QA3),write(S,'/'),
write(S,QA4),write(S,'/'),
write(S,QA5),write(S,'/'),
write(S,QA6),write(S,'/'),
write(S,QA7),write(S,'/'),
write(S,QA8),write(S,'/'),
write(S,QA9),write(S,'/'),
write(S,QA10),nl(S),
close(S).
```

19. Rule--output_cummd (Original)

This rule output total project costs in cummulative man-days to the SUMMARY.DAT file. This CUMMD value follows

its related QA distribution scheme, in the file layout, and is number with the iteration it corresponds to.

```
output_cummd(NEW,ITER):-
  open(S,'summary.dat',a),
  write(S,ITER),write(S,'. '),
  write(S,'CUMMD='),write(S,NEW),nl(S),
  close(S).
```

20. Rule--output_break (Original)

This rule is invoked by the calc_less rule at the beginning of a new cycle. It records a new cycle header into the SUMMARY.DAT file.

```
output_break:-
  open(S,'summary.dat',a),
  nl(S),write(S,
  '**********  Start of a new cycle  **********'),
  nl(S),nl(S),
  close(S).
```

21. Rule--read_cummd (Original)

This rule translates and reads into the expert system module the CUMMD value currently stored in the PROJECT.OUT file. This file records the total project costs derived by the system dynamics simulation model.

```
read_cummd:-
   open(C,'project.out',r),
   read(C,CUMMD),
   abolish(cummd/1),
   asserta(CUMMD),
   close(C).
```

22. Rule--initial_run (Modified)

This rule is only invoked once to assist the pqa rule in initializing and starting the system. Accepting the ten base-line QA values it sets both the old base (b0) and the temporary base (TPFMQA) to these values. The initial QA distribution is then recorded in the SUMMARY.DAT file and the simulation is run to derive the initial CUMMD. This CUMMD is used to initialize the base CUMMD recording system (b0_cummd and b1_cummd) along with the cummdold recording system.

```
initial_run(QA1,QA2,QA3,QA4,QA5,QA6,QA7,QA8,QA9,QA10):-
/* establishes the initial temporary base */
   asserta(tpfmqa(1,QA1)),
   asserta(tpfmqa(2,QA2)),
   asserta(tpfmqa(3,QA3)),
   asserta(tpfmqa(4,QA4)),
   asserta(tpfmqa(5,QA5)),
   asserta(tpfmqa(6,QA6)),
   asserta(tpfmqa(7,QA7)),
   asserta(tpfmqa(8,QA8)),
   asserta(tpfmqa(9,QA9)),
   asserta(tpfmqa(10,QA10)),
```

```
/* establishes the initial base for pattern search */
   asserta(b0(1,QA1)),
   asserta(b0(2,QA2)),
   asserta(b0(3,QA3)),
   asserta(b0(4,QA4)),
   asserta(b0(5,QA5)),
   asserta(b0(6,QA6)),
   asserta(b0(7,QA7)),
   asserta(b0(8,QA8)),
   asserta(b0(9,QA9)),
   asserta(b0(10,QA10)),
/* runs the system dynamics model */
   output_tpfmqa,
   shell(model),
   read_cummd,
   call(cummd(INITIAL)),
/* records initial CUMMDs for perturbation comparisons */
   asserta(b0_cummd(INITIAL)),
   asserta(b1_cummd(INITIAL)),
   asserta(cummdold(1,0)),
   asserta(cummdold(2,0)),
   asserta(cummdold(3,0)),
   asserta(cummdold(4,0)),
   asserta(cummdold(5,0)),
   asserta(cummdold(6,0)),
   asserta(cummdold(7,0)),
   asserta(cummdold(8,0)),
   asserta(cummdold(9,0)),
   asserta(cummdold(10,INITIAL)),
   retract(cummd(INITIAL)).
```

C. SYSTEM OPERATIONS

To use the pattern search ESS model the following hardware and software is required:

- IBM compatible microcomputer.
- 80286 or better microprocessor.
- math coprocessor.
- 640k RAM.
- 20M hard drive.
- PCDOS or MSDOS 2.0 or better.
- Arity Prolog 5.0 Interpreter software.
- the system dynamics simulation model and associated editors and files.
- the expert system module (PATTERN.ARI).
- the interfacing files (SUMMARY.DAT, PROJECT.DNX, PROJECT.OUT, PROJECT.DRS, and MODEL.BAT).

The interpreter files, expert system program, simulator files, and interfacing files should all be stored under a separate directory on the computer's hard drive. Next, the user can modify the system dynamics model's internal variables (project.dyn) using the pd.com file, to reflect the predicted behavior of a proposed software project. Complete information on this process is contained in Reference 11. The PROJECT.DYN file currently contains the NASA DE-A project estimates and data. To conduct further experiments using this project requires no editing.

The expert system simulation model is initiated by typing "api" at the DOS prompt, as shown below:

```
C:\(directory name)> api
```

This loads the Arity Prolog Interpreter which displays the software package information and input prompt (?-).

```
Arity/Prolog Interpreter Version 5.0
Copyright (C) 1989 Arity Corporation
?-
```

At the prompt the name of the prolog program, without the extension, is typed enclosed in square brackets and followed by a period.

```
?- [pattern].
```

The interpreter is case sensitive. All commands should be typed as lower case letters, as capitals are treated as variables. A period must follow all entries to the system. This tells the interpreter, when accompanied by a carriage return, that the input is complete and to expect nothing else at this time.

At this point the interpreter compiles the program and reviews it for syntax errors. The system will respond with a second prompt. To start the expert system model the program's initialization rule name is typed at the prompt:

```
yes
?- pqa.
```

This brings up the user input queries, that have been coded into the rule, one at a time. Each input is to be followed by a period and a carriage return. Fractional numbers require a zero prior to the decimal point. The interpreter will not accept them in any other format. The complete input screen is displayed below with sample entries:

```
What is your desired pulse size factor? 0.05.
What is the minimum QA value? 0.02.
What is the maximum QA value? 0.5.
What is your minimum desired pulse size? 0.01.
Enter the initial QA distribution. Point 1  0.15.
                                   Point 2  0.15.
                                   Point 3  0.15.
                                   Point 4  0.15.
                                   Point 5  0.15.
                                   Point 6  0.15.
                                   Point 7  0.15.
                                   Point 8  0.15.
                                   Point 9  0.15.
                                   Point 10 0.15.
```

After the tenth base-line QA value has been entered, the system shifts into an automatic mode requiring no intervention from the user. Depending on the characteristics of the software project being simulated and the quality/speed of the hardware used, the entire process could take anywhere from 45 minutes to two days to derive an optimum. When the system has terminated (screen shows "Press any key to continue...."), the actual sequence of events the model used to reach an optimal QA distribution can be retrieved from the SUMMARY.DAT file. A copy of a NASA DE-A project results is displayed in Appendix B.

D. SYSTEM TEST

1. DE-A Project

The QA allocation performance of the pattern search ESS model was evaluated using the DE-A software project. The system was initialized with the exact parameters used in the prototype's performance evaluation. The only exception was a

minimum pulse size factor value of 1%, which was an addition to the expert system module.

The results of the experiment are compared with previous experiments below:

	QA costs (man-days)	Total Costs (man-days)
Actual DE-A	524	2,200
Manually derived	161.9	1,524.5
Prototype ESS	170.4	1,521.07
Pattern Search ESS	176.6	1,489.34

The results clearly show that the performance of the pattern search ESS is far superior to any of its predecessors. Figure 4-6 displays a composite of the QA distribution patterns of the four experiments listed above. As seen in the graph, the model places an extreme emphasis on quality assurance at the beginning of the project's life cycle. This technique causes the project to avoid: (1) the excessive cost of removing design errors during the testing phase; and (2) the tendency of early committed errors creating additional errors or growing in size and complexity as the life cycle progresses. Early detection and correction of errors allows the ESS model to reduce a considerable number of the latter life cycle points' values to within the minimum QA value (established by the user at 2%).

Another advantage to the pattern search ESS model is its level of sophistication. The prototype requires that a maximum number of cycles setting be entered by the user. This

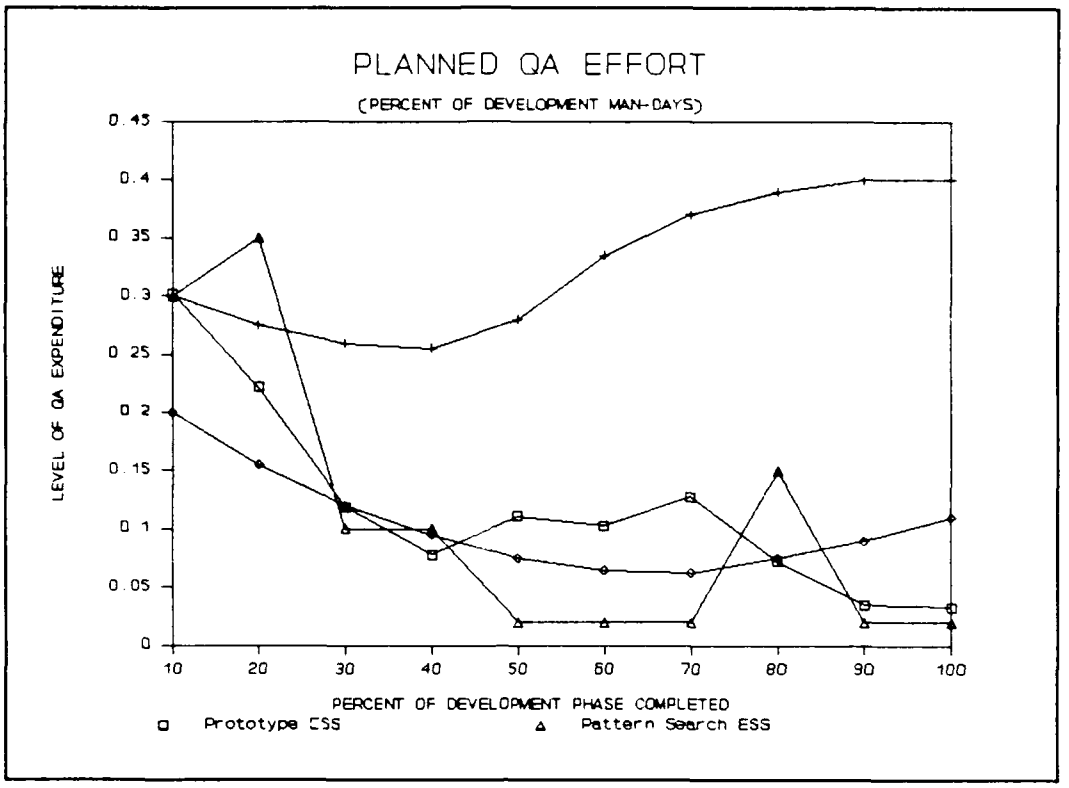


Figure 4-6 Composite of QA Distributions

parameter was used by the prototype to terminate the ESS model during all experiment runs. For this reason the prototype never successfully reached an "optimal" distribution pattern within a reasonable time limit (reasonable time limit established by the maximum cycle parameter).

The pattern search ESS model uses only one termination criteria, minimum pulse size factor (PSF). This ensures the model runs until the PSF is reduced to the level that it can no longer affect any substantial changes (e.g., below .01 PSF).

2. Performance Testing

The high/low experiments used in evaluating the performance of the prototype model were conducted on the pattern search ESS. The "high" base-line was established at 50% of total resources dedicated to QA, and the "low" at 3%. All initialization parameters were identical to those used on the prototype experiments: base line QA at .15, minimum QA level at .02, and pulse size factor at .05. Additionally, minimum pulse size was set at .01 and maximum QA level at .50.

The "high" run produced a solution after 12 cycles, and the "low" run after seven cycles. Both final QA distribution schemes are shown in Table 4-1 below. The total costs associated with these schemes were 1522.33 and 1524.65 total man-days, respectively.

TABLE 4-1

PATTERN SEARCH HIGH/LOW EXPERIMENT RESULTS

PATTERN SEARCH ESS HIGH/LOW COMPARISON										
Base	10%	20%	30%	40%	50%	60%	70%	80%	90%	100%
50%	.250	.225	.150	.050	.150	.150	.100	.087	.020	.020
3%	.205	.205	.155	.080	.155	.155	.130	.020	.020	.020

Figure 4-7 shows the graphical representation of the two distribution schemes. The similar patterns are a strong indication that the pattern search ESS is capable of deriving identical distribution patterns independent of the base-line level. This provides strong evidence that the pattern search algorithm is not vulnerable to a bi-modal (multiple local optimals) environment.

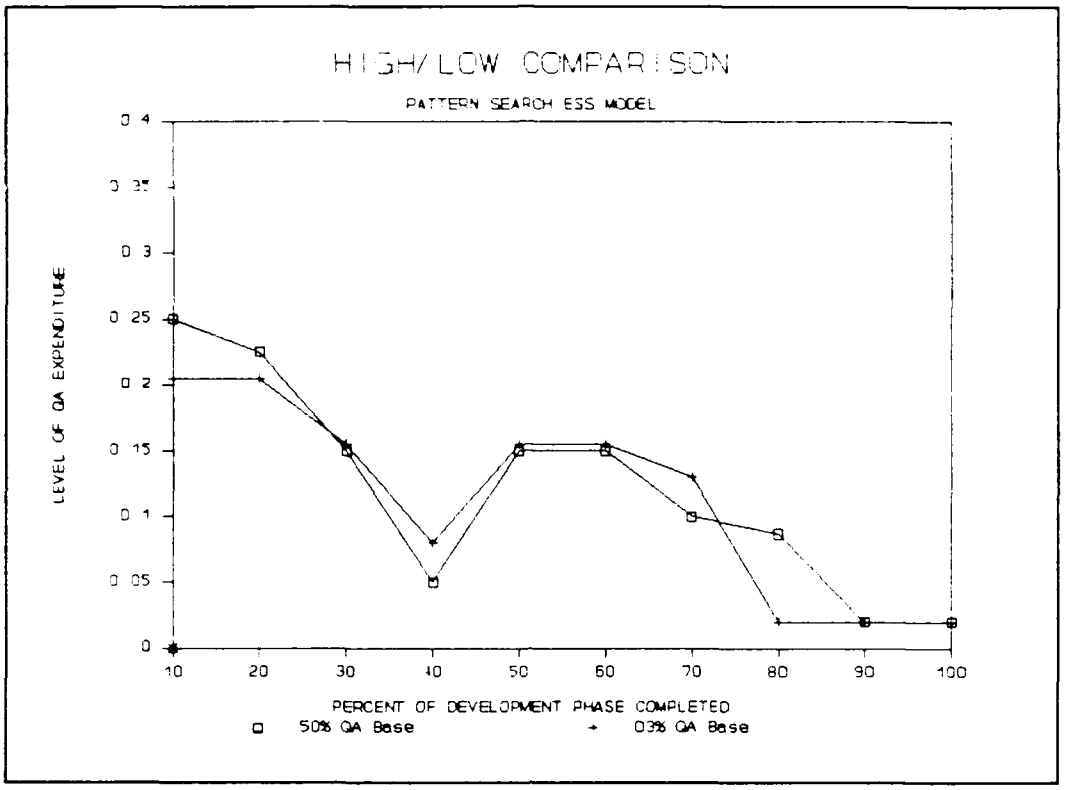


Figure 4-7 Pattern Search High/Low Comparison

V. SENSITIVITY ANALYSIS

A. INTRODUCTION

To further evaluate the performance of the pattern search ESS model, a set of six sensitivity experiments was developed. Using the DE-A project, each experiment focused on a single table function or group of related constants for altercation. The areas of focus were: (1) project performance estimates, (2) productivity of personnel, (3) nominal number of errors, (4) rework manpower required, (5) quality assurance manpower required, and (6) percentage of bad fixes.

Experiments two through six were further divided into two separate sub-experiments. Both sub-experiments addressed the same area of focus. However, the first (e.g., Experiment 2a) uniformly increased the focus areas' values while the second (e.g., Experiment 2b) did the exact opposite. These new values were compiled separately into the system dynamics portion of the ESS model and an optimal pattern search was conducted.

Resulting statistics from the 11 sub-experiments were recorded in experimental data sheets for further sensitivity analysis. These data sheets follow each experimental write-up and summarize the results in both tabular and graphic form. An ESS-derived "base case" solution (base QA), using the original DE-A project values, was established as the

performance standard for each experiment. Comparisons between the base case and experimental results will provide insights into the sensitivity of the pattern search ESS model. The data sheets contain the following displays:

- Project Statistics Tables--Two statistics tables are provided per sub-experiment. These tables provide data on estimated completion time (days), effort required (man-days), and total errors for a particular development environment. The first table uses an ESS generated QA distribution scheme (Test QA) to derive its results. The second table uses the actual project's QA allocation scheme (DE-A).

- QA Comparison Table--This table displays QA effort over ten life cycle points. Three distribution schemes are presented: (1) the actual DE-A project's QA distribution scheme (DE-A), (2) the ESS derived scheme using unaltered DE-A project values (Base QA), and (3) the ESS derived scheme with focus area values altered (Test QA).

- QA Comparison Charts--These line charts graphically display the QA comparison table's three distribution schemes. A separate chart is provided for each sub-experiment.

- Statistical Comparison Charts--These bar charts graphically display the data contained in the project statistics tables. This data is broken down into an effort/time comparison chart and an error comparison chart for each experiment. Both graphs display interest areas in groups of three along the X-axis as follows: (1) negatively altered focus area's value (e.g., experiment 2b), (2) base case value (Base QA), and (3) positively altered focus area's value (e.g., experiment 2a).

Experiment 1 consisted of a single test case and only required a single QA comparison chart. To reduce redundancy of information, the base case statistics table (Base QA) was provided only once in place of the experiment one's other comparison chart. All references to base case values (Base QA) are directed to this table.

B. RESULTS OF EXPERIMENTAL TESTS

1. Experiment 1: Accurate Estimates

This experiment examines the ESS model's optimal QA distribution when the project is accurately estimated. Four initialization constants within the system dynamics module were altered: (1) real job size in DSI (RJBDSI), (2) tasks underestimation fraction (UNDEST), (3) total man-days (TOTMD1), and (4) time to develop (TDEV1). The experimental values are shown in Table 5-1 below.

TABLE 5-1

ESTIMATED = ACTUAL

Experiment 1	
RJBDSI - 24400	UNDEST - 0
TOTMD1 - 2100	TDEV1 - 387

Actually estimating the project's size, cost, and schedule reduced the constraint pressures experienced by the development team. As expected, the system dynamics model (DE-A) produced final results that were closed to the above productivity estimates. The DE-A statistics table also shows a reduction in error generation, which is attributable to the experimentally-reduced schedule pressures. A lower error generation rate combined with actual project's high level of QA and rework ensured that very few errors escape to testing.

Comparing the ESS-produced statistics (Test QA) with the original project results show a deemphasis of QA importance by the model. QA was virtually eliminated (reduced to the minimum level of 2%) for over 80% of the project's life cycle. By eliminating all unnecessary and costly QA activities the model was able to produce a significant savings in project costs (CUMMD).

The experimental statistics (Test QA) show a substantial reduction to QA and rework levels compared to the base case results (Base QA). Although the error generation rate was lower in the experimental case, the reduced detection effort still caused many errors to escape. This increase in escaping errors required more effort in the area of project testing. Combining these factors resulted in an additional 130 man-days over the base case results.

The ESS model performed as expected in this experimental environment. By lowering the perceived pressures on the development team, fewer errors were generated and the optimal level of QA was reduced. The model emphasized QA at the beginning of the project's life cycle. This allows the most critical design errors to be detected and corrected early in the development process. Weighing all costs and benefits, the model chose to virtually eliminate QA in the latter portion of the life cycle. Most of the errors that occurred after the first two life cycle points were allowed to escape to testing. The model did not obtain a lower project cost

(CUMMD) than the base case. This is attributable to lower personnel productivity brought on by reduced performance pressures. An increase in required development effort and days needed for project completion added to the final total cost.

EXPERIMENT 1

This experiment examines the model's sensitivity when estimated job size in DSI, effort required in man-days, and time required in days, are equal to the actual project's performance.

PATTERN SEARCH ESS MODEL PROJECT STATISTICS (TEST QA):

COMPLETION TIME	377.00	DAYS
TOTAL EFFORT	1,619.22	MAN-DAYS
QA EFFORT	63.31	MAN-DAYS
DEVELOP EFFORT	1,058.00	MAN-DAYS
REWORK EFFORT	78.68	MAN-DAYS
TESTING EFFORT	376.38	MAN-DAYS
TRAINING EFFORT	42.82	MAN-DAYS
TOTAL ERRORS GENERATED	461.40	ERRORS GENERATED
TOTAL ERRORS DETECTED	85.30	ERRORS DETECTED
TOTAL ERRORS ESCAPED	376.08	ERRORS THAT ESCAPED

SYSTEM DYNAMICS MODEL PROJECT STATISTICS (DE-A):

COMPLETION TIME	384.00	DAYS
TOTAL EFFORT	2,050.93	MAN-DAYS
QA EFFORT	566.11	MAN-DAYS
DEVELOP EFFORT	954.41	MAN-DAYS
REWORK EFFORT	277.61	MAN-DAYS
TESTING EFFORT	199.87	MAN-DAYS
TRAINING EFFORT	52.94	MAN-DAYS
TOTAL ERRORS GENERATED	463.21	ERRORS GENERATED
TOTAL ERRORS DETECTED	336.66	ERRORS DETECTED
TOTAL ERRORS ESCAPED	126.20	ERRORS THAT ESCAPED

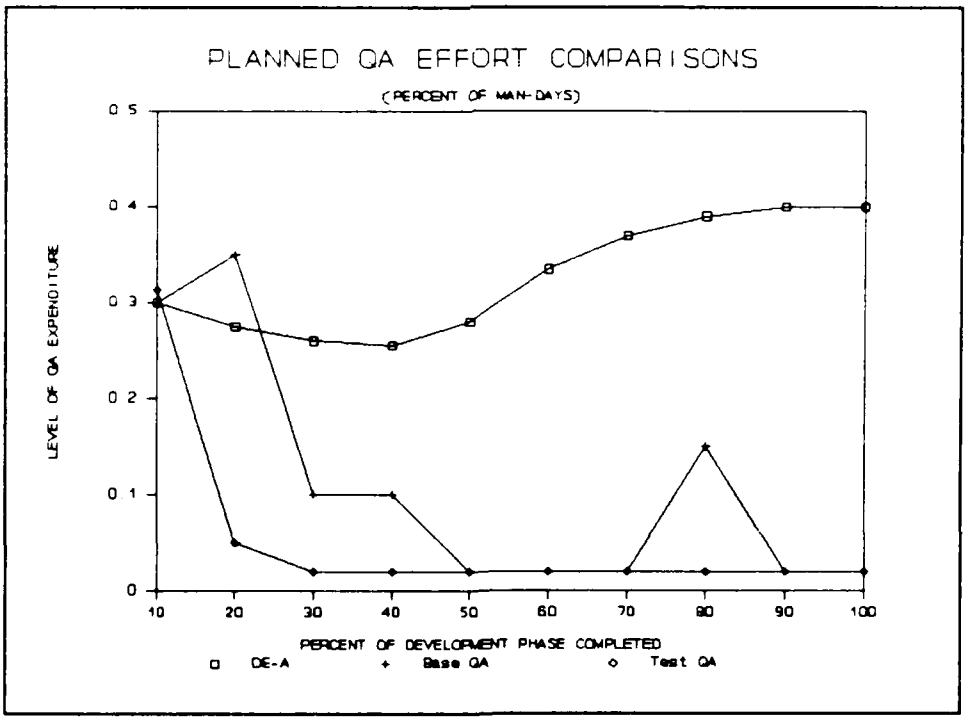
EXPERIMENT 1: QA DISTRIBUTION COMPARISON

	10	20	30	40	50	60	70	80	90	100
DE-A	.325	.290	.275	.255	.250	.275	.325	.375	.400	.400
Base QA	.300	.350	.100	.100	.020	.020	.020	.150	.020	.020
Test QA	.313	.050	.020	.020	.020	.020	.020	.020	.020	.020

EXPERIMENT 1

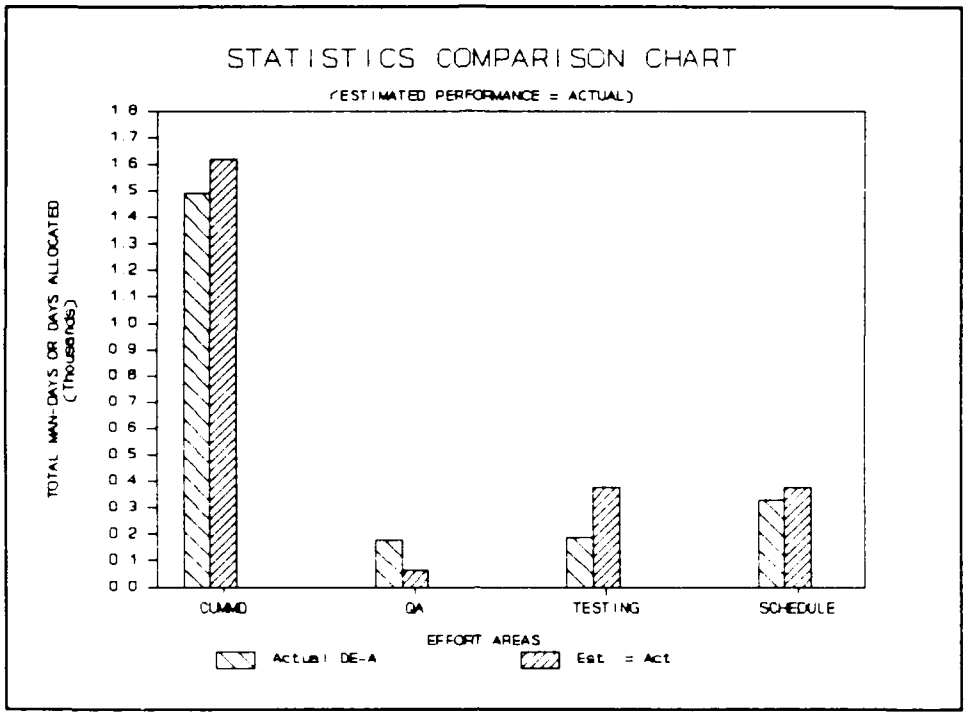
PATTERN SEARCH ESS MODEL PROJECT STATISTICS (BASE QA):

COMPLETION TIME	331.00	DAYS
TOTAL EFFORT	1,489.34	MAN-DAYS
QA EFFORT	176.60	MAN-DAYS
DEVELOP EFFORT	907.78	MAN-DAYS
REWORK EFFORT	153.66	MAN-DAYS
TESTING EFFORT	188.67	MAN-DAYS
TRAINING EFFORT	62.63	MAN-DAYS
TOTAL ERRORS GENERATED	490.72	ERRORS GENERATED
TOTAL ERRORS DETECTED	183.07	ERRORS DETECTED
TOTAL ERRORS ESCAPED	306.85	ERRORS THAT ESCAPED

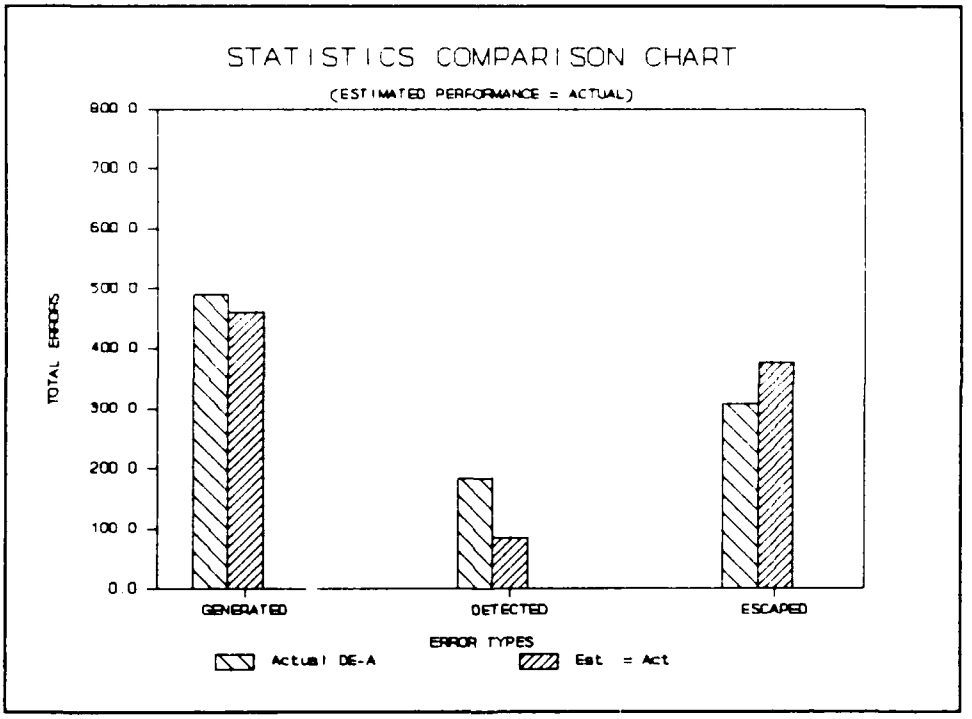


TEST 1: Estimated Equals Actual Project's Productivity

EXPERIMENT 1



TEST 1: Effort Allocation Comparison



TEST 1: Error Handling Comparison

2. Experiment 2: Productivity

This experiment examines the ESS model's sensitivity to changes in the productivity of new and experienced personnel. This analysis was divided into two sub-experiments. Experiment 2a increased productivity by 25%, and 2b decreased it by 25%. Two software development constants in the system dynamics model were altered: (1) nominal potential productivity of experienced employees (NPWPEX), and (2) nominal potential productivity new employees (NPWPNE). The experimental values are shown in table 5-2 below.

TABLE 5-2

PRODUCTIVITY

Experiment 2a	Experiment 2b
NPWPEX = 1.25	NPWPEX = 0.75
NPWPNE = 0.625	NPWPNE = 0.375

An increase in productivity (2a) significantly lowered the development effort shown in the actual project's statistics (DE-A). This lower development effort does not result in any significant increases to the error generation rate. Since the increase in productivity makes all effort more efficient, QA and rework can be increased while taking advantage of the cost effectiveness this environment offers. Diverting effort to QA and rework increases the number of

errors detected without adding greatly to final project cost (CUMMD). Both the EES model (Test QA) and the actual project (DE-A) statistics show very few errors escaping to testing.

The ESS model took full advantage of this highly productive environment. Since the relative cost of QA was low, the model assigns approximately 25% of total effort to quality assurance over the entire life cycle. This distribution scheme ensured excellent error detection, minimal error escapes, and produced a total cost lower than that of the actual DE-A project.

A decrease in personnel productivity (2b) has the reverse effect on the experiment. As shown in both the ESS (Test QA) and actual (DE-A) statistical tables, the increases experienced in all effort areas impact heavily on project cost. However, once again productivity does not impact greatly on error generation rate. The system dynamics module's results (DE-A) show the significant cost increases QA, rework, and training cause in a low productive environment. The static QA distribution scheme has almost doubled final project cost.

The ESS model effectively cuts the total project costs in half (test QA vs. DE-A). Taking into account the limited effects that QA has in this environment, and the high associated costs, it virtually does away with the effort for over 90% of the project. Cost savings in QA and rework support the transfer of error correcting to the testing phase.

EXPERIMENT 2a

This experiment examines the model's sensitivity when nonimal productivity of both experienced and new personnel is increased by 25 percent (+25%).

PATTERN SEARCH ESS MODEL PROJECT STATISTICS (TEST QA):

COMPLETION TIME	330.00	DAYS
TOTAL EFFORT	1,518.09	MAN-DAYS
QA EFFORT	321.98	MAN-DAYS
DEVELOP EFFORT	702.43	MAN-DAYS
REWORK EFFORT	254.17	MAN-DAYS
TESTING EFFORT	176.29	MAN-DAYS
TRAINING EFFORT	63.22	MAN-DAYS
TOTAL ERRORS GENERATED	494.73	ERRORS GENERATED
TOTAL ERRORS DETECTED	327.24	ERRORS DETECTED
TOTAL ERRORS ESCAPED	166.94	ERRORS THAT ESCAPED

SYSTEM DYNAMICS MODEL PROJECT STATISTICS (DE-A):

COMPLETION TIME	341.00	DAYS
TOTAL EFFORT	1,590.81	MAN-DAYS
QA EFFORT	373.59	MAN-DAYS
DEVELOP EFFORT	695.47	MAN-DAYS
REWORK EFFORT	265.38	MAN-DAYS
TESTING EFFORT	192.20	MAN-DAYS
TRAINING EFFORT	64.17	MAN-DAYS
TOTAL ERRORS GENERATED	493.70	ERRORS GENERATED
TOTAL ERRORS DETECTED	355.03	ERRORS DETECTED
TOTAL ERRORS ESCAPED	138.40	ERRORS THAT ESCAPED

EXPERIMENT 2a: QA DISTRIBUTION COMPARISON

	10	20	30	40	50	60	70	80	90	100
DE-A	.325	.290	.275	.255	.250	.275	.325	.375	.400	.400
Base QA	.300	.350	.100	.100	.020	.020	.020	.150	.020	.020
Test QA	.300	.300	.300	.200	.250	.250	.300	.250	.263	.050

EXPERIMENT 2b

This experiment examines the model's sensitivity when nominal productivity of both experienced and new personnel is decreased by 25 percent (-25%).

PATTERN SEARCH ESS MODEL PROJECT STATISTICS (TEST QA):

COMPLETION TIME	356.00	DAYS
TOTAL EFFORT	1,748.64	MAN-DAYS
QA EFFORT	55.64	MAN-DAYS
DEVELOP EFFORT	1,205.10	MAN-DAYS
REWORK EFFORT	68.88	MAN-DAYS
TESTING EFFORT	340.65	MAN-DAYS
TRAINING EFFORT	78.32	MAN-DAYS
TOTAL ERRORS GENERATED	492.99	ERRORS GENERATED
TOTAL ERRORS DETECTED	82.88	ERRORS DETECTED
TOTAL ERRORS ESCAPED	409.91	ERRORS THAT ESCAPED

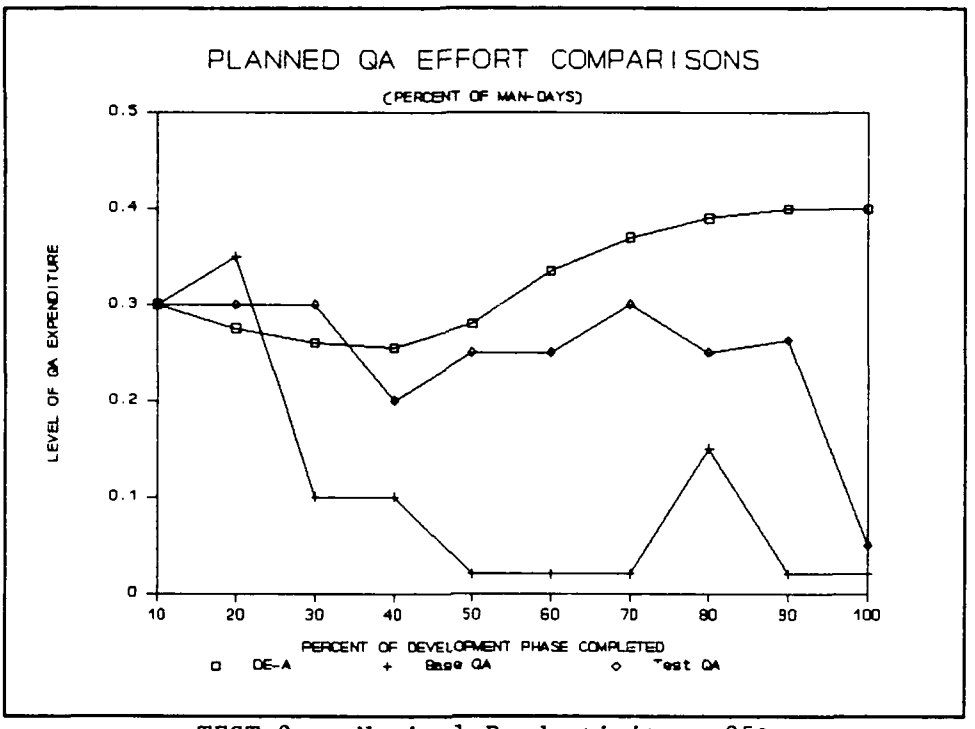
SYSTEM DYNAMICS MODEL PROJECT STATISTICS (DE-A):

COMPLETION TIME	435.00	DAYS
TOTAL EFFORT	3,341.13	MAN-DAYS
QA EFFORT	768.45	MAN-DAYS
DEVELOP EFFORT	1,440.60	MAN-DAYS
REWORK EFFORT	311.10	MAN-DAYS
TESTING EFFORT	537.05	MAN-DAYS
TRAINING EFFORT	283.97	MAN-DAYS
TOTAL ERRORS GENERATED	514.39	ERRORS GENERATED
TOTAL ERRORS DETECTED	384.68	ERRORS DETECTED
TOTAL ERRORS ESCAPED	128.62	ERRORS THAT ESCAPED

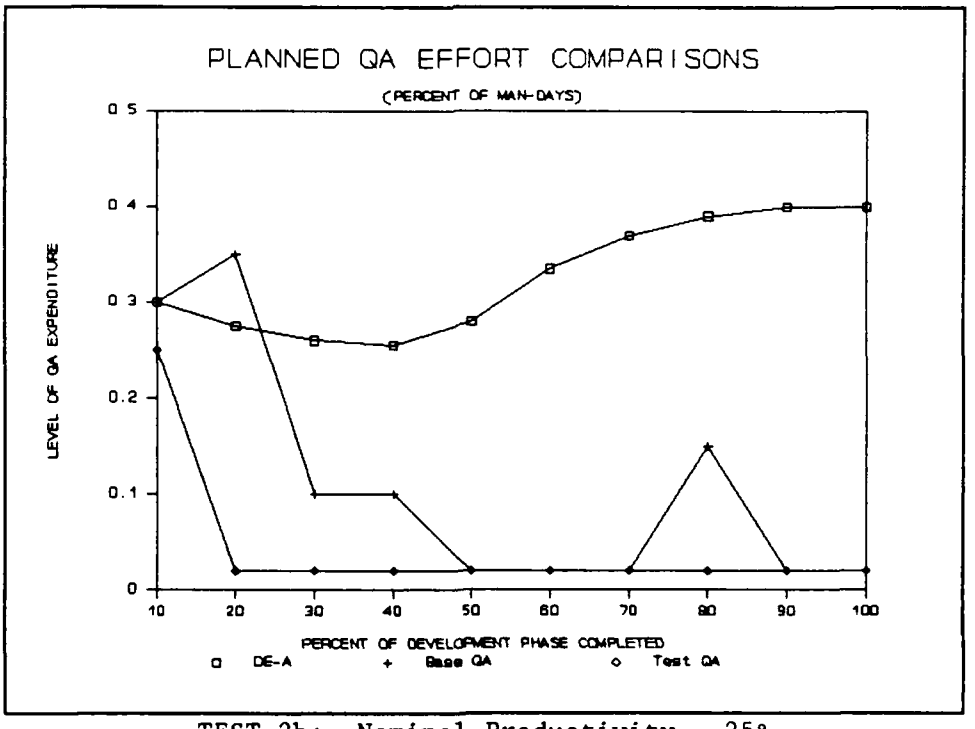
EXPERIMENT 2b: QA DISTRIBUTION COMPARISON

	10	20	30	40	50	60	70	80	90	100
DE-A	.325	.290	.275	.255	.250	.275	.325	.375	.400	.400
Base QA	.300	.350	.100	.100	.020	.020	.020	.150	.020	.020
Test QA	.250	.020	.020	.020	.020	.020	.020	.020	.020	.020

EXPERIMENT 2

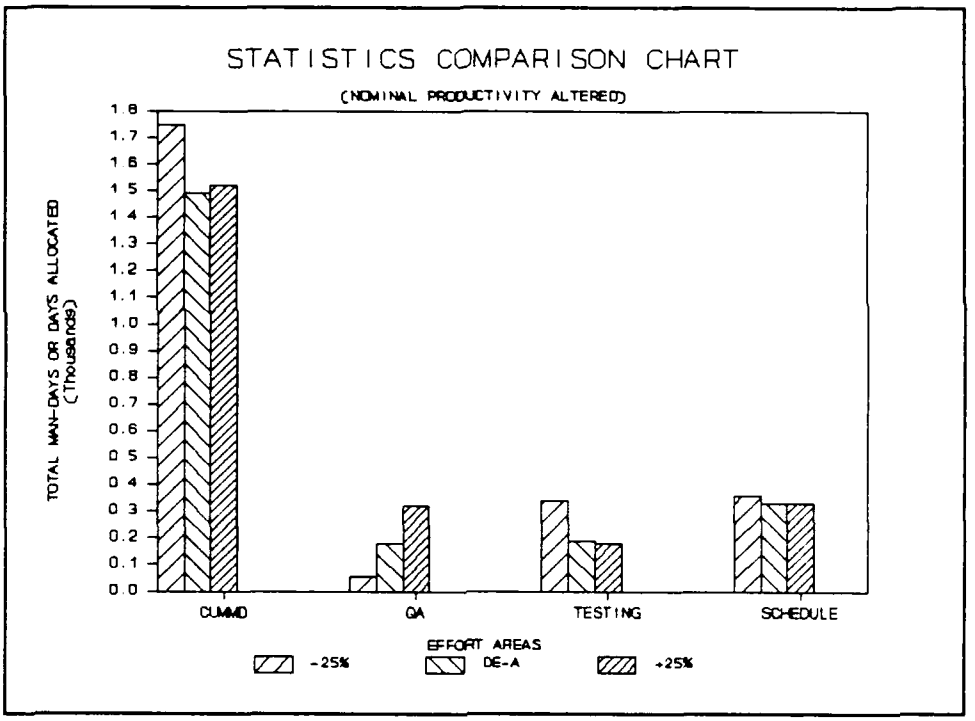


TEST 2a: Nominal Productivity + 25%

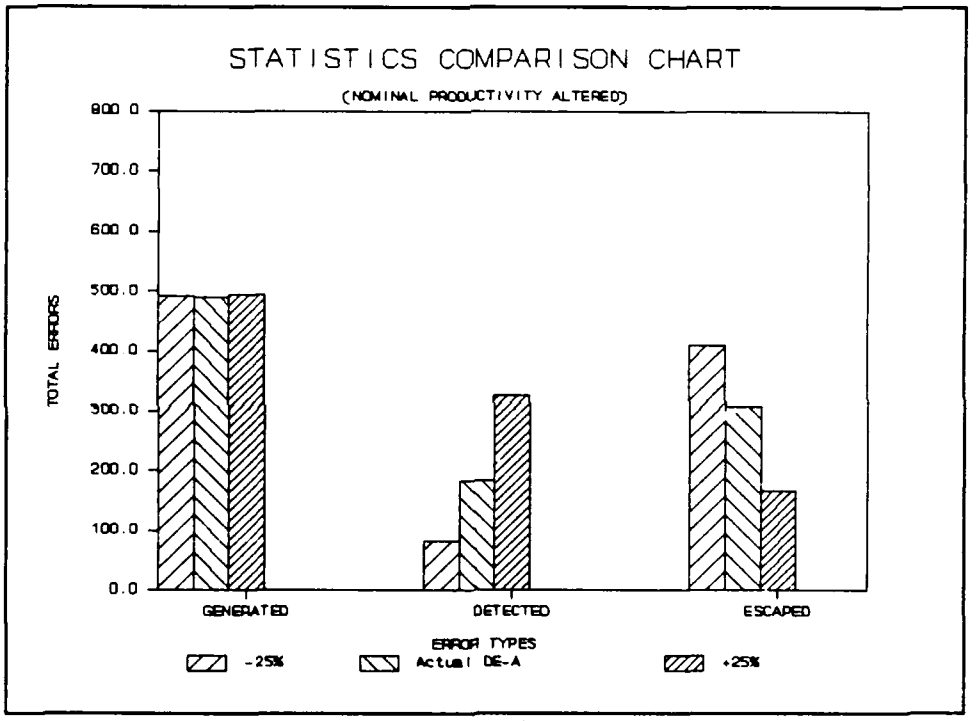


TEST 2b: Nominal Productivity - 25%

EXPERIMENT 2



TEST 2: Effort Allocation Comparison



TEST 2: Error Handling Comparison

3. Experiment 3: Error Rate

This experiment examines the ESS model's sensitivity to changes in the number of errors committed during development. This analysis was divided into two sub-experiments. Experiment 3a increased the number of errors by 50%, and 3b decreased it by 50%. A single QA and rework table function in the system dynamics model was altered: nominal errors committed per KDSI (TNERPK). The experimental values are shown in Table 5-3 below.

TABLE 5-3

ERRORS COMMITTED

	TNERPK					
Exp. 3a	36	34.35	31.125	22.875	19.65	18
Exp. 3b	12	11.45	10.375	7.625	6.55	6

An increase in the number of errors committed (3a) obviously means more effort must be allocated to detection and correction activities. As seen in the actual project's statistics (DE-A), the high allocation of QA, rework and training greatly increases total project costs.

As expected, this environment produced a significantly greater error generation rate. However, the ESS model (Test QA) ignored this fact and concentrated on reducing project costs by drastically cutting QA and rework effort. This

action also reduced the necessary training and associated costs by half (Test QA vs. DE-A). Using a form of cost-benefit analysis, the model emphasized QA during the early design phases and virtually eliminated QA from the remainder of the life cycle. This distribution scheme allowed over 83% of the errors to escape to testing.

Since the testing phase is the termination point for error generation, the model waited until development had finished to handle most errors. Surprisingly, this resulted in the need for very little added effort to handle the additional errors. By virtue of this fact, the ESS model cut over 1,000 man-days from total project costs.

A decrease in the number of errors (3b) should also decrease the man-days required in all effort areas. In this environment both the ESS model (Test QA) and the system dynamics module (DE-A) statistics, show significant reductions in effort areas, errors generated, and total costs. The only disparities between the two are in QA and rework levels.

The ESS model (Test QA) actually produces a lower project cost (CUMMD) than in the base case results (Base QA). However, the QA distribution pattern is extremely erratic. The values jump from 50% effort down to 2% over the course of the first three life cycle points. The experimentally derived total cost (Test QA) is very close to the to CUMMDs produced by the base case (Base QA) and the actual project (DE-A). However, all three patterns differ quite drastically. One

possibility is that in this environment large increases or decreases in QA effort only cause small changes in total project cost. If this is true, the pattern search model may have serious sensitivity problems and would account for the apparent "wandering" pattern the model produced. Experiment 2b failed to produce the expected results.

EXPERIMENT 3a

This experiment examines the model's sensitivity when nominal number of errors committed per KDSI is increased by 50 percent (+50%).

PATTERN SEARCH ESS MODEL PROJECT STATISTICS (TEST QA):

COMPLETION TIME	348.00	DAYS
TOTAL EFFORT	1,628.32	MAN-DAYS
QA EFFORT	119.14	MAN-DAYS
DEVELOP EFFORT	946.06	MAN-DAYS
REWORK EFFORT	139.38	MAN-DAYS
TESTING EFFORT	353.84	MAN-DAYS
TRAINING EFFORT	69.91	MAN-DAYS
TOTAL ERRORS GENERATED	732.35	ERRORS GENERATED
TOTAL ERRORS DETECTED	150.63	ERRORS DETECTED
TOTAL ERRORS ESCAPED	581.68	ERRORS THAT ESCAPED

SYSTEM DYNAMICS MODEL PROJECT STATISTICS (DE-A):

COMPLETION TIME	415.00	DAYS
TOTAL EFFORT	2,671.32	MAN-DAYS
QA EFFORT	673.57	MAN-DAYS
DEVELOP EFFORT	1,015.20	MAN-DAYS
REWORK EFFORT	462.07	MAN-DAYS
TESTING EFFORT	336.79	MAN-DAYS
TRAINING EFFORT	183.72	MAN-DAYS
TOTAL ERRORS GENERATED	763.25	ERRORS GENERATED
TOTAL ERRORS DETECTED	596.67	ERRORS DETECTED
TOTAL ERRORS ESCAPED	165.32	ERRORS THAT ESCAPED

EXPERIMENT 3a: QA DISTRIBUTION COMPARISON

	10	20	30	40	50	60	70	80	90	100
DE-A	.325	.290	.275	.255	.250	.275	.325	.375	.400	.400
Base QA	.300	.350	.100	.100	.020	.020	.020	.150	.020	.020
Test QA	.338	.225	.020	.020	.020	.020	.020	.020	.020	.020

EXPERIMENT 3b

This experiment examines the model's sensitivity when nominal number of errors committed per KDSI is decreased by 50 percent (-50%).

PATTERN SEARCH ESS MODEL PROJECT STATISTICS (TEST QA):

COMPLETION TIME	330.00	DAYS
TOTAL EFFORT	1,437.29	MAN-DAYS
QA EFFORT	297.21	MAN-DAYS
DEVELOP EFFORT	860.89	MAN-DAYS
REWORK EFFORT	76.52	MAN-DAYS
TESTING EFFORT	142.38	MAN-DAYS
TRAINING EFFORT	60.28	MAN-DAYS
TOTAL ERRORS GENERATED	245.34	ERRORS GENERATED
TOTAL ERRORS DETECTED	91.60	ERRORS DETECTED
TOTAL ERRORS ESCAPED	152.97	ERRORS THAT ESCAPED

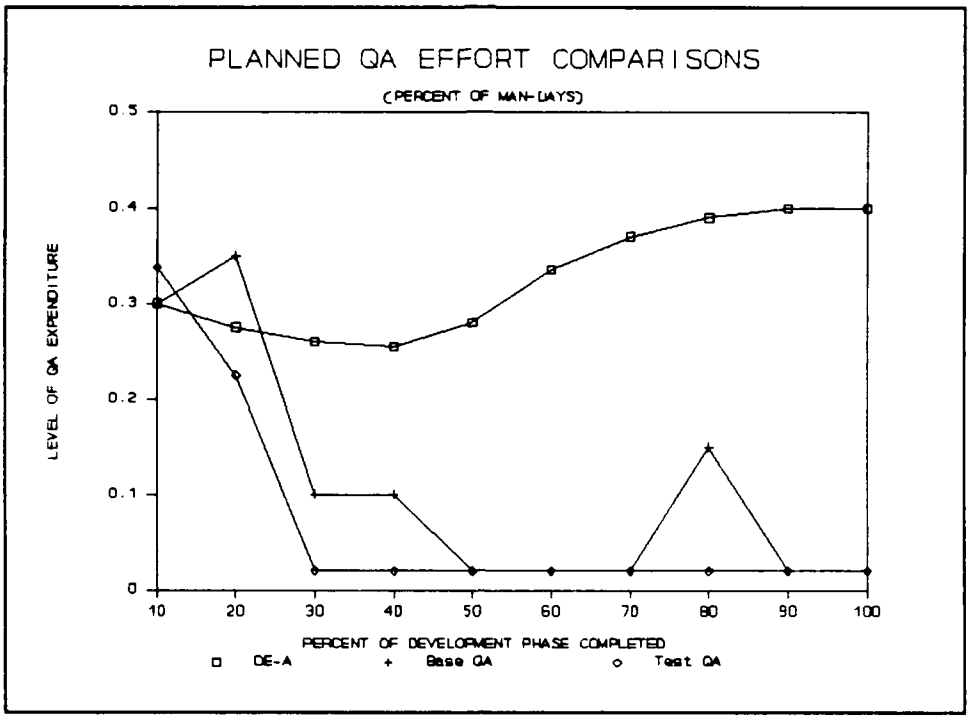
SYSTEM DYNAMICS MODEL PROJECT STATISTICS (DE-A):

COMPLETION TIME	338.00	DAYS
TOTAL EFFORT	1,583.72	MAN-DAYS
QA EFFORT	388.36	MAN-DAYS
DEVELOP EFFORT	877.84	MAN-DAYS
REWORK EFFORT	104.84	MAN-DAYS
TESTING EFFORT	147.69	MAN-DAYS
TRAINING EFFORT	64.98	MAN-DAYS
TOTAL ERRORS GENERATED	245.93	ERRORS GENERATED
TOTAL ERRORS DETECTED	137.87	ERRORS DETECTED
TOTAL ERRORS ESCAPED	107.63	ERRORS THAT ESCAPED

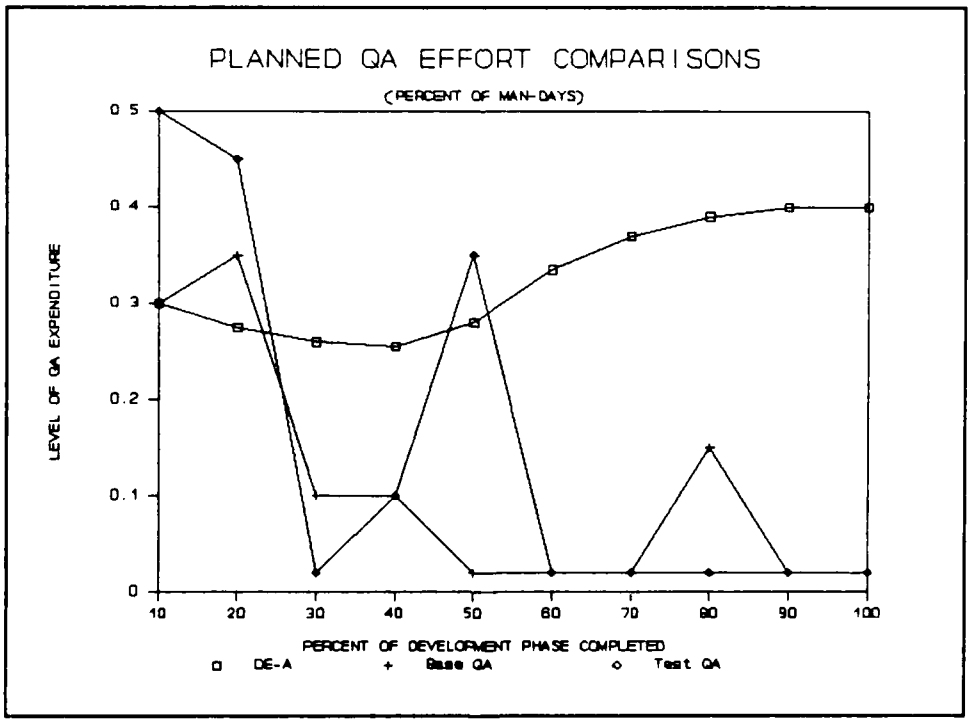
EXPERIMENT 3b: QA DISTRIBUTION COMPARISON

	10	20	30	40	50	60	70	80	90	100
DE-A	.325	.290	.275	.255	.250	.275	.325	.375	.400	.400
Base QA	.300	.350	.100	.100	.020	.020	.020	.150	.020	.020
Test QA	.500	.450	.020	.100	.350	.020	.020	.020	.020	.020

EXPERIMENT 3

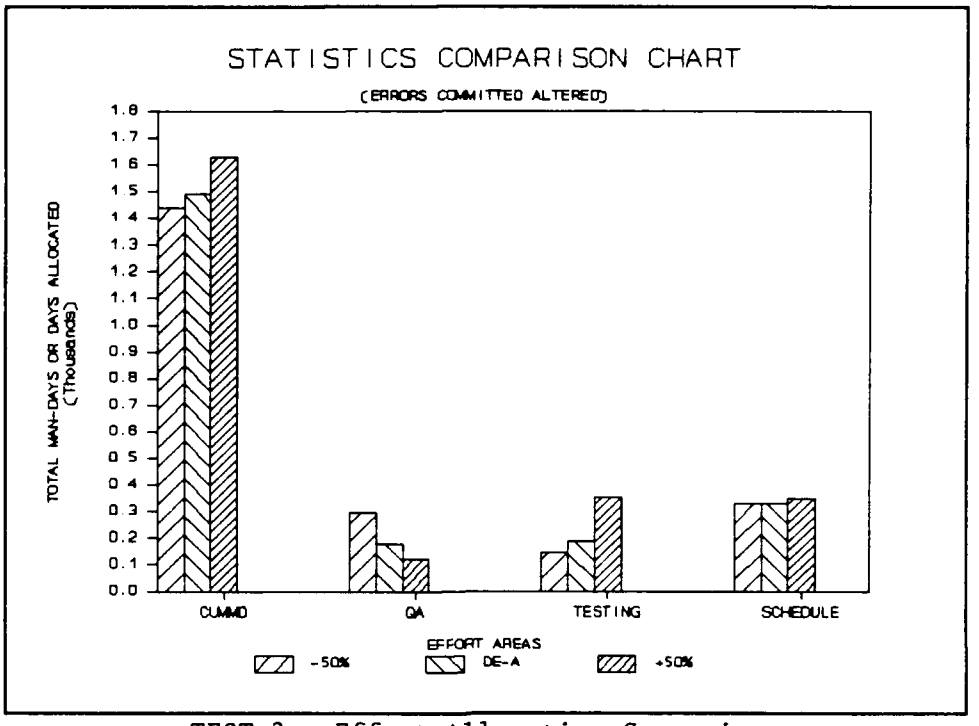


TEST 3a: Errors per KDSI + 50%

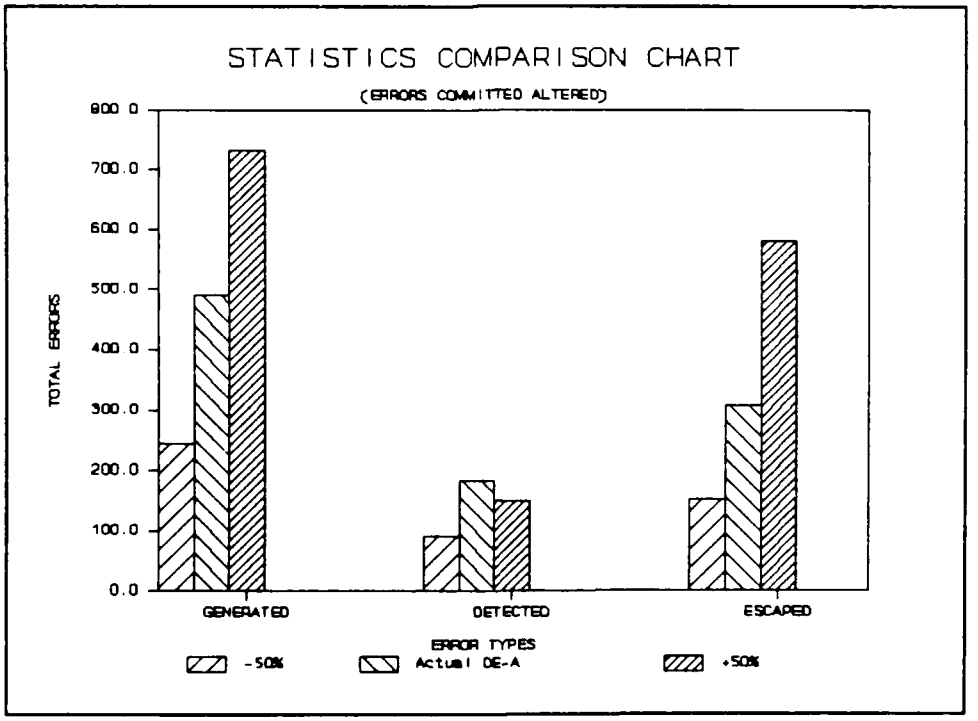


TEST 3b: Errors per KDSI - 50%

EXPERIMENT 3



TEST 3: Effort Allocation Comparison



TEST 3: Error Handling Comparison

4. Experiment 4: Rework Manpower

This experiment examines the ESS model's sensitivity to changes in rework manpower levels. This analysis was divided into two sub-experiments. Experiment 4a increased rework manpower by 50%, and 4b decreased it by 50%. A single QA and rework table function in the system dynamics model was altered: nominal rework manpower needed per error (TNRWME). The experimental values are shown in Table 5-4 below.

TABLE 5-4

REWORK MANPOWER

	TNRWME					
Exp. 4a	0.9	0.8625	0.75	0.6	0.4875	0.45
Exp. 4b	0.3	0.2875	0.25	0.2	0.1625	0.15

An increase in the rework manpower required to resolve each error (4a) means any correction activity prior to testing will be extremely costly. The actual project results (DE-A), placed significant emphasis on QA, rework, and training. As expected, the cost of rework and its related activities produced a high final project cost.

The ESS model (Test QA) effectively reduced costs to within the base case results (Base QA). This was accomplished by focusing QA and rework effort during the first third of the life cycle, and virtually eliminating this effort over the

latter two-thirds. Since the rework capabilities of the development team were reduced, leaving non-design errors in the system until testing was the most cost effective solution.

A decrease in the rework manpower required to resolve an error (4b) creates the exact opposite environment from above. Reducing the cost of rework means more errors can be handled prior to testing at a fraction of the base case cost (Base QA).

Analyzing the ESS model results (Test QA) we see cost savings were obtained by reducing QA and rework below the levels contained in the base case (Base QA). This seems to be counter intuitive. With rework being less costly it is expected that the ESS model would attempt to capitalize on resolving more errors prior to testing. However, this was not the case. Rework has been reduced to almost half of the base case value. The resulting distribution scheme is also very erratic (similar to experiment 3b). Large disparities between consecutive life cycle points indicates the ESS model is "hunting" for an optimal solution. Once again we have an environment where increases and decreases in QA values have limited impact on overall project cost. Experiment 4b failed to produce the expected results.

EXPERIMENT 4a

This experiment examines the model's sensitivity when nominal rework manpower required per error is increased by 50 percent (+50%).

PATTERN SEARCH ESS MODEL PROJECT STATISTICS (TEST QA):

COMPLETION TIME	333.00	DAYS
TOTAL EFFORT	1,490.38	MAN-DAYS
QA EFFORT	134.85	MAN-DAYS
DEVELOP EFFORT	913.09	MAN-DAYS
REWORK EFFORT	171.52	MAN-DAYS
TESTING EFFORT	208.40	MAN-DAYS
TRAINING EFFORT	62.51	MAN-DAYS
TOTAL ERRORS GENERATED	489.14	ERRORS GENERATED
TOTAL ERRORS DETECTED	126.74	ERRORS DETECTED
TOTAL ERRORS ESCAPED	361.86	ERRORS THAT ESCAPED

SYSTEM DYNAMICS MODEL PROJECT STATISTICS (DE-A):

COMPLETION TIME	413.00	DAYS
TOTAL EFFORT	2,597.02	MAN-DAYS
QA EFFORT	664.51	MAN-DAYS
DEVELOP EFFORT	1,004.90	MAN-DAYS
REWORK EFFORT	431.34	MAN-DAYS
TESTING EFFORT	325.82	MAN-DAYS
TRAINING EFFORT	170.48	MAN-DAYS
TOTAL ERRORS GENERATED	506.26	ERRORS GENERATED
TOTAL ERRORS DETECTED	373.90	ERRORS DETECTED
TOTAL ERRORS ESCAPED	131.24	ERRORS THAT ESCAPED

EXPERIMENT 4a: QA DISTRIBUTION COMPARISON

	10	20	30	40	50	60	70	80	90	100
DE-A	.325	.290	.275	.255	.250	.275	.325	.375	.400	.400
Base QA	.300	.350	.100	.100	.020	.020	.020	.150	.020	.020
Test QA	.450	.150	.100	.020	.020	.020	.020	.020	.020	.020

EXPERIMENT 4b

This experiment examines the model's sensitivity when nominal rework manpower required per error is decreased by 50 percent (-50%).

PATTERN SEARCH ESS MODEL PROJECT STATISTICS (TEST QA):

COMPLETION TIME	330.00	DAYS
TOTAL EFFORT	1,501.03	MAN-DAYS
QA EFFORT	254.03	MAN-DAYS
DEVELOP EFFORT	902.37	MAN-DAYS
REWORK EFFORT	96.07	MAN-DAYS
TESTING EFFORT	184.44	MAN-DAYS
TRAINING EFFORT	63.46	MAN-DAYS
TOTAL ERRORS GENERATED	491.69	ERRORS GENERATED
TOTAL ERRORS DETECTED	229.94	ERRORS DETECTED
TOTAL ERRORS ESCAPED	260.84	ERRORS THAT ESCAPED

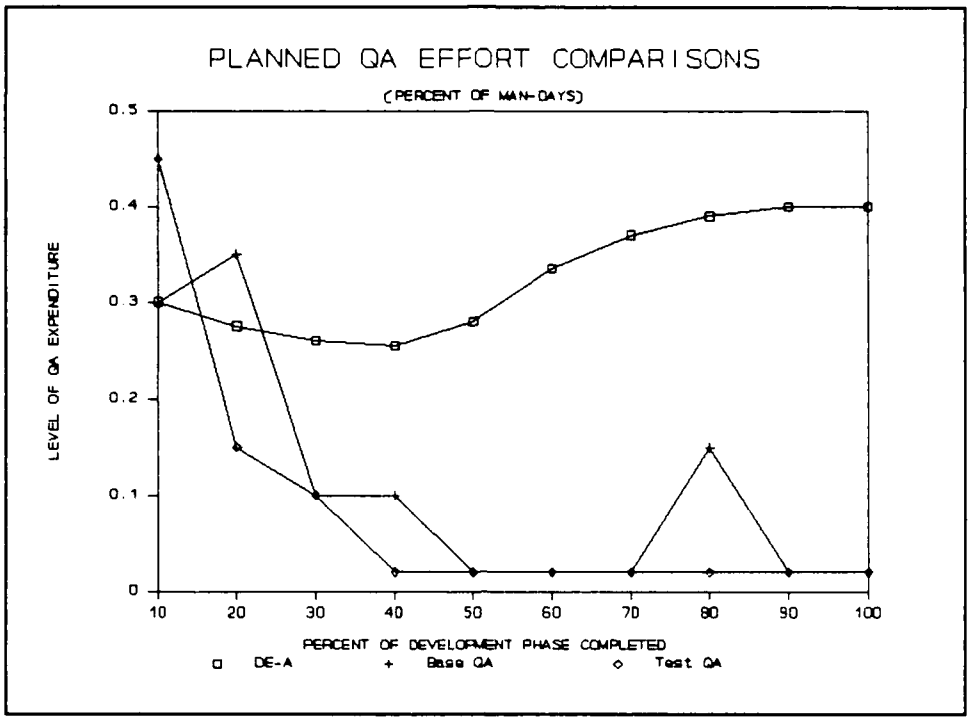
SYSTEM DYNAMICS MODEL PROJECT STATISTICS (DE-A):

COMPLETION TIME	353.00	DAYS
TOTAL EFFORT	1,689.44	MAN-DAYS
QA EFFORT	398.35	MAN-DAYS
DEVELOP EFFORT	874.97	MAN-DAYS
REWORK EFFORT	134.04	MAN-DAYS
TESTING EFFORT	212.82	MAN-DAYS
TRAINING EFFORT	69.25	MAN-DAYS
TOTAL ERRORS GENERATED	492.79	ERRORS GENERATED
TOTAL ERRORS DETECTED	356.79	ERRORS DETECTED
TOTAL ERRORS ESCAPED	135.58	ERRORS THAT ESCAPED

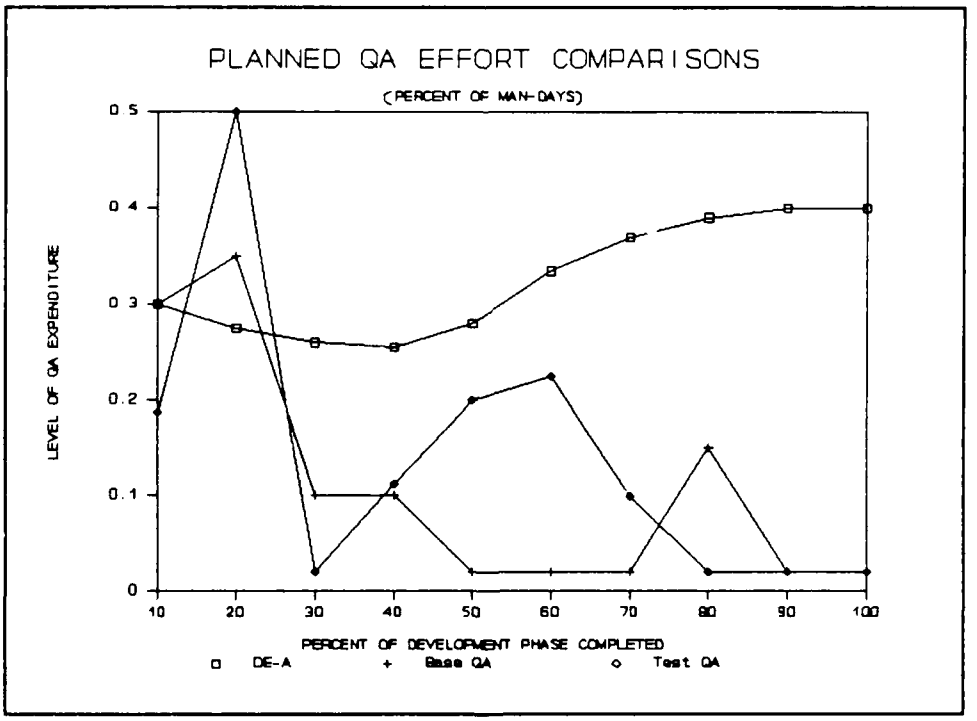
EXPERIMENT 4b: QA DISTRIBUTION COMPARISON

	10	20	30	40	50	60	70	80	90	100
DE-A	.325	.290	.275	.255	.250	.275	.325	.375	.400	.400
Base QA	.300	.350	.100	.100	.020	.020	.020	.150	.020	.020
Test QA	.187	.500	.020	.112	.200	.225	.099	.020	.020	.020

EXPERIMENT 4

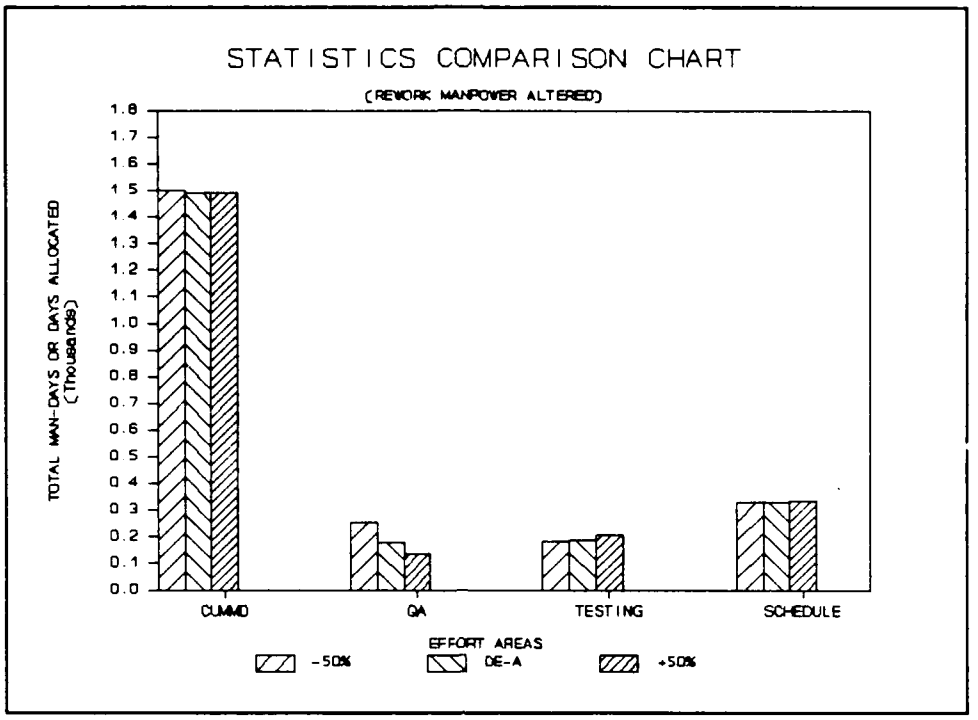


TEST 4a: Rework Manpower + 50%

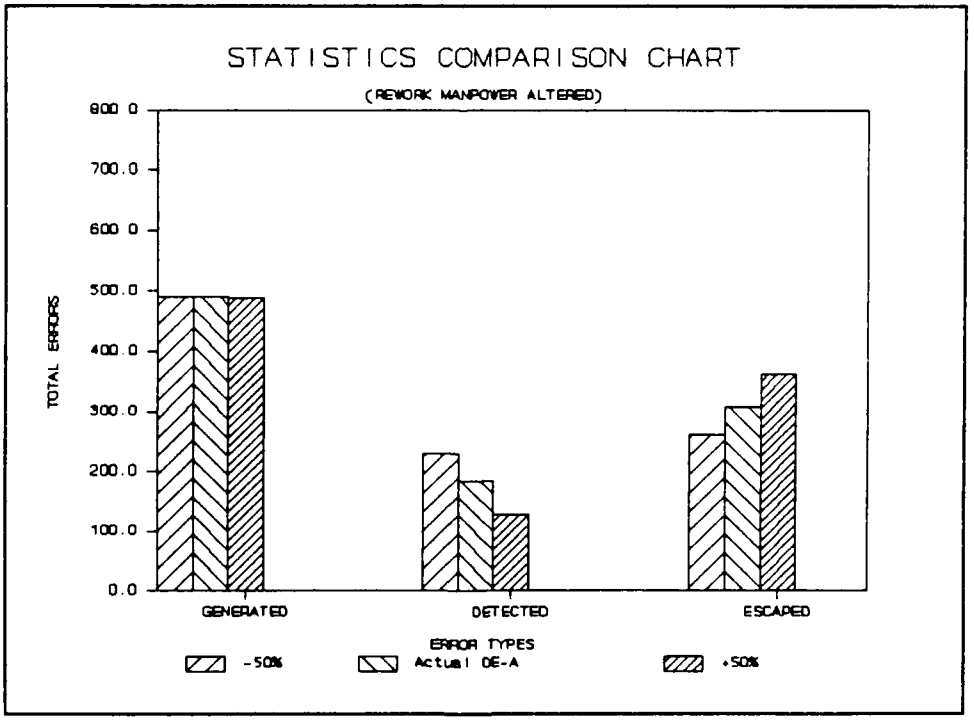


TEST 4b: Rework Manpower - 50%

EXPERIMENT 4



TEST 4: Effort Allocation Comparison



TEST 4: Error Handling Comparison

5. Experiment 5: QA Manpower

This experiment examines the ESS model's sensitivity to changes in quality assurance manpower levels. This analysis was divided into two sub-experiments. Experiment 5a increased QA manpower by 50%, and 5b decreased it by 50%. A single QA and rework table function in the system dynamics model was altered: nominal QA manpower needed to detect average errors (TNQAPE). The experimental values are shown in Table 5-5 below.

TABLE 5-5

QA MANPOWER

Experiment 5 a & b (TNQAPE)										
.6	.6	.585	.5625	.525	.45	.375	.3375	.315	.3	.3
.2	.2	.195	.1875	.175	.15	.125	.1125	.105	.1	.1

An increase in the QA manpower required to detect errors (5a) will caused a rise project costs when large levels of quality assurance are used. This is very apparent in the actual project's results (DE-A) where a high level of QA was scheduled.

The ESS model (Test QA) reduced project cost by minimizing the total man-days devoted to QA and rework. Interestingly, the model allowed a large majority of errors to escape, yet still managed to reduce testing man-days below

those required by the actual project (DE-A). The ESS model's resulting distribution scheme and statistics closely resemble the base case results (Base QA). Model performance in this environment was as expected.

A decrease in the QA manpower required to detect errors (5b) means that any additional QA effort used should be more economical and effective than in the base case (Base QA). The actual project results (DE-A) indicate that this environment has increased QA effectiveness though the detection of most error prior to testing. However, the high project cost does not indicate that any monetary advantage can be gain with increased QA.

The ESS effectively reduce QA effort with only minor increases in escaped errors. Taking advantage of the project team's efficiency at QA, the ESS model developed a distribution scheme which significantly reduced project costs. The relative position of the ESS pattern (Test QA) above the base case distribution indicates that the model is capable of capitalizing on some limited cost reduction by increasing QA. Likewise, the ESS pattern position below the actual project curve (DE-A), indicates that the model is able to discern the environmental limits when developing a QA scheme.

EXPERIMENT 5a

This experiment examines the model's sensitivity when nominal QA manpower required to detect average errors is increased by 50 percent (+50%).

PATTERN SEARCH ESS MODEL PROJECT STATISTICS (TEST QA):

COMPLETION TIME	331.00	DAYS
TOTAL EFFORT	1,491.51	MAN-DAYS
QA EFFORT	177.87	MAN-DAYS
DEVELOP EFFORT	928.74	MAN-DAYS
REWORK EFFORT	112.55	MAN-DAYS
TESTING EFFORT	208.93	MAN-DAYS
TRAINING EFFORT	63.42	MAN-DAYS
TOTAL ERRORS GENERATED	488.24	ERRORS GENERATED
TOTAL ERRORS DETECTED	125.52	ERRORS DETECTED
TOTAL ERRORS ESCAPED	362.00	ERRORS THAT ESCAPED

SYSTEM DYNAMICS MODEL PROJECT STATISTICS (DE-A):

COMPLETION TIME	384.00	DAYS
TOTAL EFFORT	2,050.56	MAN-DAYS
QA EFFORT	510.00	MAN-DAYS
DEVELOP EFFORT	902.84	MAN-DAYS
REWORK EFFORT	259.85	MAN-DAYS
TESTING EFFORT	264.74	MAN-DAYS
TRAINING EFFORT	113.13	MAN-DAYS
TOTAL ERRORS GENERATED	496.30	ERRORS GENERATED
TOTAL ERRORS DETECTED	348.65	ERRORS DETECTED
TOTAL ERRORS ESCAPED	146.68	ERRORS THAT ESCAPED

EXPERIMENT 5a: QA DISTRIBUTION COMPARISON

	10	20	30	40	50	60	70	80	90	100
DE-A	.325	.290	.275	.255	.250	.275	.325	.375	.400	.400
Base QA	.300	.350	.100	.100	.020	.020	.020	.150	.020	.020
Test QA	.325	.400	.050	.020	.020	.020	.125	.020	.020	.020

EXPERIMENT 5b

This experiment examines the model's sensitivity when nominal QA manpower required to detect average errors is decreased by 50 percent (-50%).

PATTERN SEARCH ESS MODEL PROJECT STATISTICS (TEST QA):

COMPLETION TIME	327.00	DAYS
TOTAL EFFORT	1,528.53	MAN-DAYS
QA EFFORT	160.66	MAN-DAYS
DEVELOP EFFORT	909.80	MAN-DAYS
REWORK EFFORT	239.35	MAN-DAYS
TESTING EFFORT	153.66	MAN-DAYS
TRAINING EFFORT	65.05	MAN-DAYS
TOTAL ERRORS GENERATED	492.14	ERRORS GENERATED
TOTAL ERRORS DETECTED	317.90	ERRORS DETECTED
TOTAL ERRORS ESCAPED	173.66	ERRORS THAT ESCAPED

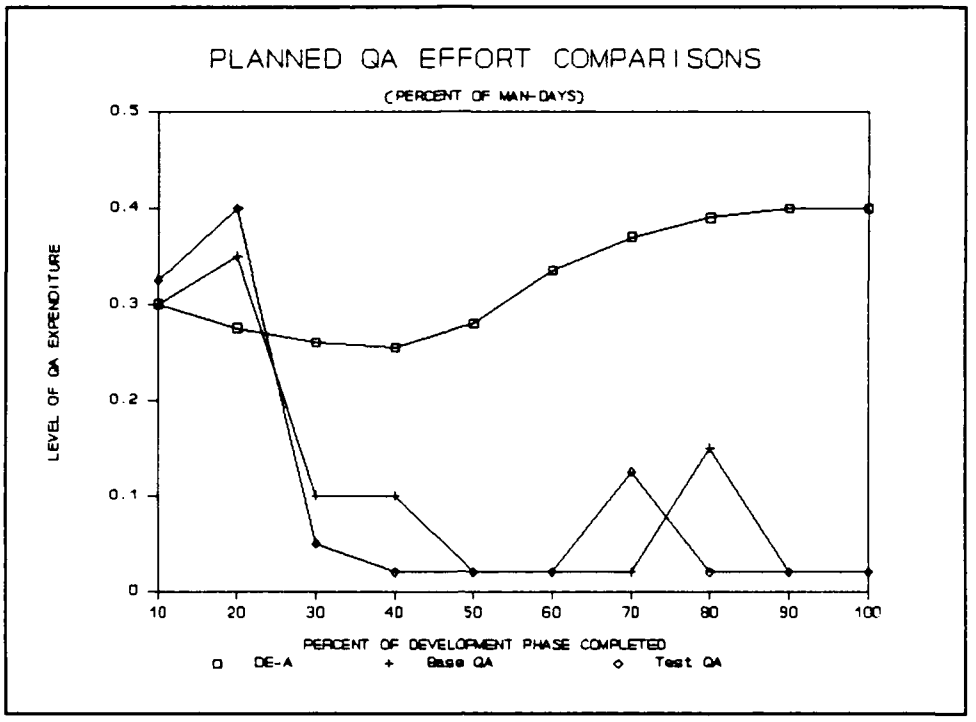
SYSTEM DYNAMICS MODEL PROJECT STATISTICS (DE-A):

COMPLETION TIME	390.00	DAYS
TOTAL EFFORT	2,139.22	MAN-DAYS
QA EFFORT	542.88	MAN-DAYS
DEVELOP EFFORT	918.70	MAN-DAYS
REWORK EFFORT	287.94	MAN-DAYS
TESTING EFFORT	268.23	MAN-DAYS
TRAINING EFFORT	121.47	MAN-DAYS
TOTAL ERRORS GENERATED	496.98	ERRORS GENERATED
TOTAL ERRORS DETECTED	386.06	ERRORS DETECTED
TOTAL ERRORS ESCAPED	109.94	ERRORS THAT ESCAPED

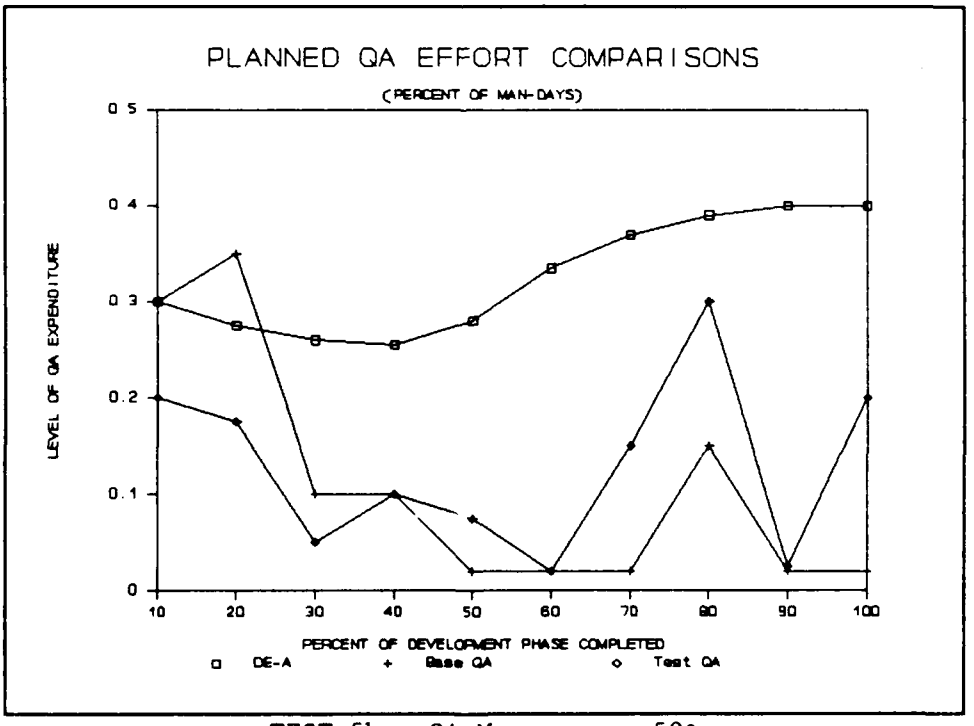
EXPERIMENT 5b: QA DISTRIBUTION COMPARISON

	10	20	30	40	50	60	70	80	90	100
DE-A	.325	.290	.275	.255	.250	.275	.325	.375	.400	.400
Base QA	.300	.350	.100	.100	.020	.020	.020	.150	.020	.020
Test QA	.200	.175	.050	.100	.075	.020	.150	.300	.025	.200

EXPERIMENT 5

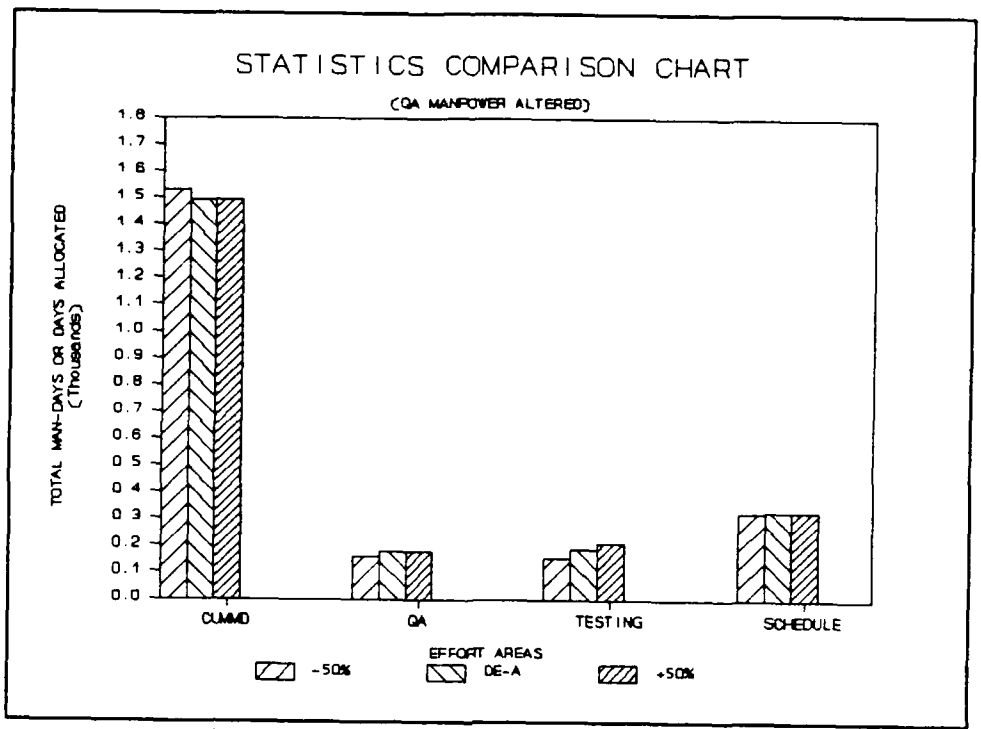


TEST 5a: QA Manpower + 50%

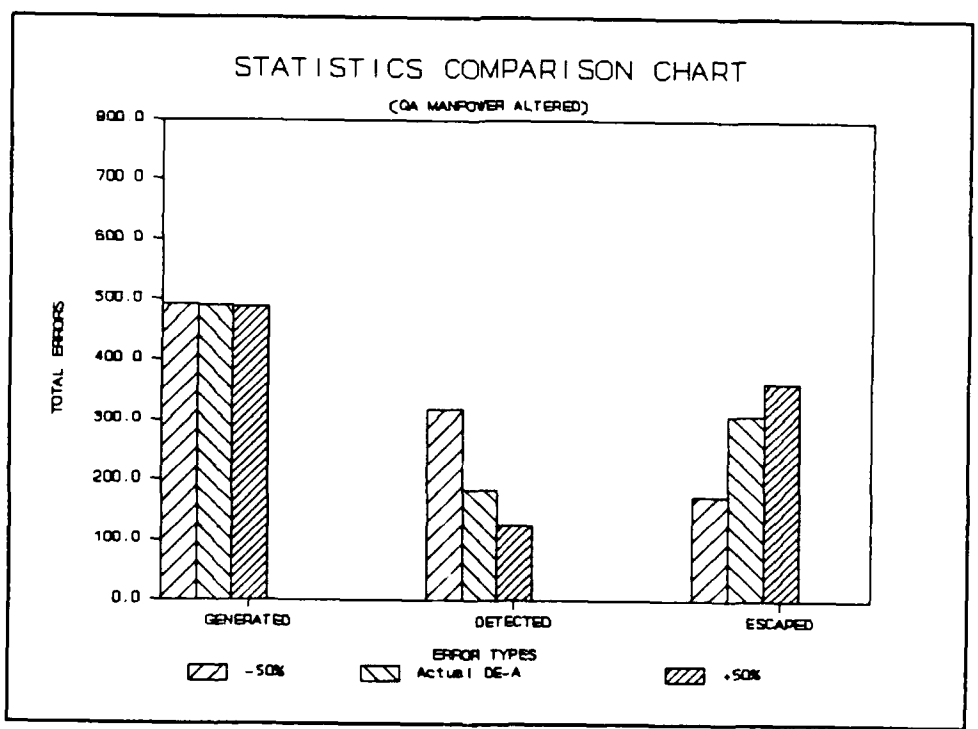


TEST 5b: QA Manpower - 50%

EXPERIMENT 5



TEST 5: Effort Allocation Comparison



TEST 5: Error Handling Comparison

6. Experiment 6: Bad Fixes

This experiment examines the ESS model's sensitivity to changes in the number of bad fixes. This analysis was divided into two sub-experiments. Experiment 6a increased the overall bad fix percentage, and 6b decreased the percentage. A single system test constant in the system dynamics model was altered: percent of bad fixes (PBADFX). The experimental values are shown in Table 5-6 below.

TABLE 5-6

BAD FIXES

Experiment 6a	Experiment 6b
PBADFX = .15	PBADFX = .05

An increase in bad fixes (6a: from 7.5% to 15%) had only minor influence on project results. In the actual project (DE-A) this environment caused increases in the QA, rework, training, and testing effort areas. These effort increases subsequently caused a rise in the project's cost.

As expected, the ESS model (Test QA) allocated more effort to QA and rework than the base case (Base QA). QA was established at a little less than half the total effort, and allowed to declined steadily over the entire design phase. Once reaching the coding phase the model virtually eliminated

QA in an attempt to reduce project cost. The total errors that escaped were comparable to the number in the base case. However, project costs were inflated due to the doubling of bad fixes.

A decrease in bad fixes (6b: from 7.5% to 5%) subsequently reduced the number of errors that escaped. This provided substantial cost savings in testing for both the ESS model (Test QA) and the actual project (DE-A) results.

The ESS model attempted to redistribute the bad fix savings into the QA and rework effort areas. Although there was a reduction in the number of errors that escaped to testing, total project cost was higher than anticipated. The model's final distribution scheme was also quite erratic (similar to experiments 3b and 4b). QA was emphasized at the beginning of the life cycle process as was done in all prior experiments. However, the model allocated a considerable amount of QA during the bulk of the coding phase. This allocation technique does not occur in any of the other experiments and is well above level of QA the base case (Base QA) allocates. It was anticipated that a reduction in bad fixes would have a more positive impact on the experiment's results. The opposite resulted, indicating the ESS model is sensitive to this environment. Experiment 6b <u>failed</u> to produce the expected results.

EXPERIMENT 6a

This experiment examines the model's sensitivity when bad fixes are estimated to be 15 percent of total fixes (original project estimated at 7.5 percent).

PATTERN SEARCH ESS MODEL PROJECT STATISTICS (TEST QA):

COMPLETION TIME	331.00	DAYS
TOTAL EFFORT	1,507.92	MAN-DAYS
QA EFFORT	150.55	MAN-DAYS
DEVELOP EFFORT	931.71	MAN-DAYS
REWORK EFFORT	144.93	MAN-DAYS
TESTING EFFORT	217.04	MAN-DAYS
TRAINING EFFORT	63.68	MAN-DAYS
TOTAL ERRORS GENERATED	488.23	ERRORS GENERATED
TOTAL ERRORS DETECTED	160.29	ERRORS DETECTED
TOTAL ERRORS ESCAPED	327.35	ERRORS THAT ESCAPED

SYSTEM DYNAMICS MODEL PROJECT STATISTICS (DE-A):

COMPLETION TIME	389.00	DAYS
TOTAL EFFORT	2,142.50	MAN-DAYS
QA EFFORT	526.36	MAN-DAYS
DEVELOP EFFORT	912.94	MAN-DAYS
REWORK EFFORT	273.49	MAN-DAYS
TESTING EFFORT	304.51	MAN-DAYS
TRAINING EFFORT	125.20	MAN-DAYS
TOTAL ERRORS GENERATED	497.37	ERRORS GENERATED
TOTAL ERRORS DETECTED	365.85	ERRORS DETECTED
TOTAL ERRORS ESCAPED	130.74	ERRORS THAT ESCAPED

EXPERIMENT 6a: QA DISTRIBUTION COMPARISON

	10	20	30	40	50	60	70	80	90	100
DE-A	.325	.290	.275	.255	.250	.275	.325	.375	.400	.400
Base QA	.300	.350	.100	.100	.020	.020	.020	.150	.020	.020
Test QA	.350	.250	.150	.050	.063	.020	.020	.020	.020	.020

EXPERIMENT 6b

This experiment examines the model's sensitivity when bad fixes are estimated to be 5 percent of total fixes (original project estimated at 7.5 percent).

PATTERN SEARCH ESS MODEL PROJECT STATISTICS (TEST QA):

COMPLETION TIME	329.00	DAYS
TOTAL EFFORT	1,526.87	MAN-DAYS
QA EFFORT	191.02	MAN-DAYS
DEVELOP EFFORT	902.66	MAN-DAYS
REWORK EFFORT	193.34	MAN-DAYS
TESTING EFFORT	175.03	MAN-DAYS
TRAINING EFFORT	64.82	MAN-DAYS
TOTAL ERRORS GENERATED	492.74	ERRORS GENERATED
TOTAL ERRORS DETECTED	245.78	ERRORS DETECTED
TOTAL ERRORS ESCAPED	246.15	ERRORS THAT ESCAPED

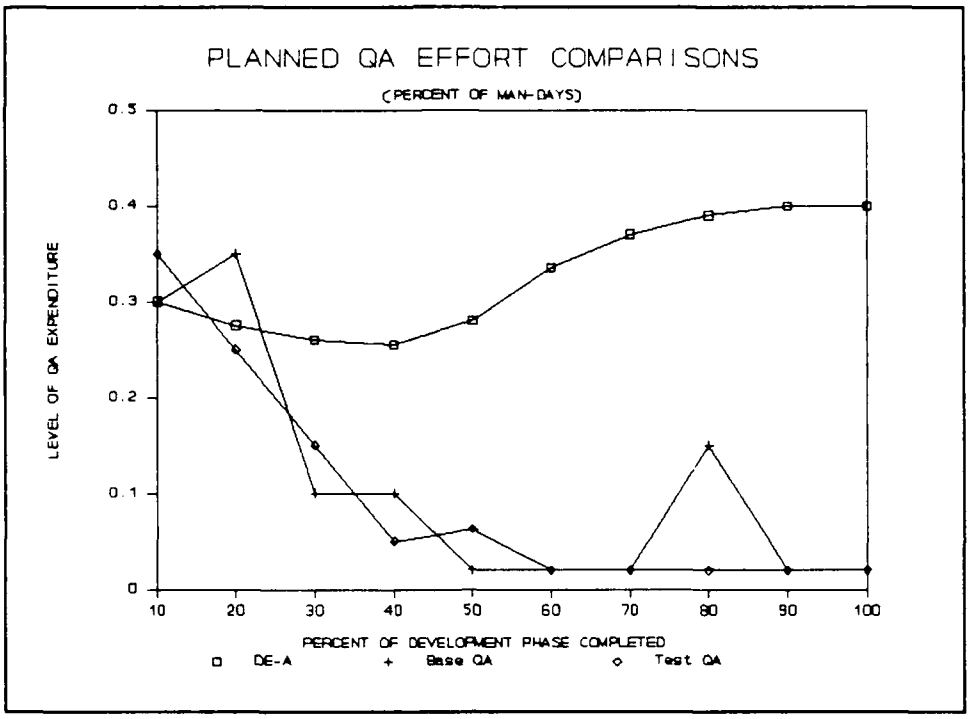
SYSTEM DYNAMICS MODEL PROJECT STATISTICS (DE-A):

COMPLETION TIME	390.00	DAYS
TOTAL EFFORT	2,139.22	MAN-DAYS
QA EFFORT	542.88	MAN-DAYS
DEVELOP EFFORT	918.70	MAN-DAYS
REWORK EFFORT	287.94	MAN-DAYS
TESTING EFFORT	268.23	MAN-DAYS
TRAINING EFFORT	121.47	MAN-DAYS
TOTAL ERRORS GENERATED	496.98	ERRORS GENERATED
TOTAL ERRORS DETECTED	386.06	ERRORS DETECTED
TOTAL ERRORS ESCAPED	109.94	ERRORS THAT ESCAPED

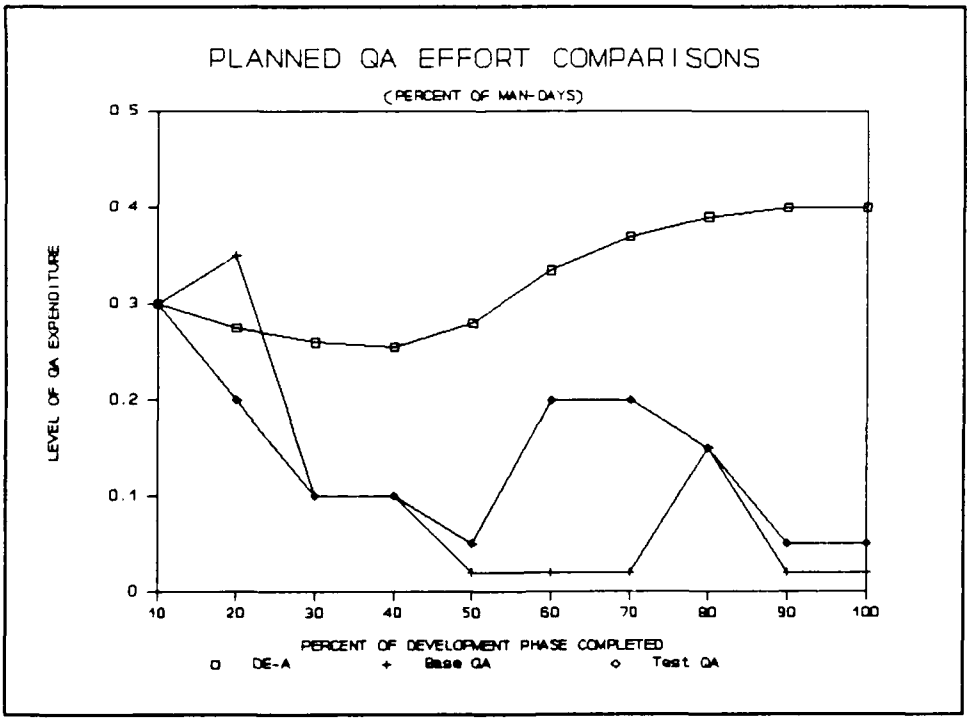
EXPERIMENT 6b: QA DISTRIBUTION COMPARISON

	10	20	30	40	50	60	70	80	90	100
DE-A	.325	.290	.275	.255	.250	.275	.325	.375	.400	.400
Base QA	.300	.350	.100	.100	.020	.020	.020	.150	.020	.020
Test QA	.300	.200	.100	.100	.050	.200	.200	.150	.050	.050

EXPERIMENT 6

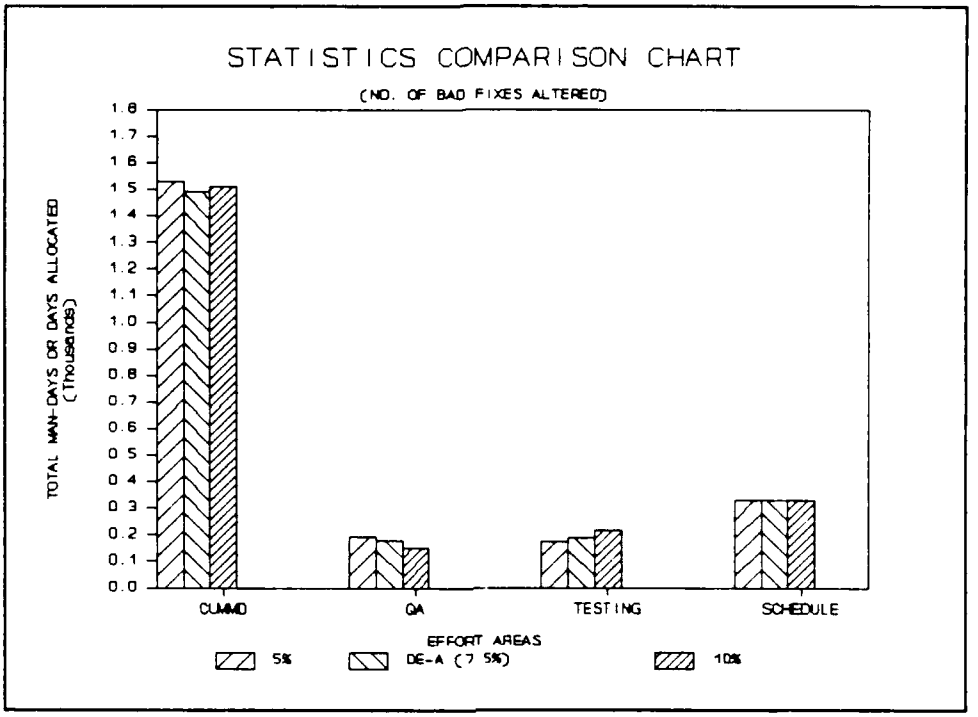


TEST 6a: Bad Fixes 15%

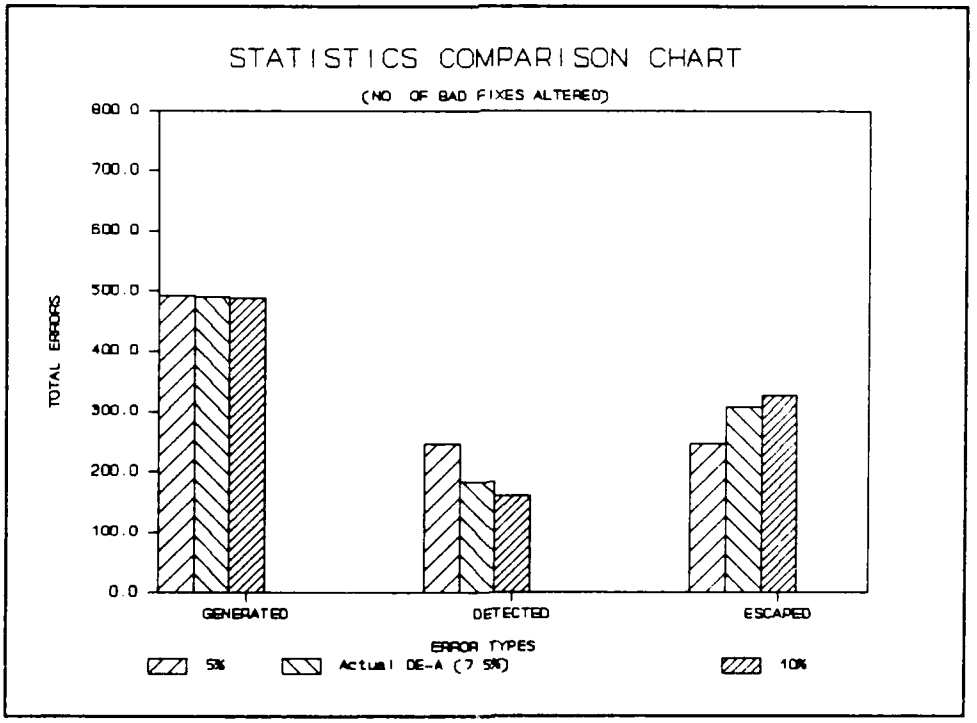


TEST 6b: Bad Fixes 5%

EXPERIMENT 6



TEST 6: Effort Allocation Comparison



TEST 6: Error Handling Comparison

C. RESULTS

The pattern search ESS model was able to effect some cost, effort, and time reductions in all 11 sensitivity experiments. Eight of these experiments produced results similar to what was anticipated. Identifying close to optimal solutions for their specific environment, these eight developed QA distribution schemes that significantly reduced project costs.

Experiment 3b (errors per KDSI reduced), experiment 4b (rework manpower required per error reduced), and experiment 6b (percentage of bad fixes reduced) all produced unexpected results. Although their QA distribution schemes did render lower project costs, their allocation patterns were very erratic. These three experiments exist in environments were significant changes in QA allocation have minimal effect on total cost (CUMMD). This would mean that a high percentage of alternate distribution schemes are capable of providing similar results.

The ESS model is vulnerable to these "low impact" environments. When operating in this environment it is possible for the ESS model to continue in the same direction that was first established for each life cycle point. Since most patterns produce similar results, the model has no reason to change directions. Once the model terminates the final QA distribution can be an erratic pattern ranging between predetermined limits (set at .02 to .5 for all experiment runs). This is exactly the product of the three unsuccessful

experiments. Further experimentation is required to identify and eliminate the ESS model's sensitivity to this environment.

VI. CONCLUSION

A. ACCOMPLISHMENTS

The focus of this thesis was to enhance the performance of an existing expert system simulation (ESS) model. Through a series of experiments, the performance limitations of the prototype system were identified. These limitations were resolved by the addition of a pattern search technique within the existing expert system program.

Evaluating the performance of the pattern search ESS model showed great improvements over the prototype system. The pattern search technique minimized the QA environment's bi-modal problems, thus enabling the model to identify similar QA distribution schemes independent of where the system was initialized. All system results were established in fewer cycles, at a lower cost, and with a "smoother" curve than the prototype was capable of achieving.

B. SUGGESTIONS FOR FURTHER RESEARCH

The pattern search methodology effectively resolved the problem areas identified in the prototype ESS model. However, this new technique is not devoid of its own sensitivity problems.

Eleven experimental scenarios were used to evaluate the performance of the new system. Of these experiments, three failed to produce the expected results. These failed

scenarios were classified as "low impact" environments, where large changes in QA effort had minimal effect on project cost. In this environment the ESS model was able to make almost any allocation decision with similar cost results. This lead to extremely erratic QA distribution schemes.

Further research can be broken down into two areas: (1) improving the pattern search ESS model, and (2) developing a new expert system model using a more sophisticated optimization technique.

Improvements to the ESS model will require further experimentation in "low impact" environments. Additions to the system should include mechanisms to identify low impact situations, and possibly bias the system towards minimizing QA when impact on costs is insignificant. The desired outcome is a model which provides a more predictable and less erratic solution to all project scenarios.

A separate and smaller expert system (seven-ten rules) can be developed as a test vehicle for further development effort. This system may provide valuable insights into the complexities surrounding the distribution of quality assurance. Any discoveries could be used to enhance the existing model or may identify a technique that better handles this environment.

One possible technique to replace pattern search is, "Optimization by Simulated Annealing." This technique makes use of statistical evaluation mechanisms to derive optimal

patterns in a complex environment. Using an analogy to metal annealing, this technique makes small adjustment to a particular variable and allows the system to move to equilibrium before making the next change. A complex optimization methodology may be the best method of handling an equally complex environment. [Ref. 14:pp. 1-9]

APPENDIX A

PROGRAM LISTING OF PATTERN.ARI

```
/*****************************************************************/
/*                                                               */
/* PROGRAM - pattern.ari                                         */
/*                                                               */
/* This program utilizes a "Pattern Search" technique,           */
/* developed by R. Hooke and T. Jeeves, to adjust percentage     */
/* of Quality Assurance (TPFMQA) allocated for each of 10        */
/* lifecycle points (Design thru Coding phases).  This           */
/* technique identifies an optimal QA scheme and subsequently    */
/* reduces total cummulative man-months (CUMMD) required to      */
/* complete the project.                                         */
/*                                                               */
/*****************************************************************/

/*****************************************************************/
/*                                                               */
/* RULE - pqa                                                    */
/*                                                               */
/* This rule initializes the system parameters and gets the      */
/* system started.                                               */
/*                                                               */
/*****************************************************************/

pqa:-

/* sets initial global values */
   asserta(number(1)),
   asserta(calc(0)),
   asserta(flag(0)),
   asserta(newcycle(1)),

/* user input queries */
   write('What is your desired pulse size factor? '),
   read(PU),
   asserta(size(PU)),

   write('What is the minimum QA value? '),
   read(MN),
   asserta(min(MN)),

   write('What is the maximum QA value? '),
   read(AX),
   asserta(max(AX)),
```

```
   write('What is your minimum desired pulse size? '),
   read(MP),
   asserta(minpsf(MP)),

   write('Enter the initial QA distribution. Point 1  '),
   read(QA1),

   write('                                  Point 2  '),
   read(QA2),

   write('                                  Point 3  '),
   read(QA3),

   write('                                  Point 4  '),
   read(QA4),

   write('                                  Point 5  '),
   read(QA5),

   write('                                  Point 6  '),
   read(QA6),

   write('                                  Point 7  '),
   read(QA7),

   write('                                  Point 8  '),
   read(QA8),

   write('                                  Point 9  '),
   read(QA9),

   write('                                  Point 10 '),
   read(QA10),

/* prints starting values into the summary output */
   print_head(PU,MN,AX,MP),

/* initializes the system with user inputs */
   initial_run(QA1,QA2,QA3,QA4,QA5,QA6,QA7,QA8,QA9,QA10),

   dopqa.

/* end of pqa */

/*****************************************************************/
/*                                                               */
/* RULE - dopqa                                                  */
/*                                                               */
/* Creates a repeat-fail loop to execute main rule until the     */
/* step size (PSF) falls below the user input minimum.           */
/*                                                               */
/*****************************************************************/
```

```
dopqa:-

   repeat,

   main,

   fail.

/* end of dopqa */

/*****************************************************************/
/*                                                               */
/* RULE - main                                                   */
/*                                                               */
/* This module controls the perturbation activity of the         */
/* pattern search technique.                                     */
/*                                                               */
/*****************************************************************/

main:-

/* gets the x value for the pulse */
   call(number(ITER)),
   call(calc(TYPE)),

/* gets the previous man days */
   PREV is [[[ITER + 8] mod 10] + 1],
   call(cummdold(PREV,CHECK)),
   call(cummdold(ITER,OLD)),

/* gets the man days from the last QA numbers */
   read_cummd,
   call(cummd(NEW)),

/* calculates the new y value (QA) for the current x value */
   case([NEW =< CHECK -> calc_less(ITER,NEW,OLD,TYPE),
         NEW > CHECK -> calc_more(ITER,NEW,OLD,CHECK,TYPE)]),

/* prints to screen module statistics for monitoring */
   statistics,

   write(' Iteration = '),write(ITER).

/* end of main */
```

```
/*****************************************************************/
/*                                                               */
/* RULE - end_cycle                                              */
/*                                                               */
/* Advances cycle count and initiates pattern search technique */
/* after initial perturbations have occurred.                    */
/*                                                               */
/*****************************************************************/

end_cycle(NEW,ITER):-

   call(newcycle(NOW)),

/* initializes the "Pattern Search" algorithm */
   ifthenelse(NOW > 1,pattern_search(NEW,ITER),
              output_cummd(NEW,ITER)),

/* advances cycle number */
   NEXT is NOW + 1,
   write('Cycle number = '),write(NOW),
   retract(newcycle(NOW)),
   asserta(newcycle(NEXT)).

/* end of end_cycle */

/*****************************************************************/
/*                                                               */
/* RULE - pattern_search                                         */
/*                                                               */
/* Evaluation rule that monitors past preformance to determine */
/* the next course of action.                                    */
/*                                                               */
/*****************************************************************/

pattern_search(NEW,ITER):-

   call(flag(TYPE)),
   case([TYPE =:= 0 -> temp_base(NEW,ITER),
         TYPE =:= 1 -> reduction_test(NEW,ITER)]),!.

/* end of pattern_search */

/*****************************************************************/
/*                                                               */
/* RULE - reduction_test                                         */
/*                                                               */
/* Further evaluation rule that determines if local              */
/* perturbations or pulse size reduction is the next logical     */
/* course of action.                                             */
/*                                                               */
/*****************************************************************/
```

```
reduction_test(NEW,ITER):-

   call(b1_cummd(BASE)),
   ifthenelse(NEW < BASE,temp_base(NEW,ITER),
                         reduce_pulse(NEW,ITER)),!.

/* end of reduction_test */

/*****************************************************************/
/*                                                               */
/* RULE - temp_base                                              */
/*                                                               */
/* "Pattern Search" algorithm that captilizes on a               */
/* preestablished pattern in determining the next QA value       */
/* for the 10 lifecycle points.                                  */
/*                                                               */
/*****************************************************************/

temp_base(NEW,ITER):-

   output_cummd(NEW,ITER),

/* resets "flag" to establish a record of last action taken */
   abolish(flag/1),
   asserta(flag(0)),

/* resets the 3 CUMMD value holders */
   call(b1_cummd(OLD)),
   abolish(b0_cummd/1),
   asserta(b0_cummd(OLD)),
   abolish(b1_cummd/1),
   asserta(b1_cummd(NEW)),

/* pattern search algorithm */
   call(b0(1,Q1)),
   call(tpfmqa(1,QA1)),
   A1 is Q1+[2*[QA1-Q1]],
   qa_test(A1,T1),
   asserta(t(1,T1)),

   call(b0(2,Q2)),
   call(tpfmqa(2,QA2)),
   A2 is Q2+[2*[QA2-Q2]],
   qa_test(A2,T2),
   asserta(t(2,T2)),

   call(b0(3,Q3)),
   call(tpfmqa(3,QA3)),
   A3 is Q3+[2*[QA3-Q3]],
   qa_test(A3,T3),
   asserta(t(3,T3)),
```

```
    call(b0(4,Q4)),
    call(tpfmqa(4,QA4)),
    A4 is Q4+[2*[QA4-Q4]],
    qa_test(A4,T4),
    asserta(t(4,T4)),

    call(b0(5,Q5)),
    call(tpfmqa(5,QA5)),
    A5 is Q5+[2*[QA5-Q5]],
    qa_test(A5,T5),
    asserta(t(5,T5)),

    call(b0(6,Q6)),
    call(tpfmqa(6,QA6)),
    A6 is Q6+[2*[QA6-Q6]],
    qa_test(A6,T6),
    asserta(t(6,T6)),

    call(b0(7,Q7)),
    call(tpfmqa(7,QA7)),
    A7 is Q7+[2*[QA7-Q7]],
    qa_test(A7,T7),
    asserta(t(7,T7)),

    call(b0(8,Q8)),
    call(tpfmqa(8,QA8)),
    A8 is Q8+[2*[QA8-Q8]],
    qa_test(A8,T8),
    asserta(t(8,T8)),

    call(b0(9,Q9)),
    call(tpfmqa(9,QA9)),
    A9 is Q9+[2*[QA9-Q9]],
    qa_test(A9,T9),
    asserta(t(9,T9)),

    call(b0(10,Q10)),
    call(tpfmqa(10,QA10)),
    A10 is Q10+[2*[QA10-Q10]],
    qa_test(A10,T10),
    asserta(t(10,T10)),

/* establishes a new temporary base */
    new_base,

/* notification in the output that a pattern search occurred */
    open(S,'summary.dat',a),
    nl(S),write(S,
    '********************* Pattern Search **********************'),
    nl(S),nl(S),close(S),
```

```
/* runs the system dynamics model for a pattern search CUMMD */
   output_tpfmqa,

   shell(model),

   read_cummd,
   call(cummd(HEAD)),
   output_cummd(HEAD,ITER),

/* if CUMMD is not an improvement, local explorations occur */
   ifthen(HEAD >= NEW,local_explorations),

/* resets CUMMD recording system for perturbation comparisons */
   retract(cummdold(ITER,NEW)),
   asserta(cummdold(ITER,HEAD)),

/* erases temporary head (T) values */
   abolish(t/2),!.

/* end of temp_base */

/*****************************************************************/
/*                                                               */
/* RULE - qa_test                                                */
/*                                                               */
/* Used in the "temp_base" rule to keep the pattern from         */
/* jumping below the minimum QA value.                           */
/*                                                               */
/*****************************************************************/

qa_test(IN,OUT):-

   call(min(MINQA)),
   ifthenelse(IN < MINQA,OUT is MINQA,OUT is IN).

/* end of qa_test */

/*****************************************************************/
/*                                                               */
/* RULE - new_base                                               */
/*                                                               */
/* This rule actually records the "Pattern Search" values        */
/* into the program.                                             */
/*                                                               */
/*****************************************************************/

new_base:-

/* calls last perturbation values and pattern search values */
   call(tpfmqa(1,QA1)),
   call(tpfmqa(2,QA2)),
   call(tpfmqa(3,QA3)),
```

```
      call(tpfmqa(4,QA4)),
      call(tpfmqa(5,QA5)),
      call(tpfmqa(6,QA6)),
      call(tpfmqa(7,QA7)),
      call(tpfmqa(8,QA8)),
      call(tpfmqa(9,QA9)),
      call(tpfmqa(10,QA10)),

      call(t(1,A1)),
      call(t(2,A2)),
      call(t(3,A3)),
      call(t(4,A4)),
      call(t(5,A5)),
      call(t(6,A6)),
      call(t(7,A7)),
      call(t(8,A8)),
      call(t(9,A9)),
      call(t(10,A10)),

   /* resets a new base */
      abolish(b0/2),

      asserta(b0(1,QA1)),
      asserta(b0(2,QA2)),
      asserta(b0(3,QA3)),
      asserta(b0(4,QA4)),
      asserta(b0(5,QA5)),
      asserta(b0(6,QA6)),
      asserta(b0(7,QA7)),
      asserta(b0(8,QA8)),
      asserta(b0(9,QA9)),
      asserta(b0(10,QA10)),

   /* resets a new temporary base */
      abolish(tpfmqa/2),

      asserta(tpfmqa(1,A1)),
      asserta(tpfmqa(2,A2)),
      asserta(tpfmqa(3,A3)),
      asserta(tpfmqa(4,A4)),
      asserta(tpfmqa(5,A5)),
      asserta(tpfmqa(6,A6)),
      asserta(tpfmqa(7,A7)),
      asserta(tpfmqa(8,A8)),
      asserta(tpfmqa(9,A9)),
      asserta(tpfmqa(10,A10)),!.

   /* end of new_base */
```

```
/*****************************************************************/
/*                                                               */
/* RULE - local_explorations                                     */
/*                                                               */
/* This rule is invoked when the last pattern search did not     */
/* result in an improved CUMMD than the previous base (b1).      */
/*                                                               */
/*****************************************************************/

local_explorations:-

/* resets "flag" to establish a record of last action taken */
   abolish(flag/1),
   asserta(flag(1)),

/* notification in output that a local exploration occurred */
   open(S,'summary.dat',a),
   nl(S),write(S,
   '****************** Local Explorations ******************'),
   nl(S),nl(S),close(S),!.

/* end of local_explorations */

/*****************************************************************/
/*                                                               */
/* RULE - reduce_pulse                                           */
/*                                                               */
/* Decreases the pulse size factor by half and reestablishes     */
/* the previous base values (TPFMQA) after local explorations    */
/* failed to produce an improved CUMMD.                          */
/*                                                               */
/*****************************************************************/

reduce_pulse(NEW,ITER):-

   output_cummd(NEW,ITER),

   call(size(PULSE)),
   call(minpsf(MPSF)),
   call(b0_cummd(OLD)),

/* reduces current pulse size (PSF) by half */
   REDUCED is round(PULSE / 2,3),

/* notification in the output that a pulse reduction occurred */
   open(S,'summary.dat',a),
   nl(S),write(S,
   '*****************      Pulse reduction     ***************'),
   nl(S),write(S,'Pulse size is: '),write(S,REDUCED),
   nl(S),nl(S),
   close(S),
```

```
/* resets old base TPFMQA values to those prior  */
/* to the pattern search and local explorations */
   call(b0(1,QA1)),
   call(b0(2,QA2)),
   call(b0(3,QA3)),
   call(b0(4,QA4)),
   call(b0(5,QA5)),
   call(b0(6,QA6)),
   call(b0(7,QA7)),
   call(b0(8,QA8)),
   call(b0(9,QA9)),
   call(b0(10,QA10)),

   abolish(tpfmqa/2),

   asserta(tpfmqa(1,QA1)),
   asserta(tpfmqa(2,QA2)),
   asserta(tpfmqa(3,QA3)),
   asserta(tpfmqa(4,QA4)),
   asserta(tpfmqa(5,QA5)),
   asserta(tpfmqa(6,QA6)),
   asserta(tpfmqa(7,QA7)),
   asserta(tpfmqa(8,QA8)),
   asserta(tpfmqa(9,QA9)),
   asserta(tpfmqa(10,QA10)),

   output_tpfmqa,

   shell(model),

   read_cummd,
   call(cummd(AFTER)),
   output_cummd(AFTER,ITER),

/* resets CUMMD recording system for perturbation comparisons */
   abolish(b1_cummd/1),
   asserta(b1_cummd(OLD)),

   retract(cummdold(10,BEFORE)),
   asserta(cummdold(10,AFTER)),
   retract(cummdold(ITER,NEW)),
   asserta(cummdold(ITER,AFTER)),

/* terminates the model if pulse is below user input mininmum */
   retract(size(PULSE)),
   ifthenelse(REDUCED >= MPSF,asserta(size(REDUCED)),
              run_stop(OLD)),

/* resets "flag" to establish a record of last action taken */
   abolish(flag/1),
   asserta(flag(0)),!.
```

```
/* end of reduce_pulse */

/*****************************************************************/
/*                                                               */
/* RULE - print_head                                             */
/*                                                               */
/* Creates summary data file (summary.dat) for recording all     */
/* TPFMQA values sent to the system dynamics module and total    */
/* project costs data (CUMMD) that is returned.  Secondly, it    */
/* records all user input data as the header of this file.       */
/*                                                               */
/*****************************************************************/

print_head(PU,MN,AX,MP):-

   create(S,'summary.dat'),

   write(S,'     Pulse size factor  = '),write(S,PU),nl(S),
   write(S,'     Minimum QA value   = '),write(S,MN),nl(S),
   write(S,'     Maximum QA value   = '),write(S,AX),nl(S),
   write(S,'     Minimum pulse size = '),write(S,MP),nl(S),nl(S),

   close(S).

/* end of print_head */

/*****************************************************************/
/*                                                               */
/* RULE - calc_less                                              */
/*                                                               */
/* Negative perturbation (pulse).                                */
/*                                                               */
/*****************************************************************/

calc_less(ITER,NEW,OLD,TYPE):-

/* records man-days for this cycle */
   retract(cummdold(ITER,OLD)),
   asserta(cummdold(ITER,NEW)),

   ifthenelse(ITER =:= 1,end_cycle(NEW,ITER),
              output_cummd(NEW,ITER)),

/* reads the current QA values */
   call(tpfmqa(ITER,QA)),

/* establishes an "initial" QA holder for perturbation use */
   abolish(holder/1),
   asserta(holder(QA)),

   call(size(PULSE)),
```

```
/* calculates a new QA value (NEWQA) */
   NEWQA is round(QA-PULSE,3),

/* checks if the new QA is less than the minimum (MINQA) */
/* add the new QA value to the database */
   call(min(MINQA)),
   retract(tpfmqa(ITER,QA)),
   ifthenelse(NEWQA < MINQA,asserta(tpfmqa(ITER,MINQA)),
              asserta(tpfmqa(ITER,NEWQA))),

/* moves to the next life cycle position */
   retract(number(ITER)),

   NEWITER is [ITER mod 10] + 1,
   asserta(number(NEWITER)),

   ifthen(ITER =:= 1,output_break),
   output_tpfmqa,

   shell(model),

/* records that last pulse was negative */
   retract(calc(TYPE)),
   asserta(calc(0)).

/* end of calc_less */

/*****************************************************************/
/*                                                               */
/* RULE - calc_more                                              */
/*                                                               */
/* Calculates a positive perturbation (pulse) if the previous    */
/* negative pulse resulted in a higher man-days value.           */
/* If the man-days resulting from the positive pulse also        */
/* results in a higher man-days value then the QA value will     */
/* be returned to its original value for this cycle.             */
/*                                                               */
/*****************************************************************/

calc_more(ITER,NEW,OLD,CHECK,TYPE):-

/* resets the life cycle position back 1 */
   NEWITER is [[[ITER + 8] mod 10] + 1],

   call(tpfmqa(NEWITER,QA)),

/* calls the "initial" QA value for perturbation use */
   call(holder(VALUE)),

   call(size(PULSE)),
```

```
/* calculates the new QA value depending on whether the last */
/* pulse was negative (TYPE = 0) or positive (TYPE = 1) */
   case([TYPE =:= 0 -> NEWQA is round(VALUE+PULSE,3),
         TYPE =:= 1 -> NEWQA is VALUE]),


/* checks if the new QA is greater than the maximum (MAXQA) */
   call(max(MAXQA)),
   retract(tpfmqa(NEWITER,QA)),
   ifthenelse(NEWQA > MAXQA,asserta(tpfmqa(NEWITER,MAXQA)),
              asserta(tpfmqa(NEWITER,NEWQA))),

   retract(calc(TYPE)),

/* resets the type of calculation */
   case([TYPE =:= 0 -> calc_up(NEW,NEWITER),
         TYPE =:= 1 -> calc_orig(NEWITER,ITER,NEW,OLD,CHECK)]).

/* end of calc_more */

/*****************************************************************/
/*                                                               */
/* RULE - calc_up                                                */
/*                                                               */
/* Runs the dynamo model after an increase in the current        */
/* QA position.                                                  */
/*                                                               */
/*****************************************************************/

calc_up(NEW,NEWITER):-

   asserta(calc(1)),

   output_cummd(NEW,NEWITER),
   output_tpfmqa,

   shell(model).

/* end of calc_up */

/*****************************************************************/
/*                                                               */
/* RULE - calc_orig                                              */
/*                                                               */
/* After the QA value at a point has been returned to its        */
/* original value, this rule continues with the next point.      */
/*                                                               */
/*****************************************************************/

calc_orig(NEWITER,ITER,NEW,OLD,CHECK):-
```

```
   asserta(calc(0)),

   output_cummd(NEW,NEWITER),
   output_tpfmqa,
   calc_less(ITER,CHECK,OLD,0).

/* end of calc_orig */

/*****************************************************************/
/*                                                               */
/* RULE - run_stop                                               */
/*                                                               */
/* Halts the program when an exit condition is met.              */
/*                                                               */
/*****************************************************************/

run_stop(NEW):-

   call(cummdold(10,BEST)),

   open(S,'summary.dat',a),
   nl(S),nl(S),

   write(S,'The best CUMMD is: '),

   ifthenelse(BEST < NEW,write(S,BEST),write(S,NEW)),

   nl(S),
   close(S),

   halt.

/* end of run_stop */

/*****************************************************************/
/*                                                               */
/* RULE - output_tpfmqa                                          */
/*                                                               */
/* Creates one file and updates one file:                        */
/*                                                               */
/*    1. project.dnx - created with the format required by the   */
/*       system dynamics simulation model.                       */
/*                                                               */
/*    2. summary.dat - updated by adding the current TPFMQA      */
/*       values to the end of the list of all previous values.   */
/*                                                               */
/*****************************************************************/

output_tpfmqa:-

   call(tpfmqa(1,QA1)),
   call(tpfmqa(2,QA2)),
```

```
    call(tpfmqa(3,QA3)),
    call(tpfmqa(4,QA4)),
    call(tpfmqa(5,QA5)),
    call(tpfmqa(6,QA6)),
    call(tpfmqa(7,QA7)),
    call(tpfmqa(8,QA8)),
    call(tpfmqa(9,QA9)),
    call(tpfmqa(10,QA10)),

    create(D,'project.dnx'),
    write(D,'T TPFMQA='),
    write(D,QA1),write(D,' '),
    write(D,QA2),write(D,' '),
    write(D,QA3),write(D,' '),
    write(D,QA4),write(D,' '),
    write(D,QA5),write(D,' '),
    write(D,QA6),write(D,' '),
    write(D,QA7),write(D,' '),
    write(D,QA8),write(D,' '),
    write(D,QA9),write(D,' '),
    write(D,QA10),nl(D),
    close(D),

    open(S,'summary.dat',a),
    write(S,'TPFMQA='),
    write(S,QA1),write(S,'/'),
    write(S,QA2),write(S,'/'),
    write(S,QA3),write(S,'/'),
    write(S,QA4),write(S,'/'),
    write(S,QA5),write(S,'/'),
    write(S,QA6),write(S,'/'),
    write(S,QA7),write(S,'/'),
    write(S,QA8),write(S,'/'),
    write(S,QA9),write(S,'/'),
    write(S,QA10),nl(S),
    close(S).

/* end of output_tpfmqa */

/*****************************************************************/
/*                                                               */
/* RULE - output_cummd                                           */
/*                                                               */
/* Outputs cumulative man-days total to the SUMMARY.DAT file.    */
/*                                                               */
/*****************************************************************/

output_cummd(NEW,ITER):-

    open(S,'summary.dat',a),
```

```
   write(S,ITER),write(S,'. '),
   write(S,'CUMMD='),write(S,NEW),nl(S),

   close(S).

/* end of output_cummd */

/*****************************************************************/
/*                                                               */
/* RULE - output_break                                           */
/*                                                               */
/* Outputs a line denoting the start of a new cycle to the       */
/* SUMMARY.DAT file.                                             */
/*                                                               */
/*****************************************************************/

output_break:-

   open(S,'summary.dat',a),

   nl(S),write(S,
   '*****************   Start of a new cycle   *****************'),
   nl(S),nl(S),

   close(S).

/* end of output_break */

/*****************************************************************/
/*                                                               */
/* RULE - read_cummd                                             */
/*                                                               */
/* Reads from PROJECT.OUT file the man-days total output by      */
/* the system dynamics simulation model.                         */
/*                                                               */
/*****************************************************************/

read_cummd:-

   open(C,'project.out',r),

   read(C,CUMMD),
   abolish(cummd/1),
   asserta(CUMMD),

   close(C).

/* end of read_cummd */
```

```
/*****************************************************************/
/*                                                               */
/* RULE - initial_run                                            */
/*                                                               */
/* Runs the system dynamics simulation model with the            */
/* initial TPFMQA values.                                        */
/*                                                               */
/*****************************************************************/

initial_run(QA1,QA2,QA3,QA4,QA5,QA6,QA7,QA8,QA9,QA10):-

/* establishes the initial temporary base for pattern search */
   asserta(tpfmqa(1,QA1)),
   asserta(tpfmqa(2,QA2)),
   asserta(tpfmqa(3,QA3)),
   asserta(tpfmqa(4,QA4)),
   asserta(tpfmqa(5,QA5)),
   asserta(tpfmqa(6,QA6)),
   asserta(tpfmqa(7,QA7)),
   asserta(tpfmqa(8,QA8)),
   asserta(tpfmqa(9,QA9)),
   asserta(tpfmqa(10,QA10)),

/* establishes the initial base for pattern search */
   asserta(b0(1,QA1)),
   asserta(b0(2,QA2)),
   asserta(b0(3,QA3)),
   asserta(b0(4,QA4)),
   asserta(b0(5,QA5)),
   asserta(b0(6,QA6)),
   asserta(b0(7,QA7)),
   asserta(b0(8,QA8)),
   asserta(b0(9,QA9)),
   asserta(b0(10,QA10)),

/* runs the system dynamics model */
   output_tpfmqa,
   shell(model),

   read_cummd,
   call(cummd(INITIAL)),

/* records initial CUMMDs for perturbation comparisons */
   asserta(b0_cummd(INITIAL)),
   asserta(b1_cummd(INITIAL)),

   asserta(cummdold(1,0)),
   asserta(cummdold(2,0)),
   asserta(cummdold(3,0)),
   asserta(cummdold(4,0)),
   asserta(cummdold(5,0)),
   asserta(cummdold(6,0)),
```

```
    asserta(cummdold(7,0)),
    asserta(cummdold(8,0)),
    asserta(cummdold(9,0)),
    asserta(cummdold(10,INITIAL)),

    retract(cummd(INITIAL)).

/* end of initial_run */

/* end of program pattern.ari */
```

APPENDIX B

PATTERN SEARCH RESULTS

```
     Pulse size factor  = 0.05
     Minimum QA value   = 0.02
     Maximum QA value   = 0.5
     Minimum pulse size = 0.01

TPFMQA=0.15/0.15/0.15/0.15/0.15/0.15/0.15/0.15/0.15/0.15
1. CUMMD=1656.71

****************   Start of a new cycle   ****************

TPFMQA=0.1/0.15/0.15/0.15/0.15/0.15/0.15/0.15/0.15/0.15
1. CUMMD=1786.84
TPFMQA=0.2/0.15/0.15/0.15/0.15/0.15/0.15/0.15/0.15/0.15
2. CUMMD=1597.04
TPFMQA=0.2/0.1/0.15/0.15/0.15/0.15/0.15/0.15/0.15/0.15
2. CUMMD=1604.8
TPFMQA=0.2/0.2/0.15/0.15/0.15/0.15/0.15/0.15/0.15/0.15
2. CUMMD=1606.44
TPFMQA=0.2/0.15/0.15/0.15/0.15/0.15/0.15/0.15/0.15/0.15
3. CUMMD=1597.04
TPFMQA=0.2/0.15/0.1/0.15/0.15/0.15/0.15/0.15/0.15/0.15
4. CUMMD=1586.96
TPFMQA=0.2/0.15/0.1/0.1/0.15/0.15/0.15/0.15/0.15/0.15
5. CUMMD=1573.78
TPFMQA=0.2/0.15/0.1/0.1/0.1/0.15/0.15/0.15/0.15/0.15
6. CUMMD=1556.6
TPFMQA=0.2/0.15/0.1/0.1/0.1/0.1/0.15/0.15/0.15/0.15
7. CUMMD=1551.05
TPFMQA=0.2/0.15/0.1/0.1/0.1/0.1/0.1/0.15/0.15/0.15
7. CUMMD=1558.14
TPFMQA=0.2/0.15/0.1/0.1/0.1/0.1/0.2/0.15/0.15/0.15
7. CUMMD=1557.33
TPFMQA=0.2/0.15/0.1/0.1/0.1/0.1/0.15/0.15/0.15/0.15
8. CUMMD=1551.05
TPFMQA=0.2/0.15/0.1/0.1/0.1/0.1/0.15/0.1/0.15/0.15
8. CUMMD=1557.77
TPFMQA=0.2/0.15/0.1/0.1/0.1/0.1/0.15/0.2/0.15/0.15
8. CUMMD=1557.66
TPFMQA=0.2/0.15/0.1/0.1/0.1/0.1/0.15/0.15/0.15/0.15
9. CUMMD=1551.05
TPFMQA=0.2/0.15/0.1/0.1/0.1/0.1/0.15/0.15/0.1/0.15
9. CUMMD=1557.69
TPFMQA=0.2/0.15/0.1/0.1/0.1/0.1/0.15/0.15/0.2/0.15
```

```
9. CUMMD=1557.72
TPFMQA=0.2/0.15/0.1/0.1/0.1/0.1/0.15/0.15/0.15/0.15
10. CUMMD=1551.05
TPFMQA=0.2/0.15/0.1/0.1/0.1/0.1/0.15/0.15/0.15/0.1
10. CUMMD=1557.68
TPFMQA=0.2/0.15/0.1/0.1/0.1/0.1/0.15/0.15/0.15/0.2
10. CUMMD=1557.74
TPFMQA=0.2/0.15/0.1/0.1/0.1/0.1/0.15/0.15/0.15/0.15
1. CUMMD=1551.05

********** Pattern Search **********

TPFMQA=0.25/0.15/0.05/0.05/0.05/0.05/0.15/0.15/0.15/0.15
1. CUMMD=1570.05

********** Local Explorations **********


**********   Start of a new cycle   **********

TPFMQA=0.2/0.15/0.05/0.05/0.05/0.05/0.15/0.15/0.15/0.15
1. CUMMD=1579.31
TPFMQA=0.3/0.15/0.05/0.05/0.05/0.05/0.15/0.15/0.15/0.15
2. CUMMD=1560.65
TPFMQA=0.3/0.1/0.05/0.05/0.05/0.05/0.15/0.15/0.15/0.15
2. CUMMD=1585.35
TPFMQA=0.3/0.2/0.05/0.05/0.05/0.05/0.15/0.15/0.15/0.15
3. CUMMD=1545.7
TPFMQA=0.3/0.2/0.02/0.05/0.05/0.05/0.15/0.15/0.15/0.15
3. CUMMD=1554.31
TPFMQA=0.3/0.2/0.1/0.05/0.05/0.05/0.15/0.15/0.15/0.15
4. CUMMD=1537.04
TPFMQA=0.3/0.2/0.1/0.02/0.05/0.05/0.15/0.15/0.15/0.15
4. CUMMD=1541.09
TPFMQA=0.3/0.2/0.1/0.1/0.05/0.05/0.15/0.15/0.15/0.15
5. CUMMD=1529.88
TPFMQA=0.3/0.2/0.1/0.1/0.02/0.05/0.15/0.15/0.15/0.15
5. CUMMD=1531.51
TPFMQA=0.3/0.2/0.1/0.1/0.1/0.05/0.15/0.15/0.15/0.15
5. CUMMD=1540.31
TPFMQA=0.3/0.2/0.1/0.1/0.05/0.05/0.15/0.15/0.15/0.15
6. CUMMD=1529.88
TPFMQA=0.3/0.2/0.1/0.1/0.05/0.02/0.15/0.15/0.15/0.15
6. CUMMD=1530.47
TPFMQA=0.3/0.2/0.1/0.1/0.05/0.1/0.15/0.15/0.15/0.15
6. CUMMD=1535.59
TPFMQA=0.3/0.2/0.1/0.1/0.05/0.05/0.15/0.15/0.15/0.15
7. CUMMD=1529.88
TPFMQA=0.3/0.2/0.1/0.1/0.05/0.05/0.1/0.15/0.15/0.15
7. CUMMD=1530.21
TPFMQA=0.3/0.2/0.1/0.1/0.05/0.05/0.2/0.15/0.15/0.15
```

```
7. CUMMD=1536.25
TPFMQA=0.3/0.2/0.1/0.1/0.05/0.05/0.15/0.15/0.15/0.15
8. CUMMD=1529.88
TPFMQA=0.3/0.2/0.1/0.1/0.05/0.05/0.15/0.1/0.15/0.15
8. CUMMD=1529.95
TPFMQA=0.3/0.2/0.1/0.1/0.05/0.05/0.15/0.2/0.15/0.15
8. CUMMD=1536.42
TPFMQA=0.3/0.2/0.1/0.1/0.05/0.05/0.15/0.15/0.15/0.15
9. CUMMD=1529.88
TPFMQA=0.3/0.2/0.1/0.1/0.05/0.05/0.15/0.15/0.1/0.15
9. CUMMD=1529.89
TPFMQA=0.3/0.2/0.1/0.1/0.05/0.05/0.15/0.15/0.2/0.15
9. CUMMD=1529.91
TPFMQA=0.3/0.2/0.1/0.1/0.05/0.05/0.15/0.15/0.15/0.15
10. CUMMD=1529.88
TPFMQA=0.3/0.2/0.1/0.1/0.05/0.05/0.15/0.15/0.15/0.1
1. CUMMD=1529.88

********** Pattern Search **********

TPFMQA=0.4/0.25/0.1/0.1/0.02/0.02/0.15/0.15/0.15/0.05
1. CUMMD=1524.59

**********   Start of a new cycle   **********

TPFMQA=0.35/0.25/0.1/0.1/0.02/0.02/0.15/0.15/0.15/0.05
2. CUMMD=1516.34
TPFMQA=0.35/0.2/0.1/0.1/0.02/0.02/0.15/0.15/0.15/0.05
2. CUMMD=1528.66
TPFMQA=0.35/0.3/0.1/0.1/0.02/0.02/0.15/0.15/0.15/0.05
2. CUMMD=1530.23
TPFMQA=0.35/0.25/0.1/0.1/0.02/0.02/0.15/0.15/0.15/0.05
3. CUMMD=1516.34
TPFMQA=0.35/0.25/0.05/0.1/0.02/0.02/0.15/0.15/0.15/0.05
3. CUMMD=1524.42
TPFMQA=0.35/0.25/0.15/0.1/0.02/0.02/0.15/0.15/0.15/0.05
3. CUMMD=1528.3
TPFMQA=0.35/0.25/0.1/0.1/0.02/0.02/0.15/0.15/0.15/0.05
4. CUMMD=1516.34
TPFMQA=0.35/0.25/0.1/0.05/0.02/0.02/0.15/0.15/0.15/0.05
4. CUMMD=1516.94
TPFMQA=0.35/0.25/0.1/0.15/0.02/0.02/0.15/0.15/0.15/0.05
4. CUMMD=1534.82
TPFMQA=0.35/0.25/0.1/0.1/0.02/0.02/0.15/0.15/0.15/0.05
5. CUMMD=1516.34
TPFMQA=0.35/0.25/0.1/0.1/0.02/0.02/0.15/0.15/0.15/0.05
6. CUMMD=1516.34
TPFMQA=0.35/0.25/0.1/0.1/0.02/0.02/0.15/0.15/0.15/0.05
7. CUMMD=1516.34
TPFMQA=0.35/0.25/0.1/0.1/0.02/0.02/0.1/0.15/0.15/0.05
8. CUMMD=1510.08
TPFMQA=0.35/0.25/0.1/0.1/0.02/0.02/0.1/0.1/0.15/0.05
```

```
8. CUMMD=1510.13
TPFMQA=0.35/0.25/0.1/0.1/0.02/0.02/0.1/0.2/0.15/0.05
8. CUMMD=1516.53
TPFMQA=0.35/0.25/0.1/0.1/0.02/0.02/0.1/0.15/0.15/0.05
9. CUMMD=1510.08
TPFMQA=0.35/0.25/0.1/0.1/0.02/0.02/0.1/0.15/0.1/0.05
10. CUMMD=1510.08
TPFMQA=0.35/0.25/0.1/0.1/0.02/0.02/0.1/0.15/0.1/0.02
1. CUMMD=1510.08

********** Pattern Search **********

TPFMQA=0.4/0.3/0.1/0.1/0.02/0.02/0.05/0.15/0.05/0.02
1. CUMMD=1505.19

**********   Start of a new cycle   **********

TPFMQA=0.35/0.3/0.1/0.1/0.02/0.02/0.05/0.15/0.05/0.02
2. CUMMD=1497.86
TPFMQA=0.35/0.25/0.1/0.1/0.02/0.02/0.05/0.15/0.05/0.02
2. CUMMD=1510.4
TPFMQA=0.35/0.35/0.1/0.1/0.02/0.02/0.05/0.15/0.05/0.02
2. CUMMD=1510.18
TPFMQA=0.35/0.3/0.1/0.1/0.02/0.02/0.05/0.15/0.05/0.02
3. CUMMD=1497.86
TPFMQA=0.35/0.3/0.05/0.1/0.02/0.02/0.05/0.15/0.05/0.02
3. CUMMD=1511.74
TPFMQA=0.35/0.3/0.15/0.1/0.02/0.02/0.05/0.15/0.05/0.02
3. CUMMD=1510.2
TPFMQA=0.35/0.3/0.1/0.1/0.02/0.02/0.05/0.15/0.05/0.02
4. CUMMD=1497.86
TPFMQA=0.35/0.3/0.1/0.05/0.02/0.02/0.05/0.15/0.05/0.02
4. CUMMD=1504.83
TPFMQA=0.35/0.3/0.1/0.15/0.02/0.02/0.05/0.15/0.05/0.02
4. CUMMD=1509.73
TPFMQA=0.35/0.3/0.1/0.1/0.02/0.02/0.05/0.15/0.05/0.02
5. CUMMD=1497.86
TPFMQA=0.35/0.3/0.1/0.1/0.02/0.02/0.05/0.15/0.05/0.02
6. CUMMD=1497.86
TPFMQA=0.35/0.3/0.1/0.1/0.02/0.02/0.05/0.15/0.05/0.02
7. CUMMD=1497.86
TPFMQA=0.35/0.3/0.1/0.1/0.02/0.02/0.02/0.15/0.05/0.02
7. CUMMD=1498.02
TPFMQA=0.35/0.3/0.1/0.1/0.02/0.02/0.1/0.15/0.05/0.02
7. CUMMD=1504.01
TPFMQA=0.35/0.3/0.1/0.1/0.02/0.02/0.05/0.15/0.05/0.02
8. CUMMD=1497.86
TPFMQA=0.35/0.3/0.1/0.1/0.02/0.02/0.05/0.1/0.05/0.02
8. CUMMD=1497.91
TPFMQA=0.35/0.3/0.1/0.1/0.02/0.02/0.05/0.2/0.05/0.02
8. CUMMD=1504.2
TPFMQA=0.35/0.3/0.1/0.1/0.02/0.02/0.05/0.15/0.05/0.02
```

```
9. CUMMD=1497.86
TPFMQA=0.35/0.3/0.1/0.1/0.02/0.02/0.05/0.15/0.02/0.02
10. CUMMD=1497.86
TPFMQA=0.35/0.3/0.1/0.1/0.02/0.02/0.05/0.15/0.02/0.02
1. CUMMD=1497.86

********** Pattern Search **********

TPFMQA=0.35/0.35/0.1/0.1/0.02/0.02/0.02/0.15/0.02/0.02
1. CUMMD=1497.32

**********   Start of a new cycle   **********

TPFMQA=0.3/0.35/0.1/0.1/0.02/0.02/0.02/0.15/0.02/0.02
2. CUMMD=1489.34
TPFMQA=0.3/0.3/0.1/0.1/0.02/0.02/0.02/0.15/0.02/0.02
2. CUMMD=1508.86
TPFMQA=0.3/0.4/0.1/0.1/0.02/0.02/0.02/0.15/0.02/0.02
2. CUMMD=1506.9
TPFMQA=0.3/0.35/0.1/0.1/0.02/0.02/0.02/0.15/0.02/0.02
3. CUMMD=1489.34
TPFMQA=0.3/0.35/0.05/0.1/0.02/0.02/0.02/0.15/0.02/0.02
3. CUMMD=1502.6
TPFMQA=0.3/0.35/0.15/0.1/0.02/0.02/0.02/0.15/0.02/0.02
3. CUMMD=1502.07
TPFMQA=0.3/0.35/0.1/0.1/0.02/0.02/0.02/0.15/0.02/0.02
4. CUMMD=1489.34
TPFMQA=0.3/0.35/0.1/0.05/0.02/0.02/0.02/0.15/0.02/0.02
4. CUMMD=1496.1
TPFMQA=0.3/0.35/0.1/0.15/0.02/0.02/0.02/0.15/0.02/0.02
4. CUMMD=1507.7
TPFMQA=0.3/0.35/0.1/0.1/0.02/0.02/0.02/0.15/0.02/0.02
5. CUMMD=1489.34
TPFMQA=0.3/0.35/0.1/0.1/0.02/0.02/0.02/0.15/0.02/0.02
6. CUMMD=1489.34
TPFMQA=0.3/0.35/0.1/0.1/0.02/0.02/0.02/0.15/0.02/0.02
7. CUMMD=1489.34
TPFMQA=0.3/0.35/0.1/0.1/0.02/0.02/0.02/0.15/0.02/0.02
8. CUMMD=1489.34
TPFMQA=0.3/0.35/0.1/0.1/0.02/0.02/0.02/0.1/0.02/0.02
8. CUMMD=1489.4
TPFMQA=0.3/0.35/0.1/0.1/0.02/0.02/0.02/0.2/0.02/0.02
8. CUMMD=1495.67
TPFMQA=0.3/0.35/0.1/0.1/0.02/0.02/0.02/0.15/0.02/0.02
9. CUMMD=1489.34
TPFMQA=0.3/0.35/0.1/0.1/0.02/0.02/0.02/0.15/0.02/0.02
10. CUMMD=1489.34
TPFMQA=0.3/0.35/0.1/0.1/0.02/0.02/0.02/0.15/0.02/0.02
1. CUMMD=1489.34
```

```
********** Pattern Search **********

TPFMQA=0.25/0.4/0.1/0.1/0.02/0.02/0.02/0.15/0.02/0.02
1. CUMMD=1498.23

********** Local Explorations **********


**********   Start of a new cycle   **********

TPFMQA=0.2/0.4/0.1/0.1/0.02/0.02/0.02/0.15/0.02/0.02
2. CUMMD=1495.38
TPFMQA=0.2/0.35/0.1/0.1/0.02/0.02/0.02/0.15/0.02/0.02
2. CUMMD=1509.26
TPFMQA=0.2/0.45/0.1/0.1/0.02/0.02/0.02/0.15/0.02/0.02
2. CUMMD=1511.97
TPFMQA=0.2/0.4/0.1/0.1/0.02/0.02/0.02/0.15/0.02/0.02
3. CUMMD=1495.38
TPFMQA=0.2/0.4/0.05/0.1/0.02/0.02/0.02/0.15/0.02/0.02
3. CUMMD=1501.95
TPFMQA=0.2/0.4/0.15/0.1/0.02/0.02/0.02/0.15/0.02/0.02
3. CUMMD=1514.79
TPFMQA=0.2/0.4/0.1/0.1/0.02/0.02/0.02/0.15/0.02/0.02
4. CUMMD=1495.38
TPFMQA=0.2/0.4/0.1/0.05/0.02/0.02/0.02/0.15/0.02/0.02
4. CUMMD=1495.67
TPFMQA=0.2/0.4/0.1/0.15/0.02/0.02/0.02/0.15/0.02/0.02
4. CUMMD=1513.95
TPFMQA=0.2/0.4/0.1/0.1/0.02/0.02/0.02/0.15/0.02/0.02
5. CUMMD=1495.38
TPFMQA=0.2/0.4/0.1/0.1/0.02/0.02/0.02/0.15/0.02/0.02
6. CUMMD=1495.38
TPFMQA=0.2/0.4/0.1/0.1/0.02/0.02/0.02/0.15/0.02/0.02
7. CUMMD=1495.38
TPFMQA=0.2/0.4/0.1/0.1/0.02/0.02/0.02/0.15/0.02/0.02
8. CUMMD=1495.38
TPFMQA=0.2/0.4/0.1/0.1/0.02/0.02/0.02/0.1/0.02/0.02
9. CUMMD=1495.37
TPFMQA=0.2/0.4/0.1/0.1/0.02/0.02/0.02/0.1/0.02/0.02
10. CUMMD=1495.37
TPFMQA=0.2/0.4/0.1/0.1/0.02/0.02/0.02/0.1/0.02/0.02
1. CUMMD=1495.37

**********      Pulse reduction      **********
Pulse size is: 0.025

TPFMQA=0.3/0.35/0.1/0.1/0.02/0.02/0.02/0.15/0.02/0.02
1. CUMMD=1489.34

**********   Start of a new cycle   **********

TPFMQA=0.275/0.35/0.1/0.1/0.02/0.02/0.02/0.15/0.02/0.02
```

```
1. CUMMD=1494.57
TPFMQA=0.325/0.35/0.1/0.1/0.02/0.02/0.02/0.15/0.02/0.02
1. CUMMD=1496.56
TPFMQA=0.3/0.35/0.1/0.1/0.02/0.02/0.02/0.15/0.02/0.02
2. CUMMD=1489.34
TPFMQA=0.3/0.325/0.1/0.1/0.02/0.02/0.02/0.15/0.02/0.02
2. CUMMD=1499.24
TPFMQA=0.3/0.375/0.1/0.1/0.02/0.02/0.02/0.15/0.02/0.02
2. CUMMD=1498.23
TPFMQA=0.3/0.35/0.1/0.1/0.02/0.02/0.02/0.15/0.02/0.02
3. CUMMD=1489.34
TPFMQA=0.3/0.35/0.075/0.1/0.02/0.02/0.02/0.15/0.02/0.02
3. CUMMD=1495.97
TPFMQA=0.3/0.35/0.125/0.1/0.02/0.02/0.02/0.15/0.02/0.02
3. CUMMD=1495.52
TPFMQA=0.3/0.35/0.1/0.1/0.02/0.02/0.02/0.15/0.02/0.02
4. CUMMD=1489.34
TPFMQA=0.3/0.35/0.1/0.075/0.02/0.02/0.02/0.15/0.02/0.02
4. CUMMD=1492.77
TPFMQA=0.3/0.35/0.1/0.125/0.02/0.02/0.02/0.15/0.02/0.02
4. CUMMD=1498.49
TPFMQA=0.3/0.35/0.1/0.1/0.02/0.02/0.02/0.15/0.02/0.02
5. CUMMD=1489.34
TPFMQA=0.3/0.35/0.1/0.1/0.02/0.02/0.02/0.15/0.02/0.02
6. CUMMD=1489.34
TPFMQA=0.3/0.35/0.1/0.1/0.02/0.02/0.02/0.15/0.02/0.02
7. CUMMD=1489.34
TPFMQA=0.3/0.35/0.1/0.1/0.02/0.02/0.02/0.15/0.02/0.02
8. CUMMD=1489.34
TPFMQA=0.3/0.35/0.1/0.1/0.02/0.02/0.02/0.125/0.02/0.02
8. CUMMD=1489.37
TPFMQA=0.3/0.35/0.1/0.1/0.02/0.02/0.02/0.175/0.02/0.02
8. CUMMD=1495.65
TPFMQA=0.3/0.35/0.1/0.1/0.02/0.02/0.02/0.15/0.02/0.02
9. CUMMD=1489.34
TPFMQA=0.3/0.35/0.1/0.1/0.02/0.02/0.02/0.15/0.02/0.02
10. CUMMD=1489.34
TPFMQA=0.3/0.35/0.1/0.1/0.02/0.02/0.02/0.15/0.02/0.02
1. CUMMD=1489.34

********** Pattern Search **********

TPFMQA=0.3/0.35/0.1/0.1/0.02/0.02/0.02/0.15/0.02/0.02
1. CUMMD=1489.34

********** Local Explorations **********


**********   Start of a new cycle   **********

TPFMQA=0.275/0.35/0.1/0.1/0.02/0.02/0.02/0.15/0.02/0.02
1. CUMMD=1494.57
```

```
TPFMQA=0.325/0.35/0.1/0.1/0.02/0.02/0.02/0.15/0.02/0.02
1. CUMMD=1496.56
TPFMQA=0.3/0.35/0.1/0.1/0.02/0.02/0.02/0.15/0.02/0.02
2. CUMMD=1489.34
TPFMQA=0.3/0.325/0.1/0.1/0.02/0.02/0.02/0.15/0.02/0.02
2. CUMMD=1499.24
TPFMQA=0.3/0.375/0.1/0.1/0.02/0.02/0.02/0.15/0.02/0.02
2. CUMMD=1498.23
TPFMQA=0.3/0.35/0.1/0.1/0.02/0.02/0.02/0.15/0.02/0.02
3. CUMMD=1489.34
TPFMQA=0.3/0.35/0.075/0.1/0.02/0.02/0.02/0.15/0.02/0.02
3. CUMMD=1495.97
TPFMQA=0.3/0.35/0.125/0.1/0.02/0.02/0.02/0.15/0.02/0.02
3. CUMMD=1495.52
TPFMQA=0.3/0.35/0.1/0.1/0.02/0.02/0.02/0.15/0.02/0.02
4. CUMMD=1489.34
TPFMQA=0.3/0.35/0.1/0.075/0.02/0.02/0.02/0.15/0.02/0.02
4. CUMMD=1492.77
TPFMQA=0.3/0.35/0.1/0.125/0.02/0.02/0.02/0.15/0.02/0.02
4. CUMMD=1498.49
TPFMQA=0.3/0.35/0.1/0.1/0.02/0.02/0.02/0.15/0.02/0.02
5. CUMMD=1489.34
TPFMQA=0.3/0.35/0.1/0.1/0.02/0.02/0.02/0.15/0.02/0.02
6. CUMMD=1489.34
TPFMQA=0.3/0.35/0.1/0.1/0.02/0.02/0.02/0.15/0.02/0.02
7. CUMMD=1489.34
TPFMQA=0.3/0.35/0.1/0.1/0.02/0.02/0.02/0.15/0.02/0.02
8. CUMMD=1489.34
TPFMQA=0.3/0.35/0.1/0.1/0.02/0.02/0.02/0.125/0.02/0.02
8. CUMMD=1489.37
TPFMQA=0.3/0.35/0.1/0.1/0.02/0.02/0.02/0.175/0.02/0.02
8. CUMMD=1495.65
TPFMQA=0.3/0.35/0.1/0.1/0.02/0.02/0.02/0.15/0.02/0.02
9. CUMMD=1489.34
TPFMQA=0.3/0.35/0.1/0.1/0.02/0.02/0.02/0.15/0.02/0.02
10. CUMMD=1489.34
TPFMQA=0.3/0.35/0.1/0.1/0.02/0.02/0.02/0.15/0.02/0.02
1. CUMMD=1489.34

**********      Pulse reduction      **********
Pulse size is: 0.013

TPFMQA=0.3/0.35/0.1/0.1/0.02/0.02/0.02/0.15/0.02/0.02
1. CUMMD=1489.34

**********   Start of a new cycle   **********

TPFMQA=0.287/0.35/0.1/0.1/0.02/0.02/0.02/0.15/0.02/0.02
1. CUMMD=1492.09
TPFMQA=0.313/0.35/0.1/0.1/0.02/0.02/0.02/0.15/0.02/0.02
1. CUMMD=1492.86
TPFMQA=0.3/0.35/0.1/0.1/0.02/0.02/0.02/0.15/0.02/0.02
```

```
2. CUMMD=1489.34
TPFMQA=0.3/0.337/0.1/0.1/0.02/0.02/0.02/0.15/0.02/0.02
2. CUMMD=1494.53
TPFMQA=0.3/0.363/0.1/0.1/0.02/0.02/0.02/0.15/0.02/0.02
2. CUMMD=1490.42
TPFMQA=0.3/0.35/0.1/0.1/0.02/0.02/0.02/0.15/0.02/0.02
3. CUMMD=1489.34
TPFMQA=0.3/0.35/0.087/0.1/0.02/0.02/0.02/0.15/0.02/0.02
3. CUMMD=1492.78
TPFMQA=0.3/0.35/0.113/0.1/0.02/0.02/0.02/0.15/0.02/0.02
3. CUMMD=1492.24
TPFMQA=0.3/0.35/0.1/0.1/0.02/0.02/0.02/0.15/0.02/0.02
4. CUMMD=1489.34
TPFMQA=0.3/0.35/0.1/0.087/0.02/0.02/0.02/0.15/0.02/0.02
4. CUMMD=1491.14
TPFMQA=0.3/0.35/0.1/0.113/0.02/0.02/0.02/0.15/0.02/0.02
4. CUMMD=1493.83
TPFMQA=0.3/0.35/0.1/0.1/0.02/0.02/0.02/0.15/0.02/0.02
5. CUMMD=1489.34
TPFMQA=0.3/0.35/0.1/0.1/0.02/0.02/0.02/0.15/0.02/0.02
6. CUMMD=1489.34
TPFMQA=0.3/0.35/0.1/0.1/0.02/0.02/0.02/0.15/0.02/0.02
7. CUMMD=1489.34
TPFMQA=0.3/0.35/0.1/0.1/0.02/0.02/0.02/0.15/0.02/0.02
8. CUMMD=1489.34
TPFMQA=0.3/0.35/0.1/0.1/0.02/0.02/0.02/0.137/0.02/0.02
8. CUMMD=1489.36
TPFMQA=0.3/0.35/0.1/0.1/0.02/0.02/0.02/0.163/0.02/0.02
8. CUMMD=1495.67
TPFMQA=0.3/0.35/0.1/0.1/0.02/0.02/0.02/0.15/0.02/0.02
9. CUMMD=1489.34
TPFMQA=0.3/0.35/0.1/0.1/0.02/0.02/0.02/0.15/0.02/0.02
10. CUMMD=1489.34
TPFMQA=0.3/0.35/0.1/0.1/0.02/0.02/0.02/0.15/0.02/0.02
1. CUMMD=1489.34

********** Pattern Search **********

TPFMQA=0.3/0.35/0.1/0.1/0.02/0.02/0.02/0.15/0.02/0.02
1. CUMMD=1489.34

********** Local Explorations **********


**********   Start of a new cycle   **********

TPFMQA=0.287/0.35/0.1/0.1/0.02/0.02/0.02/0.15/0.02/0.02
1. CUMMD=1492.09
TPFMQA=0.313/0.35/0.1/0.1/0.02/0.02/0.02/0.15/0.02/0.02
1. CUMMD=1492.86
TPFMQA=0.3/0.35/0.1/0.1/0.02/0.02/0.02/0.15/0.02/0.02
```

```
2. CUMMD=1489.34
TPFMQA=0.3/0.337/0.1/0.1/0.02/0.02/0.02/0.15/0.02/0.02
2. CUMMD=1494.53
TPFMQA=0.3/0.363/0.1/0.1/0.02/0.02/0.02/0.15/0.02/0.02
2. CUMMD=1490.42
TPFMQA=0.3/0.35/0.1/0.1/0.02/0.02/0.02/0.15/0.02/0.02
3. CUMMD=1489.34
TPFMQA=0.3/0.35/0.087/0.1/0.02/0.02/0.02/0.15/0.02/0.02
3. CUMMD=1492.78
TPFMQA=0.3/0.35/0.113/0.1/0.02/0.02/0.02/0.15/0.02/0.02
3. CUMMD=1492.24
TPFMQA=0.3/0.35/0.1/0.1/0.02/0.02/0.02/0.15/0.02/0.02
4. CUMMD=1489.34
TPFMQA=0.3/0.35/0.1/0.087/0.02/0.02/0.02/0.15/0.02/0.02
4. CUMMD=1491.14
TPFMQA=0.3/0.35/0.1/0.113/0.02/0.02/0.02/0.15/0.02/0.02
4. CUMMD=1493.83
TPFMQA=0.3/0.35/0.1/0.1/0.02/0.02/0.02/0.15/0.02/0.02
5. CUMMD=1489.34
TPFMQA=0.3/0.35/0.1/0.1/0.02/0.02/0.02/0.15/0.02/0.02
6. CUMMD=1489.34
TPFMQA=0.3/0.35/0.1/0.1/0.02/0.02/0.02/0.15/0.02/0.02
7. CUMMD=1489.34
TPFMQA=0.3/0.35/0.1/0.1/0.02/0.02/0.02/0.15/0.02/0.02
8. CUMMD=1489.34
TPFMQA=0.3/0.35/0.1/0.1/0.02/0.02/0.02/0.137/0.02/0.02
8. CUMMD=1489.36
TPFMQA=0.3/0.35/0.1/0.1/0.02/0.02/0.02/0.163/0.02/0.02
8. CUMMD=1495.67
TPFMQA=0.3/0.35/0.1/0.1/0.02/0.02/0.02/0.15/0.02/0.02
9. CUMMD=1489.34
TPFMQA=0.3/0.35/0.1/0.1/0.02/0.02/0.02/0.15/0.02/0.02
10. CUMMD=1489.34
TPFMQA=0.3/0.35/0.1/0.1/0.02/0.02/0.02/0.15/0.02/0.02
1. CUMMD=1489.34

**********      Pulse reduction      **********
Pulse size is: 0.007

TPFMQA=0.3/0.35/0.1/0.1/0.02/0.02/0.02/0.15/0.02/0.02
1. CUMMD=1489.34


The best CUMMD is: 1489.34
```

LIST OF REFERENCES

1. Schlender, B.R., "How to Break the Software Logjam," Fortune, 25 September 1989.

2. Evans, M.W., Marciniak, J., Software Quality Assurance and Management, New York: John Wiley & Sons Inc., 1987.

3. Softcon '89, Managing Software Into the '90's...Acquiring the Competitive Edge, New Jersey: American Defense Preparedness Association, 26 January 1989.

4. Abdel-Hamid, T.K., Liedy, F.H. "An Expert-Simulator for Allocating the Quality Assurance Effort in Software Development," May 1989.

5. Flitman, A.M., Hurrion, R.D. "Linking Discrete-Event Simulation Models with Expert Systems," Journal of Operational Research Society, Vol. 38, No. 8, 1987.

6. Harvey, J.J., "Expert Systems: Present and Future," Computers and People, Vol. 36, Nos. 1-2, January-February 1987.

7. Doukidis, G.I., "An Anthology on the Homology of Simulation With Artificial Intelligence," Journal of Operational Research Society, Vol. 38, No. 8, 1987.

8. O'Keefe, R., "Simulation and Expert Systems," Simulation, Vol. 46, January 1986.

9. Abdel-Hamid, T.K., Madnick, S.E. "Lessons Learned From Modeling the Dynamics of Software Development," Communications of the ACM, Vol. 32, No. 12, December 1989.

10. Abdel-Hamid, T.K., "Investigating the Cost/Schedule Trade-Off in Software Development," IEEE Software, Vol. VII, January 1990.

11. Abdel-Hamid, T.K., Madnick, S.E., Dynamics of Software Project Management. Prentice-Hall, Englewood Cliffs, N.J.: In press, 1990.

12. Liedy, F.H., "Design and Development of an Expert System Based Quality Assurance Model for the Dynamo Model of Software Project Management," Department of Administrative Sciences, Naval Postgraduate School, Monterey, California, March 1986.

13. Wilde, D.J., Optimum Seeking Methods, Prentice-Hall, Englewood Cliffs, N.J., 1964.

14. Kirkpatrick, S., Gelatt, C.D., Vecchi, M.P. Optimization by Simulated Annealing, New York: IBM Research Division, RC 9355, No. 41093, April 1982.

INITIAL DISTRIBUTION LIST

		No. Copies
1.	Defense Technical Information Center Cameron Station Alexandria, Virginia 22304-6145	2
2.	Library, Code 0142 Naval Postgraduate School Monterey, California 93943-5002	2
3.	Prof. Tarek Abdel-Hamid, Code 54Ah Department of Administrative Sciences Naval Postgraduate School Monterey, California 93943-5000	5
4.	Prof. Tung Bui, Code 54Bd Department of Administrative Sciences Naval Postgraduate School Monterey, California 93943-5000	1
5.	CDR T.J. Hoskins Computer Technology (Code 37) Naval Postgraduate School Monterey, California 93943-5000	1
6.	LT Raymond K. Buzzard 1371 Home Avenue Menasha, Wisconsin 54952	2