NAVAL POSTGRADUATE SCHOOL
Monterey, California

AD-A273 193

THESIS

APPLICATION OF A GENETIC ALGORITHM
TO OPTIMIZE QUALITY ASSURANCE IN
SOFTWARE DEVELOPMENT

by

Donald M. Elliott

September 1993

Principal Advisor: Balasubramaniam Ramesh

93-29252

REPORT DOCUMENTATION PAGE

1a Report Security Classification: Unclassified	1b Restrictive Markings
2a Security Classification Authority	3 Distribution/Availability of Report
2b Declassification/Downgrading Schedule	Approved for public release; distribution is unlimited.

4 Performing Organization Report Number(s)	5 Monitoring Organization Report Number(s)

6a Name of Performing Organization	6b Office Symbol	7a Name of Monitoring Organization
Naval Postgraduate School	(if applicable) 37	Naval Postgraduate School

6c Address (city, state, and ZIP code)	7b Address (city, state, and ZIP code)
Monterey CA 93943-5000	Monterey CA 93943-5000

8a Name of Funding/Sponsoring Organization	6b Office Symbol (if applicable)	9 Procurement Instrument Identification Number

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) APPLICATION OF A GENETIC ALGORITHM TO OPTIMIZE QUALITY ASSURANCE IN SOFTWARE DEVELOPMENT

12 Personal Author(s) Donald M. ELLIOTT

13a Type of Report	13b Time Covered	14 Date of Report (year, month, day)	15 Page Count
Master's Thesis	From To	September 1993	86

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	Subgroup	Genetic Algorithm, Software Development, Simulation, Software Quality Assurance, Fitness Measure, Generation

19 Abstract (continue on reverse if necessary and identify by block number)

Quality Assurance is an important aspect of the software development lifecycle. With declining Department of Defense dollars, the development of a Quality Assurance scheme, that minimizes total software development project costs in large scale systems, is extremely valuable. This research aims at developing such a scheme that will provide a staffing profile for Quality Assurance. As there are no analytical solutions available to solve this nonlinear optimization problem and the potential search space is extremely large, a genetic algorithm is used to arrive at an optimal solution. The results indicate that the solution obtained using this approach performs better than several other approaches, such as expert simulators and pattern search techniques, that have been attempted. The scheme is developed using a software project simulation model that incorporates data from an actual software development project. The simulation model allows for the examination of the consequence of staffing profile decisions on total project cost.

20 Distribution/Availability of Abstract	21 Abstract Security Classification
_X_ unclassified/unlimited __ same as report __ DTIC users	Unclassified

22a Name of Responsible Individual	22b Telephone (include Area Code)	22c Office Symbol
Balasubramaniam Ramesh	(408)656-2439	AS/RA

DD FORM 1473, 84 MAR

83 APR edition may be used until exhausted
All other editions are obsolete

Application of a Genetic Algorithm to Optimize
Quality Assurance in Software Development

by

Donald M. Elliott
Major, United States Marine Corps
B.S., Auburn University

Submitted in partial fulfillment
of the requirements for the degree of

MASTER OF SCIENCE IN INFORMATION TECHNOLOGY MANAGEMENT

from the

NAVAL POSTGRADUATE SCHOOL
September 1993

Author: _____
Donald M. Elliott

Approved by: _____
Balasubramaniam Ramesh, Principal Advisor

_____
Tarek K. Abdel-Hamid, Associate Advisor

_____
David R. Whipple, Chairman
Department of Administrative Sciences

ABSTRACT

Quality Assurance is an important aspect of the software development lifecycle. With declining Department of Defense dollars, the development of a Quality Assurance scheme, that minimizes total software development project costs in large scale systems, is extremely valuable. This research aims at developing such a scheme which will provide a staffing profile for Quality Assurance. As there are no analytical solutions available to solve this nonlinear optimization problem and the potential search space of all possible solutions is extremely large, a genetic algorithm is used to arrive at an optimal solution. The results indicate that the solution obtained using this approach performs better than several other approaches, such as expert simulators and pattern search techniques, that have been attempted. The scheme is developed using a software project simulation model that incorporates data from an actual software development project. The simulation model allows for the examination of the consequence of staffing profile decisions on total project cost.

TABLE OF CONTENTS

I. INTRODUCTION

A. BACKGROUND

In the recent years of rapid computer growth, software development has not enjoyed

the same degree of success as hardware development. Large software development

projects have frequently failed to meet expectations and are often completed over

schedule, over budget or both. Furthermore, once the product is delivered it frequently

fails to meet the expectations of the user. What has not occurred during this growth period

is the evolvement of methodologies to adequately manage the software development

process. The software development problem can largely be attributed to this lack of

evolving methodologies in software development. Since software has now passed

hardware as the critical component in the success of many computer systems, better

techniques in the management of software development become imperative. [Ref. 1:p. 3]

This lack of success in software development may be attributed to many reasons and

include the following:

- Lack of corporate understanding of software development thus leading to a lack of commitment to quality software engineering.

- Few individuals with an adequate level of software project management education and experience, thus leading to poor management of software development.

- Inadequate employment of software engineering principles.

- Lack of software quality control (in comparison to hardware).

1

Two of the above shortcomings address the quality issue. Quality assurance, a significant

part of the software development cycle, is clearly an area in need of improvement.

[Ref. 2:p. 10]

While there are many definitions for quality assurance, the following emphasizes the

central themes:

> Quality assurance is a planned and systematic pattern of all actions necessary to provide
> adequate confidence that the item or product conforms to established technical
> requirement. [Ref. 3:p. 5]

The implementation of a sound quality assurance (QA) program is proven to have an

impact on the success of a software development project [Ref. 4:p. 395]. QA embraces

the concept of systematic development to assure that quality is built into the system from

the beginning. Walkthroughs, inspections and code checks are just a few of the many

aspects of QA that occur during the development process and are designed to avoid,

detect and eliminate errors as early in the in the software development lifecycle as

possible. Since the earlier in the life cycle that an error is detected, the less expensive it is

to correct, the economic advantage of a sound quality assurance program is desirable.

[Ref. 4:p. 403]

B. OBJECTIVES

There are two principal reasons for placing a strong emphasis on software quality.

First, the Department of Defense is increasing its dependence on software based systems

and this rate of dependence keeps accelerating. Second, there is an ever increasing cost of

software failure. The failure of software when there are human life-consequences (such as

safety critical applications) necessitates a very high quality product. Economic considerations are not the sole justification for developing quality software and the QA effort necessary to bring it about. [Ref. 2:p. 11]

However, QA does not come without a cost. The introduction of a QA program into the development process requires that personnel, that could be used for other functions, must now perform QA duties. This reallocation of personnel, if not done correctly, could in fact lead to even higher project costs. If too much effort is allocated to QA, the project cost will increase since after a point, a higher level of QA does not produce enough savings to offset the cost of the additional QA effort. The project manager must decide what level of QA is appropriate to minimize total project costs. The purpose of this thesis is to develop a system, that will predict the optimal QA level in the software development process, to minimize total project cost. The system will consist of a dynamic simulation model and a genetic algorithm that will interact with each other to achieve this optimal level. While the system has been developed to minimize total project cost from a QA standpoint, the principles and techniques could equally apply to other aspects of the development process such as staffing decisions for design, rework or coding.

C. THE RESEARCH QUESTION

The primary research question addressed in this thesis is how to compute the optimal QA effort that will minimize total project costs. A dynamic simulation model of software development is used to investigate this question The advantage of using a simulation, with

data from an actual project, is that it serves as a convenient and reliable surrogate to experimenting with various QA schemes on an actual project. A genetic algorithm will be used as the optimization technique as it is considered a suitable tool for nonlinear optimization problems such as this. Therefore, the modified research question is how to compute the optimal QA scheme on a PC using a dynamic simulation model with a genetic algorithm. Other important questions that will be addressed include evaluating the significance of the results that are obtained and evaluating the effectiveness of the genetic algorithm at minimizing the total project cost from a QA standpoint.

D. METHODOLOGY

Simulation is the process of developing a model of reality and then using this model to experiment with different inputs that can be introduced into the model. Simulation implies experimentation to develop better solutions to real problems. If it were feasible to simulate the effects of changing the level of QA effort on total project cost, it is reasonable to conclude that through experimentation it would be possible to discover the optimal level of QA. In this study, a dynamic simulation model will be used that can take as input varying quality assurance schemes, with all other parameters kept constant, and calculate the projected total project cost. The simulation model selected for use is based on several actual software development project case studies and validated on other case studies. This allows the simulation model to precisely characterize several different project management situations and accurately predict the outcome of the simulated management decisions.

The model can simulate different levels of QA and compute the total project cost. There needs to be a method that will propose various QA schemes to be evaluated by the simulation in order to find an optimal solution. This is a nonlinear optimization problem for which there are no known analytical solutions available. The obvious answer would be to run the model with all possible QA schemes and find the best one. However, the search space of all possible QA schemes is extremely large. In this problem, the QA scheme is defined as the % of total effort over 10 phases of a project. Assuming this value in a range from 10% to 64% (54 possible values for each phase), the total number of possibilities is 54^{10} making it a computationally intractable problem. An intelligent search technique to arrive at an optimal solution is needed. The genetic algorithm has been chosen for this purpose.

The genetic algorithm has the characteristics of being able to solve problems involving large search spaces. The genetic algorithm (GA) will present various schemes of quality assurance to the simulation model which in turn computes the total project cost for each such scheme. The GA uses the total project cost associated with each scheme as its fitness or "efficiency". As the GA learns from the fitness of each particular QA scheme presented, it generates potentially better schemes to the simulation model for evaluation.

E. THESIS ORGANIZATION

This chapter discusses the general background of the study and the focus of the research. Chapter II serves to provide an introduction to the GA. Also, an overview of the

GA and simulation model used in the research is provided. Chapter III provides a detailed

account of the interface between the systems that has been developed. Chapter IV

addresses the experiments that are conducted and an analysis of the results. Chapter V will

address a sensitivity analysis of the findings. The purpose of the sensitivity analysis is to

validate the usefulness of the results against additional test cases. Finally, Chapter VI

discusses the conclusions and recommendations for continued research in this area.

II. SYSTEM ARCHITECTURE COMPONENTS

A. THE GENETIC ALGORITHM

1. The Genetic Algorithm Defined

Genetic algorithms are search algorithms that share their functioning with the theory of natural evolution. They employ the principles of survival of the fittest and natural selection and apply these principles to a string of characters. In each new generation that evolves, the algorithms combine bits and pieces from the fittest strings of the last generation to produce a new population of individuals.

The *genetic algorithm* is a highly parallel mathematical algorithm that transforms a set (*population*) of individual mathematical objects (typically fixed length character strings patterned after chromosome strings), each with an associated *fitness* value, into a new population (i.e. the next *generation*) using operations patterned after the Darwinian principle of reproduction and survival of the fittest and after naturally occurring genetic operations (notably sexual recombination). [Ref. 5:p 18]

While the initial population is normally randomly selected, each new generation has the advantage of using past historical data to speculate on new search points. As each new generation evolves, there is an expectation of better performance from that generation than from the previous generation. Genetic algorithms were initially developed by John Holland and his colleagues and students at the University of Michigan in the mid 1970's. [Ref. 6:pp. 1-2]

As a search and optimization tool, the genetic algorithm is proving to be robust and efficient in arriving at solutions in complex search spaces. GA's have proved that they

7

are capable of outperforming gradient techniques and other forms of random search on more difficult problems such as optimization on discontinuous, noisy, high dimensional and multimodal objective functions [Ref. 7:p. 4]. Whereas many techniques require additional information or data to solve the problem, the GA needs no inherent knowledge or information about the problem. The only knowledge required by the GA to perform properly is gained by the GA during its execution. [Ref. 6:p. 9]

The GA has been described as an algorithm that arrives at an "optimal" solution to the function or process to be optimized. A valid question would be what exactly is an "optimal" solution?

Optimization seeks to improve performance toward some optimal point or points. Note that this definition has two parts: (1) we seek improvement to approach some (2) optimal point. There is a clear distinction between the process of improvement and the destination or optimum itself. Yet in judging optimization procedures, we commonly focus solely upon convergence (does the method reach the optimum?) and forget entirely about the interim performance. This emphasis stems from the origins of optimization in the calculus. It is not however a natural emphasis. [Ref. 6:p. 6]

The important goal that optimization seeks to obtain is an improvement in performance. While it would be preferable to obtain the absolute optimum solution, it is not necessarily reasonable. In the following discussion, a distinction will be drawn between the **optimum solution** (the one best solution) and an **optimal solution** (desirable or favorable solution). In complex systems, the cost involved in arriving at the optimum solution may not be justified when an optimal solution exists. Second, it may well be impossible to determine if a solution is the optimum or only an optimal solution. Striving for perfection is a worthy goal but the improvement of the process that can lead to this

8

goal is important as well. The GA possesses the qualities that allows the goal of improvement to be realized. [Ref. 6:p. 7]

2. Generations

The GA evolves from generation to generation by processing a population of strings during each new generation to produce a successive population. Each new generation is produced by using operations that are modeled after the Darwinian principals of reproduction incorporating survival of the fittest techniques [Ref. 5:p. 10]. It is evolutionary in nature and in each generation, several distinct activities occur in a step by step process. These activities include reproduction, crossover and mutation. It is through these processes that the GA is able to arrive at a solution. [Ref. 6:p. 10]

3. Reproduction

The mechanics of a genetic algorithm are not difficult to understand. The first step in the process is to build a string of characters. The string will be representative of some value or condition that can exist (describes the problem variable) and in genetic terms would be the equivalent of a chromosome. To illustrate the mechanics of a genetic algorithm, an example will be used [Ref. 5:pp. 18-30]. This example will be artificially simple so the functioning of the algorithm can be clearly presented. In this example, the aim will be to maximize the number from a string 3 characters in length. As is traditional with the GA, binary representation of these strings will be used. Given these parameters, the strings would have the makeup of binary characters ranging from 000 to 111 and

these strings would assume the numeric values (decimal equivalent) in the range of 0 to 7 and are shown in Table 2-1.

TABLE 2-1

BINARY STRINGS AND VALUES

String	Value	String	Value
0 0 0	0	1 0 0	4
0 0 1	1	1 0 1	5
0 1 0	2	1 1 0	6
0 1 1	3	1 1 1	7

The possible number of strings that can exist in this search space is defined by the equation 2^L, where L is equal to the string length. In our example, this would be a very small number of 2^3 or 8. While it is quite possible to compute all the possible values that a string of this length can have, this is certainly not the case in larger strings. A string of length 50 would have 2^{50} or greater than $1 * 10^{15}$ possible combinations. Even with today's modern computer hardware, it would not be feasible to evaluate all the possible combinations for larger strings. [Ref. 5:p. 18]

A small number of strings are initially randomly generated to produce the makeup of generation 0. In our example, we may assume that four strings will be evaluated in each generation and the initial random selection for generation 0 produced the strings of 001, 011, 100 and 110. To evaluate how well a particular string performs, there must be a fitness measure. In the example the fitness measure will simply be the numeric

value of the binary string. The selected strings and their associated fitness values for generation 0 are displayed in Table 2-2.

TABLE 2-2

INITIAL POPULATION AND FITNESS VALUES

Number	String	Fitness	% of Total
1	001	1	7.1
2	011	3	21.4
3	100	4	28.6
4	110	6	42.9
Total		14	100.0

The highest value is much greater than the lowest. While the greatest value that can be achieved is evident in this example, in larger examples it may not be as easy to determine if a value is close to the best possible. It is however possible to compare the value of each string with that of the other strings. By summing up the fitness of the four strings and computing what percentage to the total is contributed by each individual string, a determination of fitness can be made based on this contribution. As the genetic algorithm functions by improving the population, this information will be used in producing the next generation. A roulette wheel is used to illustrate which strings will survive [Ref. 6:p. 11]. If a weighted roulette wheel was constructed based on the fitness of the four strings, it would appear as shown in Figure 2-1.
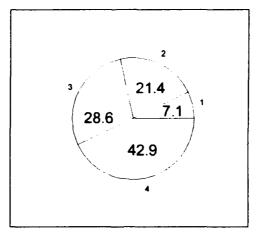
Figure 2-1: Roulette Wheel

Now the object is to generate a new set of strings by spinning the roulette wheel. If the roulette wheel illustrated were to be spun, the probability of any one area being selected is contained within each slice of the pie on the wheel. Since the population is of size 4, it will be necessary to spin the roulette wheel 4 times to select the new population. While the selection of the 4 strings is probabilistic, for this example it could reasonably be expected that slice 2 and slice 3 will each be selected once while slice 4 would be selected twice. The new population would now be composed of the following strings shown in Table 2-3.

TABLE 2-3

POPULATION AFTER REPRODUCTION

Number	String	Fitness	% of Total
1	011	3	7.1
2	100	4	21.4
3	110	6	28.6
4	110	6	42.9
Total		19	100.0

It is observed that as a consequence of reproduction, the overall fitness of the population has increased and the string with the poorest performance has now disappeared. What has not occurred though is the creation of any new strings. Unless it was fortunate enough for the optimal solution to have been generated in the initial random selection of strings, it will never be attained. The probability of that occurrence with larger strings is extremely low. New strings must be generated to arrive at an optimal solution. In order to generate new strings, the genetic algorithm employs the process of crossover. The newly created strings from Table 2-3 will form the mating pool for the crossover procedure. [Ref. 5:p. 23]

4. Crossover

Crossover occurs in a two step process. The first step of the process is the mating of individual strings. During this stage, one string is selected to mate with another string. For crossover to occur, there must be at least 2 parents to contribute to the offspring. The selection of the two parents that will participate in the crossover operation, as with the process of reproduction, is based on their proportionate fitness. The second step of the process is the determination of the point within the string where the crossover procedure will occur. In larger strings it is possible to have more than one crossover point. It is possible for crossover to occur at any position in the string between two bits. The number of available crossover positions is determined by the equation L-1 where L is the string length. In the example, this would be 3-1 or 2 possible crossover positions (between bit 1 and bit 2 or between bit 2 and bit 3). The selection of the exact crossover point is

13

also a random selection. It will be assumed that string 1 was selected to crossover with string 2 between bit positions 1 and 2, and string 3 was selected to crossover with string 4 between bit positions 2 and 3. The strings before crossover would appear as follows with crossover points indicated by a |.

String 1 = 0 | 1 1   String 2 = 1 | 0 0

String 3 = 1 1 | 0   String 4 = 1 1 | 0

Four new strings will now result from the crossover procedure. Those strings, which are indicated with a ('), are shown below:

String 1' = 0 0 0 (bit 1 from string 1, bits 2 and 3 from string 2)

String 2' = 1 1 1 (bit 1 from string 2, bits 2 and 3 from string 1)

String 3' = 1 1 0 (bits 1 and 2 from string 3, bit 3 from string 4)

String 4' = 1 1 0 (bits 1 and 2 from string 4, bit 3 from string 3)

The significance of the crossover procedure can now be examined. The first obvious result is that the maximum value that can be attained from a string with a three character combination has been created in String 2'. From Table 2-1, it can be observed that it is not be possible to achieve a higher value than 7 with a three bit combination. This is shown for illustrative purposes only and does not purport that the optimum level would be achieved in only the second (or any) generation. The important facet to observe with the crossover procedure is that two entirely different strings were created. Since string 3 was selected to crossover with string 4, and these strings are the same, no new strings were created. However when string 1 crossed over with string 2, the resulting strings of 1'

14

and 2' were created. It is with this creation of new strings that the GA is able to search for a solution.

TABLE 2-4

POPULATION AFTER CROSSOVER

Number	String	Fitness	% of Total
1'	000	0	0.0
2'	111	7	36.8
3'	110	6	31.6
4'	110	6	31.6
Total		19	100.0

5. Mutation

The last step that occurs during each generation is the process of mutation. Mutation is introduced into the GA to prevent premature convergence. Premature convergence occurs when the population too quickly becomes inhabited with strings of the same genetic make up. When this happens, the search space that the GA explores can become too limited and the solution that is obtained may well not be an optimal solution [Ref. 6:p. 14]. The frequency at which the mutation operation occurs is governed by the mutation probability. The significance of the mutation operation in GA's is questionable and the operation is used only sparingly [Ref. 5:p. 26].

Mutation is an asexual operation and begins with the random selection of a string from the mating pool. A character within the string is then also randomly selected

for mutation. In binary representation, the mutation of a character simply means to complement it. While the example did not illustrate mutation, a simple example is as follows. If string 2 (1 0 0) was selected for mutation at bit position 3, the result would be string (1 0 1). The mutation operation results in greater genetic diversity in the population by introducing a new individual to the population. [Ref. 5:p. 26]

6. Schema

While the random nature of genetic algorithms (GA) has been mentioned several times, it should not be assumed that the search procedure followed by the GA is completely random. To more clearly understand this concept, a discussion of schema (plural schemata) is necessary. A schema is a comparison template that describes the similarities that exist at certain positions within the string.

> First, we are seeking similarities among strings in a population. Second, we are looking for casual relationships between these similarities and high fitness. In so doing we admit a wealth of new information to help guide the search. [Ref. 6:p. 18]

A schema is a pattern matching device that serves to identify which bits in a string are important in determining the fitness of that string.

A schema is constructed by adding the third character * or don't care symbol to the alphabet. The don't care symbol means that the bit position occupied by a * can be either a 1 or a 0. The addition of this symbol allows for strings to be grouped together. For instance, the *11 string would represent both strings 111 and 011 and the string *1* would represent the subset of strings composed of 010, 110, 011 and 111. What should be observed is that with the concept of schema, it is possible to address the similarities of well

16

defined strings over a finite alphabet. It should be noted that the * character is only a symbol that represents other symbols and is not processed by the GA. [Ref. 6:p. 19]

A schema represents a set of points from the problem's searchspace that have certain similarities. If there is a population of size L and an alphabet of size K, then a schema is identified by the string of length L and the extended alphabet of size K+1. The addition to the alphabet size is due to the addition of the * character. In the example L=3 and K=2. The alphabet size refers to the possible number of different characters that can occupy a bit position which is 0 and 1 in the example. So the number of possible schema in the example problem is $(K+1)^L$ or $(2+1)^3 = 27$ different schemata. When L=3, it is possible to geometrically show the 8 possible strings as the corners of a hypercube of 3 dimensions as seen in Figure 2-2. [Ref. 5:pp. 33-34]
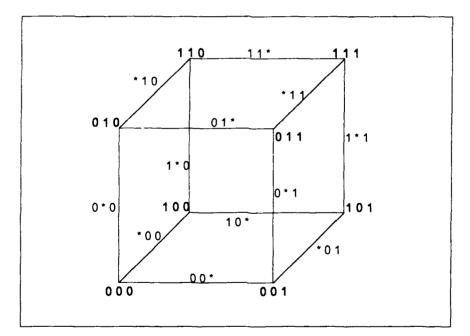
Figure 2-2: Schema Search Space for Three Bit Problem

Figure 2-2 shows the 8 possible strings or schema in bold at the corners of the

hypercube. The edges of the cube between each corner each display one of the 12

schemata that contain one "don't care" position. This accounts for twenty of the schema.

One schema with two "don't care" positions is represented by each plane made up by the

faces of the cube and this accounts for 6 more of the schema. An example of one such

schema is contained in Figure 2-3. The last schema is the * * * schema which for

simplicity reasons is not shown. [Ref. 5:p. 34]
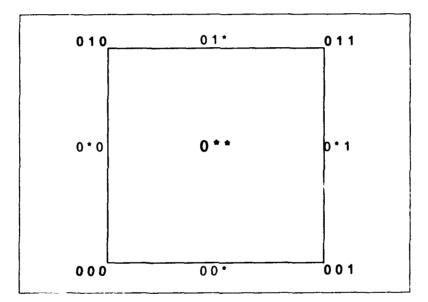
Figure 2-3: One of Six Schemata (0 * *) Represented by Plane

The use of the schema allows for multiple strings to be represented by a single

character string when the "don't care" character is used. During the execution of the GA,

certain strings will prove to be more fit than other strings. The concept of the schema

allows for the GA to exploit the similarities found in fit strings. What has been observed to

occur is that highly fit schema propagate from generation to generation and continually

18

show up in the observed best solutions. What makes this particularly interesting is that all this occurs without any extra processing requirements by the GA. The schema concept provides a mathematical explanation as to why the GA works. As it is beyond the scope of this paper to fully discuss this issue, the reader is referred to Reference 5 pages 30-51, which contains a detailed explanation. [Ref. 6:p. 20]

7. The GAUCSD

The GA program selected to use for this research is the GAUCSD (Genetic Algorithm, University of California, San Diego). This is a GA that was jointly developed by Dr. Nicol N. Schraudolph currently at the Computer Science and Engineering Department of the University of California, San Diego and Dr. John J. Greffenstette who works with the Naval Research Lab in Washington, DC. GAUCSD is a refinement of the Genesis GA that was initially developed by Dr. Greffenstette. It was developed for the purpose of encouraging experimental work with GA's on realistic optimization problems and identify both the strengths and weakness' of GA's. [Ref. 7:p. 1]

There are several GA programs available in public domain. The selection of this GAUCSD was based on several reasons. It is a robust and documented code that comes in a ready to use format. While it was specifically written to function in the UNIX environment, it is easily ported to work in the PC based environment as well. In order to use the code in this research, it is only necessary to insert the fitness measure for the particular problem to be solved into the code. Modifications to the code itself are usually not necessary. While it is important to understand the functioning of the GA, it is possible

to view the code as a black box once it is compiled on the selected platform. Finally, since

the developers of the code are anxious for user feedback, it is possible to obtain answers

to technical questions and functioning through e-mail with relative ease.

B. SYSTEM DYNAMICS MODEL

1. System Dynamics Model of Software Development

Simulation modeling provides a feasible solution to experiment with proposed

solutions to difficult problems.

> In software engineering it is remarkably easy to propose hypothesis and remarkably difficult to test them. Controlled experiments have proven to be too costly and too time consuming. Furthermore, even when affordable, the isolation of the effect an d the evaluation of the impact of any given practice within a large, complex and dynamic project environment can be exceedingly difficult. Accordingly, it is useful to seek other methods of testing software engineering hypothesis. [Ref. 4:p. 396]

This is the premise for the development of the system dynamics model for software

development. In addition to allowing for a less costly and time consuming approach, it is

possible to control the parameters of experimentation. [Ref. 4:p. 396]

A comprehensive system dynamics computer model for software development

has been developed based on this model. The model was developed based on field

interviews of software project managers in five organizations and complemented by an

extensive database of empirical findings from literature. The model integrates the varying

functions of the software development process into four major subsystems. These

subsystems include 1) the human resource management subsystem, 2) the software

production subsystem, 3) the controlling subsystem and 4) the planning subsystem. Figure

20

2-2 depicts these subsystems and the interrelationships that exist between them.

[Ref. 4:pp. 396-397]
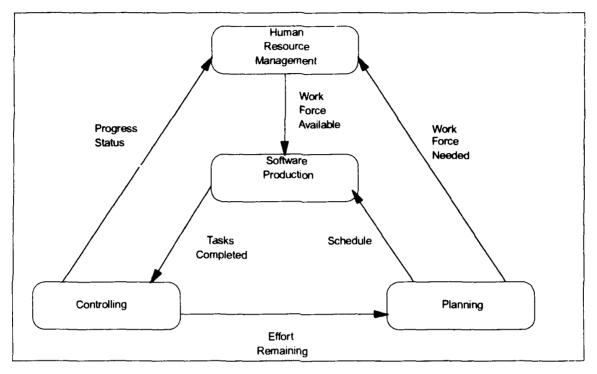
Figure 2-2 Overview of Model Structure

While an in depth discussion of the model is not intended here, a high level

overview will be given on the four subsystems. The human resource management

subsystem reflects the practices of hiring, training and transfer of the human resource. It

segregates the work force into different types of employees such as newly hired or

experienced. The segregation of the work force is done for two reasons. First, it allows for

the reflection of situations where newly added team members are less productive than

those teammembers who have experience on the project. Second, segregation better

allows the model to capture the training process of assimilating the new team members.

21

Since these actions are not carried out in isolation, the actions in one subsystem can affect another. Figure 2-2 depicts how other subsystems are affected by or affect each subsystem. [Ref. 8:p. 1430]

The software production subsystem models the software development process by addressing the designing, coding and testing phases of the software development lifecycle. The requirements, maintenance and operation phases are not included. This is done since these areas are not within control of the development group and it is the goal of the model to reflect the actual decisions, policies and actions of the software development organization. Enhancing productivity is the principal focus of this subsystem. QA is a component of the production subsystem. Through the implementation of QA activities, it is possible to detect errors. Once these errors are detected the components are reworked. Any errors that are not detected by QA activities will normally be corrected during the testing phase. The basic premise of QA is to detect and correct errors as early as possible in the development process. Since these errors can carry through and have an escalation effect, the importance of early detection and correction of errors to minimize costs is clear. [Ref. 4:p. 397]

The control subsystem differentiates between the two types of model variables: actual or perceived. The perceived or apparent conditions may very well be a poor indicator of the actual state. Since software is basically an intangible product during most of its development, it is extremely difficult to measure the exact status of a project during

22

intermediate stages. The purpose of this subsystem is to evaluate many of the system variables and make a determination of the actual project status. [Ref. 8:p. 1431]

The final subsystem to be considered is planning. In this subsystem, initial project estimates in areas such as project completion time and staffing are made through the use of a variety of techniques. These estimates are revised as required during the project's life cycle. Plans may very well be revised as a result of these estimates. These decisions on revising plans are driven by a variety of variables that can change throughout the project lifecycle. [Ref. 8:p. 1431]

2. Quality Assurance

The quality assurance component is part of the software production subsystem. It is one of four major activities that occur in this subsystem and its primary objective is the detection and correction of errors that have been generated. Once code is received from the software development sector, the QA activity uses accepted techniques such as planned group meetings and walkthroughs to detect errors. Once the errors are detected, the rework portion of the activity will make corrections. Since the objective of the QA activity is to detect and correct errors, the success of the QA activity cannot really be judged until the testing phase when the remaining errors are detected. [Ref. 4:p. 401]

The allocation of resources, specifically manpower, to the QA activity is regulated by the variable Planned Fraction of Manpower for QA (TPFMQA). Different schemes may exist to allocate manpower to the QA effort. For lack of a better way, it is often allocated in a uniform manner as seen in Figure 2-3. [Ref. 4:p. 404]

23

PLANNED QA EFFORT
(Percent of Development Man-Days)
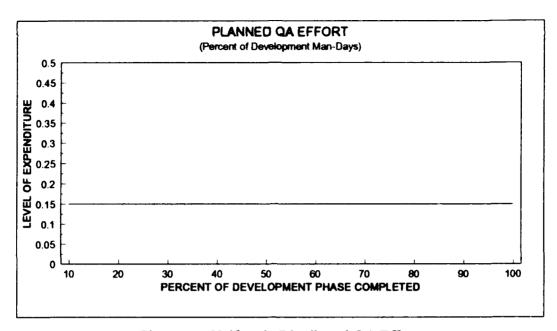
Figure 2-3 Uniformly Distributed QA Effort

As can be seen in Figure 2-3, the QA level is measured at 10 points during the

project development lifecycle. As previously mentioned, the simulation model allocation of

resources to QA is represented by the Planned Fraction of Manpower QA (TPFMQA)

input variable. This variable allows for the input at the 10 points at designated life cycle

completion points starting at 10% and going to 100% in 10% increments. The specific

values assigned to the TPFMQA variable at each 10% interval are percentages of the total

Man-Days available that are designated for the QA effort.

This allocation is most likely not the optimal way to distribute manpower to the

QA activity. When managers lack the tools to determine the optimal level, a scheme like

the one above may well result. With the systems dynamic model, it is possible to introduce

a variety of different schemes to the model for evaluation. As previously discussed in

Chapter I, QA does not come without a cost. If QA is under allocated, too many errors

24

will be left to be corrected when system testing begins resulting in higher costs. If over allocated, too much manpower will be dedicated to the QA effort to identify the last few remaining errors in the system. At a certain point in error detection, it becomes increasingly difficult, time consuming and costly to detect and correct these last few elusive errors and it would be better and more economical to correct these errors during the system testing phase. The system dynamics model allows for experimentation with varying levels of QA to find this optimal level. The DYNAMICA personal computer software program has the capability to conduct these simulations. [Ref. 4:p. 403]

C. THE NASA DE-A SOFTWARE PROJECT

1. Background Of NASA DE-A Software Project

While the use of the dynamic simulation model and specifically the DYNAMICA program is an extremely useful tool, a test platform is needed to experiment with the capabilities of the program. This research uses the results of an in depth case study of the NASA DE-A software development project that was conducted at the Systems Development Section of the Goddard Flight Test Center at Greenbelt, Maryland. In this fortran based project, a program was developed to support spacecraft attitude determination and control. The estimated and actual cost and development times for the project are in Table 2-5. [Ref. 9:pp. 139-140]

The project was completed well above the projected estimates and, though the system was rated as performing very reliably, the success of the project is questioned.

25

TABLE 2-5

DE-A PROJECT STATISTICS

	<u>ESTIMATED</u>	<u>ACTUAL</u>
Project Size (DSI)	16,000	24,000
Development Cost (Man-Days)	1,100	2,200
Completion Time (Days)	320	380

Development cost for the project was twice the estimated cost, the project size in delivered source instructions (DSI) was one and half times the estimate and the project went well beyond the estimated completion time. A likely contributing factor to the overruns and schedule slippage was the 30% average allocation of available resources to the quality assurance effort, which is well above the industry average. The actual QA allocation is shown in Figure 2-3. Excessive QA was intentionally planned into the project
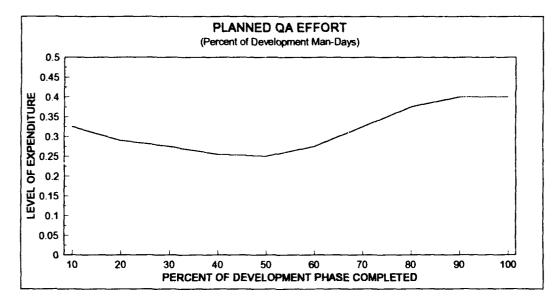
Figure 2-3   QA Distribution for NASA DE-A Project

26

to detect hard to find errors. This experience raises interesting questions on the impact of this high level of QA on the project, and the implications this has to management. [Ref. 4:p. 396]

2. NASA DE-A Software Project Testing

The use of the NASA DE-A software project as a test vehicle can now be seen to be very useful. The specific values assigned to the TPFMQA at each 10% interval in the project is a known quantity. By changing the value of the TPFMQA variables at each interval, it will be possible to determine what effect this new QA scheme would have on the project. The remainder of the inputs to the model will remain constant so that only the TPFMQA variable changes are effecting the outcome. This introduction of varying levels of QA at each of the ten life cycle points allows for the development and evaluation of different QA schemes.

27

III. PREPARATION FOR EXPERIMENTATION

A. OVERVIEW OF SYSTEM

This chapter discusses how the GA and simulation components will function together. The genetic algorithm requires that thousands of evaluations be completed to achieve the desired results. Since each proposed QA scheme is dependent upon the DYNAMICA program for a fitness evaluation on each QA scheme, the interaction between the GA and the DYNAMICA program must be smooth and efficient. The overall design of the system and how each component interacts with the other is displayed in Figure 3-1. For ease of discussion, the complete model will now be referred to as the GASD Model (Genetic Algorithm Software Development).
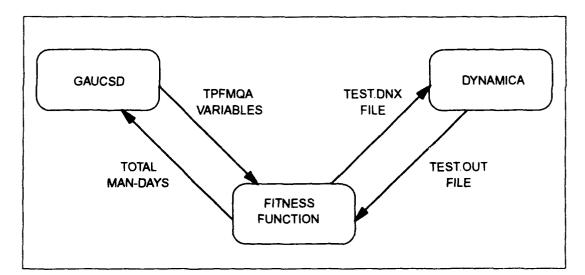
Figure 3-1: System Architecture Overview of GASD Model

28

As seen in Figure 3-1, the GA sends the TPFMQA variables to the fitness function. The fitness function then writes these variables into the TEST.DNX file and this file is sent to the DYNAMICA program for execution. The execution of the program produces the TEST.OUT file containing the Total Man-Days value and this file is sent back to the fitness function. The fitness function then extracts the Total Man-Days value from the TEST.OUT file and sends it to the GA for processing. This procedure occurs for each QA scheme that is produced by the GA and comprises a single trial.

B. EXECUTING THE GAUCSD

1. Required Steps to Execute the GA

There are four major steps involved in using the conventional genetic algorithm to solve a problem employing fixed length strings and these steps are outlined below. [Ref. 5:p. 27]

- Determine the representation scheme.

- Determine the fitness measure.

- Determine the parameters and variables for controlling the algorithm

- Determine the way of designating the result and criterion for terminating a run.

2. Determination of Representation Scheme

The success of the GA in solving a problem can be very dependent upon the representation scheme that is selected. Since the GA processes binary strings, it is necessary to have a representation scheme that allows for the QA values to be defined as a binary string. The representation scheme is not a difficult issue in this problem since the

29

QA variables must be real numbers that range between 0.00 to 1.00 (can have more digits after decimal point). With six binary digits, it is possible to have a representation scheme that would allow for the input of integer numbers in the range of 0 to 63. By taking these numbers and dividing by 100, the range of 0.00 to 0.63 is achieved. Since there are ten points in the life cycle with the required TPFMQA variable, it is necessary to have the string length equal to 10 (# of QA variables) times 6 (# of bits per variable) for a string (chromosome) length of 60. Another option is to use seven digit binary strings that would allow for a range of 0 to 127. As the option of allocating more than 63% of personnel to QA is not reasonable, only six digit strings were used per variable. The return on the QA investment normally flattens out when it exceeds 20-30% of development effort [Ref. 9:p. 204]. While it is not necessary for the GA to have this domain knowledge to function, it will make for a more efficient run when it is available. Caution must be used when introducing domain knowledge to limit the GA search space as it is possible to preclude the GA from looking at potential optimal solutions.

3. Determination of Fitness Measure

The next step in the process is the determination of a fitness measure. Since the intent is to find the most economical allocation of QA, the Total Man-Days variable was selected. The Total Man-Days is sum of the man-days required in the design, code, QA, rework and testing effort. Since the allocation of QA affects other areas , the Total Man-Days figure was selected for evaluation of fitness. The Total Man-Days figure is a direct reflection of the total project cost.

30

4. Controlling the GA

The primary parameters that are used for controlling the GA are the population

size and the total number of generations to be run. Secondary parameters, such as the

frequency of crossover and mutation, can also be used to control the functioning of the

GA. In the case of the GAUCSD, all these parameters are governed in the SAMPLE.IN

file that must be input to the GA at the initiation of each run. An example of a

SAMPLE.IN file is contained in Figure 3-2.

```
Experiments = 1
Total Trials = 100000
Population Size = 1000
Structure Length = 60
Crossover Rate = 0.600000
Mutation Rate = 0.005000
Generation Gap = 1.000000
Scaling Window = -1
Report Interval = 1000
Structures Saved = 10
Max Gens w/o Eval = 2
Dump Interval = 1
Dumps Saved = 1
Options = Aacelru
Random Seed = 3436473682
Maximum Bias = 0.990000
Max Convergence = 30
Conv Threshold = 0.950000
DPE Time Constant = 40
Sigma Scaling = 2.000000
```

Figure 3-2: Example of SAMPLE.IN File

All the parameters in the file will not be addressed since most of them seldom change from run to run but a short explanation of critical control parameters will be given. In GAUCSD, there is no input parameter for the number of generations to be run. Instead, the number of generations is equal to the total trials divided by the population size. From Figure 3-2 this is 100,000/1000 or 100 generations. In this problem, with larger population sizes (greater than 500), it is not necessary to have a large number of generations. Also, for smaller population sizes, it would be required to increase the total number of trials. The question of whether to have large populations and fewer generations or small populations and a greater number of generations is often problem dependent.

More complex problems generally require larger population sizes to solve. These more complex problems are usually the problems which entail exceedingly time-consuming fitness calculations. Thus, the problem of limited computer resources becomes especially acute for these problems because both the population size and the amount of time required to evaluate the fitness is large. [Ref. 5:p. 98]

Initial test runs indicated that population sizes beyond 1000 were not improving the results significantly. Therefore, subsequent tests were performed with a population size of 1000.

The crossover rate was varied between the range of .6 to .9. Higher crossover rates than 1.0 will result in crossing over at more than one point. The literature indicates that in a string length of 60, multiple crossover points are not necessary. While it is subjective, a good rule of thumb on the crossover rate is to keep it between .60 to .70. The mutation rate is kept intentionally low for the reasons discussed in Chapter II. Arriving at the structure length of 60 was previously discussed. Reference 7 pages 14-16 presents a detailed discussion of each of the other variables.

5. Terminating a Run and Designating Results

The final step in executing the GA is to determine the criterion for terminating a run and a method for designating the results. The SAMPLE.IN file contains two primary methods to end a run. The first method is to terminate the run when the total number of trials are completed. The second method involves convergence factors. A convergence threshold of 95% stipulates that the run will be terminated when that percentage of the strings in any generation have converged. Convergence implies that the genetic makeup of the population is not significantly different enough to produce any better results and the only way to introduce variety to the population is through mutation [Ref. 5:p. 104]. It is possible to determine how well the GA is performing by examining the SAMPLE.OUT file during its execution. This file is updated at the end of each generation. The critical columns to examine are the best of generation and the generation average fitness. A partial SAMPLE.OUT file is displayed in Figure 3-3.

Designating the results is dependent on examining the saved results that are created by the GA during its execution. The saved results are stored in the SAMPLE.MIN file. This file stores the best results achieved during the execution of the GA. The number of results that are saved is specified by the variable Structures Saved in the file SAMPLE.IN. In this example, the best 10 results are saved. It is necessary to refer to the SAMPLE.MIN file in order to determine the exact makeup of a QA scheme. The SAMPLE.OUT lists only the minimum result obtained, not the actual GA strings. The

33

SAMPLE.MIN file contain ' ᵗʰ a number and its binary representation for each of the 10

TPFMQA variables in the QA scheme.

gen	trial						Best of Gen	Avg of Gen
0	1000	0	0	-5.510	2.45245e+03	1.61266e+03	1.535830e+03	2.452454e+03
1	2000	0	0	-5.601	2.35133e+03	1.57424e+03	1.534230e+03	2.250203e+03
2	3000	0	0	-5.748	2.26463e+03	1.56091e+03	1.534230e+03	2.091245e+03
3	4000	0	0	-5.899	2.18957e+03	1.55179e+03	1.520460e+03	1.964376e+03
4	5000	0	2	-6.035	2.12599e+03	1.54552e+03	1.520460e+03	1.871696e+03
5	6000	0	5	-6.121	2.07408e+03	1.53616e+03	1.488480e+03	1.814531e+03
6	7000	0	6	-6.234	2.02965e+03	1.52672e+03	1.466890e+03	1.763038e+03
7	8000	0	6	-6.293	1.99271e+03	1.51924e+03	1.466890e+03	1.734103e+03
8	9000	0	8	-6.398	1.95903e+03	1.51343e+03	1.466890e+03	1.689653e+03
9	10000	0	8	-6.556	1.92841e+03	1.50877e+03	1.466890e+03	1.652782e+03
10	11000	0	9	-6.611	1.90212e+03	1.50496e+03	1.466890e+03	1.639259e+03
11	12000	0	9	-6.651	1.87966e+03	1.50178e+03	1.466670e+03	1.632549e+03
12	13000	0	9	-6.688	1.85952e+03	1.49908e+03	1.466670e+03	1.617882e+03
13	14000	0	10	-6.788	1.84045e+03	1.49676e+03	1.466670e+03	1.592478e+03
14	15000	0	11	-6.918	1.82226e+03	1.49466e+03	1.465070e+03	1.567701e+03
15	16000	0	11	-7.033	1.80560e+03	1.49268e+03	1.461090e+03	1.555650e+03

Figure 3-3: Example of SAMPLE.OUT File

As the GAUCSD incorporates a form of gray coding in its programming, translation of the

binary strings into a decimal number is impractical. An example of a portion of a

SAMPLE.MIN file is shown in Figure 3-4.

The first line of each entry is the gray coded binary number followed by the

fitness value of the string in Man-Days, generation that the value was produced and the

specific trial number. The numerical equivalent of the gray coded strings for the ten QA

points are all contained in the second line. Here is a brief description of how the 6 bit

strings in the GAUCSD were mapped to the QA scheme. Using gray coding, GAUCSD

produces 64 values ranging from -0.32 to +0.32 for each variable. The range of the QA

34

variables that are desired in the experiment are initially 0.0 to 0.64 (0% to 64%).

Therefore, in most cases this figure is a negative number. This is a reflection of the

GAUCSD coding as well. It is necessary to add .32 to the values returned by the

GAUCSD to arrive at the desired output.

```
110101 110000 011010 001111 000000 000010 000000 000111 000010 000001   1.4523e+03   25 25526
0.0642981 0.00829779 -0.129446 -0.215234 -0.313446 -0.286263 -0.316608 -0.267231 -0.285762 -0.30931

010110 010101 001101 000010 001100 011101 001111 000110 000000 000010   1.4580e+03   46 46512
-0.0407483 -0.0602402 -0.223317 -0.282795 -0.23693 -0.0914721 -0.216825 -0.270359 -0.313887 -0.287876

110101 110000 011010 001111 000000 000010 000000 000111 000010 000001   1.4536e+03   26 26919
0 0617531 0.00603136 -0.129624 -0.213781 -0.312949 -0.283305 -0.31924 -0.261908 -0.285353 -0.306355

010110 010111 000111 001110 001011 001110 000100 000111 000001 000011   1.4555e+03   40 40072
-0.0486596 -0.0507936 -0.261798 -0.204884 -0.185786 -0.204618 -0.241072 -0.262133 -0.309104 -0.298557

010000 010110 000010 000110 001001 001000 000011 001000 000001 000011   1.4585e+03   47 47079
-0.008099 -0.045853 -0.282085 -0.273652 -0.172593 -0.160513 -0.290294 -0.163346 -0.303555 -0.292155

110101 110000 011010 001111 000000 000010 000000 000111 000010 000001   1.4505e+03   27 2707C
0.0670907 0.00920402 -0.121939 -0.215764 -0.319365 -0.283706 -0.312954 -0.260583 -0.284211 -0.30399

010011 111011 000101 000001 000010 000111 001111 000011 000100 000111   1.4586e+03   17 17228
-0.0237072 0 131148 -0.25851 -0.309917 -0.28588 -0.267313 -0.212606 -0.290767 -0.242238 -0.268387
```

Figure 3-4: Example of SAMPLE.MIN File

The values from the SAMPLE.MIN file are read into a spread sheet so that the

QA scheme could be easily computed (by adding .32) . Spreadsheets also have the

capability to quickly graph the results. An example of a spreadsheet output is contained in

Figure 3-5. Since it would not be unusual to save 50 or more values from each GA run

that are all relatively close to the minimum result obtained, the ability to rapidly graph the

results was very helpful. Graphing the results enables the determination of which set of

35

1460	-0.0640331	-0.0523766	-0.229818	-0.297636	-0.143108	-0.220198	-0.250414	-0.262079	-0.200847	-0.293354
1459.3	-0.0409072	-0.0551557	-0.24383	-0.279118	-0.17307	-0.177435	-0.260492	-0.262063	-0.312823	-0.247226
1459.7	-0.0114129	-0.089575	-0.247427	-0.211139	-0.181665	-0.246257	-0.186999	-0.243678	-0.318236	-0.279074
1455.9	0.00632882	-0.0732518	-0.247822	-0.248901	-0.210987	-0.165083	-0.215224	-0.268166	-0.310823	-0.282778
1458.7	-0.0457895	-0.0529342	-0.242691	-0.273493	-0.179744	-0.17343	-0.263593	-0.262126	-0.317507	-0.240235
1460	0.0425679	-0.0878271	-0.240986	-0.308664	-0.306285	-0.184548	-0.168236	-0.305675	-0.260643	-0.053566
1459.8	-0.0248794	-0.0528819	-0.269087	-0.283874	-0.0957797	-0.2750087	-0.177332	-0.235196	-0.299387	-0.291162
1459	-0.0451357	-0.0553777	-0.24394	-0.270445	-0.170498	-0.172572	-0.265222	-0.265884	-0.311747	-0.247358
1460.1	0.078906	-0.0641142	-0.301697	-0.285297	-0.280396	-0.238233	-0.240256	-0.113323	-0.298859	-0.213928
1460	0.2559669	0.2676234	0.090182	0.022364	0.176892	0.099802	0.069586	0.057921	0.119153	0.026646
1459.3	0.2790928	0.2648443	0.07617	0.040882	0.14693	0.142565	0.059508	0.057937	0.007177	0.072774
1459.7	0.3085871	0.230425	0.072573	0.108861	0.138335	0.073743	0.133001	0.076322	0.001764	0.040926
1455.9	0.32632882	0.2467482	0.072178	0.071099	0.109013	0.154917	0.104776	0.051834	0.009176	0.037222
1458.7	0.27742105	0.2670658	0.077309	0.046507	0.140256	0.14657	0.056407	0.057874	0.002492	0.079765
1460	0.3625679	0.2310856	0.057355	0.095067	0.007093	0.159774	0.119639	0.085341	0.121813	0.029495
1459.3	0.3844221	0.23221729	0.079014	0.011336	0.013715	0.135452	0.151764	0.014325	0.059357	0.2664338
1459.8	0.2951206	0.2671181	0.050913	0.036126	0.2242203	0.044913	0.142668	0.084804	0.020613	0.028838
1459	0.27748643	0.2646223	0.07606	0.049555	0.149502	0.147428	0.054778	0.054116	0.008253	0.072642
1460.1	0.398906	0.25558858	0.018303	0.034703	0.039604	0.081767	0.079744	0.206677	0.021141	0.106072

Figure 3-5: Example of Spreadsheet Conversion

36

values would produce a scheme that is most desirable for implementation reasons. Also, macros could be developed to automate the process. The top portion of Figure 3-5 contains the output produced by the model. The bottom portion of Figure 3-5 contains the numbers after they have been processed in the spreadsheet and had .32 added to them. These figures are used to evaluate the results.

As previously mentioned, it is relatively easy to use a spreadsheet to graph the results and examine the different QA schemes. Since it was found that a particular GA run might produce over 100 results all within 1% of the optimal solution, this becomes a very convenient step. Graphically viewing the QA schemes makes it possible to eliminate many solutions if the scheme has an irregular shape. When selecting a solution, it is desirable to have a relatively smooth curve that does not contain irregularities (sudden increases and decreases). From a managerial standpoint, implementation of irregular shaped schemes will be difficult. Figures 3-6 and 3-7 illustrate this point by displaying two QA schemes with almost identical results for Total Man-Days but significantly different schemes.
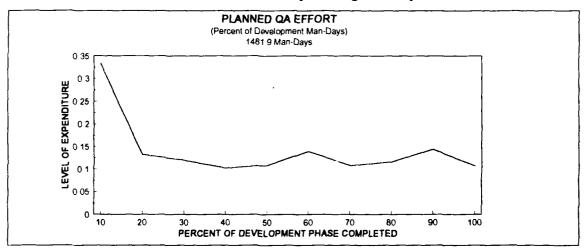
Figure 3-6: Sample QA Scheme

37

PLANNED QA EFFORT
(Percent of Development Man-Days)
1481 4 Man-Days
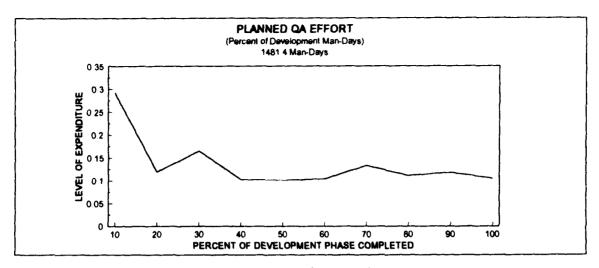
Figure 3-7: Sample QA Scheme

There are two other files created by running the GAUCSD. The checkpoint file,

SAMPLE.CPT, contains all the binary strings that were produced during the last

generation and this file is updated at the completion of each generation. The update writes

over the last generation with the data on all the binary strings from the just completed

generation. The purpose of the checkpoint file this is to allow for the restart of the

program if it is interrupted with the computation loss of only the current generation. The

last file to be discussed is the SAMPLE.LOG file. This file contains the historical

information on when the GA run began, when it was restarted if it was halted and what

generation it converged if it did converge. Figure 3-8 displays how each files is

manipulated by the GA.
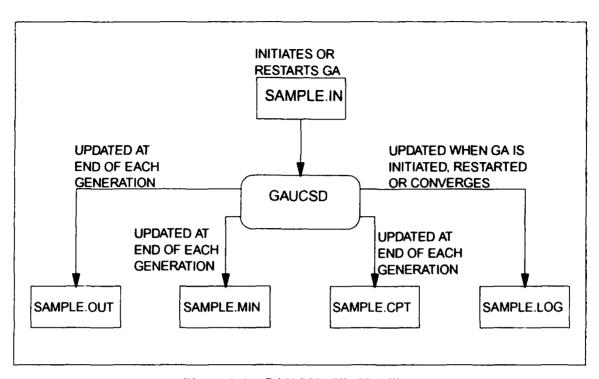
Figure 3-8: GAUCSD File Handling

C. EXECUTING THE DYNAMICA PROGRAM

The DYNAMICA SD (Dynamica Model of Software Development) is a PC based

program that permits the simulation of software project models. The particular version of

the program incorporated into the GASD model is specifically meant to execute the

simulations contained in Reference 9. If it is desired to simulate a model that is structurally

different than those contained in Reference 9, then a more complete version of the

program is required. The DE-A software project is one of the examples incorporated in

the DYNAMICA SD version.

The simulation program can be used in two modes: interactive and batch. In the

interactive mode, the user can utilize an interface that consists of a hierarchy of menus.

39

Each simulation would require the user to make several manual entries on the keyboard to start the program, select the example to simulate, make the desired changes to the model, run the model and produce the desired reports. This is not a difficult task when it is desired to simulate a small number of proposed changes to a project. However, this method is totally infeasible when working with the GA which requires that thousands of simulations be run, each time with a new input.

In the batch mode, it is necessary to create an input file that is read by the program each time it is executed. An example of this file, the TEST.DNX file, is shown in Figure 3-9. It should be observed that one of the parameters contained in the TEST.DNX file is TPFMQA. With the creation of the TEST.DNX file, it is possible to execute the

```
C RJBDSI=24400
C UNDEST=.35
C TOTMD1=1111
C DEVPRT=0.85
T TPFMQA=0.502508 0.593546 0.113032 0.345407 0.361586 0.426484 0.441874
            0.489284 0.418452 0.204185 0
C INUDST=0.4
C TDEV1=320
C MXSCDX=1.16
C ADMPPS=0.5
C HIREDY=30
C AVEMPT=1000
C TRPNHR=0.25
C ASIMDY=20
C DSIPTK=40
T TNERPK=24 22.9 20.75 15.25 13.1 12
```

Figure 3-9: Example of TEST.DNX File

simulation through DOS commands. The required DOS batch commands are shown in Figure 3-10.

```
DYNEX EXAMPLE1 TEST -OUT TEST.DTM -D TEST.DRS


SMLT EXAMPLE1 -GO TEST.RSL -DTM TEST


REP TEST.RSL REPORT.DRS -T
```

Figure 3-10: DYNAMICA DOS Execution Commands

The first line in Figure 3-10 initiates the DYNAMICA program. This command specifies the project to be simulated and the name of the TEST.DNX file that will be used. In this case, the TEST.DNX is shown in Figure 3-10 as the input file with the .DNX assumed and EXAMPLE1 is the DE-A example. The -OUT switch specifies that the file TEST.DTM contains the output and the -D switch redirects the text output to the TEST.DRS file. The second line invokes the simulation of EXAMPLE1. The -GO switch executes the simulation without waiting for user confirmation and the results of the simulation are written to the TEST.RSL file. The -DTM switch is necessary to designate the TEST.DTM file as the .DTM file to be used in the simulation. The final line in Figure 3-10 produces the report containing the information necessary for the GA. The report is given the name TEST.OUT, a sample of which is contained in Figure 3-11.

41

```
PROJECT STATISTICS:
_____

COMPLETION TIME          320.00              DAYS

TOTAL MAN-DAYS           1,477.53            MAN-DAYS

    TOTAL DEV'T MD             1,348.69      MAN-DAYS

        DESIGN & CODE              988.74   MAN-DAYS
        QA MD                      166.36   MAN-DAYS
        REWORK MD                  193.59   MAN-DAYS

    TOTAL TESTING MD           128.84        MAN-DAYS

OVERALL-PRODUCTIVITY     16.24               DSI/MAN-DAYS

TOTAL ERRORS             490.00              ERRORS

% ERRORS DETECTED BY QA  52.05               PERCENT
```

Figure 3-11: Example of TEST.OUT File

D. ROLE OF THE FITNESS FUNCTION

The final component of the GASD model is the fitness function. While the fitness

function is addressed as a sperate component, it is in fact embedded in the GAUCSD

when the GA is compiled. The fitness function is written in the C programming language

and is contained in the Appendix. The fitness function serves three principal duties. The

first function is to take the TPFMQA variables provided by the GAUCSD and create the

TEST.DNX file like the one seen in Figure 3-9. The second function is to call and execute

the DYNAMICA program. The command instructions in Figure 3-10 necessary to

execute the program are contained in the fitness function coding. Finally, after the

DYNAMICA SD program is executed and the TEST.OUT file is created like that in

42

Figure 3-11, the fitness function opens the file and retrieves the Total-Man Days value and returns this value to the GAUCSD.

E. MODEL EXECUTION

1. Hardware and Software Requirements

Given the understanding of how all the components of the GASD model fit together and the role that each component fulfills, the steps necessary to execute a GA run will be covered. The hardware platform that was used for the GA experimentation was a 486 CPU personal computer set up in a network configuration. While it is possible to run the GASD model on less capable machines, the time required to execute a run would increase substantially. Software requirements, of course include the GAUCSD and the DYNAMICA program as well as a C compiler.

2. Compiling the Program

As discussed in Chapter II, the GAUCSD was developed to function in the UNIX environment but can easily be ported to function on any system that has a C compiler. The make and awk utilities commonly used in the UNIX environment are also required. Both of these utilities are available as source distributions free of charge for most types of machines. These utilities must have the specific names mentioned for the program to compile since they are called from deep within the GAUCSD package. Four directories must be created on the computer to hold the required code. The selection of directory

names is discretionary. The names of the directories selected for use in the GASD model are GAWK, SRC, ETC and USR.

The first directory is the GAWK directory and will contain the gawk utility (GNU awk for DOS) previously mentioned. The SRC directory contains all the GAUCSD source files as well as the DYNAMICA program. The USR directory will contain the fitness function. The fitness function must have a name with a .C extension. The fitness function has the name of GOLD.C in the GASD model. The ETC directory must contain the wrapper necessary to embed the fitness function in the GAUCSD. This script takes the fitness function (GOLD.C) and creates a wrapped function (GOLD_GA.C) that is in a format used by the GAUCSD. The wrapper is included in GAUCSD as an awk script. Once the directories are created and the appropriate files are inserted into the directories, it is possible to compile the program. The compiling procedure is listed in the following steps.

Step 1. Change directory location to C: (top directory)

Step 2. Issue command C: GAWK\GAWK -f ..\ETC\WRAPPER GOLD.C > GOLD_GA.C

Step 3. Change directory to SRC:

Step 4. Call C compiler from location where compiler is installed

Step 5. Compile program

A final note on compiling the program is to pay particular attention to the memory specifications of the program. Most compilers have ranges that allow the user to specify how much memory that the compiled program will be able to access in RAM and how

44

much storage space will be permitted on the hard drive. Default settings may be much

smaller than the memory space required to execute the GASD model. This smaller

memory space may have severe limits on the population size and the number of minimum

results that can be saved in the SAMPLE.MIN file.

3. Executing the Program

Once the program is compiled, it is ready for execution. The command

execution for the model is:

GAUCSD F:\TEST\SAMPLE

where F:\TEST\SAMPLE.IN is the initialization file (see Figure 3-2 page 31). The GA

will create and write to the SAMPLE.MIN, SAMPLE.OUT, SAMPLE.LOG and

SAMPLE.CPT files in the same directory where SAMPLE.IN is located.

The F:\ directory was located in a network configuration. Since the GA is being

executed on a PC, it is not possible to monitor the progress of the GA from that PC

during its execution. As the intermediate results of the GA written to output files are

accessible from a network, the use of a network offers the significant advantage of

allowing the GA to be monitored from any other computer on the network during its

execution. It is also possible to have multiple runs going simultaneously utilizing a

network by using different SAMPLE.IN files from different directories. The conditions for

ending a GA run and evaluation of results has already been discussed in pages 33-38.

IV. EXPERIMENTATION AND RESULTS

A. INITIAL TESTING

1. GA Observations

Initial testing done with the GASD model quickly demonstrated that the implementation was successful and that the GA was performing well in developing a QA scheme that minimized total project cost. No constraints were placed on the GA and no domain knowledge was introduced into the coding that would allow for the GA to arrive at a solution faster. The initial runs were conducted with populations of size 1000 and set to run for 100 generations. A total of eight initial runs were conducted.

It was not necessary to run the simulation for a full 100 generations as all eight tests converged between generation 47 and generation 58. In these runs the convergence factors were set at Max Convergence=30, Conv Threshold=.95 and Max Gens w\o eval=2 just as seen in Figure 3-2, page 31. When the GAUCSD was forced to run longer with Max Convergence=100, Conv Threshold=.99 and Max Gens w\o eval=0 (disabled), there was no improvement in the performance of the GA in arriving at a solution. The convergence constraints were useful as they minimized the computer time necessary to execute a run. In the initial simulations it was possible to run approximately 720 trials per hour which resulted in a new generation approximately every one and one third hour. At this level of effort, it required 3 days of computer time to simulate a run of population size

1000 for 54 generations. Allowing the GA to stop execution due to convergence proved to save significant computer processing time without any loss of optimization to the solution.

Crossover rates were also observed to determine if any improvement in performance could be achieved. The rule of thumb is to use crossover rates in the 60%-70% range. Crossover rates were selected at .6, .7, .8, and .9. Two runs of the eight were allocated to each crossover rate. Examination of the results of the eight runs failed to show any conclusive results about which crossover rate was most productive. For that reason, all subsequent testing was conducted using all four crossover rates.

2. Development of QA Scheme

The GASD model produced a large number of solutions that were better when compared to other methods. Since the DE-A is a well-studied project, it is possible to draw comparisons with other solutions that have been developed to this problem. A comparison of the solution arrived at by the GA with other experiments is contained in Table 4-1 [Ref. 11:p. 73]. The manually derived method involved the user input of QA schemes into the DYNAMICA simulation with manual perturbations introduced. The prototype expert simulator used an expert system module incorporating heuristic rules that is interfaced with the DYNAMICA simulation. The pattern search expert simulator is a refinement of the prototype expert simulator that identifies patterns to make further refinements in improvement. [Ref. 10:pp. 17-20,37]

TABLE 4-1

COMPARISON OF DIFFERENT METHODS

	Total Cost (Man-Days)
Actual DE-A Project	2,200
Manually Derived	1.524.5
Prototype Expert Simulator	1,521.07
Pattern Search Expert Simulator	1,489.34
GAUCSD	1,437.8

These results indicate that the GASD model clearly outperformed the other methods designed to solve the problem. The best results produced by the GASD model demonstrate that it is able to reach an **optimal** solution that is sought. The goal of improving performance has clearly been achieved in this case. The goal of reaching the one global **optimum** solution for this problem has probably not been attained. However, the GASD model has produced a significantly better result than was obtained with the other heuristic, expert knowledge and pattern search techniques. As there is no analytical model available for the problem, it is impossible to predict whether there even exists a global optimum.

It is possible to draw some interesting conclusions by examining the QA scheme if Figure 4-1 both on the performance of the GA and on the significance of the derived scheme. The decision to limit the QA effort to no greater than 64% proved to be reasonable. The highest level obtained was only 59.8%. This proved to be a higher level

than expected and emphasizes the point that caution must be used when domain

knowledge is introduced to constrain the search space.

PLANNED QA EFFORT
(Percent of Development Man-Days)
1437.8 Man-Days

10	20	30	40	50	60	70	80	90	100
0.5984	0.2764	0.0987	0.0079	0.0133	0.0352	0.0102	0.0306	0.0139	0.0189
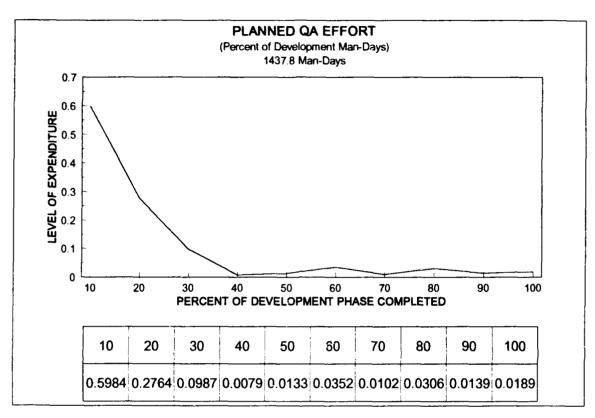
Figure 4-1: QA Effort for Initial GA Run

The shape of the curve shows that there is a great deal of emphasis on QA in

the initial development phases but that the requirement for QA drops very quickly after

30% of the project has been completed. In fact, the amount of effort allocated to QA is

less than 1% at the 40 % point of development. The question arises if it is reasonable to

lower the QA effort to less than 1% of the effort and still have a valid QA program. To

lower the QA effort to less than 1% of the effort in essence terminates the QA function

and the personnel assigned to it for the specified time period when the effort is that small.

49

When the effort rises to the 3% level at the 60 % development phase completion point, the QA effort would have to be reintroduced.

This is not at all reasonable for two significant reasons. The first is that it is not possible to eliminate the QA effort at a point during the development phase then reintroduce at a later time. As discussed in Chapter I, QA is a planned and systematic effort to detect and correct errors early as possible in the development phase. If the QA effort is eliminated, there may well be an explosion rate on the number of errors that are generated and not detected during development. The result would be a significantly greater amount of effort expended in testing due to this higher rate of error generation. The second reason is the effect this action would have on the personnel working on the project. If they see that the QA effort has been eliminated, there will be the perception that QA is no longer a priority and the emphasis for a quality product may be lost. Furthermore, quality personnel would not want to be assigned to QA if they sensed that the QA effort was not continuous. Therefore, even if the QA scheme produced by the GASD model is an optimal one, it may not be enforceable. So the GA did produce a scheme that minimized the total project cost but it did not produce a QA scheme that could be implemented.

It should be noted that the GA was not provided with any knowledge about the implementation of plans, and therefore did not rule out this solution. An interesting way to incorporate this type of knowledge would be to include a penalty in the computation of fitness of individuals, penalizing "unimplementable" QA schemes. This would eliminate

such schemes from the solution set as it evolves. With the initial testing complete and the functioning of the GASD model confirmed, it was possible to focus efforts on deriving an implementable QA plan.

B. TESTING WITH CONSTRAINTS

1. Invoking the Constraint

Since it was determined that an implementable QA scheme could not be developed with the model without further refinement, it was necessary to constrain the GA to obtain a more desirable solution. The problem with the initial solution was that it placed too little effort into the QA effort after 30% of project development was complete. An implementable solution would be one that did not allow the QA effort to drop to such a low level that it was no longer possible to maintain a viable QA program. The selection of a minimum baseline is difficult to determine. Some projects have reported expending an average as low as 6% of resources on QA while other projects have expended as high as 25% of resources on QA. The selection of a minimum baseline of 10% expenditure of resources on QA was the figure deemed most appropriate to maintain a vibrant QA program [Ref. 9:p. 71]. The QA scheme was constrained to a range between 10% and 74% with this limitation.

The implementation of the constraint in the GAUCSD proved easy. The modification of the code would be in the fitness function contained in the Appendix. The initial range for the QA variables in the GA was from 0.00 to 0.64. As previously

discussed and shown in Figure 3-5 on page 36, it was necessary to add .32 to the GASD model generated numbers to keep the values in the 0% to 64% range. Imposing the constraint on the GA meant that the range the GA would be allowed to search would be from 0.10 to 0.74. The code in the Appendix on page 72 displays where the constraint (.42) is added. The only change to the code necessary was changing the .32 to a .42 and the constraint was imposed. It was of course necessary to recompile the program before execution.

2. GA Runs with Constraints

With the 10% minimum QA effort constraint on the GA, four runs were conducted. The crossover rates were set at .6, .7, .8, and .9 as previously discussed, but with a larger population size of 2000 vice the 1000 of the previous runs. It was hoped that the larger population size would bring about an optimal answer sooner by reducing the number of generations required to arrive at a solution. The inclusion of the constraint resulted in a significant change in results obtained. The first result was that each individual trial took an average of 1 to 1.5 seconds to execute. While this may seem trivial, this means an additional 14 hours of run time every 100,000 trials.

The decision to expand the population size to 2000 to reduce run time was not supported. In fact, it is arguable that 2000 was too large a population. By introducing a population size of 2000, each generation required approximately 3 hours of run time to complete a single generation. The results indicated that these larger population sizes did not aid in reducing the number of generations required to reach an optimal solution. As a

52

result, these runs with population size 2000 were required to execute for six days before they converged in comparison to the three days for population size 1000.

The results obtained from these runs were not quite as good as those obtained in the initial tests, nor were they expected to be. Even with the constraints imposed, the resulting solutions were significantly lower than those previously found (see Table 4-1). A total of 200 results were graphed and examined in the spread sheet to select the QA scheme with the smoothest curve that would allow for the easiest implementation. The scheme with the smoothest curve was not the one with the minimal value. In fact, there were several results with a lower value than the scheme selected as the best. Figures 4-2 and 4-3 illustrate some of the lowest values that were obtained.

Many of the results had characteristics similar to those shown. The purpose of displaying the schemes in Figures 4-2 and 4-3 is to show the variety of solutions that are possible with the GA. The top graph in Figure 4-3 illustrates the minimum value that was obtained. The bottom graph displays a change that was found in several solutions. At the 90% of development phase completed, a spike was observed to occur that was 3% to 5% higher than surrounding points. The top graph in Figure 4-2 displays where another spike was often observed at the 70% of development phase completed. This spike was usually only 3% to 4%. There were a small number of solutions that contained both spikes. The bottom graph in Figure 4-3 shows a significantly higher amount of QA going into the initial stage of development than was seen in most solutions. All four solutions are within 4 development Man-Days of each other but offer significantly different solutions.

PLANNED QA EFFORT
(Percent of Development Man-Days)
1473.5 (Minimum Value Found)

10	20	30	40	50	60	70	80	90	100
0.2577	0.1695	0.1383	0.1019	0.1004	0.1163	0.1051	0.1139	0.1110	0.1056

PLANNED QA EFFORT
(Percent of Development Man-Days)
1477.2 Man-Days

10	20	30	40	50	60	70	80	90	100
0.2520	0.1755	0.1163	0.1192	0.1005	0.1027	0.1009	0.1034	0.1680	0.1117
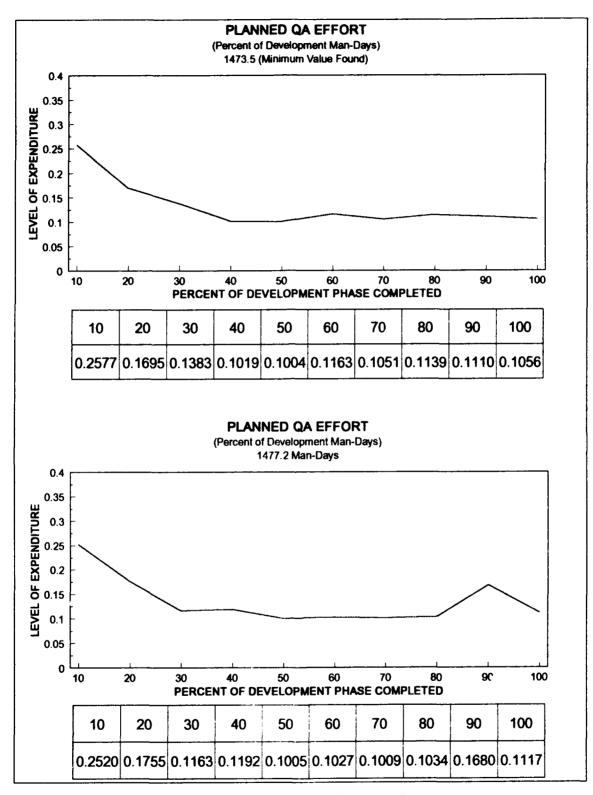
Figure 4-2: QA Effort With Constraint

PLANNED QA EFFORT
(Percent of Development Man-Days)
1476.4 Man-Days

10	20	30	40	50	60	70	80	90	100
0.2714	0.1729	0.1222	0.1050	0.1072	0.1097	0.1391	0.1062	0.1107	0.1069

PLANNED QA EFFORT
(Percent of Development Man-Days)
1477.4 Man-Days

10	20	30	40	50	60	70	80	90	100
0.3444	0.1512	0.1018	0.1035	0.1014	0.1001	0.1078	0.1228	0.1343	0.1110
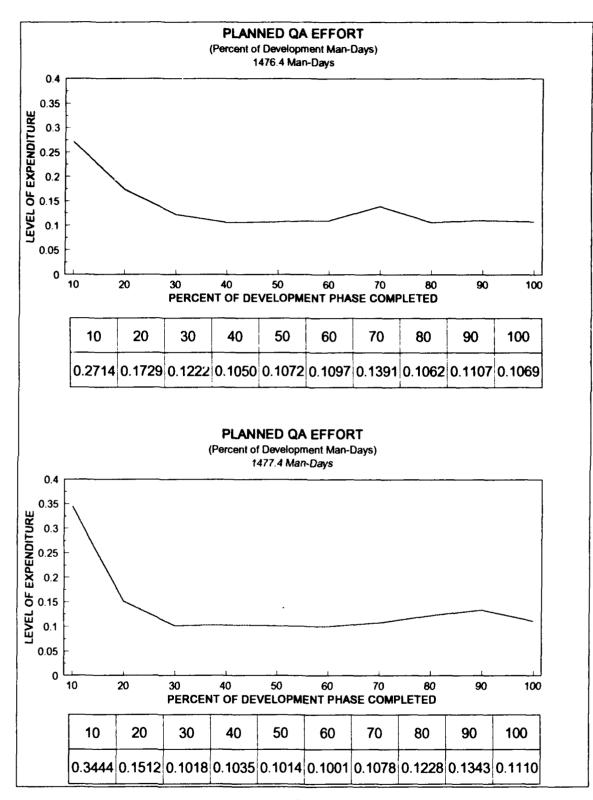
Figure 4-3: QA Effort With Constraint

55

After observing all the available solutions, one QA scheme was selected as the one with the best characteristics. The scheme selected as the best had some distinctive characteristics but it was not the one with the lowest cost. The selected scheme offered a curve that contained no sharp jumps or irregularities that would make implementation difficult with only a 0.3% higher cost than the best result obtained with constraint and 2.5% higher cost of the best result found without constraint. The curve gradually declined over three time periods to a level just above 10% of effort and remained there with only minimal fluctuation. The initial allocation of QA effort was only slightly above 25% and this initial effort was one of the lowest initial values found. This allows for the QA effort to decline over the next two time periods at a much lower rate than observed in most of the solutions which allows for a more gradual transition. The selected optimal QA scheme is contained in Figure 4-4.

The results of the experimentation resulted in some clear observations. The most significant is that the solution places an emphasis on QA early on the development life cycle. First, the early detection and correction of errors helps alleviate the effect of errors committed early from carrying through and resulting in additional errors. Second, the solution results in a reduction in the cost of locating and redesigning errors during the testing phase. Early detection and correction of errors are seen as the critical aspect of QA in reducing cost.

PLANNED QA EFFORT
(Percent of Development Man-Days)
1475.5 Man-Days

10	20	30	40	50	60	70	80	90	100
0.2546	0.1684	0.1319	0.1083	0.1035	0.1123	0.1036	0.1120	0.1167	0.1035
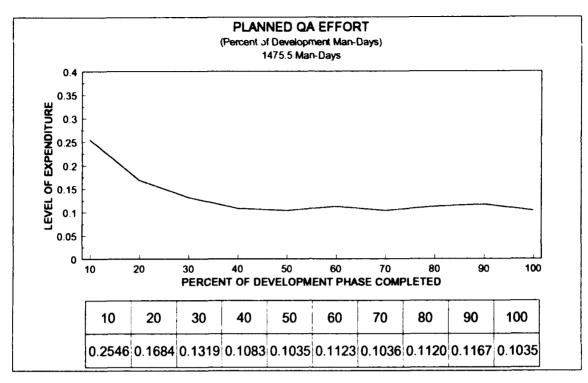
Figure 4-4: Selected Optimal QA Scheme

V. SENSITIVITY ANALYSIS

A. INTRODUCTION

To further evaluate the performance of the GA, two sensitivity analysis experiments were performed. These experiments still used the DE-A project but values of some selected variables in the in the project were permanently changed. Each experiment focused on a particular area with the intent of defining the significance these changes had on the generated QA scheme. The first experiment addresses the consequence of making testing less costly. The second experiment investigates the results of having the number of errors committed per task increase as the project develops.

To conduct this experimentation, it is necessary to recompile the DE-A example. This procedure involves utilizing a more complete version of the DYNAMICA program to make the desired permanent changes. Once the changes are made, the menu selection in the software allows for the program to be recompiled. To perform this procedure, the DYNAMICA PD Plus software version is required. Once the examples are compiled in the PD Plus version, they may be ran in the SD version.

The settings for the GA remained the same as in the previous tests. Both test cases were conducted using four GA runs for each test case and with crossover rates of .6, .7, .8, and .9. The populations sizes remained at 1000. The only discernible difference in the test cases was that a 50% longer runtime was required due to DYNAMICA's coding.

58

B. TEST CASE 1

The first test focused on exploring the results of reducing the cost associated with testing and increasing the quality assurance effort needed to detect errors. Testing serves the function of detecting and then correcting errors that were not detected by the QA effort during the development phase. If testing becomes a less expensive task and quality assurance becomes more costly, it should be expected that less effort would be placed into QA with the expectation of correcting more errors during the testing phase.

The following variables were changed to conduct the test.

- TMPNPE: Testing Manpower Needed Per Error (Man-Days/Error)

- TSTOVH: Testing Effort Overhead (Man-Days/KDSI)

- TNQAPE: Nominal QA Manpower Needed to detect Avg. Error (Man-Days/Error)

Table 5-1 contains the values of these variables for the DE-A project and what value they were change to for test case 1.

TABLE 5-1

TEST CASE 1 CHANGED VARIABLES

DE-A PROJECT										
TMPNPE			.15							
TSTOVH			1.0							
TNQAPE	4	.4	.39	.375	.35	.3	.25	.225	.21	.2	.2
TEST CASE 1										
TMPNPE			.075							
TSTOVH			.5							
TNQAPE	.6	.6	.585	.5625	.525	.45	.375	.3375	.315	.3	.3

Table 5-1 indicates that the testing variables TMPNPE and TSTOVH have both been decreased by 50% for the test case. The QA variable TNQAPE has been increased by 50%. Testing is now less expensive and the QA effort costs more. This should induce the expected results of greater emphasis on testing and less emphasis on QA for correcting errors. Table 5-2 contains the TEST.OUT file produced during initial testing with constraints and the TEST.OUT file produced with the new variables for test case 1.

TABLE 5-2

PROJECT STATISTICS FOR DE-A AND TEST CASE 1

PROJECT STATISTICS FOR DE-A EXAMPLE

COMPLETION TIME	320.00	DAYS
TOTAL MAN-DAYS	1,475.53	MAN-DAYS
TOTAL DEV'T MD	1,354.03	MAN-DAYS
DESIGN & CODE	976.11	MAN-DAYS
QA MD	167.99	MAN-DAYS
REWORK MD	209.93	MAN-DAYS
TOTAL TESTING MD	121.50	MAN-DAYS
OVERALL-PRODUCTIVITY	16.27	DSI/MAN-DAYS
TOTAL ERRORS	491.00	ERRORS
% ERRORS DETECTED BY QA	56.11	PERCENT

PROJECT STATISTICS FOR TEST CASE 1

COMPLETION TIME	332.00	DAYS
TOTAL MAN-DAYS	1,455.49	MAN-DAYS
TOTAL DEV'T MD	1,320.57	MAN-DAYS
DESIGN & CODE	925.04	MAN-DAYS
QA MD	241.90	MAN-DAYS
REWORK MD	153.63	MAN-DAYS
TOTAL TESTING MD	134.92	MAN-DAYS
OVERALL-PRODUCTIVITY	16.49	DSI/MAN-DAYS
TOTAL ERRORS	491.00	ERRORS
% ERRORS DETECTED BY QA	40.60	PERCENT

As expected for the test case, a greater emphasis was placed on correcting errors

during the testing phase as can be seen in Figure 5-1. Since QA became a more costly

option, it became necessary to allocate more effort to QA to achieve optimal results.

There was a significant change in the effort allocated to rework during the development

phase as more errors were corrected during testing. The newly developed QA scheme for

the test case is contained in Figure 5-2 along with a comparison of this scheme to the

previously developed scheme.
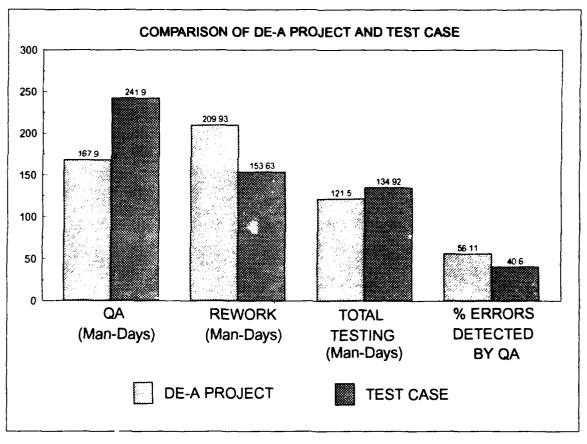
Figure 5-1: Test Case 1 Statistical Comparison With DE-A

PLANNED QA EFFORT FOR TEST CASE
(Percent of Development Man-Days)
1455.5 Man-Days

10	20	30	40	50	60	70	80	90	100
0.4984	0.3207	0.1054	0.1005	0.1023	0.1024	0.1032	0.1094	0.1036	0.1004

COMPARISON OF PLANNED QA EFFORT BETWEEN
OPTIMAL BASE CASE AND TEST CASE
(Percent of Development Man-Days)
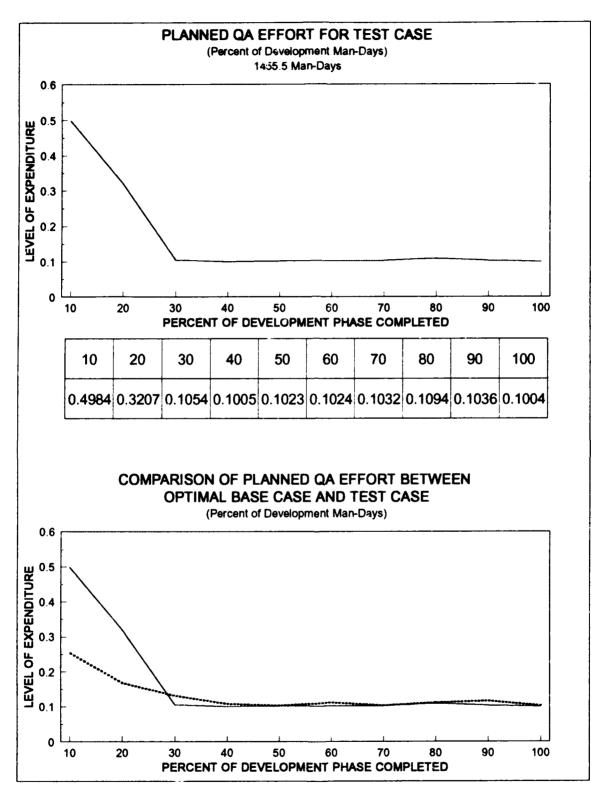
Figure 5-2: Test Case 1 QA Scheme Comparison With Optimal Base Case

C. TEST CASE 2

The second test focused on determining the effects of changing the error generation rate and the amount of rework effort required to correct the errors. Errors committed early in the development cycle are less costly to correct than those committed later in the process. Therefore if more errors are committed later in the development lifecycle, it should be expected that the QA effort required would be greater and more errors would be corrected during testing.

The following variable were changed to conduct the test.

- TNERPK: Nominal Number of Errors Committed per KDSI (Errors/KDSI)

- TNRWME: Nominal Rework Manpower Needed per Error (Man-Days/Error)

Table 5-3 contains the value of these variable for the DE-A project and the values they were changed to for test case 2.

TABLE 5-3

TEST CASE 2 CHANGED VARIABLES

DE-A PROJECT						
TNERPK	24	22.9	20.75	15.25	13.1	12
TNRWME	.6	.575	.5	.4	.325	.3
TEST CASE 2						
TNERPK	12	13.1	15.25	20.75	22.9	24
TNRWME	.3	.325	.4	.5	.575	.6

Table 5-3 indicates that the variables TNERPK and TNRWME have both been reversed resulting in a greater number of errors being committed later in the project development. Figure 5-3 contains a graphical comparison of the results of reversing the TNERPK and TNRWME variables.

COMPARISON OF BASE CASE AND
TEST CASE 2 TNERPK VALUES

1	2	3	4	5	6
24.00	22.90	20.75	15.25	13.10	12.00
12.00	13.10	15.25	20.75	22.90	24.00

COMPARISON OF BASE CASE AND
TEST CASE 2 TNRWME VALUES

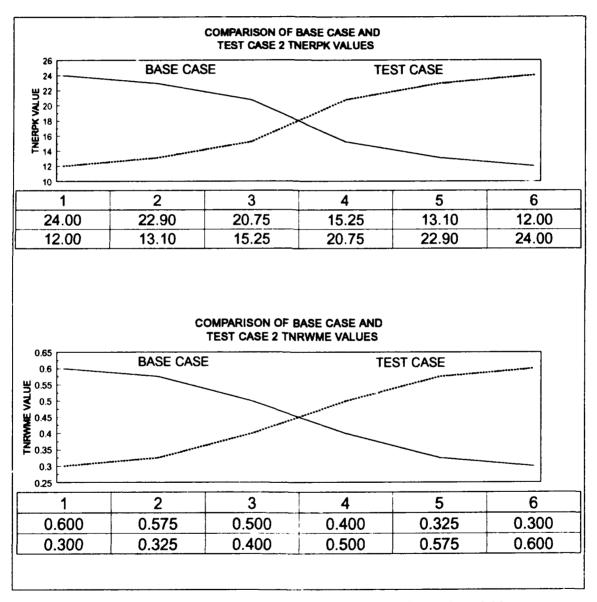| 1	2	3	4	5	6
0.600	0.575	0.500	0.400	0.325	0.300
0.300	0.325	0.400	0.500	0.575	0.600

Figure 5-3: Graphical Comparison of Test Case 2 Changed Variables

An initial expectation was that these changes may bring about a greater QA effort later in the project development lifecycle to correct these errors. Table 5-4 contains the TEST.OUT file produced during the initial testing with constraints and the TEST.OUT file produced with the new variables for test case 2.

TABLE 5-4

PROJECT STATISTICS FOR DE-A EXAMPLE AND TEST CASE 2

PROJECT STATISTICS FOR DE-A EXAMPLE

COMPLETION TIME	320.00		DAYS
TOTAL MAN-DAYS	1,475.53		MAN-DAYS
TOTAL DEV'T MD		1,354.03	MAN-DAYS
DESIGN & CODE		976.11	MAN-DAYS
QA MD		167.99	MAN-DAYS
REWORK MD		209.93	MAN-DAYS
TOTAL TESTING MD		121.50	MAN-DAYS
OVERALL-PRODUCTIVITY	16.27		DSI/MAN-DAYS
TOTAL ERRORS	491.00		ERRORS
% ERRORS DETECTED BY QA	56.11		PERCENT

PROJECT STATISTICS FOR TEST CASE 2

COMPLETION TIME	328.00		DAYS
TOTAL MAN-DAYS	1,520.25		MAN-DAYS
TOTAL DEV'T MD		1,378.15	MAN-DAYS
DESIGN & CODE		972.44	MAN-DAYS
QA MD		242.76	MAN-DAYS
REWORK MD		162.96	MAN-DAYS
TOTAL TESTING MD		142.09	MAN-DAYS
OVERALL-PRODUCTIVITY	15.79		DSI/MAN-DAYS
TOTAL ERRORS	495.00		ERRORS
% ERRORS DETECTED BY QA	47.20		PERCENT

The results of the changes can be seen if figure 5-4. A greater QA effort was required to handle the larger number of errors that were generated later in the development lifecycle. The effort dedicated to rework decreased but the effort required for testing increased by an even greater margin than the decrease. This is emphasized by the 9% decrease in the number of errors that were detected during development. The newly developed QA scheme along with a comparison of this scheme with the DE-A scheme is contained in Figure 5-5.

COMPARISON OF DE-A PROJECT AND TEST CASE 2
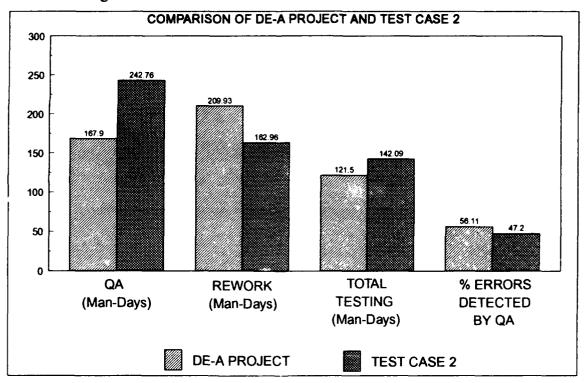
Figure 5-4: Test Case 2 Statistical Comparison With DE-A

A somewhat surprising aspect of Figure 5-5 is that while the QA effort was indeed higher for the test case, this increase was not seen at the end of the development lifecycle as expected, but was at the beginning. While it is not possible to determine the exact cause

66

PLANNED QA EFFORT FOR TEST CASE 2
(Percent of Development Man-Days)
1520.2 Man-Days

10	20	30	40	50	60	70	80	90	100
0.4164	0.4132	0.1479	0.1087	0.1077	0.1025	0.1016	0.1039	0.1039	0.1137

COMPARISON OF PLANNED QA EFFORT BETWEEN
OPTIMAL BASE CASE AND TEST CASE 2
(Percent of Development Man-Days)
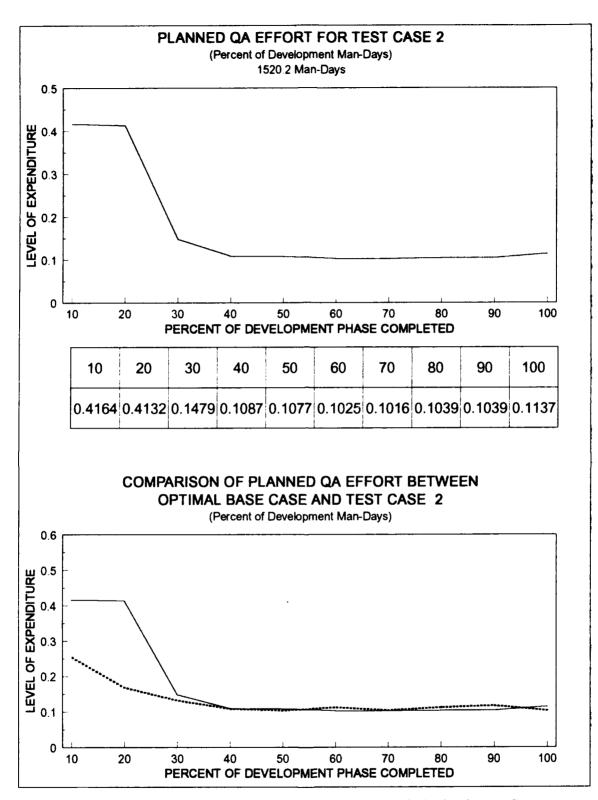
Figure 5-5: Test Case 2 QA Scheme Comparison With Optimal Base Case

of this result, there are two probable reasons. The first is that as a result of the 10%

percent minimum constraint placed on the GA, the amount of effort dedicated to QA was

already being kept higher than required at the end of the development lifecycle. The other

possibility is that there is little to be gained by increasing the QA effort later in the

development phase enforcing the need to for a strong QA program early on in the project's

lifecycle.

VI. CONCLUSIONS AND RECOMMENDATIONS

A. CONCLUSIONS

The focus of this thesis was the development of GASD model that would produce an optimal Quality Assurance scheme in the software development cycle. This was accomplished by using a genetic algorithm to interact with the DYNAMICA software project simulator. Initial testing was completed to determine how to enhance the performance of the GA by determining the best mix of input parameters to be used by the GA. With this accomplished, the model was able to develop optimal solutions in the development of the QA scheme.

Careful examination of the initial QA schemes suggested that the feasibility implementation of a scheme into the software development cycle should be taken into account. The next set of tests focused on developing a QA scheme that could in practice be implemented. This was accomplished by constraining the minimum value that could be assigned to the QA effort at any stage in the development life cycle. The results of these tests demonstrated that the model was still able to develop a QA scheme that added only 2.5% to the project cost as compared to the unconstrained solution but that could be implemented in a real life project.

Finally, the GASD model was used to evaluate the effects of changing some parameters in the DE-A example and to evaluate the sensitivity of the results. A

69

comparison of these results with the DE-A example clearly indicated that the GASD

model was responsive to these changes. In addition, the comparisons highlighted the

importance of the necessity to adjust the QA effort to prevailing conditions of a particular

project and not to develop an all encompassing cookbook solution.

B. RECOMMENDATIONS FOR FURTHER RESEARCH

1. Expand the Number of Inputs

All the testing with the GA was done by changing the TPFMQA variables in the

GASD model. The selection of this variable was based on the desire to study the

significance of the QA effort in software development. However, QA is certainly not the

only parameter influencing the cost of development. With the use of the GA, it is possible

to either select other variables to be looked at by the GA or to consider other variables

along with the TPFMQA variables. Total project cost could then be minimized based on

any of the other inputs or based on a combination of inputs.

Changes in the GASD model necessary to accomplish this would not be difficult.

The TPFMQA variables were represented as a 60 bit binary string that was used by the

GA for computational purposes. The GA is not constrained to any particular string length.

Changing the inputs in the TEST.DNX file would require modifications to the fitness

function in the Appendix. If it was desired to change the input to be optimized by the GA,

it would be necessary to determine the desired representation scheme and write these

inputs to the TEST.DNX in the same manner the QA variables are currently written to the

file. TPFMQA variables would have to be set to a specific values in the file and the code in GOLD.C that generates the TPFMQA variables changed to suit the problem. If it was desired to examine multiple inputs, including TPFMQA, it would be necessary to just write these inputs to the TEST.DNX file as stated above. The only other changes would be in the SAMPLE.IN file that would have to have the string length changed to match to new representation scheme. Optimizing parameters not contained in the TEST.DNX file will require a more advance version of the DYNAMICA software development program.

2. Rule Generation

The Genetic Algorithm proved very useful in solving the problem of developing an optimal QA scheme. The answer that is developed does little to enhance the knowledge of why a particular QA scheme worked well or why another did not. The interactions of the different processes that occur within the DYNAMICA simulation do not concern the GA as the GA is only interested in a fitness measure to gauge its performance. The interaction of different variables and the relationships that develop as a result of these interactions contains significant information on how the overall project management process functions. The identification of rules or procedures that govern these relationships would be extremely helpful to a project manager.

One approach to developing such rules and procedures would be by using a genetic program. The genetic algorithm is able to develop solutions based on a specific representation scheme. The genetic program functions by breeding computer programs much as the genetic algorithm bred binary strings. The development of the genetic

71

program would require the examination of all available data that is used in the

development of the QA scheme. The genetic program would use this data to develop a

function that describes the process. By examining the function, it should then be possible

to determine the rules associated with the process. There are currently genetic programs

under development that are directed toward rule generation. The use of one of these

programs would make this a feasible problem to solve.

APPENDIX

/* This code is called by the GAUCSD genetic algorithm. It takes a bit string containing 10 double precision floating point numbers representing the input parameters from the genetic algorithm and converts the string to a single double precision floating point representing the output value. It does this by calling the DYNAMICA DOS based simulation program. The DYNAMICA program generates a report file during its execution from which this code extracts relevant information from and is used by the genetic algorithm to perform the fitness measure portion of execution.*/

#include <math.h>

#include <stdio.h>

double gold(x) /*Designates function gold as a double precision float with x as the input
            parameter.* /

    /*GAUCSD sends array x to this code*/

    register double *x; {

    double p1,p2,p3,p4,p5,p6,p7,p8,p9,p10; /*space allocation for function gold*/

    char dea_file_name[80]; /*space allocation for DEA File*/

    char rpt_file_name[80]; /*space allocation for Report out_file_name*/

    char file_line[80]; /*space allocation for the string file_line*/

    char command_line[80]; /*space allocation for string command_line*/

    char *c; /*space allocation for character pointer c*/

    double result,mult; /*Makes result double precsion and multiplies result*/

    FILE *f;

/*This portion of the code takes the numbers from the array x and transforms them into valid parameters for DYNAMICA. It then places the values into the fields of p1 thru p10*/

    p1 = x[0] + .42 ; p2 = x[1] + .42 ; p3 = x[2] + .42 ;

    p4 = x[3]+ .42 ; p5 = x[4]+ .42 ; p6 = x[5]+ .42 ;

    p7 = x[6]+ .42 ; p8 = x[7]+ .42 ; p9 = x[8]+ .42 ; p10 = x[9]+ .42 ;

73

```c
        sprintf(dea_file_name,"TEST.dnx");   /* Names the file with the .DNX extension
                neccesary for DYNAMICA to input the file .*/

        f = fopen(dea_file_name,"w");   /* Opens the file for writing .*/

        if(!f) { perror("gold"); exit(1); }   /* Error condition to ensure file is opened. If file file is
                not opened, the programs exits the simulation.*/

        /* The following data is printed directly iinto the test.DNX file. This data remains
        constant each time the file is created.*/

        fprintf(f,"\n\

C RJBDSI=24400\n\

C UNDEST=.35\n\

C TOTMD1=1111\n\

C DEVPRT=0.85\n\

T TPFMQA=");

        fprintf(f,"%lf %lf %lf %lf %lf %lf %lf %lf %lf %lf 0\n",

                p1,p2,p3,p4,p5,p6,p7,p8,p9,p10);

        /* The remaining data entries remain constant each time the file is created.*/

        fprintf(f,"\

C INUDST=0.4\n\

C TDEV1=320\n\

C MXSCDX=1.16\n\

C ADMPPS=0.5\n\

C HIREDY=30\n\

C AVEMPT=1000\n\

C TRPNHR=0.25\n\

C ASIMDY=20\n\
```

```
C DSIPTK=40\n\

T TNERPK=24 22.9 20.75 15.25 13.1 12\n");
        close(f);   /* Closes the file f */


        /* Prints the DYNEX command to a character array named command_line.*/

        sprintf(command_line,"DYNEX EXAMPLE1 TEST -OUT TEST.DTM -D TEST.DRS");
        /* Sends command_line to DOS which calls the DYNAMICA program for execution. If
        an error code is detected, program is exited and an error message is sent.*/
        if(system(command_line)) {

                perror("system:");

                exit(1);

        }
        /* Prints the SMLT  (simulate) command to a character array named command_line.*/

        sprintf(command_line,"SMLT EXAMPLE1 -GO TEST.RSL -DTM TEST");

        /* Sends command_line to the DOS.  If an error code is detected, program is exited and
        an error message is sent.*/
        if(system(command_line)) {

                perror("system");

                exit(1);

        }

        /* Prints the REP (report) command to a character array named command_line.*/

        sprintf(command_line,"REP TEST.RSL REPORT.DRS -T");
        /* Sends command_line to DOS. If an error code is detected, program is exited and an
        error message is sent.*/
        if(system(command_line)) {

                perror("system");
```

```
        exit(1);

}

sprintf(rpt_file_name,"TEST.out");  /* Assigns test file name TEST.out.*/

f = fopen(rpt_file_name,"r");   /* Opens TEST.out for reading.*/


/* Within the file TEST.out is the data on total man days. That information is retrieved
from the file to be returned to the GA.*/
while(fgets(file_line,79,f)) {

        if(!strncmp(file_line,"   TOTAL MAN-DAYS",18)) {

                printf(" --- %s",file_line); // compute result

                /* Skips over file data until a number is reached. Once the the correct
                number is found, that string of numbers is turned into a floating point
                number. */

                for(c = file_line; *c && !isdigit(*c); c++)

                        ;

                result = 0.0;

                /* c presumably points to the integer. */

                for(;*c && (isdigit(*c) || *c==','); c++)  {

                        if(*c == ',')

                                continue;

                        result *= 10.0;

                        result += *c-'0';

                }

                if(*c == '.') {

                        c++;
```

```c
                        for(mult = 0.1; *c && isdigit(*c); c++) {

                                result += (*c-'0')*mult;

                                mult = mult/10.0;

                        }

                }

                /* Error condition to check if retrieved result is less than zero. If this
                occurs an error condition exists and the program is exited.*/

                if(result <= 0.0) {

                        fprintf(stderr,"check this out!\n");

                        exit(1);

                }

                fclose(f);   /* Closes the file.*/

                /* prints the number that was retrieved from above */

                fprintf(stderr,"result -- %lf\n",result);

                return(result);   /* Returns the number to GA for evaluation.*/

        }

}

/* This portion of code executed only if it was not possible to locate the total man days
figure. If it gets here, the program will be terminated.*/

fclose(f);

fprintf(stderr,"gold\n");

exit(1);

}

/* GAeval gold 6:0.32dg10 */    //**AWK script necessary to process the gold.c program. This is
        a necessary directive for the post processor to ensure that this entire function is embedded
        correctly in the GAUCSD code**//
```

LIST OF REFERENCES

1. Pressman, Roger S., *Software Engineering a Practictioners Approach* , McGraw-Hill Inc., 1992.

2. Carrazoni, Joseph A., *Initial Definition of a Knowledge-Based Software Quality Assistant*, Rome Laboratories, Air Force Material Command, Griffis Air Force Base, New York, 1993.

3. Schulmeyer, G. Gordon , and Mcmanus, James G., *Handbook of Software Quality Assurance*, Van Nostrand Reinhold, 1992.

4. Abdel-Hamid, Tarek K., and Madnick, Stuart E. "The Economics Of Software Quality Assurance: A Simulation Based Case Study", *MIS Quarterly*, Vol. 32, September 1988.

5. Koza, John R., *Genetic Programming*, MIT Press, 1992.

6. Goldberg, David A., *Genetic Algorithims in Search, Optimization, and Muchine Learning*, Addison-Wesley Publishing Company, Inc., 1989.

7. Schraudolph, Nicol N., and Greffenstette, John J., *A User's Guide to GAUCSD 1.4*, CSE Department, UC San Diego, LA Jolla, CA, 1992

8. Abdel-Hamid, Tarek K., and Madnick, Stuart E., "Lessons Learned from Modeling the Dynamics of Software Development", Communications of the ACM, Vol 32, Number 12, December 1989.

9. Abdel-Hamid, Tarek K., and Madnick, Stuart E., *Softeware Project Dynamics an Integrated Approach*, Prentice-Hall, 1991.

10. Buzzard, Raymond Karl, *A Prolog Implementation of Pattern Search to Optimize Quality Assurance*, M.S. Thesis, Naval Postgraduate School, Monterey, California, March 1990.

DISTRIBUTION LIST

1. Defense Technical Information Center   2
   Cameron Station
   Alexandria, Virginia 22304-6145

2. Library, Code 52   2
   Naval Postgraduate School
   Monterey, California 93943-5002

3. Balasubramaniam Ramesh, Code AS/RA   5
   Naval Postgraduate School
   Monterey, California 93943-5002

4. Tarek K. Abdel-Hamid, Code AS/AH   1
   Naval Postgraduate School
   Monterey, California 93943-5002

5. Major Donald M. Elliott   2
   12708 Flagship Court
   Herndon, Virginia 22070