Experimenting with Genetic Algorithms to Devise Optimal Integration Test Orders

By

Jie Feng, B. Eng.

A thesis submitted to the Faculty of Graduate Studies and
Research in partial fulfillment of the requirements of the
degree of
Master of Applied Science in Electrical Engineering

Ottawa-Carleton Institute of Electrical and Computer Engineering
Department of Systems and Computer Engineering
Carleton University
Ottawa Ontario, K1S 5B6
Canada

August 2002

National Library
of Canada

Acquisitions and
Bibliographic Services

395 Wellington Street
Ottawa ON K1A 0N4
Canada

Bibliothèque nationale
du Canada

Acquisitions et
services bibliographiques

395, rue Wellington
Ottawa ON K1A 0N4
Canada

Your file Votre référence

Our file Notre référence

The author has granted a non-exclusive licence allowing the National Library of Canada to reproduce, loan, distribute or sell copies of this thesis in microform, paper or electronic formats.

The author retains ownership of the copyright in this thesis. Neither the thesis nor substantial extracts from it may be printed or otherwise reproduced without the author's permission.

L'auteur a accordé une licence non exclusive permettant à la Bibliothèque nationale du Canada de reproduire, prêter, distribuer ou vendre des copies de cette thèse sous la forme de microfiche/film, de reproduction sur papier ou sur format électronique.

L'auteur conserve la propriété du droit d'auteur qui protège cette thèse. Ni la thèse ni des extraits substantiels de celle-ci ne doivent être imprimés ou autrement reproduits sans son autorisation.

0-612-79700-7

Canada

The undersigned recommend to the
Faculty of Graduate Studies and Research
Acceptance of the thesis:

**Experimenting with Genetic Algorithms to Devise
Optimal Integration Test Orders**

Submitted by
Jie Feng, B. Eng.

In partial fulfillment of the requirements of the degree of
Master of Applied Science

Chair, Professor Rafik A. Goubran

Thesis Supervisor, Professor L. Briand

Thesis Supervisor, Professor Y. Labiche

Carleton University

September 2002

ABSTRACT

This work presents an improved strategy to devise optimal integration test orders in object-oriented systems in the presence of dependency cycles. The goal is to minimize the complexity of stubbing during integration testing as this has been shown to be a major source of expenditure. The strategy adopted in this work is based on the combined use of inter-class coupling measurement and genetic algorithms. The former is used to assess the complexity of stubs (each coupling measure capturing a dimension of this complexity) and the latter is used to minimize complex cost functions based on coupling measurement. Using a precisely defined procedure, we investigate this approach in case studies investigating five real systems. The obtained results are very encouraging as the approach clearly helps obtain, in a systematic way, integration orders with low stubbing complexity.

Acknowledgments

First, I would like to express my deep gratitude to my supervisor Dr. Lionel Briand. It was my very first time speaking English for more than five minutes when I first met Dr. Briand less than two years. Dr. Briand believed in my work and me when I most needed support.

I am deeply indebted to Dr. Briand and my co-supervisor, Dr. Yvan Labiche, for my intellectual formation. Their guidance and tolerance made my study in Carleton University enjoyable.

I would also like to thank my mother for her unconditional love and support.

And most of all, I give special thanks to my little daughter for her good behaviour.

All my colleagues in SQUALL (Software QUALity Engineering Laboratory) lab supported me during my study and research work. One of my case studies came from Leeshawn O'Sullivan's course project. Niannian Ding provided me part of the Java implementation for graph-based algorithms I needed in this thesis. Tun Yang gave me a hand in correcting my English. Thank you, Leeshawn, Niannian and Tun. Last, I want to thank my friends Hong Sun, Peng Xie and Yufeng Yang for their help, support and hints.

Table of Contents

CHAPTER 1 INTRODUCTION.....	1
CHAPTER 2 APPLYING GENETIC ALGORITHMS TO THE TEST ORDER PROBLEM.....	3
2.1 MOTIVATIONS	3
2.1.1 <i>The integration order problem</i>	3
2.1.2 <i>Graph-based solutions are not satisfactory</i>	5
2.1.3 <i>Problem Statement</i>	7
2.1.4 <i>Moving to Genetic Algorithms</i>	8
2.2 FUNDAMENTAL PRINCIPLES AND APPLICATION TO SCHEDULING PROBLEMS.....	10
2.3 TAILORING AND APPLICATION PROCEDURE	16
2.3.1 <i>Parameter settings</i>	16
2.3.2 <i>Measuring Stubbing Complexity</i>	16
2.3.3 <i>Constraints</i>	19
CHAPTER 3 AUTOMATION TOOLS	22
3.1 TAILORED GA TOOL	22
3.2 TOOLS FOR ANALYZING CLASS DEPENDENCIES	27
CHAPTER 4 CASE STUDIES.....	29
4.1 DESCRIPTION OF THE CASE STUDIES	29
4.2 THE APPLICATION OF GENETIC ALGORITHMS	34
4.3 RESULTS.....	37
4.3.1 <i>The ATM system</i>	38
4.3.2 <i>The Ant system</i>	40
4.3.3 <i>The SPM System</i>	44
4.3.4 <i>The BCEL System</i>	46
4.3.5 <i>The DNS System</i>	49
4.4 DETERMINING WEIGHTS IN OCPLX.....	51
CHAPTER 5 CONCLUSIONS	53
REFERENCES	55
APPENDIX A.1: CLASS DIAGRAM AND RELATIONSHIP MATRIX FOR ATM.....	57
APPENDIX A.2: PRECEDENCE TABLE FOR ATM.....	60
APPENDIX A.3: DEPENDENCY MATRIX FOR ATM.....	61

APPENDIX A.4: ATTRIBUTE COUPLING MATRIX FOR ATM	62
APPENDIX A.5: METHOD COUPLING MATRIX FOR ATM	63
APPENDIX A.6: TEST ORDERS OBTAINED WITH NUMBER OF BROKEN DEPENDENCIES AS COST FUNCTION (ATM).....	64
APPENDIX A.7: TEST ORDERS OBTAINED WHEN USING THE NUMBER OF STUBBED ATTRIBUTES AS THE COST FUNCTION (ATM)	65
APPENDIX A.8: TEST ORDERS OBTAINED WHEN USING THE NUMBER OF STUBBED METHODS AS THE COST FUNCTION (ATM)	66
APPENDIX A.9: TEST ORDERS OBTAINED WHEN USING A WEIGHTED GEOMETRIC AVERAGE OF ATTRIBUTE AND METHOD COUPLING MEASURES AS THE COST FUNCTION (ATM).....	67
APPENDIX B.1: CLASS DIAGRAM AND RELATIONSHIP MATRIX FOR ANT	68
APPENDIX B.2: PRECEDENCE TABLE FOR ANT	71
APPENDIX B.3: DEPENDENCY MATRIX FOR ANT.....	72
APPENDIX B.4: ATTRIBUTE COUPLING MATRIX FOR ANT	73
APPENDIX B.5: METHOD COUPLING MATRIX FOR ANT.....	74
APPENDIX B.6: TEST ORDERS OBTAINED WITH NUMBER OF BROKEN DEPENDENCIES AS COST FUNCTION (ANT).....	75
APPENDIX B.7: TEST ORDERS OBTAINED WHEN USING THE NUMBER OF STUBBED ATTRIBUTES AS THE COST FUNCTION (ANT)	76
APPENDIX B.8: TEST ORDERS OBTAINED WHEN USING THE NUMBER OF STUBBED METHODS AS THE COST	

FUNCTION (ANT)	77
APPENDIX B.9: TEST ORDERS OBTAINED WHEN USING A WEIGHTED GEOMETRIC AVERAGE OF ATTRIBUTE AND METHOD COUPLING MEASURES AS THE COST FUNCTION (ANT).....	78
APPENDIX C.1: CLASS DIAGRAM AND RELATIONSHIP MATRIX FOR SPM	79
APPENDIX C.2: DEPENDENCY MATRIX FOR SPM	82
APPENDIX C.3: PRECEDENCE TABLE FOR SPM.....	83
APPENDIX C.4: ATTRIBUTE COUPLING MATRIX FOR SPM	84
APPENDIX C.5: METHOD COUPLING MATRIX FOR SPM	85
APPENDIX C.6: TEST ORDERS OBTAINED WITH NUMBER OF BROKEN DEPENDENCIES AS COST FUNCTION (SPM).....	86
APPENDIX C.7: TEST ORDERS OBTAINED WHEN USING THE NUMBER OF STUBBED ATTRIBUTES AS THE COST FUNCTION (SPM)	87
APPENDIX C.8: TEST ORDERS OBTAINED WHEN USING THE NUMBER OF STUBBED METHODS AS THE COST FUNCTION (SPM)	88
APPENDIX C.9: TEST ORDERS OBTAINED WHEN USING A WEIGHTED GEOMETRIC AVERAGE OF ATTRIBUTE AND METHOD COUPLING MEASURES AS THE COST FUNCTION (SPM).....	89
APPENDIX D.1: RELATIONSHIP MATRIX FOR BCEL	90
APPENDIX D.2: PRECEDENCE TABLE FOR BCEL	93
APPENDIX D.3: DEPENDENCY MATRIX FOR BCEL	94
APPENDIX D.4: ATTRIBUTE COUPLING MATRIX FOR BCEL....	96

APPENDIX D.5: METHOD COUPLING MATRIX FOR BCEL	98
APPENDIX D.6: TEST ORDERS OBTAINED WITH NUMBER OF BROKEN DEPENDENCIES AS COST FUNCTION (BCEL)	100
APPENDIX D.7: TEST ORDERS OBTAINED WHEN USING THE NUMBER OF STUBBED ATTRIBUTES AS THE COST FUNCTION (BCEL).....	101
APPENDIX D.8: TEST ORDERS OBTAINED WHEN USING THE NUMBER OF STUBBED METHODS AS THE COST FUNCTION (BCEL).....	102
APPENDIX D.9: TEST ORDERS OBTAINED USING A WEIGHTED GEOMETRIC AVERAGE OF ATTRIBUTE AND METHOD COUPLING MEASURES AS THE COST FUNCTION (BCEL).....	103
APPENDIX E.1: RELATIONSHIP MATRIX FOR DNS	104
APPENDIX E.2: PRECEDENCE TABLE FOR DNS	112
APPENDIX E.3: DEPENDENCY MATRIX FOR DNS	114
APPENDIX E.4: ATTRIBUTE COUPLING MATRIX FOR DNS.....	120
APPENDIX E.5: METHOD COUPLING MATRIX FOR DNS	126
APPENDIX E.6: TEST ORDERS OBTAINED WITH NUMBER OF BROKEN DEPENDENCIES AS COST FUNCTION (DNS)	132
APPENDIX E.7: TEST ORDERS OBTAINED WHEN USING THE NUMBER OF STUBBED ATTRIBUTES AS COST FUNCTION (DNS).....	133
APPENDIX E.8: TEST ORDERS OBTAINED WHEN USING THE NUMBER OF STUBBED METHODS AS COST FUNCTION (DNS).....	134
APPENDIX E.9: TEST ORDERS OBTAINED USING A WEIGHTED	

GEOMETRIC AVERAGE OF ATTRIBUTE AND METHOD COUPLING MEASURES AS THE COST FUNCTION (DNS)135

List of Figures

Figure 1 Genetic Algorithm – Pseudo Code [Kantrowitz 1997].....	11
Figure 2 State design pattern.....	20
Figure 3 Dependency matrix of ATM.....	23
Figure 4 Attribute coupling matrix of ATM	24
Figure 5 Method coupling matrix of ATM	24
Figure 6 Precedence table of ATM	25
Figure 7 Weight table.....	25
Figure 8 Ajustable range and the cost	26
Figure9 Results of 100 executions	27
Figure 10 Data Flow Diagram for the usage of the DA and MC tools	28
Figure 11 Distributions of coupling for ATM.....	32
Figure 12 Distributions of coupling for Ant.....	33
Figure 13 Distributions of coupling for SPM.....	33
Figure 14 Distributions of coupling for BCEL	33
Figure 15 Distributions of coupling for DNS.....	34
Figure 16 Distributions of test stubs from the graph-based algorithms for the ATM system.....	40
Figure 17 M Distributions when the cost function is M, A and OCplx	42
Figure 18 Distribution of OCplx across 100 GA executions for the Ant system.....	43
Figure 19 Distributions of test stubs from graph-based algorithms for the Ant system ...	44
Figure 20 Distributions of test stubs from graph-based algorithms for the SPM system .	45
Figure 21 Distribution of <i>OCplx</i> for the BCEL system.....	47
Figure 22 Distributions of test stubs from graph-based algorithms for the BCEL system	48
Figure 23 Distributions of test stubs from graph-based algorithms for the DNS system .	51
Figure 24 Class Diagram of the ATM System.....	57
Figure 25 Class Diagram of the Ant System.....	68
Figure 26 Class Diagram for the SPM system	79

List of Tables

Table 1 System Descriptions.....	32
Table 2 Coupling levels of the five systems	34
Table 3 Summary of Results for the ATM (Appendices A.6, A.7, A.8, A.9).....	39
Table 4 Summary of Results for the Ant (Appendices B.6, B.7, B.8, B.9)	42
Table 5 Summary of Results for the SPM (Appendices C.6, C.7, C.8, C.9)	45
Table 6 Summary of Results for the BCEL (Appendices D.6, D.7, D.8, D.9)	47
Table 7 Summary of Results for the DNS System (Appendices E.6, E.7, E.8, E.9)	50

Chapter 1 Introduction

One important problem when integrating and testing object-oriented software is to decide the order of class integration [Binder 1999]. A number of papers have provided strategies and algorithms to derive an integration and test order from dependencies among classes in a system [Kung 1995, Tai 1999, Labiche 2000, Le Traon 2000, Briand 2001]. The objective of all these approaches is to minimize the number of test stubs to be produced, as this is perceived to be a major cost factor of integration testing.

One specific issue is how to deal with dependency cycles, which prevent any topological ordering of classes. This is important as class diagrams (e.g., UML class diagrams defined during Analysis or Design, or reverse-engineered) of most object-oriented software systems contain dependency cycles. All the solutions proposed in [Tai 1999, Le Traon 2000, Briand 2001] are based on the principle of “breaking” some dependencies to obtain acyclic dependencies between classes. A broken dependency implies that the target class will have to be stubbed when integrating and testing the source class. Furthermore, all these solutions are based on search algorithms in directed graph representations of class dependencies; The limitations of such an approach are discussed below.

In practice, we would like to refine graph-based algorithms in order, for example, to account for the complexity of dependencies between classes. In other words, different dependencies that we consider breaking may lead to stubs of widely varying complexity. Such complexity is driven by the coupling that exists between the client and server class of a dependency. If the client uses a lot of features of the server, then breaking the dependency is expected to lead to an expensive stub. Given our objective, modifying

graph-based algorithms to account for stub complexity based on coupling measurement turns out to be very complex and leads to intractable algorithms.

In order to address this problem, and for reasons that are described below, we then turned our attention to Genetic Algorithms (GA's), a global optimization technique based on heuristics, which has been developed by the artificial intelligence community over the years [Chambers 1995]. As further discussed below, the main motivation for using these algorithms is their flexibility and practicality in solving a large range of optimization problems. Their popularity in software engineering is growing as many of the problems we face can be re-expressed as optimization problems. This is further illustrated by a recent journal issue dedicated to the applications of metaheuristics algorithms to software engineering [Jones 2001].

This thesis focuses on the best ways to use GA's in order to devise optimal test orders based on dependency coupling measurement between client and server classes. We provide motivations and a methodology in Chapter 2. Chapter 3 gives a brief description of the automation of the experiments; Chapter 4 then provides the detailed results of five case studies where we demonstrate that the approach is not only feasible but also yields low stubbing complexity test orders. Chapter 5 concludes and outlines future research.

Chapter 2 Applying Genetic Algorithms to the Test Order

Problem

First some motivations that lead us to select genetic algorithms over alternative approaches are provided in Section 2.1. We then present an overview of the relevant, fundamental principles we will be using in the remainder of the thesis (Section 2.2). We then describe how we tailored Genetic Algorithms to our specific problem and report a procedure to use them to devise low stubbing complexity test orders (Section 2.3).

2.1 Motivations

First the integration order problem is discussed in more depth, then why it is difficult to adapt existing graph-based algorithms to accommodate our needs in determining test orders is analyzed. Last, why genetic algorithms seem to be a good alternative to investigate is justified.

2.1.1 The integration order problem

Once classes have been developed and tested in isolation, one important problem is to integrate them into a (sub)system. Due to the usual problems associated with big-bang integration, classes need to be integrated one at a time or, in some cases, in small clusters. A practical question is how to define precisely what an optimal integration order actually is. We need to define evaluation criteria to compare orders and find algorithms that can identify (near) optimal orders automatically.

A natural evaluation criterion is the stubbing effort required to integrate classes according to a specific order. If an order integrates classes when some of the classes they depend on are not yet integrated, the development of stubs to substitute for those classes is

necessary to perform integration testing. It has been observed that such stubbing is error-prone and significantly increases the cost of integration testing [Beizer 1990]. Existing algorithms that devise class test orders are typically based on analyzing dependencies between classes [Kung 1995, Tai 1999, Labiche 2000, Le Traon 2000, Briand 2001]. If there are no cycles of dependencies among classes, the problem becomes a simple topological sorting issue, a well-known graph theory problem [Deo 1974], and it is then possible to obtain integration orders that do not require the stubbing of any class. In the case where dependency cycles can be observed, there is no choice but to integrate some classes when certain classes they depend on are not yet integrated. This leads to the development of stubs and significantly increases the cost of integration testing. In such a situation, we need to devise integration orders that *minimize* stubbing effort. Since such effort cannot be directly measured or estimated, we need to resort to indicators.

One type of indicator is the dependency type. Previous papers [Kung 1995, Tai 1999, Labiche 2000, Briand 2001] on class integration order differentiate several kinds of binary dependencies between classes. In a UML class diagram, Compositions (C_p) and Inheritance (I) dependencies are considered unbreakable as such relationships usually entail tight dependencies between the client/parent and server/child classes [Briand 2001] so that breaking them would likely lead to complex stubbing. Remaining dependencies, including Usages (U_s), Associations and Simple Aggregations (A_s), are considered breakable to obtain an acyclic class diagram. The strategy in this thesis refines this approach by accounting for coupling information between breakable dependencies (Section 2.3.2 and Section 2.3.3). Another important point to mention is that, though we focus on ordering class integration, the method we investigate can be applied at higher levels of integration.

For example, in a very large system, we can easily imagine two or more levels of integration. A first level is concerned with integrating classes into subsystems, within each sub-team of the project. The second level would then address the integration of subsystems into a complete system. In large, complex systems, the dependencies among subsystems may be complex and show cycles. Those dependencies are determined by the public classes within subsystems, which form their public interface to other subsystems.

2.1.2 Graph-based solutions are not satisfactory

Previous works considered the number of stubs as an estimator for the cost associated with the construction of stubs, that is the number of dependencies to break so as to obtain an acyclic class dependency graph, and then applying topological sorting to obtain an integration test order. As a consequence, the problem was a graph-searching problem and was solved using typical graph algorithms. Basically, the proposed solutions in [Briand 2001] and [Le Traon 2000] consist in recursively identifying strongly connected components (SCC) and then, in each SCC, removing *one* dependency that is most likely to maximize the number of broken cycles. These solutions consider two distinct but simple cost functions to select the dependency to break at each recursion step, but they share the commonality of always optimizing their choice of dependencies to break at a given stage of recursion, without determining the consequences on the ultimate results (i.e., they are both greedy algorithms). How this becomes a serious limitation in the context we stated in Chapter 1 is discussed below.

In addition, one issue in the approach proposed by Le Traon et al [Le Traon 2000] is that this algorithm is not fully deterministic in the sense that, depending on some arbitrary decision (e.g., the initial vertex (class) of the search of SCC, and the search itself),

the algorithms may yield significantly different results. Furthermore, since the model used does not have any information on the kind of dependency, this approach may lead to the removal of an Inheritance or Composition relationship [Briand 2001].

Tai and Daniels [Tai 1999] proposed a 2-stage algorithm that deals with dependency cycles. However, in cases where usages, association and simple aggregation relationships are not involved in cycles, their solution is sub-optimal in terms of the required number of test stubs [Labiche 2000]. The strategy proposed by Briand et al is based on the two presented above and addresses some of their weaknesses. The five case studies presented in Section 4.3 make use of these three approaches in order to provide a baseline of comparison to the solution based on genetic algorithms in this thesis.

Let us first recall the needs in terms of devising optimal integration orders. As mentioned above the coupling associated with breakable client-server dependencies, for which we can provide several complementary and alternative measures, e.g., number of calls made by the client class to the server class, number of distinct methods invoked. If such coupling is accounted for, there may be a situation where, for instance, breaking two dependencies has a lower cost (i.e., coupling) than breaking only one dependency that would make the graph acyclic in one step. Furthermore, we would like to consider constraints, due to organizational or contractual reasons, which would then result in some classes not being available (developed or unit tested) before others during integration. In practice, such situations constrain the optimal ordering solution we are searching for.

Given the above requirements, adapting the previous graph-based, stepwise strategies seems difficult or even impossible. The cost functions which are being minimized here (see Section 2.3.2) are highly non-linear and are likely to present many

local minima in which a stepwise algorithm may get stuck¹. This is a typical shortcoming for greedy algorithms.

Consequently, using coupling information in the cost function would require the identification, in each SCC, of all the *subsets* of breakable dependencies (e.g., pairs, triplets) that would make the graph acyclic, and the determination of their associated cost to select an optimal solution. This is equivalent to an exhaustive search, which is not a viable solution for any realistic class diagram. Similarly, when using a stepwise search algorithm, it is difficult to determine whether breaking a dependency at a given step will not lead to orders, at subsequent steps, that will transgress some of the constraints which limit the search.

2.1.3 Problem Statement

The first question we want to investigate is whether GA's, as parameterized in this thesis, work as well as deterministic graph algorithms such as the ones described above and experimented with in [Briand 2001]. So we define the cost function as the number of dependencies to be broken in a given test order and run the GA. Since such algorithms are heuristic, the results may differ from run to run and we therefore run the GA 100 times and check the percentage of times the algorithm converges towards the orders using the graph-based search algorithms. This question is important as it is a preliminary but necessary validation of the approach we describe in this thesis.

The second question we investigate is related to the use of three different coupling measurement in the cost function. We want to know whether such measurements make a significant difference in terms of the test orders that are generated, as compared to the

¹ Any small change in the order (e.g., swapping two classes) may have a large impact on the cost function.

orders we obtain with the graph-based algorithm. In other words, these four cost functions (three coupling measures and the number of broken dependencies) are used in turn to produce sets of orders on which we compute the values of the three other functions. If we observe *significant* differences in terms of order stubbed complexity when only the number of dependencies is used as a cost function and when the other measures are used, then we can conclude that using coupling measurement may lead to significantly different test orders, involving less coupling to be broken and hopefully leading to lower integration test expenditures.

2.1.4 Moving to Genetic Algorithms

The solution space to the integration test order problem is the set of all possible permutations of N classes (test orders) for a given system. The size of this search space is $N!$. Any single permutation of classes yields a solution. Given the cost of breaking the dependency between each pair of classes, our goal is to search for an order that minimizes the cost due to broken dependencies. A typical permutation problem is the *Travelling Salesman Problem* (TSP) where a traveling salesman must visit every city in his territory exactly once and then return home to his starting point [Michalewicz 2000]. The representation of the TSP is also an order, more precisely a sequence of towns being traversed. The problem is also to minimize the distance (cost function) covered by the salesman while achieving this objective. Though there are significant differences between the integration test order problem and the TSP, it is analogous since it looks for an optimal order minimizing a cost function.

The TSP is a well-known member of the NP-Complete class of problems. There are no polynomial-time algorithms that can generate perfect solutions though many

approximate solutions exist. These include nearest neighbor, greedy, nearest insertion, farthest insertion, double minimum spanning trees, strip, and space-filling curves [Michalewicz 2000]. Alternatively, the Genetic Algorithm community proposed and experimented with solutions for the TSP [Michalewicz 2000]. Furthermore, reported examples of scheduling and routing problems (e.g., [Thangiah 1992]) suggest that genetic algorithms provide satisfactory results. However, there are important differences between our integration order problem and the TSP problem. This mainly stems from differences in the cost function:

- In the TSP, if the order places A before B or vice-versa, the cost is the same as it is the distance between the two towns. This is not the case in our context where one solution may entail a stub whereas the other one does not.
- Similarly, since the distance is the cost in the TSP, the triangle inequality holds where the distance of ABC is equal or larger than that of AC. Such inequality has no reason to hold in our context.

Those differences could have significant impact on the ability of the GA to converge towards an optimal solution and this further justifies the need to perform case studies to assess the ability of GA's to solve the integration test order problem. Another difference with the TSP problem is the fact that we will need to find an optimal solution under constraints specifying whether orders are acceptable or not. Section 2.3.3 shows how this can be practically achieved with GA's. Furthermore, it is interesting to note that, though deterministic algorithms to compute near-optimal tours in the TSP problem have been proposed [Lawler 1985], the differences above prevent us from using them.

Another practical motivation to use GA's is that commercial tools have reached a high level of maturity and provide convenient features for solving real-scale problems. This is further discussed in Section 4.1. Genetic algorithms (GA's) are a specific type of optimization technique that are based on a set of heuristics and involve a non-trivial and careful setting of a number of parameters. The basic principles that are relevant to our problem will be summarized in the next section. In the context of this work, one of their interesting characteristics is that they prevent the optimization search from getting stuck in local minima and this is why they are often referred to as *global* optimization techniques. GA's are also known to help solve complex, non-linear² problems that often lead to cases where the search space shows a curvy, noisy "landscape" with numerous local minima [Jones 2001]. However, as they are based on heuristics there is no guarantee they will find the absolute, global minimum.

2.2 Fundamental Principles and Application to Scheduling Problems

A genetic algorithm is a model of machine learning which derives its behavior from a metaphor of the processes of *evolution* in nature. It is inspired by Darwin's theory about evolution [Kantrowitz 1997]. With such an algorithm, the solution to a problem to be solved is evolved by creating a *population* of individuals represented by chromosomes (the chromosomes in the population then go through a process of evolution). Given a fitness function (i.e., the function to be optimized), some chromosomes are better than others. Those that are better are more likely to survive and propagate their genetic material from one generation of the population to the next. . This propagation of genetic material is performed by means of a recombination operation, which is generally referred to as

² Linear problems describe cases where the cost function to minimize is linearly related to input parameters.

crossover because of the way that genetic material crosses over from one chromosome to another (in nature, sexual reproduction allows the creation of genetically different offspring). After a crossover is performed, *mutation* takes place. These mechanisms prevent the genetic algorithms from getting stuck in a local optimum in the solution space.

The implementation of a genetic algorithm can then be described by the pseudo-algorithm in . The algorithm mainly contains a loop that (1) evaluate the *fitness* of all of the chromosomes in the population, (2) create a new population by performing operations such as crossover and mutation on the individuals whose fitness has just been measured, (3) discard the old population and iterate using the new population. One iteration of this loop is usually referred to as a *generation*. The first generation of this process operates on a population of randomly generated chromosomes.

```

Begin GA
  t := 0;                // start with an initial time
  initpopulation P (t); // initialize population of individuals
                        // (usually randomly)
  evaluate P (t);        // evaluate the fitness of all initial
                        // individuals in the population
  While (not done) Do    // test for termination criterion (time,
                        // fitness, etc.)
    t := t + 1;          // increase the time counter
    P' := selectparents P (t); // select a sub-population for
                        // offspring production
    recombine P' (t);    // recombine the "genes" of
                        // selected parents
    mutate P' (t);       // perturb the mated population
    evaluate P' (t);     // evaluate it's new fitness
    P := survive P,P' (t); // select the survivors
  End do
End GA.

```

Figure 1 Genetic Algorithm – Pseudo Code [Kantrowitz 1997]

A genetic algorithm (GA) is an optimization technique that has the ability to find a *global* optimum, and avoid getting stuck in a *local* optimum. However, finding the global optimum is not guaranteed as GA's are based on heuristics. Therefore, for each new problem to be solved, it is necessary to investigate empirically how GA's perform in representative situations. A GA allows an initial population composed of many solutions to the stated problem (called *chromosomes*) to evolve under specified selection rules to a state that optimizes (say, minimizes for the rest of this discussion) a cost function. The parameters involved in the cost function one wants to minimize are first encoded into a chromosome (e.g., a chromosome can be seen as an ordered list of parameter values). There are several possible *encoding* strategies to specify how a solution is represented and their adequacy depends on the problem under consideration (e.g., continuous function optimization or order optimization). In all cases, a chromosome is composed of a number of *genes*. When the initial population evolves from one generation to the other, the best chromosomes are preserved whereas the others are discarded to make room for the new offspring(s). This emulates natural selection that keeps the best-fitted individuals. New offsprings are produced using an evolution operator, named *crossover*, so as to keep the same number of individuals in the next generation. A crossover produces two new chromosomes that share part of the genes coming from two of the best chromosomes in the original population. Another operator, called *mutation*, affects the population by mutating some of the genes of the chromosomes. These operators allow the algorithm to leave the area, in the solution space, of a minimum that may turn out to be local. Though the overall principles make sense, a number of questions arise in practice:

- What is an adequate encoding for the problem?

- What should be the size of the initial population?
- What should be the number of chromosomes kept from one generation to the other?
- What are possible crossover and mutation operators, which are adapted to the selected encoding?
- How do we consider whether we have obtained an acceptable solution?

Let us now consider our integration test problem and answer these questions in context. Recall we need to determine an order of class integration that is at least near-optimal in the sense that it minimizes stubbing complexity (our cost function). The first step is to decide how to encode the solution to our problem. It seems natural in our case to select a *permutation* encoding: Every chromosome is a string of class labels. For a set of classes {A, B, C, D}, a possible chromosome is the sequence (B C A D): The population from which new generations are generated is a set of such chromosomes, e.g., {(B C A D), (B C D A), ...}.

Recall that in the context of GA's, *crossover* and *mutation* operators need to be performed on the original population and subsequent generations until we believe we have obtained an acceptable (possibly optimal) solution. These operators are specific to permutation encoding as they need to ensure that correct sequences are produced, that is sequences with strictly one occurrence of each class. Mutation is implemented by selecting two classes and swapping their positions in the chromosome. For example, highlighting a randomly selected pair of genes (classes in our test order), (B C **D** A) is mutated into (B **D** C A). Crossover is a more complex operator and several crossover operators have been proposed in the literature for permutation encoding [Goldberg 1989]. One of them, which is

used in our study, is described in [Davis 1991] and works as follows: Genes (classes in our context) are randomly selected from a first parent chromosome (test order), their places are found in the other parent, and the remaining genes are copied into the first parent in the same order as they appear in the second parent. This preserves some of the sub-orderings in the original parents while creating some new sub-orderings. For example, if we assume the following two parent chromosomes:

Parent 1: (A B C D)

Parent 2: (D C B A)

If we further assume that A and C are selected in the first parent, we produce a first child chromosome as follows:

- (1) We obtain (A - C -), where A and C have been selected and '-' are gaps;
- (2) We fill in the gaps with B and D in the order they are encountered in parent 2 and obtain (A D C B).

A related point to mention is that we use steady-state replacement strategy [Davis 1991]: A large proportion of the chromosomes in a population should survive to the next generation. The underlying rationale is that we should ensure we preserve the best-found solutions in the next generation. In our particular case, one chromosome (the worst one) gets replaced at each trial after two parent chromosomes are selected. The selection of parents is based on the rank of each parent, in other words, the best-fitted chromosomes are more likely to be selected.

If Parent 2 in the above example is (B C D A), the same offspring as Parent 1 will be obtained. Since genes to crossover are randomly selected, this does not necessarily

happens in the next population. Furthermore, when the length of chromosome increases, the probability decreases.

GA's also require that a number of parameters be set. The first one is the *population size*. It has an impact on the speed of the GA convergence towards an optimal solution and its ability to avoid local optima. Larger population sizes increase the amount of variation present in the initial population at the expense of requiring more cost function evaluations and longer execution times. Typical population sizes in the literature range between 25 and 100 [Davis 1991, Back 1992, Michalewicz 2000]. More specifically, for TSP-related studies [Michalewicz 2000], population size ranges from 30 to 80. Consistent with the literature, we use the upper bound of that interval, that is a size of 100.

Another parameter is the crossover rate, which is the probability that a chromosome will undergo a crossover. Typical rates in the literature for the Traveling Salesperson Problem range from 0.5 to 0.6, consistent with De Jong's simulation results [De Jong 1975]. He reported that, on a number of different optimization problems, simulation results suggested that a crossover rate of roughly 0.6 was a good compromise between a number of performance measures.

Mutation prevents the GA search from falling into local minima, but they should not happen too often or the search will become a random search. The mutation rate is defined as the probability that a chromosome will undergo a mutation. Typically, for the Traveling Salesman Problem, the literature reports rates around 0.15. Recent theoretical work provides a rule of thumb of $1/N$, where N is the number of genes in the chromosomes [Back 1992]. [Tate 1993] argues that for more complex encoding, such as order encoding, high mutation rates can be both desirable and necessary.

2.3 Tailoring and Application Procedure

In this section, we present how the Genetic Algorithm was parameterized based on the specifics of our study and results reported in the literature. We then define what cost functions will be minimized and how we proceed to a priori restrain our search space.

2.3.1 Parameter settings

We selected a mutation rate of 0.15, consistent with the literature on the TSP. We also tried 0.05 and 0.1, to be more in line with the $1/N$ heuristic stated above, but it did not make any significant difference. Regarding the crossover rate, we tried both 0.5 and 0.6, noticing no significant difference and decided to pursue the experiments with 0.5. Our population is made of 100 chromosomes, this being the largest population size in most case studies in the literature.

Last, we allow for 500 trials to converge towards an acceptable result. Preliminary experiments using larger numbers showed that this was sufficient to converge, thus saving us significant computation time.

2.3.2 Measuring Stubbing Complexity

A number of dependencies can be found between classes in a UML class diagram. Compositions and inheritance relationships are considered unbreakable in our strategy as, according to our heuristic, breaking them would likely lead to complex stubs. Such relationships usually entail tight dependencies between the client/parent and server/child classes [Briand 2001]. For remaining dependencies, that is associations, simple aggregations, and usage dependencies, we compute a complexity based on the level of coupling they involve. In this thesis, we measure coupling in two simple, intuitive ways:

1. $A(Dependency)$: The number of attributes *locally* declared³ in the target class when references/pointers to instances of the target class appear in the argument list of some methods in the source class, as the type of their return value, in the list of attributes (data members) of the source class, or as local variables of methods. This complexity measure counts the (maximum) number of attributes that would have to be handled in the stub if the dependency were broken.⁴
2. $M(Dependency)$: The number of methods (including constructors) *locally* declared³ in the target class which are invoked by the source class methods (including constructors). This complexity measure counts the number of methods that would have to be emulated in the stub if the dependency were broken.

More measures could be defined in the future [Briand 1999], but those two measures are enough to illustrate the benefits of our approach. Based on the two measures $A()$ and $M()$, we can then define a cost function to be minimized by the GA. We define the complexity of a dependency as being the geometric average of all complexity measures, i.e., $A()$ and $M()$ in the current study. However, before we compute such an average we need to make sure the resulting complexity is not sensitive to the measurement units of each complexity measure. For example, if the number of methods tend to be on average higher than the number of attributes, $M()$ will have a larger weight than $A()$ in the total complexity of a dependency, though this was not originally intended. To address this issue, we normalize $A()$ and $M()$ so that their range is between 0 and 1. In mathematical terms, for a measure $Cplx()$, we compute its corresponding *normalized* measure $\overline{Cplx()}$. If we assume

³ We do not count inherited attributes (and methods) as this would lead to counting them several times when measuring the stubbing complexity of an order, as described in the $OCplx$ formula below.

that complexity information is represented as a matrix $Cplx(i,j)$ where rows and columns are classes and i depends on j , we have to compute $Cplx_{\min} = \text{Min}\{Cplx(i, j), i, j = 1, 2, \dots\}$ and $Cplx_{\max} = \text{Max}\{Cplx(i, j), i, j = 1, 2, \dots\}$ and then perform the following computation:

$$\overline{Cplx(i, j)} = \frac{Cplx(i, j)}{Cplx_{\max} - Cplx_{\min}}$$

For the two complexity measures we define here, the minimum complexity value is 0 and the equation can be simplified:

$$\overline{Cplx(i, j)} = \frac{Cplx(i, j)}{Cplx_{\max}}$$

Then, based on $A()$ and $M()$, the overall stubbing complexity $SCplx(i,j)$, for a dependency linking a pair of classes (i,j) , can be computed as a weighted geometric average of the normalized measures:

$$SCplx(i, j) = (W_A \cdot \overline{A}(i, j)^2 + W_M \cdot \overline{M}(i, j)^2)^{1/2}$$

where W_A and W_M are weights and $W_A + W_M = 1$.

For a given test order o , a set of d dependencies (i.e., denoted above as pairs of classes) to be broken is identified and an overall complexity can be computed for the order as:

$$OCplx(o) = \sum_{k=1}^d SCplx(k)$$

The measure $OCplx()$ is the *cost function* we try to minimize by using GA. Fully defining such a cost function requires the determination of weights for all complexity measures

⁴ Another measure, which requires more reverse engineering, would count the number of attributes used in the source class.

involved, e.g., W_A and W_M . On the one hand, those weights allow the user to tailor the cost function to his own intuitions about the kinds of complexity that have more bearing on the stubbing effort. On the other hand, this is difficult as setting such weights is a subjective task. This is further discussed in Section 4.4 based on the results presented in Section 4.3.

2.3.3 Constraints

Recall that, according to our strategy, *Inheritance* and *Composition* dependencies cannot be broken. This means the base/container classes must precede child/contained classes in any order that is generated by the GA. In other words, we must optimize the integration order under certain constraints. As we will see, many GA tools such as Evolver [Palisade 1998] allow for the specification of a *precedence table* where the conditions for a new offspring to be acceptable are specified under the form of partial orders. If after applying crossover and mutation operators to selected parent chromosomes an offspring does not fulfill the constraints in the precedence table, the tool backtracks and a new offspring is re-generated until it conforms with the precedence table. The details of such backtracking algorithms vary from tool to tool and are usually proprietary.

Another important set of constraints comes from the fact that we only accept breaking *Association* and *Usage* dependencies that are part of at least one dependency cycle. Integration orders that do not fulfill such constraints are by definition suboptimal. In the previous work of [Briand 2001], where graph-based approaches were explored for optimizing test orders, Tarjan's algorithm [Tarjan 1972] is used to detect strongly connected components (SCC's) in the graph formed by classes and their dependencies. This algorithm can be used again here to detect SCC's and only dependencies between classes part of at least one SCC should be considered for breaking by the GA. While this

type of constraints is in theory not required by the GA to converge towards optimal orders we have observed that the convergence was significantly faster by doing so and fewer generations and less execution time were therefore required. In other words, constraints turned out to be a very efficient and effective way to optimize the GA algorithm.

Another interesting source of constraints comes from the use of design patterns in the class diagram. Let us take as an example the state design pattern [Gamma 1995], which is typically used to design classes with a state dependent behavior. The *context* class, whose state is being modeled, is related through an aggregation relationship to an abstract class *State* (see Figure 2). Every time the context class receives a message, its state is likely to change, thus requiring a message being sent to an instance of one of the subclasses of *State*. In turn, the subclasses instances may invoke *action* methods (e.g., *Action1*) in the context class. In this situation, even if the aggregation is a candidate for breaking when involved in a cycle, it does not seem reasonable to separate the test of classes *Context* and *State* during integration testing. Though this needs to be further studied, a similar conclusion can be drawn when using other design patterns.

Figure 2 State design pattern

In practice, yet another source of constraints stems from the availability of personnel or other development resources (e.g., off-the-shelf or outsourced software). The integration order is going to drive the development order as classes and subsystems need to

be ready and tested when they are to be integrated. For example, though according to some optimal integration order some classes need to be developed before others, no personnel with suitable skills may be available and, therefore, such an integration order is simply not applicable. So it is important that such practical considerations be taken into account when finding optimal integration orders as they may play a key role in determining their feasibility.

All constraints, regardless of their source, may easily be accounted for when using GA tools through the use of the precedence table and backtracking mechanisms.

Chapter 3 Automation Tools

In order to facilitate the experiments with the five case studies, two tools were developed. The first one is simply a tailored version of the Evolver GA tool and was developed using its API. The second tool was implemented to perform reverse engineering of Java code and derive the information required by the GA.

3.1 Tailored GA Tool

In our work, we do not use the user interface provided by Evolver 4.0. Instead, we developed a Microsoft Visual Basic Application (*VBA*) to access the Evolver API to adapt the GA tool for our uses.

The *VBA* module we developed performs the experiments presented in our case studies. The module executes Evolver 4.0 100 times to obtain the distribution of stubbing complexities. The basic control flow is as follows:

1. Set all the parameters and the stop routine used by GA (Section 2.3.1);
2. Define the cost functions (Section 2.3.2);
3. Read input of the weight for each cost factor;
4. Provide an initial randomtest order to GA;
5. Run GA and store the results;
6. Repeat from Step 4 for 100 times.

The tool requires six tables supplied to the tool as individual Excel worksheets. Three of the tables provide dependency information, and these are the dependency matrix (Figure 3), attribute coupling matrix (Figure 4), and method coupling matrix (Figure 5). One table provides constraints information and is the precedence table (Figure 6). One

table provides the weight assignments, and is called the weight table (Figure 7). The final table required provides the adjustable range (test order) for the GA which is highlighted in Figure 8 and the target cell (cell E25 in Figure 8) holds the value of the cost as the tool is running.

ATM Example: Dependency Matrix																					
ID	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21
1																					
2																1				1	
3																1					
4																				1	
5			1																		
6																	1				
7																	1			1	
8																					
9																				1	
10		1	1	1	1	1	1		1	1						1				1	
11								1												1	1
12								1	1		1	1	1	1	1						1
13								1	1	1										1	1
14								1	1	1	1									1	1
15								1	1	1	1									1	1
16								1	1	1	1									1	1
17								1	1	1	1									1	1
18																					
19																1					
20								1	1											1	
21								1	1											1	
22																					
23																					
24																					

Figure 3 Dependency matrix of ATM

Microsoft Excel - atm0326D

File Edit View Insert Format Tools Data Window Help

A1 = ATM Example: Attribute-Coupling Matrix

	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W
1	ATM Example: Attribute-Coupling Matrix																						
2	ID	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	
3	1																2					1	
4	2																1						
5	3																				1		
6	4		2																				
7	5																	2				1	
8	6																	2					
9	7																						
10	8									13	9											1	
11	9								13														8
12	10								13	13				2	2	3	1						4
13	11								13	13	9											2	
14	12								13	13	9											1	2
15	13								13	13	9											1	2
16	14								13	13	9											1	1
17	15								13	13	9											1	1
18	16																						
19	17																						
20	18																					1	
21	19																					1	
22	20																						
23	21																						
24																							

Figure 4 Attribute coupling matrix of ATM

Microsoft Excel - atm0326D

File Edit View Insert Format Tools Data Window Help

A1 = ATM Example: Method-Coupling Matrix

	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W
1	ATM Example: Method-Coupling Matrix																						
2	ID	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	
3	1																0					4	
4	2																0						
5	3																				1		
6	4		1																				
7	5																	2				1	
8	6																	2					
9	7																						
10	8									1	2						1					1	
11	9								7														
12	10								7	2				2	2	2	2						0
13	11								2	1	2												0
14	12								4	4	2											1	0
15	13								4	4	2											0	0
16	14								3	3	2											0	0
17	15								2	3	2											0	0
18	16																						
19	17																	1					
20	18																					1	
21	19																					1	
22	20																						
23	21																						
24																							

Figure 5 Method coupling matrix of ATM

Microsoft Excel - atm0326D

File Edit View Insert Format Tools Data W

A1 = Precedence Table

	A	B	C	D	E	F	G	H	I	J	K
1	Precedence Table										
2	Class ID	Precedent Classes									
3	1	16	20								
4	2	16									
5	3	2	20								
6	4	17									
7	5	17	20								
8	6										
9	7	20									
10	8	1	2	3	4	5	6	7	16	20	
11	9	20	21								
12	10	11	21								
13	11	20	21								
14	12	11	20	21							
15	13	11	20	21							
16	14	11	20	21							
17	15	11	20	21							
18	16										
19	17	16									
20	18	8	9	20							
21	19	8	9	20							
22	20										
23	21										
24											

Figure 6 Precedence table of ATM

Microsoft Excel - atm0326D

File Edit View Insert Format Tools Data Window Help

A1 = Weight Table

	A	B	C
1	Weight Table		
2	VD-Dependence	VA-Attribute	VM-Method
3	0	0.5	0.5
4			

Figure 7 Weight table

Microsoft Excel - atm0326D					
File Edit View Insert Format Tools Data Window Help					
<div> </div>					
TestOrder		= 16			
	A	B	C	D	E
1	Test Order				
2	Class ID	Class			
3	16	GUILayout			
4	6	EnvelopeAcceptor			
5	20	Money			
6	1	ReceiptPrinter			
7	17	QuestionDialog			
8	5	OperatorPanel			
9	4	CardReader			
10	7	CashDispenser			
11	2	Display			
12	3	Keyboard			
13	8	ATM			
14	21	Status			
15	9	Bank			
16	18	ATMMain			
17	19	ATMApplet			
18	11	Transaction			
19	10	Session			
20	12	WithdrawTransaction			
21	13	DepositTransaction			
22	14	TransferTransaction			
23	15	InquiryTransaction			
24					
25			Total Complexity		2.68
26			Attributes needed		
27					
28					
29					

Figure 8 Ajustable range and the cost

The tool obtains input data from the six tables supplied, and then outputs the results of the 100 executions of the GA tool, as shown in Figure 9. Each blue row represents one test order derived from one execution of the GA; the four uncolored cells that follow the blue row are calculated costs using different cost functions.

	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z
1																							D	A	M	OCplx
2	1	20	6	7	16	17	4	1	2	3	5	8	21	9	19	18	11	10	15	14	13	12	7	39	13	2.682
3	2	16	17	6	4	2	20	1	3	5	7	8	21	9	18	11	10	15	13	12	14	19	7	39	13	2.682
4	3	16	2	6	17	4	20	5	7	3	1	8	21	9	18	11	10	19	15	14	12	13	7	39	13	2.682
5	4	21	6	16	17	4	2	20	3	5	1	7	8	9	19	11	10	14	12	13	18	15	7	39	13	2.682
6	5	16	17	4	2	6	20	3	7	1	5	8	21	9	19	18	11	10	13	15	14	12	7	39	13	2.682
7	6	20	21	6	16	17	4	1	5	2	7	3	8	9	18	11	10	15	13	14	19	12	7	39	13	2.682
8	7	20	16	1	17	4	5	2	3	6	7	8	21	9	18	19	11	10	14	13	15	12	7	39	13	2.682
9	8	6	20	16	2	3	17	4	5	1	7	8	21	9	11	10	14	18	15	19	13	12	7	39	13	2.682
10	9	21	16	20	1	17	4	2	6	5	7	3	8	9	18	19	11	10	13	14	12	15	7	39	13	2.682
11	10	6	16	2	17	4	20	5	1	7	3	8	21	9	18	11	10	15	12	14	19	13	7	39	13	2.682
12	11	6	21	20	7	16	2	1	3	17	4	5	8	9	18	11	10	14	19	15	12	13	7	39	13	2.682
13	12	16	2	6	20	1	17	4	7	5	3	8	21	9	19	11	10	14	18	15	13	12	7	39	13	2.682
14	13	16	20	2	1	6	17	4	5	7	3	8	21	9	18	19	11	10	14	13	15	12	7	39	13	2.682
15	14	16	6	2	17	4	20	1	5	3	7	8	21	9	19	11	10	15	13	14	12	18	7	39	13	2.682
16	15	16	21	2	17	20	1	7	6	5	4	3	8	9	18	19	11	10	15	12	13	14	7	39	13	2.682
17	16	6	20	7	16	2	3	1	17	5	4	8	21	9	18	19	11	10	13	14	12	15	7	39	13	2.682
18	17	16	21	2	6	20	3	17	4	5	1	7	8	9	18	11	10	13	19	15	14	12	7	39	13	2.682
19	18	6	16	17	20	1	7	4	5	2	3	8	21	9	19	18	11	10	12	14	13	15	7	39	13	2.682

Figure 9 Results of 100 executions

With the aid of this tool, we can automate data collection and analysis. The time for one execution depends on the size and complexity of the system under analysis, and on a typical personal computer (500MHz, 128 MB) is typically between several minutes, for simple systems such as ATM, to more than twenty minutes for more complex systems, such as BCEL.

3.2 Tools for Analyzing Class Dependencies

We developed reverse engineering tools to analyze the code of the five systems under study. The first tool is referred to as Dependency Analysis (DA) and uses the parser generated using the Java Compiler Compiler (JavaCC) (http://www.webgain.com/products/java_cc/) to read source code and the corresponding post-compilation .class files to identify all the relevant dependency and coupling information between classes. DA accepts the name(s) of the directories in which source

code files and the corresponding .class files are located, and then produces the dependency matrix in text format. In order to be used with Evolver, the output is converted to a *Microsoft Excel* file that can be processed by a *Microsoft Excel VBA* making use of Evolver. DA is not as complete as we would like and is not able, as it is based on static analysis, to capture dependencies due to dynamic bindings, which have to be added manually to the matrices.

The second tool is referred to as Matrix Computation (MC), and it reads the corrected dependency matrix and recomputes the new matrices based on the corrected dependency matrix, the source code, and the corresponding .class files. The basic procedure is shown in Figure 10 below.

Figure 10 Data Flow Diagram for the usage of the DA and MC tools

Chapter 4 Case Studies

We describe in the first subsection our case studies. Then, we apply our strategy and the abovementioned graph-based approaches and compare the results.

4.1 Description of the Case Studies

Five systems are studied and reported in this thesis. All input data the GA needs as well as sample output are shown in appendices A, B, C, D and E. Appendix A.1 is the class diagram and relationship matrix for the ATM system. The relationship matrix summarizes the relationships between the classes in the system. Each class in the system is identified by a *ClassID* which is simply an assigned number for the specific class to make the matrix more readable. In the matrix, *ClassID*'s go across both the x and y axes; any relationship between combinations of classes will be placed in the intersecting cell. The valid relationships that may be placed in the relationship matrix are Cp (*Composition*), I (*Inheritance*), As (*Association and Simple aggregation*) or Us (*Usage*). For example, in Appendix A.1, there is a Us relationship between class 3 and class 2 (class 3 depends on class 2). The *precedence table* describing constraints discussed in Section 2.3.3 is shown in Appendix A.2. Once again, each class in the system is identified by a *ClassID* as in the relationship matrix. Each row contains *ClassID*'s that must precede the class specified in the first column. The dependency matrix, shown in Appendix A.3, is created using the relationship matrix by replacing any specific relationships with a value of 1. Thus, this is a simplified version of the relationship matrix that allows one to see whether or not there are any dependencies between classes of the matrix. The GA needs the dependency matrix to calculate the number of broken dependencies for a test order. The attribute coupling matrix is shown in Appendix A.4. It is derived from the relationship matrix by first removing any I

or Cp relationships, and in the place of the remaining relationships, which may be either Us or As, replacing the names of relationships with the attribute coupling values. The method coupling matrix is shown in Appendix A.5, and is derived in the same manner as the attribute coupling matrix except that the method coupling values are used to replace Us and As relationships instead of attribute coupling values. The derivation of attribute coupling and method coupling values is discussed in Section 2.3.2. Sample test orders and their corresponding complexity values are included in Appendices A.6, A.7, A.8 and A.9 for four different cost functions: i.e., the number of broken dependencies, the number of stubbed attributes, the number of stubbed methods and the weighted geometric average of attribute and method coupling values. Appendices B, C, D and E contain data for the remaining four systems under examination with the exact same structure as Appendix A except for Appendices D and E which do not include class diagrams⁵.

Appendix A contains the data for the first system, which is an Automated Teller Machine (ATM) simulation (the classes connected to hardware devices are missing). The class diagram is made of 21 classes and 67 relationships. The class diagram also contains 30 cycles⁶, involving 8 of the 21 classes (see Figure 24 in Appendix A.1). Appendix B contains the data for the second system, named Ant, which is part of the Jakarta project (<http://jakarta.apache.org/>). Ant creates and maintains open source solutions on the Java platform for distribution to the public at no charge. The Ant system is a Java based build tool similar to the make tool on Unix platforms: It maintains, updates, and regenerates related programs and files according to their dependencies (e.g., compilation units). The

⁵ Class diagrams for the BCEL system and DNS system are highly complex and are not provided as they are not legible, but all the relationships between classes are shown in their corresponding dependency matrices.

⁶ We count here the number of *elementary* circuits (algorithm provided in [Tarjan 1973]), i.e., each class appears once and only once in each circuit, and self loops are not considered.

class diagram contains 25 classes and 83 relationships. The class diagram also contains 654 cycles, involving 12 of the 25 classes (see Figure 25 in Appendix B.1). Appendix C contains the data for the third example, SPM (*Security Patrol Monitoring*) system, which is a course project implemented by Mr. Leeshawn O'Sullivan (<http://www.sce.carleton.ca/courses/94578/project.txt>). It is part of a building security manager system which may detect a security event or a security alarm. It contains 19 classes and 72 relationships. The class diagram contains 1,178 cycles, involving 15 out of the 19 classes (see Figure 26 in Appendix C.1). The fourth example, BCEL (*Byte Code Engineering Library*), also comes from a subproject of Jakarta Project which is intended to give users a convenient tool to analyze, create, and manipulate binary Java class files. We used the *org.apache.bcel.classfile* package of version 5.0 as our example (See <http://jakarta.apache.org/bcel/index.html>). In this example, there are 45 classes, 294 relationships. It is the most complicated among the five systems, containing 416,091 cycles, involving 41 out of 45 classes. Appendix E contains data for the last example which is the DNS system. DNS contains 61 classes and 276 relationships. Although the DNS system has more classes than other examples in this study, as well as a large number of relationships, there are only 16 cycles that involve 10 out of 61 classes in the system. We did not supply the class diagrams for the last two examples because there were either too many edges, or too many classes to be readable.

These five systems were chosen because they were deemed to be of sufficient size and of varying complexity. This allows us to assess the effect of complexity on the performance of GA's by comparing the results of the five case studies. Further details about the systems' structure are provided in Table 1.

System	Classes	Usages	Associations and Simple Aggregations	Compositions	Inheritance	Cycles
ATM	21	39	9	15	4	30
Ant	25	54	16	2	11	654
SPM	19	24	34	10	4	1178
BCEL	45	18	226	4	46	416,091
DNS	61	211	23	12	30	16

Table 1 System Descriptions

For the five systems, Figure 11, Figure 12, Figure 13, Figure 14 and Figure 15 provide the distributions, in the form of histograms, of attribute and method coupling values for each system according to their corresponding attribute coupling matrices and method coupling matrices (Appendices A.4 and A.5, B.4 and B.5, C.4 and C.5, D.4 and D.5 and E.4 and E.5). The height of each bar represents the number of breakable relationships (brkbls in figures) with the x-axis specified attribute or method coupling.

Attribute Coupling

Method Coupling

Figure 11 Distributions of coupling for ATM

Attribute Coupling

Method Coupling

Figure 12 Distributions of coupling for Ant

Attribute Coupling

Method Coupling

Figure 13 Distributions of coupling for SPM

Attribute Coupling

Method Coupling

Figure 14 Distributions of coupling for BCEL

Figure 15 Distributions of coupling for DNS

We summarize the five figures above in Table 2 to show the different coupling levels of these five systems. However, as we will see from the results in Section 4.3, what distinguishes the complexity of the five systems is not only their coupling distributions but also the number of dependency cycles (Table 1) which influence stubbing complexity.

System	Attribute Coupling Range	Average of Attribute Coupling	Method Coupling Range	Average of Method Coupling
ATM	[1, 13]	6.15	[0, 7]	1.79
Ant	[0, 31]	8.36	[0, 14]	2.52
SPM	[1, 16]	7.53	[0, 8]	2.33
BCEL	[0, 20]	1.89	[0, 7]	1.52
DNS	[0, 12]	3.36	[0, 10]	1.46

Table 2 Coupling levels of the five systems

4.2 The Application of Genetic Algorithms

Consistent with Section 1.3, we use the class diagrams or relationship matrices to identify compositions and inheritance relationships to build the precedence tables in Appendices A.2, B.2, C.2, D.2 and E.2 (see Section 2.3.3). The class diagrams or relationship matrices are further analyzed to identify association and usage dependencies and Strongly Connected Components (SCC's) in the graphs formed by the class diagrams. The source code of the system is then analyzed to compute the coupling values of each

usage, association or simple aggregation dependency, as reported in Appendices A.4 and A.5, B.4 and B.5, C.4 and C.5, D.4 and D.5, E.4 and E.5, for each of the five systems.

The first question we want to investigate is whether GA's, as parameterized in this thesis, work as well as deterministic graph algorithms such as the ones described above and experimented with in [Briand 2001]. So we define the cost function as the number of dependencies to be broken (D) in a given test order and run the GA 100 times. Since such algorithms are heuristic, the results may differ from run to run and we therefore check the percentage of times the algorithm converges towards the orders using the graph-based search algorithms. This question is important as it is a preliminary but necessary validation of the approach we describe in this thesis.

The second question we investigate is related to the use of coupling measurement in the cost function ($OCplx$), as described in Section 2.3.2. We want to know whether such measurements make a significant difference in terms of the test orders that are generated, as compared to the orders we obtain with the graph-based algorithm. To answer this question, we compare the orders produced by each of the following cost functions:

- Only the number of broken dependencies is used as a cost function: $SCplx(i,j) \in \{0,1\}$, depending on whether i depends on j or not.
- Attribute and Method coupling are used in turn as the cost function: $SCplx(i,j) = A(i,j)$, and $SCplx(i,j)=M(i,j)$.
- A weighted geometric average of attribute and method coupling ($OCplx$) is used in the definition of $SCplx$.

In other words, these four cost functions are used in turn to produce sets of orders on which we compute the values of the three other functions. If we observe *significant* differences in

OCplx (order stubbed complexity) when only the number of dependencies is used as a cost function and when *OCplx* itself is used, then we can conclude that using coupling measurement may lead to significantly different test orders, involving less coupling to be broken and hopefully leading to lower integration test expenditures. Furthermore, if using *OCplx* as a cost function leads to test orders which are a reasonable compromise between the number of broken dependencies and their corresponding number of attributes and methods, then we can conclude that cost functions as defined in *OCplx* can be useful to achieve practical test orders in the context where one has multiple optimization objectives, as captured by the coupling measures. To address the latter question, we must compare the results obtained with a weighted geometric average and those obtained when using Attribute and Method coupling alone as cost functions. Such a comparison tells us whether a normalized, weighted average used as a cost function can achieve results that are close to the best-found values⁷ of broken dependencies, Attribute coupling, and Method coupling.

Furthermore, since GA's are based on heuristics, it is important we investigate the reliability and repeatability of the results we obtain. Ideally, we would also like to know how far the minima we find are from the actual global minima, which are unknown. This is unfortunately impossible, unless we use simplistic examples. But in light of the complexity of the systems we use as case studies, we may be able to assess whether the results we obtain are plausible and likely to be close to actual global minima.

In order to automate our study, we used a commercial tool: Evolver 4.0 [Palisade 1998]. The choice of tool was made based on a number of criteria. First, we needed the tool to handle permutation encoding. Second we needed to be able to handle possibly large

⁷ As obtained by the GA, which may not necessarily be the global minimum values. Unless otherwise specified, in the remainder of the text, we will use the term minimum with this restriction in mind.

numbers of classes and we needed to make sure the number of genes in the chromosomes could be large enough as many tools have limitations with this respect. Similar requirements for the maximum size of populations had to be considered since when the number of genes grows, one needs to generate larger populations. Other practical features included the random generation of initial populations (in such a way that the diversity of chromosomes is ensured) and the graphical display of the average and minimum cost function value from generation to generation. The latter allows the user to stop the production of new generations when no noticeable improvements can be observed over a sequence of generations. Last, a feature that appeared more and more important over time was the possibility of specifying constraints defining acceptable orders and backtracking mechanisms to account for those constraints when generating new chromosomes.

4.3 Results

The results presented here are based on the precedence table and the dependency and coupling matrices presented in appendices A, B, C, D, and E for the ATM, Ant, SPM, BCEL and DNS systems, respectively. The precedence tables specify constraints so that it is not possible to break inheritance or composition dependencies as well as dependencies that are *not* involved in at least one cycle. Additional constraints (e.g., due to resource management or organizational issues) could be easily added as we would just need to specify, for each class, what other classes must precede it in the integration order. The coupling matrices can simply be extracted by a static code analyzer or a UML model analyzer (e.g., loading XMI files and analyzing the models) if the specified coupling measures can be obtained from such models.

4.3.1 The ATM system

We ran the search algorithm 100 times and found that, in *every* case, we obtain the best order found with the graph-based search algorithm [Briand 2001]: 7 dependencies are broken and stubbed (see Figure 16). What this suggests is that, under similar conditions, GA's can perform as well as deterministic algorithms. This result is of course dependent on the way we tailored and parameterized them.

Table 3 summarizes the overall results we obtain. Appendices A.6, A.7, A.8, and A.9 provide 10 sample orders for each cost function. In the table, columns represent the different ways to measure stubbing complexity, in terms of broken dependencies and stubbed methods and attributes, whereas rows show the different cost functions that can be minimized using GA's, as described in Section 2.1 and 2.2. Highlighted cells show the cases where minimal *OCplx* values are obtained. In some cases, intervals are shown as the results are not consistent across the 100 executions of the GA. We also provide, below the interval, the mean and median values.

From Table 3, we can notice that the best result is consistently obtained for each cost function (no interval). When *D* is used as the cost function, the number of dependencies broken is the best (7) but this is not systematically the case for the values of *A*, *M*, and *OCplx* which are going up to 67, 19, and 4.18 instead of their best-found values of 39, 13, and 2.68, respectively⁸. Therefore, when using the number of dependencies broken as a cost function, we frequently obtain orders that are significantly suboptimal in terms of attributes and methods stubbed. As we see next, this can be fixed by accounting for both *A* and *M* in the cost function, which leads to orders that are systematically the best.

⁸ These minimum values are also determined from executing the Genetic Algorithms 100 times with *OCplx*, *A* and *M*, as cost functions.

When the cost function is M , we systematically obtain the best-found value for M (13) and the number of dependencies, but not for A (and $OCplx$). When the cost function is A , we systematically obtain the best-found value for A (39) and the number of dependencies, but not for M (and $OCplx$).

When using $OCplx$, with $W_M=W_A=0.5$, as the cost function, we obtain a result which is best-found in terms of dependencies, methods, and attributes. Though we may have been lucky in this example, this results suggests that such cost functions, where all coupling measures are given an equal weight and a geometric average is computed, may be a useful, practical means to achieve a reasonable compromise among multiple objectives, as formalized by the coupling measures.

		Stubbing Complexity Values			
		D	A	M	OCplx
Cost Functions	D	7	[39-67]	[13-19]	[2.68-4.18]
		-	53, 53	19, 16.72	3.44, 3.47
	A (OCplx, $W_A=1$)	7	39	[13-19]	[2.68-2.98]
		-	-	19, 17	2.98, 2.88
	M (OCplx, $W_M=1$)	7	[46-67]	13	[2.98-3.88]
		-	61, 62.8	-	3.61, 3.7
	OCplx, $W_A=W_M=0.5$	7	39	13	2.68
		-	-	-	-

Table 3 Summary of Results for the ATM (Appendices A.6, A.7, A.8, A.9)

Comparing the results from the graph-based algorithms mentioned in Section 2.1.2, we can see this strategy works at least as well as, or better than, the three algorithms whose results are shown in Figure 16. Le Traon et al.'s approach does not consistently break the same number of stubs: recall their algorithm has two levels of non-determinism. Also,

some of the orders it generates (4 among the 100 produced) consistently break 4 inheritance relationships and 1 composition relationship.

Figure 16 Distributions of test stubs from the graph-based algorithms for the ATM system

4.3.2 The Ant system

We followed the exact same procedure to experiment with the Ant system which is more complex. Sample orders, for each cost function, are presented in Appendices B.6, B.7, B.8, and B.9, and results are summarized in Table 4. This table has the same structure as Table 3 but one important difference is that, in this case, we do not consistently obtain the best orders for any of the cost functions. Each cell therefore contains the interval of values we obtain across 100 GA executions. Average and median values are also provided below the intervals.

Results show that values for D are always close to be the best-found value (10, the minimum of results determined by the GA and other three graph-based algorithms), regardless of the cost function selected. The results in terms of number of attributes (A) are obviously very good when A is the cost function but also when a weighted function of A and M ($OCplx$ with $W_A=W_M=0.5$) is used (this is highlighted in Table 4 with dark gray cells). However these two functions perform poorly in terms of methods (M). With respect to M , the results are reversed as the results are best when using M or D as cost functions

(see light gray cells in Table 4), and these two functions produce poor results in terms of attributes (A) and the weighted function ($OCplx$).

What we can conclude is that the results when using $OCplx$ as a cost function seem to be mostly driven by A . Though using $OCplx$ leads to very good results in terms of D and A , they are also significantly different from the best-found value with respect to M . After investigating the reasons in more depth, we have come to realize that orders that minimize M (in the [19-26] range) tend to break large numbers of relationships that are associated with the maximum number of attributes and therefore dramatically increase A . Recall from Figure 12 that the attribute coupling distribution for Ant showed a cluster of 12 extremely large values, far above most of the other relationships. No such pattern was observed for method coupling or the coupling distributions of the ATM. We can therefore understand that any order breaking such relationships is unlikely to be optimal when using $OCplx$ as a cost function, even though we normalize and account for A and M with the same weight. In other words, due to the distributions of attribute coupling in Ant, $OCplx$ is strongly driven by A , and optimizing M systematically leads to significantly poorer results with respect to A . Using $OCplx$ yields good results with respect to D and A , but leads to a [29-35] range (with a 29 median) for M , instead of the [19-26] range (with a 22 median) we obtain using M as a cost function. Whether this is an acceptable compromise among the multiple objectives of our optimization (D , A , M) is a subjective call. What we can say objectively is that using $OCplx$ ($W_A=W_M=0.5$) is still a compromise. This is not visible when looking at the intervals but is very clear when looking at the distributions of M values in Figure 17. From left to right, we see the distribution when M , A , and $OCplx$ are used as a cost function. The latter is clearly a compromise between the first two cases.

Figure 17 M Distributions when the cost function is M, A and Ocplx

		Stubbing Complexity Values			
		D	A	M	OCplx
Cost Functions	D	[10-13]	[152-274]	[19-32]	[3.97-6.42]
		11, 10.98	187, 192	26, 24.68	4.63, 4.69
	A (OCplx, $W_A=1$)	[12-14]	[131-137]	[29-37]	[3.59-3.95]
		12, 12.27	131, 132.7	33, 32.7	3.74, 3.74
	M (OCplx, $W_M=1$)	[10-14]	[163-235]	[19-26]	[4.1-5.53]
		13, 12.5	197, 204.4	22, 21.67	4.66, 4.83
	OCplx, $W_A = W_M=0.5$	[12-14]	[131-143]	[29-35]	[3.59-3.86]
		12, 12.16	136, 136.25	29, 29.45	3.59, 3.62

Table 4 Summary of Results for the Ant (Appendices B.6, B.7, B.8, B.9)

Another interesting point is that, because Ant is much more complex with respect to dependencies and cycles than the ATM system, we do not systematically obtain the best results across the 100 GA executions. However, we see that for best-found results (e.g., $A \in [131-143]$ and the cost function is *OCplx*), intervals are also narrow. This implies that the variations observed in the outputs of the GA executions do not hamper its practical use as the orders obtained differ little in terms of stubbing complexity. Furthermore, we have observed that if the GA is executed a small number of times (say 10), it is likely to obtain the minimal bound of the interval at least once. This is illustrated by Figure 18 where the distribution of *OCplx* values is shown for the 100 executions of the GA with *OCplx* as a cost function. We see that though the interval is [3.59 3.93], the minimal bound is very likely to occur.

Figure 18 Distribution of OCplx across 100 GA executions for the Ant system

When results significantly differ from the best-found values (e.g., $A \in [163-235]$ when the cost function is M), intervals tend to be significantly larger as well, thus creating more uncertainty in the results the GA may produce.

To conclude, we see that the results for Ant are not as clear-cut as for ATM. This is to be expected due to the higher complexity of Ant and is also explained by the specific pattern of dependencies within Ant that makes it difficult to completely optimize M and A within the same order. Despite those differences, GA's still appear to be useful to achieve useful compromises between D , A , and M .

The distributions of stubs needed using graph-based algorithms are shown below in Figure 19, and it can be seen that our strategy still works better than those algorithms in terms of class stubs. The data also show that only 8 among the 100 test orders produced by Le Traon et al.'s technique do not break inheritance or composition relationships: 9 of them break 1 inheritance and 1 composition; 61 break 2 inheritances and 1 composition; 13 break 3 inheritances and 1 composition; 2 break 1 inheritance and no composition; 7 break 2 inheritances and no composition.

Figure 19 Distributions of test stubs from graph-based algorithms for the Ant system

4.3.3 The SPM System

Like in the Ant system, we do not consistently obtain best-found orders for any of the cost functions. As shown in Table 5, each cell therefore contains the interval of values we obtain across 100 GA executions. Average and median values are also provided below the intervals.

Results show that test orders with any one of D , A , M and $OCplx$ as the cost function make the other three obtain good results near optimum. This is just a coincidence as the attribute coupling matrix and the method coupling matrix are independent of each other. Results from other examples also explain this.

It may seem strange that test orders with D as the cost function is even worse than those with A as the cost function when the number of broken dependencies is under consideration. It may also seem strange that test orders with M as the cost function is a bit worse than those with $OCplx$ as the cost function when the number of stubbed methods is under consideration (This is highlighted in gray cells in Table 5). This is because the variety of test orders increases along with the complexity of the system (size, the number of dependency cycles); see Appendix C.1 for more details. Deriving test orders optimizing A happens to screen out a small number of test orders which are not “good” for D in this

example. Similarly, deriving test orders optimizing *OCplx* happens to screen out a small number of test orders which are not “good” for *M*.

		Stubbing Complexity Values			
		D	A	M	OCplx
Cost Functions	D	[16-20]	[146-232]	[27-47]	[5.82-9.02]
		16.76, 16	161.88, 149	30.34, 28	6.42, 5.95
	A (OCplx, $W_A=1$)	[16-17]	[146-167]	[26-31]	[5.77-6.57]
		16.07, 16	148.32, 149	27.63, 28	5.91, 5.95
	M (OCplx, $W_M=1$)	[16-21]	[146-227]	[26-37]	[5.77-8.48]
		17.4, 17	158.36, 151	27.72, 27	6.15, 5.9
	OCplx, $W_A=W_M=0.5$	[16-18]	[146-169]	[26-30]	[5.77-6.52]
		16.94, 17	150, 151	26.74, 27	5.87, 5.9

Table 5 Summary of Results for the SPM (Appendices C.6, C.7, C.8, C.9)

As shown in the following distributions of stubs resulting from using the graph-based algorithms (Figure 20), our strategy still works better than those algorithms in terms of stubs. Note that the 100 orders produced by Le Traon et al.’s strategy consistently break 2 compositions.

Figure 20 Distributions of test stubs from graph-based algorithms for the SPM system

4.3.4 The BCEL System

Once again, with the BCEL system, we do not consistently obtain the best-found orders for any of the cost functions. Each cell therefore contains the interval of values we obtain across 100 GA executions. Average and median values are also provided below the intervals, see Table 6.

Results show that GA generated values close to the best-found values for both D and M when D or M is used as the cost function. However, the results also show that GA generated values for both D and M are comparatively poor when A is used as the cost function. It seems that orders that minimize the number of broken dependencies are likely to minimize the number of stubbed methods and vice versa, but they cause significantly more attributes to be stubbed than those using A as the cost function. Meanwhile, orders that minimize the number of stubbed attributes make D or M perform poorer too. This is also the reason why the orders determined by A , D and M cost functions cannot minimize $OCplx$. Unlike the Ant example, the results when using $OCplx$ as a cost function do not seem to be driven by any single factor. But even then, using $OCplx$ leads to reasonably good results in terms of every other cost function. Like in the Ant example, after investigating the attribute coupling distribution and method coupling distribution (Figure 14), we find that there are more relationship clusters with tight attribute coupling than those with tight method coupling. This is why breaking dependencies with M as the cost function highly increases the number of stubbed attributes.

		Stubbing Complexity Values			
		D	A	M	OCplx
Cost Functions	D	[63-70]	[101-143]	[70-87]	[8.59-10.28]
		65.53, 65.5	126.74, 127	76.59, 76	9.22, 9.14
	A (OCplx, $W_A=1$)	[71-82]	[46-55]	[76-105]	[8.06-10.99]
		75.11, 75	49.09, 49	89.68, 89.5	9.45, 9.44
	M (OCplx, $W_M=1$)	[63-72]	[49-144]	[67-84]	[8.08-10.14]
		67.83, 68	120.88, 127	74.43, 75	8.95, 8.96
	OCplx, $W_A=W_M=0.5$	[69-77]	[46-96]	[70-84]	[7.56-9.08]
		72.19, 72	56.55, 53	78, 77	8.33, 8.29

Table 6 Summary of Results for the BCEL (Appendices D.6, D.7, D.8, D.9)

The result intervals for this example are much wider than those found in the Ant example for each cost function (e.g., $A \in [46-96]$, $OCplx \in [7.56-9.08]$, and the cost function is $OCplx$). Once again, the complexity of the system (size, the number of dependency cycles) described in Table 1 determines the big variety.

After some investigation of the $OCplx$ distribution when $OCplx$ is used as a cost function (see Figure 21), we found that there is a 15% chance that the $OCplx$ will be in the range of between the best-found result of $OCplx$ and 5% more than the best-found result ([7.56, 7.94]). Unlike in the Ant system, larger number of executions (say, 20) is required to obtain acceptable solutions. Since it takes less than 20 minutes for each execution of the tool with the BCEL system on a typical personal computer (500MHz, 128 MB), the slightly larger number of executions is acceptable.

Figure 21 Distribution of *OCplx* for the BCEL system

This system is the most complex system both in terms of size and number of dependency cycles. It is also expected that systems of this complexity are typical in the real world. Therefore, 20 executions of the GA are suggested as the standard number of executions in the methodology to get an acceptable test order for systems of simple to typical complexity.

We also compare this approach with the three graph-based algorithms, and their results are shown in Figure 22 below. In the BCEL system, the result of the Le Traon algorithm is comparatively better than all other algorithms. As we mentioned in Section 2.1.2, however, this algorithm does not account for the impact of different stub types. Examining the result, we found that 22 inheritance relationships are broken, which causes increased stubbing complexity.

Figure 22 Distributions of test stubs from graph-based algorithms for the BCEL system

4.3.5 The DNS System

There are more classes in the DNS system than other examined systems. Although there are more classes in this example, this system has the least number of cycles among our five examples. Consequently, there is not much variety in the result, just like in the ATM example. The results from Appendices E.6, E.7, E.8 and E.9 are summarized in Table 7. Notice that the best-found result is consistently obtained for each cost function (no interval). When D is used as the cost function, the number of dependencies broken is the best (6). However, using A as the cost function causes an increased number of broken dependencies from the low of 19 to the high of 28, while its $OCplx$ is increased as well from a value of 1.47 to a value of 1.99. The values generated using the M cost function always remains at the best-found value. By examining the method coupling distribution, shown in Figure 15, we realize that the result is always the best because most method coupling has the same values. For instance, there are 113 out of 234 (see Table 1) breakable relationships with only one method coupling between each other, and 56 out of 234 breakable relationships with 0 method coupling between each other. The suboptimal results for A may be explained with attribute coupling. The number of relationships with the same attribute coupling values in this system is less than those with the same method coupling values so that the result interval for A is narrow. Therefore, when using the number of dependencies broken as a cost function, we frequently obtain orders that are suboptimal in terms of attributes stubbed. As we see next, this can be fixed by accounting for both A and M in the cost function, which leads to orders that are consistently best-found. When the cost function is M , we consistently obtain the best-found result for M (11) as well as the best-found result in terms of the number of dependencies for A and $OCplx$.

When the cost function is A , we consistently obtain the best-found result for A (39) as well as the best-found result for M , but not for the number of dependencies and $OCplx$. When using $OCplx$, with $W_M=W_A=0.5$, as the cost function, we obtain a result which is almost the same as that when using A as cost function. Although we are not as lucky as in the ATM example, there is actually only one test order leading to 22 attribute stubs (the interval is from 19 to 22). This result once again suggests that such cost functions, where all coupling measures are given an equal weight and a geometric average is computed, may be a useful, practical method to achieve a reasonable compromise among multiple objectives, as formalized by the coupling measures.

		Stubbing Complexity Values			
		D	A	M	OCplx
Cost Functions	D	6	[19-28]	11	[1.47-1.99]
		-	23.62, 22	-	1.73, 1.64
	A (OCplx, $W_A=1$)	[6-7]	19	11	1.47
		6.88, 7	-	-	-
	M (OCplx, $W_M=1$)	[6-7]	[19-28]	11	[1.47-1.99]
		6.9, 7	23.74, 22	-	1.74, 1.64
	OCplx, $W_A=W_M=0.5$	[6-7]	[19-22]	11	1.47
		6.89, 7	19.03, 19	-	-

Table 7 Summary of Results for the DNS System (Appendices E.6, E.7, E.8, E.9)

Last, we compare results from other algorithms with those from our strategy, as shown in Figure 23. Once again, our strategy works as well as, or better than the graph-based algorithms. The 100 orders produced by Le Traon et al.'s strategy consistently break at least 2 inheritance relationships.

Figure 23 Distributions of test stubs from graph-based algorithms for the DNS system

4.4 Determining Weights in OCplx

We have seen that a practical issue when using a multiple objectives cost function is to determine appropriate weights. Equal weights may be adequate but higher flexibility may be required in some cases. If the number of coupling measures is small it is always possible to perform an exhaustive search to determine optimal weights. If the number of coupling measures is too large to consider such an option, a practical procedure could be as follows:

- 1: Run the Genetic Algorithm as parameterized in this thesis, using each coupling measure (objective) independently as a cost function, in order to identify their minimal value for a set of classes.
- 2: Assign an equal weight to all coupling measures and use the geometric average as the cost function.
- 3: Check whether, for each objective (coupling measure), the value of *OCplx* is *reasonably* close to the minimal value (what is reasonable is of course subjective).
- 4: If this is the case, keep the weights as they are. If a particular coupling measure shows a departure from the minimal value that is deemed too large, then increase its relative weight and rerun the Genetic Algorithm.

5: Repeating this procedure (from 3) will hopefully converge towards acceptable values for each coupling measure, thus leading to acceptable test order.

Another possibility is of course to assign weights based on some other rationale, that is something that is an estimation (e.g., based on expert opinion) of the relative cost of each measurement unit of each coupling measure. But it is difficult to envisage at this point how this could be achieved. The procedure described above can be automated and helps achieve a balanced compromise between all coupling measures, each of them representing an optimization objective.

Chapter 5 Conclusions

It has been shown in this thesis that, if we want to use more sophisticated and realistic criteria than those used so far in the literature, to optimize integration orders, we cannot keep relying on graph-based algorithms as described in [Briand 2001]. As an alternative, a novel solution based on Genetic Algorithms and coupling measurement that seems to address the needs is proposed. Coupling measurement helps us differentiate stubs of varying complexity and Genetic Algorithms allow us to minimize complex cost functions based on such complexity measurement.

We select a specific type of Genetic Algorithms (with permutation encoding), and focus on the best ways to use that Genetic Algorithm so as to fit our application. We also assess the results of our strategy on five real case studies of non-trivial size and complexity. Results are very encouraging as they consistently show that genetic algorithms reach acceptable compromises among multiple objectives (i.e., coupling measures). They also provide a lot of flexibility in terms of allowing the user to optimize the integration order under constraints. This is important in practice as integration orders have an impact on development and unit testing orders and the latter may be constrained by available resources shared with other projects.

Results also show that, as dependencies and cycles become more complex, genetic algorithms tend to produce less consistent results from run to run. This is to be expected as they are based on heuristics and those heuristics are expected to be sensitive to the complexity of the ordering problem to solve. As mentioned in Section 4.4, because of the uncertainty associated with the GA heuristics, we suggest that a GA be used a small number of times to ensure a near-optimal order will be identified. On the case studies we

investigated, no more than 20 executions are necessary to obtain a high probability to generate a near-optimal order (i.e., within the 5% lowest quantile of the stubbing cost distribution).

A limitation to consider is the fact that case studies as the one we present here are performed without knowing the actual global minimum, as this cannot be derived with problems of such complexity. To alleviate the problem, we observe the best-found values when using each coupling measure in isolation as a cost function and look at the optimality and consistency of the results we obtain when running the Genetic Algorithm a large number of times with complex, multi-objectives cost functions. Furthermore, in light of the complexity of the case studies (as measured by dependencies and cycles), the minima we obtain seem realistically low comparing with results from three existing graph-based algorithms. Furthermore, with respect to the number of dependencies that are “broken”, we compared our results to those of graph-based algorithms and show we consistently obtain equivalent or better results.

Another result of this thesis is the comparison of the costs of test orders derived by the three existing graph-based algorithms. Their advantages and drawbacks have already been reported by means of an analytical evaluation, and the observations made are here confirmed using the five selected case studies.

Future work includes:

- Performing large-scale simulations, with known minima, in order to confirm the generality of the results we have obtained here with case studies.
- Devising methods to define more realistic fitness/evaluation functions

References

- [Back 1992] T. Back, "The Interaction of Mutation Rate, Selection, and Self-Adaptation Within a Genetic Algorithm," *Proc. Parallel Problem Solving from Nature (PPSN'92)*, Brussels (Belgium), pp. 85-94, September 28-30, 1992.
- [Beizer 1990] B. Beizer, *Software Testing Techniques*, Van Nostrand Reinhold, New York, 2nd Ed., 1990.
- [Binder 1999] R. V. Binder, *Testing Object-Oriented Systems - Models, Patterns, and Tools*, Addison-Wesley, 1999.
- [Briand 1999] L. Briand, J. Daly and J. Wuest, "A Unified Framework for Coupling Measurement in Object-Oriented Systems," *IEEE Transactions on Software Engineering*, vol. 25 (1), pp. 91-121, 1999.
- [Briand 2001] L. C. Briand, Y. Labiche and Y. Wang, "Revisiting Strategies for Ordering Class Integration Testing in the Presence of Dependency Cycles," Carleton University, Technical Report SCE-01-02, <http://www.sce.carleton.ca/faculty/briand/SCE-01-02.pdf>, 2001, a short version appeared in the proceedings of IEEE ISSRE 2001.
- [Chambers 1995] L. Chambers, *Practical Handbook of Genetic Algorithms*, vol. 1, CRC Press, 1995.
- [Davis 1991] L. Davis, *Handbook of Genetic Algorithms*, Van Nostrand Reinhold, 1991.
- [De Jong 1975] K. A. De Jong, *Analysis of the Behavior of a Class of Genetic Adaptive Systems*, Ph.D. Dissertation, The University of Michigan, 1975.
- [Deo 1974] N. Deo, *Graph Theory With Applications to Engineering and Computer Science*, 1974.
- [Gamma 1995] E. Gamma, R. Helm, R. Johnson and J. Vlissides, *Design Patterns - Elements of Reusable Object-Oriented Software*, Addison-Wesley, 1995.
- [Goldberg 1989] D. E. Goldberg, *Genetic Algorithms in Search, Optimization & Machine Learning*, Addison Wesley, 1989.
- [Jones 2001] B. F. Jones (Ed.), *Special Issue on Metaheuristic Algorithms in Software Engineering*, Information and Software Technology, vol. 43 (14), 2001.
- [Kantrowitz 1997] Mark Kantrowitz, Genetic Algorithms FAQ, [http:// www-2.cs.cmu.edu/Groups/AI/html/gaqs/ai/genetic/top.html](http://www-2.cs.cmu.edu/Groups/AI/html/gaqs/ai/genetic/top.html)
- [Kung 1995] D. Kung, J. Gao, P. Hsia, J. Lin and Y. Toyoshima, "Class Firewall, test order, and regression testing of object-oriented programs," *Journal of Object-Oriented Programming*, vol. 8 (2), pp. 51-65, 1995.
- [Labiche 2000] Y. Labiche, P. Thévenod-Fosse, H. Waeselynck and M.-H. Durand, "Testing Levels for Object-Oriented Software," *Proc. 22nd IEEE*

- International Conference on Software Engineering (ICSE'2000)*, Limerick (Ireland), pp. 136-145, June, 2000.
- [Lawler 1985] E. L. Lawler, J. K. Lenstra, A. H. G. Rinnooy Kan and D. B. Shmoys, *The Traveling Salesman Problem*, Wiley, 1985.
- [Le Traon 2000] Y. Le Traon, J.-M. Jézéquel and P. Morel, "Efficient Object-Oriented Integration and Regression Testing," *IEEE Trans. on Reliability*, vol. 49 (1), pp. 12-25, 2000.
- [Michalewicz 2000] Z. Michalewicz, D.B. Fogel, *How to Solve It: Modern Heristics*, Springer, 2001
- [Palisade 1998] Palisade, *Evolver, the Genetic Algorithm Super Solver*, 1998.
- [Tai 1999] K.-C. Tai and F. J. Daniels, "Interclass Test Order for Object-Oriented Software," *Journal of Object-Oriented Programming*, vol. 12 (4), pp. 18-25, 35, 1999.
- [Tarjan 1972] R. Tarjan, "Depth-First Search and Linear Graph Algorithms," *SIAM Journal on Computing*, vol. 1 (2), pp. 146-160, 1972.
- [Tarjan 1973] R. Tarjan, "Enumeration of the Elementary Circuits of a Directed Graph," *SIAM Journal on Computing*, vol. 2 (3), pp. 211-216, 1973.
- [Tate 1993] D. M. Tate and A. E. Smith, "Expected Allele Coverage and the Role of Mutation in Genetic Algorithms," *Proc. Fifth International Conference on Genetic Algorithms*, pp. 31-37, 1993.
- [Thangiah 1992] S. R. Thangiah and K. E. Nygard, "School bus routing using genetic algorithms," *Proc. Applications of Artificial Intelligence SPIE Conference*, pp. 387-398, 1992.

APPENDIX A.1: CLASS DIAGRAM AND RELATIONSHIP MATRIX FOR ATM

Figure 24 Class Diagram of the ATM System

ID	Class
1	ReceiptPrinter
2	Display
3	Keyboard
4	CardReader
5	OperatorPanel
6	EnvelopeAcceptor
7	CashDispenser
8	ATM
9	Bank
10	Session
11	Transaction
12	WithdrawlTransaction
13	DepositTransaction
14	TransferTransaction
15	InquiryTransaction
16	GUILayout
17	QuestionDialog
18	ATMMain
19	ATMApplet
20	Money
21	Status

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21
1																Us				Us	
2																Us					
3		Us																		Us	
4																	Us				
5																	Us			Us	
6																					
7																				Cp	
8	Cp	Cp	Cp	Cp	Cp	Cp	Cp		As	Us						Us				Us	
9								Us												Cp	Us
10								As	As		Cp	Us	Us	Us	Us						Us
11								As	As	As										Cp	Us
12								Us	Us	Us	I									As	Us
13								Us	Us	Us	I									As	Us
14								Us	Us	Us	I									As	Us
15								Us	Us	Us	I									Us	Us
16																					
17																Us					
18								Cp	Cp											Us	
19								Cp	Cp											Us	
20																					
21																					

Description: **As:** Associations and simple Aggregations, **Cp:** Compositions, **Us:** Use dependencies, **I:** Inheritance.

APPENDIX A.2: PRECEDENCE TABLE FOR ATM

Class ID	Precedent Classes								
1	16	20							
2	16								
3	2	20							
4	17								
5	17	20							
6									
7	20								
8	1	2	3	4	5	6	7	16	20
9	20	21							
10	11	21							
11	20	21							
12	11	20	21						
13	11	20	21						
14	11	20	21						
15	11	20	21						
16									
17	16								
18	8	9	20						
19	8	9	20						
20									
21									

APPENDIX A.3: DEPENDENCY MATRIX FOR ATM

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21
1																1				1	
2																1					
3		1																		1	
4																	1				
5																	1			1	
6																					
7																				1	
8	1	1	1	1	1	1	1		1	1						1				1	
9								1												1	1
10								1	1		1	1	1	1	1						1
11								1	1	1										1	1
12								1	1	1	1									1	1
13								1	1	1	1									1	1
14								1	1	1	1									1	1
15								1	1	1	1									1	1
16																					
17																1					
18								1	1											1	
19								1	1											1	
20																					
21																					

APPENDIX A.4: ATTRIBUTE COUPLING MATRIX FOR ATM

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21
1																2				1	
2																1					
3		2																		1	
4																	2				
5																	2			1	
6																					
7																					
8									13	9						0				1	
9								13													8
10								13	13			2	2	3	1						4
11								13	13	9											2
12								13	13	9										1	2
13								13	13	9										1	2
14								13	13	9										1	1
15								13	13	9										1	1
16																					
17																0					
18																				1	
19																				1	
20																					
21																					

APPENDIX A.5: METHOD COUPLING MATRIX FOR ATM

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21
1																0				4	
2																0					
3		1																		1	
4																	2				
5																	2			1	
6																					
7																					
8									1	2						1				1	
9								7													0
10								7	2			2	2	2	2						0
11								2	1	2											0
12								4	4	2										1	0
13								4	4	2										0	0
14								3	3	2										0	0
15								2	3	2										0	0
16																					
17																1					
18																				1	
19																				1	
20																					
21																					

**APPENDIX A.6: TEST ORDERS OBTAINED WITH NUMBER OF BROKEN
DEPENDENCIES AS COST FUNCTION (ATM)**

Test Orders									
1	2	3	4	5	6	7	8	9	10
20	21	6	21	6	6	6	16	21	6
7	6	20	20	21	21	21	17	20	20
6	20	7	9	20	16	16	4	9	16
16	7	16	6	16	17	17	20	6	1
1	9	2	7	17	4	4	1	16	7
2	16	3	16	9	2	2	7	1	17
17	1	1	1	1	20	20	5	17	4
5	17	17	2	2	5	7	21	4	5
4	5	5	3	3	7	5	9	7	2
21	4	4	17	7	9	3	6	2	3
9	2	8	4	4	3	1	2	5	8
3	3	21	5	5	1	9	3	3	21
8	8	9	8	8	8	8	8	8	9
18	19	19	18	19	19	18	19	19	19
19	18	18	11	18	11	19	11	18	18
11	11	11	14	11	13	11	14	11	11
12	13	14	15	14	10	12	10	12	15
14	15	15	13	12	15	10	12	14	10
10	10	13	12	10	12	15	13	13	12
15	14	12	10	15	14	14	18	10	13
13	12	10	19	13	18	13	15	15	14
Cost Function: Number of Dependencies	7	7	7	7	7	7	7	7	7
Attributes (min = 39)	52	54	67	67	52	46	46	45	59
Methods (min = 13)	19	19	13	19	19	19	19	19	13

APPENDIX A.7: TEST ORDERS OBTAINED WHEN USING THE NUMBER OF STUBBED ATTRIBUTES AS THE COST FUNCTION (ATM)

Test Orders									
1	2	3	4	5	6	7	8	9	10
6	21	6	20	16	20	20	16	6	20
20	6	20	6	2	16	7	17	21	7
16	20	16	7	6	2	6	4	20	16
1	7	17	16	20	17	16	6	9	2
7	16	2	1	3	1	2	2	7	1
2	1	1	2	1	21	3	20	16	6
17	2	3	21	7	5	1	3	1	3
4	3	4	17	21	4	17	7	17	17
5	17	5	5	9	6	5	5	5	5
3	5	7	4	17	3	4	21	4	4
8	4	8	9	5	9	8	1	2	8
21	9	21	3	4	7	21	8	3	21
9	8	9	8	8	8	9	9	8	9
19	18	19	18	19	19	18	18	11	19
18	19	18	11	11	18	19	11	10	18
11	11	11	10	10	11	11	10	14	11
10	10	10	12	12	10	10	14	12	10
12	14	15	15	14	13	15	13	15	12
14	13	14	14	15	14	14	15	13	13
13	15	12	13	13	12	12	12	19	14
15	12	13	19	18	15	13	19	18	15
Cost Function: Number of Attributes	39	39	39	39	39	39	39	39	39
Methods (min = 13)	13	19	13	19	19	19	13	13	19
Dependencies (min = 7)	7	7	7	7	7	7	7	7	7

APPENDIX A.8: TEST ORDERS OBTAINED WHEN USING THE NUMBER OF STUBBED METHODS AS THE COST FUNCTION (ATM)

Test Orders									
1	2	3	4	5	6	7	8	9	10
21	6	6	16	20	16	20	20	20	16
20	21	16	20	6	2	7	7	6	2
7	16	20	1	16	6	6	16	7	20
6	17	1	6	17	20	16	1	16	7
16	2	7	7	4	7	17	2	2	1
2	4	2	2	1	3	5	6	3	3
17	20	3	3	5	1	1	3	17	6
4	1	17	17	7	17	2	17	5	17
3	5	4	5	2	4	3	4	4	5
1	7	5	4	3	5	4	5	1	4
5	3	8	8	8	8	8	8	8	8
8	8	21	21	21	21	21	21	21	21
9	9	9	9	9	9	9	9	9	9
18	18	18	18	19	19	18	18	18	19
19	11	19	11	18	18	19	19	19	18
11	15	11	14	11	11	11	11	11	11
10	14	12	12	12	13	14	12	10	14
14	12	13	19	10	10	13	13	13	12
12	13	14	15	14	15	15	15	15	15
13	10	10	13	13	12	10	10	12	13
15	19	15	10	15	14	12	14	14	10
Cost function: Methods (min = 13)	13	13	13	13	13	13	13	13	13
Attributes (min = 39)	39	67	59	67	46	46	60	61	39
Dependencies (min = 7)	7	7	7	7	7	7	7	7	7

**APPENDIX A.9: TEST ORDERS OBTAINED WHEN USING A WEIGHTED
GEOMETRIC AVERAGE OF ATTRIBUTE AND METHOD COUPLING
MEASURES AS THE COST FUNCTION (ATM)**

	Test Orders									
	1	2	3	4	5	6	7	8	9	10
	6	16	16	6	16	6	20	16	6	16
	16	17	2	20	17	16	6	17	20	21
	2	4	6	16	4	2	7	21	7	20
	17	20	20	17	20	17	16	6	16	6
	4	7	1	4	6	4	1	4	2	2
	20	1	17	7	1	20	17	20	1	17
	5	2	3	1	21	3	5	1	3	4
	3	3	4	5	5	5	4	5	17	3
	1	5	7	2	2	7	2	7	4	1
	7	6	5	3	3	1	3	2	5	5
	8	8	8	8	7	8	8	3	8	7
	21	21	21	21	8	21	21	8	21	8
	9	9	9	9	9	9	9	9	9	9
	11	19	19	19	19	18	19	19	18	18
	18	18	11	18	18	11	11	11	19	19
	10	11	10	11	11	10	10	10	11	11
	12	10	15	10	10	15	13	15	10	10
	19	14	12	13	14	13	18	12	15	12
	15	13	13	14	13	19	15	13	13	15
	13	15	14	15	12	12	14	18	12	14
	14	12	18	12	15	14	12	14	14	13
Cost Function: OCplx ($W_M=W_A=0.5$)	2.682	2.682	2.682	2.682	2.682	2.682	2.6818	2.682	2.682	2.6818
Attributes (min = 39)	39	39	39	39	39	39	39	39	39	39
Methods (min = 13)	13	13	13	13	13	13	13	13	13	13
Dependencies (min = 7)	7	7	7	7	7	7	7	7	7	7

APPENDIX B.1: CLASS DIAGRAM AND RELATIONSHIP MATRIX FOR ANT

Figure 25 Class Diagram of the Ant System

ID	Class
1	AntClassLoader
2	BuildEvent
3	BuildException
4	BuildListener
5	BuildLogger
6	DefaultLogger
7	DemuxOutputStream
8	DirectoryScanner
9	FileScanner
10	IntrospectionHelper
11	Launcher
12	Location
13	Main
14	NoBannerLogger
15	PathTokenizer
16	Project
17	ProjectComponent
18	ProjectHelper
19	RuntimeConfigurable
20	Target
21	Task
22	TaskAdapter
23	TaskContainer
24	UnknownElement
25	XmlLogger

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25
1			Us	I												As									
2																As				As	As				
3												As													
4		Us																							
5				I																					
6		Us	Us		I											Us									
7																As									
8			Us						I																
9																									
10			Us	I								Us				Us					Us		Us		
11	Us																								
12																									
13			Us	Us	Us											Us		Us		Us					
14		Us				I																			
15																									
16		Us	Us	As											Us		Us			Cp	As	Us			
17																As									
18			Us							Us		Us				As			Us	Us	Us	Us	Us	Us	Us
19			Us													Us		Us							
20			Us													As		Us	Us		Cp		I	Us	
21			Us									As				Us	I		As	As					
22			Us													Us					I				
23																					Us				
24			Us							Us						Us			Us		I	Us	Us		
25		Us	Us	I												Us				As	As				

Description: **As:** Associations and simple Aggregations, **Cp:** Compositions, **Us:** Use dependencies, **I:** Inheritance.

APPENDIX B.2: PRECEDENCE TABLE FOR ANT

Class ID	Precedent Classes					
1	4	3	16			
2						
3	12					
4						
5	4					
6	5	2	3	16		
7	16					
8	9	3				
9						
10	4	3	12			
11	1					
12						
13	3	4	5	16	18	20
14	6	2				
15						
16	20	3	15			
17						
18	3	12				
19	3					
20	21	23	3			
21	17	3	12			
22	21	3				
23						
24	21	3				
25	4	2	3	16	20	21

APPENDIX B.3: DEPENDENCY MATRIX FOR ANT

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25
1			1	1												1									
2																1				1	1				
3												1													
4		1																							
5				1																					
6		1	1		1											1									
7																1									
8			1						1																
9																									
10			1	1								1				1					1		1		
11	1																								
12																									
13			1	1	1											1		1		1					
14		1				1																			
15																									
16		1	1	1											1		1			1	1	1			
17																1									
18			1							1		1				1			1	1	1	1	1	1	1
19			1													1		1							
20			1													1		1	1		1		1	1	
21			1									1				1	1		1	1					
22			1													1					1				
23																					1				
24			1							1						1			1		1	1	1		
25		1	1	1												1				1	1				

APPENDIX B.4: ATTRIBUTE COUPLING MATRIX FOR ANT

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25
1			2													31									
2																31				7	6				
3												3													
4		5																							
5																									
6		5	2													3									
7																31									
8			2																						
9																									
10			2									0				31					0		0		
11	15																								
12																									
13			2	0	0											31		6		7					
14		5																							
15																									
16		5	2	0											3		1				6	1			
17																31									
18			2							8		3				31			5	7	6	1	0	3	
19			2													31		6							
20			2													31		6	5					3	
21			2									3				31			5	7					
22			2													31									
23																					6				
24			2							8						31			5			1	0		
25		5	2													3				7	6				

APPENDIX B.5: METHOD COUPLING MATRIX FOR ANT

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25
1			1													4									
2																1				1	2				
3												2													
4		3																							
5																									
6		3	1													0									
7																2									
8			1																						
9																									
10			2									0				3					0		0		
11	4																								
12																									
13			2	0	4											11		1		2					
14		4																							
15																									
16		5	2	7											3		1				5	1			
17																2									
18			7							6		1				14			4	5	6	1	1	2	
19			0													1		3							
20			1													5		1	2					1	
21			3									1				4			2	0					
22			2													1									
23																					0				
24			1							2						3			3			1	1		
25		9	1													0				1	1				

**APPENDIX B.6: TEST ORDERS OBTAINED WITH NUMBER OF BROKEN
DEPENDENCIES AS COST FUNCTION (ANT)**

Test Orders									
1	2	3	4	5	6	7	8	9	10
4	12	12	17	9	9	17	15	15	15
17	17	3	12	4	17	4	17	12	9
15	3	19	4	17	12	12	4	17	12
12	19	4	3	5	4	3	12	9	3
3	21	15	5	12	5	19	3	3	17
19	9	17	9	3	3	21	19	19	19
21	23	21	19	15	8	23	21	4	21
22	4	23	21	19	19	22	23	5	4
23	8	20	22	8	21	15	20	8	23
20	15	16	23	21	23	20	16	21	8
16	22	1	20	22	22	16	22	23	20
2	20	5	8	23	20	1	7	20	16
9	16	7	15	20	2	7	9	16	22
5	5	9	16	16	15	2	1	7	2
10	2	8	1	1	16	10	2	1	5
8	1	11	11	10	25	25	10	2	10
24	7	10	7	24	7	5	5	6	7
1	10	2	10	2	1	9	6	10	25
18	25	6	2	6	11	24	24	22	6
25	24	14	24	14	10	18	25	24	14
13	11	25	6	11	6	13	14	11	1
6	6	22	14	25	14	8	18	18	24
14	14	24	18	18	24	11	13	13	11
11	18	18	13	13	18	6	8	25	18
7	13	13	25	7	13	14	11	14	13
Cost Function: Number of Dependencies	11	11	11	11	11	11	11	11	11
Attributes (min = 131)	187	187	157	187	187	213	187	157	157
Methods (min = 19)	26	26	26	26	26	22	26	26	26

APPENDIX B.7: TEST ORDERS OBTAINED WHEN USING THE NUMBER OF STUBBED ATTRIBUTES AS THE COST FUNCTION (ANT)

Test Orders									
1	2	3	4	5	6	7	8	9	10
15	17	15	9	17	9	15	15	12	17
12	9	12	15	12	17	17	17	3	15
17	12	3	17	3	15	12	12	17	12
3	3	17	12	9	12	3	3	21	3
21	15	21	3	21	3	21	21	15	9
23	21	23	21	15	21	23	23	23	21
20	23	20	23	23	23	20	20	20	23
16	20	16	20	20	20	16	16	16	20
9	16	7	16	16	16	9	22	7	16
19	19	19	8	19	22	2	2	2	22
2	2	2	7	2	2	7	19	19	2
4	7	4	2	4	4	8	4	4	4
10	4	10	4	8	10	4	7	10	1
7	8	22	25	1	5	22	25	5	10
25	22	24	5	11	8	25	9	1	8
5	10	5	1	22	19	1	8	22	11
6	24	18	11	10	25	11	10	25	7
22	5	13	22	24	24	19	1	6	19
14	6	9	19	5	18	10	24	24	25
24	25	8	10	6	13	5	18	14	24
1	14	1	6	25	1	6	5	11	18
11	1	11	14	18	11	24	6	18	5
18	11	6	24	13	6	18	14	13	6
13	18	14	18	7	14	13	13	9	13
8	13	25	13	14	7	14	11	8	14
Cost Function: Number of Attributes	131	131	131	131	131	131	131	131	131
Methods (min = 19)	33	33	33	33	33	33	33	33	33
Dependencies (min =10)	12	12	12	12	12	12	12	12	12

APPENDIX B.8: TEST ORDERS OBTAINED WHEN USING THE NUMBER OF STUBBED METHODS AS THE COST FUNCTION (ANT)

Test Orders									
1	2	3	4	5	6	7	8	9	10
12	23	17	9	15	12	23	9	23	15
9	17	23	2	2	23	15	2	17	2
15	12	12	23	23	2	12	23	12	23
3	9	3	15	17	9	2	15	3	12
19	3	19	12	4	4	3	17	21	4
17	19	21	4	5	3	19	12	2	5
21	21	20	17	12	5	4	4	15	9
23	20	2	3	3	19	17	5	4	3
20	2	4	8	19	17	5	3	5	19
2	4	5	5	21	21	21	8	19	17
4	5	9	19	22	22	20	19	22	21
8	15	15	21	20	20	16	21	20	20
16	16	16	20	16	15	25	22	16	16
25	22	1	16	7	16	7	20	7	1
22	8	11	10	1	8	10	16	1	25
1	1	25	25	6	25	6	1	11	7
7	7	7	7	10	7	22	11	9	6
10	10	6	6	14	6	24	6	8	8
24	24	14	14	24	14	18	14	10	11
5	6	22	22	11	10	14	10	6	10
6	18	10	24	25	24	13	25	14	22
14	13	24	1	18	18	9	24	24	24
18	14	8	11	13	13	8	18	18	18
13	25	18	18	9	1	1	13	13	13
11	11	13	13	8	11	11	7	25	14
Cost function: Methods (min = 19)	19	19	19	22	22	22	22	22	22
Attributes (min = 131)	178	184	184	197	227	227	197	229	197
Dependencies (min = 10)	10	11	11	13	13	13	13	13	13

**APPENDIX B.9: TEST ORDERS OBTAINED WHEN USING A WEIGHTED
GEOMETRIC AVERAGE OF ATTRIBUTE AND METHOD COUPLING
MEASURES AS THE COST FUNCTION (ANT)**

Test Orders									
1	2	3	4	5	6	7	8	9	10
15	15	4	4	4	9	15	4	4	15
9	12	15	5	12	12	17	5	5	12
12	17	17	17	3	17	4	15	9	3
3	9	5	12	15	15	12	17	12	17
17	3	12	15	17	3	3	12	3	21
21	8	3	3	21	4	21	3	15	23
8	21	21	21	23	8	9	21	17	20
23	4	23	23	20	21	23	23	21	4
20	5	20	20	16	23	20	20	23	5
4	23	16	16	22	20	16	16	20	9
16	20	1	7	7	16	1	22	16	8
19	16	2	1	5	1	11	19	8	16
5	19	25	11	10	11	7	10	1	7
10	7	19	2	2	2	10	24	22	10
1	2	10	25	6	25	19	1	10	1
22	25	22	19	25	22	5	18	19	11
2	22	24	9	19	7	22	7	24	2
7	10	18	10	9	10	8	2	2	6
24	24	13	6	8	19	24	25	25	22
6	6	9	14	14	24	2	11	6	25
11	14	8	22	1	18	6	6	18	14
25	18	6	8	11	5	18	14	13	19
18	13	14	24	24	13	13	9	14	24
14	1	7	18	18	6	14	8	7	18
13	11	11	13	13	14	25	13	11	13
Cost Function: OCplx ($W_M=W_A=0.5$)	3.59	3.59	3.59	3.59	3.59	3.59	3.59	3.59	3.59
Attributes (min = 131)	136	136	136	136	136	136	136	136	136
Methods (min = 19)	29	29	29	29	29	29	29	29	29

APPENDIX C.1: CLASS DIAGRAM AND RELATIONSHIP MATRIX FOR SPM

Figure 26 Class Diagram for the SPM system

ID	Class
1	CardReaderInterface
2	Checkpoint
3	CheckpointController
4	CheckpointControllerIdleState
5	CheckpointControllerNormalCheckingState
6	CheckpointControllerState
7	CheckpointControllerViolationCheckingState
8	CheckpointControllerViolationState
9	CheckpointReferences
10	ControlCentre
11	ControlCentreThread
12	CreateZone
13	DoorLockInterface
14	GuardDueTimer
15	GuardLateTimer
16	PatrolSchedule
17	ScheduleEntry
18	SecurityTime
19	SecurityZone

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19
1			As						Us										
2	Cp		Cp						Cp				Cp	Cp	Cp	Cp	Us		
3				Us	Us	Cp	Us	Us	Us										
4			Us			I			Us				As	As	As	As			
5			Us			I			Us				As	As	As	As			
6			Us						Us										
7			Us			I			Us				As	As	As	As			
8			Us			I			Us				As	As		As			
9	As	As	As										As	As	As	As			
10											Us								
11																			
12																			Us
13		As	As						Us										
14		As	As						Us							As		Cp	
15			As						Us										
16		As	As						Us								Cp	Us	
17																		As	
18																			
19		As										As							

Description: **As:** Associations and simple Aggregations, **Cp:** Compositions, **Us:** Use dependencies, **I:** Inheritance.

APPENDIX C.2: DEPENDENCY MATRIX FOR SPM

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19
1			1						1										
2	1		1						1				1	1	1	1	1		
3				1	1	1	1	1	1										
4			1			1			1				1	1	1	1			
5			1			1			1				1	1	1	1			
6			1						1										
7			1			1			1				1	1	1	1			
8			1			1			1				1	1		1			
9	1	1	1										1	1	1	1			
10											1								
11																			
12																			1
13		1	1						1										
14		1	1						1							1		1	
15			1						1										
16		1	1						1								1	1	
17																		1	
18																			
19		1										1							

APPENDIX C.3: PRECEDENCE TABLE FOR SPM

	Precedent Classes							
1								
2	1	3	9	13	14	15	16	17
3	6							
4	6							
5	6							
6								
7	6							
8	6							
9								
10	11							
11								
12								
13								
14	18							
15								
16	17	18						
17	18							
18								
19								

APPENDIX C.4: ATTRIBUTE COUPLING MATRIX FOR SPM

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19
1			12						7										
2																	4		
3				4	4		4	3	7										
4			12						7				5	6	3	5			
5			12						7				5	6	3	5			
6			12						7										
7			12						7				5	6	3	5			
8			12						7				5	6		5			
9	2	21	12										5	6	3	5			
10											1								
11																			
12																			5
13		21	12						7										
14		21	12						7							5			
15			12						7										
16		21	12						7									11	
17																		11	
18																			
19		21										2							

APPENDIX C.5: METHOD COUPLING MATRIX FOR SPM

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19
1			2						1										
2																	6		
3				8	1		1	1	0										
4			6						4				3	2	2	3			
5			3						4				2	2	3	2			
6			0						0										
7			3						4				2	2	2	2			
8			6						4				3	2		3			
9	1	1	1										1	1	1	1			
10											1								
11																			
12																			3
13		3	2						2										
14		2	2						3							2			
15			2						1										
16		2	2						2									3	
17																		1	
18																			
19		4										2							

**APPENDIX C.6: TEST ORDERS OBTAINED WITH NUMBER OF BROKEN
DEPENDENCIES AS COST FUNCTION (SPM)**

Test Orders									
1	2	3	4	5	6	7	8	9	10
11	18	9	12	9	9	11	12	12	9
12	9	12	9	18	6	10	9	11	6
10	6	11	11	6	3	18	6	10	3
9	3	18	6	3	18	12	3	9	12
18	12	17	3	17	17	17	18	6	18
6	11	6	18	13	16	9	17	18	13
3	17	3	15	16	14	6	13	3	17
17	16	16	13	14	11	3	16	15	16
15	14	10	17	1	15	13	14	13	15
13	1	14	16	15	10	16	8	17	14
16	15	15	14	2	13	14	15	1	11
1	13	1	1	4	8	15	1	16	1
14	4	13	7	8	1	8	4	14	2
2	5	2	4	19	2	4	5	2	5
19	7	8	5	11	7	5	11	5	7
5	2	4	10	10	4	7	2	8	19
7	10	19	2	5	5	1	7	19	8
4	8	7	19	7	19	2	10	7	4
8	19	5	8	12	12	19	19	4	10
Cost Function: Number of Dependencies	16	16	16	16	16	16	16	16	16
Attributes (min = 146)	149	149	149	149	146	146	149	149	149
Methods (min = 26)	28	28	28	28	27	27	28	28	28

APPENDIX C.7: TEST ORDERS OBTAINED WHEN USING THE NUMBER OF STUBBED ATTRIBUTES AS THE COST FUNCTION (SPM)

Test Orders									
1	2	3	4	5	6	7	8	9	10
18	9	11	11	9	9	18	11	9	9
9	6	9	9	6	6	17	18	18	6
11	3	18	6	3	3	11	17	6	3
6	13	6	3	18	11	9	9	3	15
3	11	3	18	15	10	6	6	11	18
17	15	1	17	17	13	3	3	15	13
16	18	13	16	13	18	16	16	13	1
15	17	17	13	16	17	13	13	1	17
10	16	15	1	14	16	15	15	17	16
13	1	16	14	5	14	1	14	16	14
14	14	14	8	11	8	14	7	14	2
1	2	10	15	10	15	5	4	2	8
4	8	4	2	8	4	4	1	8	11
5	7	2	19	1	1	2	8	7	19
2	4	5	7	2	7	19	2	5	12
7	19	8	5	4	2	12	19	4	4
8	12	19	10	19	19	10	5	19	5
19	5	12	4	12	5	7	12	12	10
12	10	7	12	7	12	8	10	10	7
Cost Function: Number of Attributes	146	146	146	146	146	146	146	146	146
Methods (min = 26)	27	27	27	27	27	27	27	27	27
Dependencies (min = 16)	16	16	16	16	16	16	16	16	16

**APPENDIX C.8: TEST ORDERS OBTAINED WHEN USING THE NUMBER OF
STUBBED METHODS AS THE COST FUNCTION (SPM)**

Test Orders									
1	2	3	4	5	6	7	8	9	10
6	18	6	6	18	11	6	11	6	18
3	17	3	3	17	6	3	6	3	17
18	6	9	9	6	10	11	10	9	11
9	3	15	13	3	3	10	3	18	10
1	9	1	1	9	9	9	9	15	6
17	15	13	18	15	13	13	15	17	3
16	13	18	17	13	18	15	18	16	9
15	16	17	15	16	17	18	17	14	1
14	14	16	16	14	15	17	16	13	16
13	8	14	14	7	16	16	14	1	14
8	4	5	2	1	14	14	13	5	13
5	7	11	11	2	4	4	1	2	15
4	1	10	10	8	5	1	2	8	2
7	2	2	19	19	1	2	4	19	4
2	19	19	12	11	2	19	19	7	19
19	11	4	5	10	19	12	8	11	5
11	10	7	7	4	7	8	7	10	12
10	12	8	4	5	12	5	12	12	8
12	5	12	8	12	8	7	5	4	7
Cost Function: Number of Methods	26	26	26	26	26	26	26	26	26
Attributes (min = 146)	148	148	148	148	148	148	148	148	148
Dependencies (min = 16)	17	17	17	17	17	17	17	17	17

**APPENDIX C.9: TEST ORDERS OBTAINED WHEN USING A WEIGHTED
GEOMETRIC AVERAGE OF ATTRIBUTE AND METHOD COUPLING
MEASURES AS THE COST FUNCTION (SPM)**

	Test Orders									
	1	2	3	4	5	6	7	8	9	10
	11	18	18	18	11	6	6	6	18	11
	6	17	17	17	6	11	18	3	6	10
	3	6	6	6	3	3	17	9	3	6
	18	3	3	3	18	9	3	11	9	3
	17	9	9	9	9	1	9	18	17	9
	9	13	16	1	13	15	13	17	16	13
	16	16	15	13	17	13	16	13	15	1
	14	15	14	16	16	18	14	1	13	18
	13	14	1	14	14	17	8	16	14	15
	15	11	13	15	15	16	15	14	8	17
	10	7	5	2	7	14	11	15	5	16
	7	5	2	19	4	2	4	7	7	14
	5	1	11	8	8	19	7	2	1	2
	4	2	10	7	5	12	5	19	2	19
	1	19	19	5	1	4	1	12	11	4
	2	10	4	11	2	8	2	5	10	8
	8	8	12	10	19	10	19	8	19	5
	19	4	7	12	10	5	12	10	4	7
	12	12	8	4	12	7	10	4	12	12
Cost Function: OCplx ($W_M=W_A=0.5$)	5.77	5.77	5.77	5.77	5.77	5.77	5.77	5.77	5.77	5.77
Attributes (min = 146)	148	148	148	148	148	148	148	148	148	148
Methods (min = 26)	26	26	26	26	26	26	26	26	26	26
Dependencies (min = 16)	17	17	17	17	17	17	17	17	17	17

APPENDIX D.1: RELATIONSHIP MATRIX FOR BCEL

ID	Class
1	AccessFlags
2	Attribute
3	ClassParser
4	Code
5	CodeException
6	Constant
7	ConstantClass
8	ConstantCP
9	ConstantDouble
10	ConstantFieldref
11	ConstantFloat
12	ConstantInteger
13	ConstantInterfaceMethodref
14	ConstantLong
15	ConstantMethodref
16	ConstantNameAndType
17	ConstantObject
18	ConstantPool
19	ConstantString
20	ConstantUtf8
21	ConstantValue
22	Deprecated
23	DescendingVisitor
24	EmptyVisitor
25	ExceptionTable
26	Field
27	FieldOrMethod
28	InnerClass
29	InnerClasses
30	JavaClass
31	LineNumber
32	LineNumberTable
33	LocalVariable
34	LocalVariableTable
35	Method
36	Node
37	PMGClass
38	Signature
39	SourceFile
40	StackMap
41	StackMapEntry
42	StackMapType
43	Synthetic
44	Unknown
45	Visitor

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23
1																							
2				Us														As		Us	Us	Us	
3		Cp																Cp					
4		I			As													Us					
5																		Us					
6							Us		Us	Us	Us	Us	Us	Us	Us	Us			Us	Us			
7						I												I	Us		Us		
8						I													Us				
9						I												I	Us				
10								I															
11						I												I	Us				
12						I												I	Us				
13								I															
14						I												I	Us				
15								I															
16						I													Us				
17																			Us				
18						As	Us	Us	Us	Us	Us	Us	Us	Us	Us	Us			Us	Us			
19						I												I	Us		Us		
20						I																	
21		I				Us			Us		Us	Us		Us					Us	Us	Us		
22		I																	Us				
23		Us		Us	Us	Us	Us		Us	Us	Us	Us	Us	Us	Us	Us		Us	Us	Us	Us	Us	
24				Us	Us		Us		Us	Us	Us	Us	Us	Us	Us	Us		Us	Us	Us	Us	Us	
25		I																	Us				
26		Us																	Us		Us		
27	I	As																	As		Us		
28																			Us		Us		
29		I																	Us				
30	I	As								Us									As		Us		
31																							
32		I																	Us				
33						I													As		Us		
34		I																	Us				
35		Us		Us															Us		Us	Us	
36																							
37		I																	Us		Us		
38		I																	Us		Us		
39		I																	Us		Us		
40		I																	Us				
41																			As				
42																			As				
43		I																	Us				
44		I																	Us		Us		
45				Us	Us		Us		Us		Us	Us	Us	Us	Us	Us			Us	Us	Us	Us	Us

	24	25	26	27	28	29	30	31	32	33	34	35	36	37	38	39	40	41	42	43	44	45
1																						
2		Us				Us			Us	Us	Us		I	Us	Us	Us	Us			Us	Us	Us
3			Cp				Us					Cp										
4									Us		Us											Us
5													I									Us
6													I									Us
7																						Us
8																						
9																						Us
10																						Us
11																						Us
12																						Us
13																						Us
14																						Us
15																						Us
16																						Us
17																						
18													I									Us
19																						Us
20																						Us
21																						Us
22																						Us
23		Us	Us		Us	Us	As	Us	Us	Us	Us	Us				Us	Us	Us		Us	Us	I
24		Us	Us		Us	Us	Us	Us	Us	Us	Us	Us				Us	Us	Us		Us	Us	I
25																						Us
26				I																		Us
27													I									
28													I									Us
29					As																	Us
30			As									As	I			Us						Us
31													I									Us
32								As														Us
33													I									Us
34										As												Us
35		Us		I					Us		Us											Us
36																						Us
37																						Us
38																						Us
39																						Us
40													I					As				Us
41																			As			Us
42																						
43																						Us
44																						Us
45		Us	Us		Us	Us	Us	Us	Us	Us	Us	Us				Us	Us	Us		Us	Us	

APPENDIX D.2: PRECEDENCE TABLE FOR BCEL

[illegible]

APPENDIX D.3: DEPENDENCY MATRIX FOR BCEL

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23
1																							
2				1														1		1	1	1	
3		1																1					
4		1			1													1					
5																		1					
6							1		1	1	1	1	1	1	1	1			1	1			
7						1											1	1		1			
8						1												1					
9						1											1	1					
10								1															
11						1											1	1					
12						1											1	1					
13								1															
14						1											1	1					
15								1															
16						1												1					
17																		1					
18						1	1	1	1	1	1	1	1	1	1	1			1	1			
19						1											1	1		1			
20						1																	
21		1				1			1		1	1		1				1	1	1			
22		1																1					
23		1		1	1	1	1		1	1	1	1	1	1	1	1		1	1	1	1	1	
24				1	1		1		1	1	1	1	1	1	1	1		1	1	1	1	1	
25		1																1					
26		1																1			1		
27	1	1																1		1			
28																		1		1			
29		1																1					
30	1	1							1									1		1			
31																							
32		1																1					
33						1												1		1			
34		1																1					
35		1		1														1		1	1		
36																							
37		1																1		1			
38		1																1		1			
39		1																1		1			
40		1																1					
41																		1					
42																		1					
43		1																1					
44		1																1		1			
45				1	1		1		1		1	1	1	1	1	1		1	1	1	1	1	

	24	25	26	27	28	29	30	31	32	33	34	35	36	37	38	39	40	41	42	43	44	45
1																						
2		1				1			1	1	1		1	1	1	1	1			1	1	1
3			1				1					1										
4									1		1											1
5													1									1
6													1									1
7																						1
8																						
9																						1
10																						1
11																						1
12																						1
13																						1
14																						1
15																						1
16																						1
17																						
18													1									1
19																						1
20																						1
21																						1
22																						1
23		1	1		1	1	1	1	1	1	1	1				1	1	1		1	1	1
24		1	1		1	1	1	1	1	1	1	1				1	1	1		1	1	1
25																						1
26				1																		1
27													1									
28													1									1
29					1																	1
30			1									1	1			1						1
31													1									1
32								1														1
33													1									1
34										1												1
35		1		1					1		1											1
36																						1
37																						1
38																						1
39																						1
40													1					1				1
41																			1			1
42																						
43																						1
44																						1
45		1	1		1	1	1	1	1	1	1	1				1	1	1		1	1	

APPENDIX D.4: ATTRIBUTE COUPLING MATRIX FOR BCEL

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23
1																							
2				8														2		1	1	1	
3																							
4					4													2					
5																		2					
6							1		1	0	1	1	0	1	1	2			1	1			
7																		2		1			
8																		2					
9																		2					
10																							
11																		2					
12																		2					
13																							
14																		2					
15																							
16																		2					
17																		2					
18						1	1	1	1	0	1	1	0	1	0	2			1	1			
19																		2		1			
20																							
21						1			1		1	1		1				2	1	1			
22																		2					
23		4		8	4	1	1		1	0	1	1	0	1	0	2		2	1	1	1	1	
24				8	4		1		1	0	1	1	0	1	0	2		2	1	1	1	1	
25																		2					
26		4																2			1		
27		4																2		1			
28																		2		1			
29																		2					
30		4								0								2		1			
31																							
32																		2					
33																		2		1			
34																		2					
35		4		8														2		1	1		
36																							
37																		2		1			
38																		2		1			
39																		2		1			
40																		2					
41																		2					
42																		2					
43																		2					
44																		2		1			
45				8	4		1		1		1	1	0	1	0	2		2	1	1	1	1	

	24	25	26	27	28	29	30	31	32	33	34	35	36	37	38	39	40	41	42	43	44	45
1																						
2		2				2			2	6	2			1	1	1	2			1	3	0
3							20															
4									2		2											0
5																						0
6																						0
7																						0
8																						
9																						0
10																						0
11																						0
12																						0
13																						0
14																						0
15																						0
16																						0
17																						
18																						0
19																						0
20																						0
21																						0
22																						0
23		2	0		4	2	20	2	2	6	2	0				1	2	6		1	3	
24		2	0		4	2	20	2	2	6	2	0				1	2	6		1	3	
25																						0
26																						0
27																						
28																						0
29					4																	0
30			0									0				1						0
31																						0
32								2														0
33																						0
34										6												0
35		2							2		2											0
36																						0
37																						0
38																						0
39																						0
40																		6				0
41																			3			0
42																						
43																						0
44																						0
45		2	0		4	2	20	2	2	6	2	0				1	2	6		1	3	

APPENDIX D.5: METHOD COUPLING MATRIX FOR BCEL

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23
1																							
2				1														2		2	1	1	
3																							
4					4													1					
5																		2					
6							1		1	1	1	1	1	1	1	1			1	1			
7																		2		2			
8																		2					
9																		1					
10																							
11																		1					
12																		1					
13																							
14																		1					
15																							
16																		2					
17																		1					
18						5	2	3	2	1	2	2	1	2	1	3			2	2			
19																		1		2			
20																							
21						1			2		2	2		2				1	2	2			
22																		1					
23		2		4	2	2	2		2	2	2	2	2	2	2	2		2	2	2	2	2	
24				1	1		1		1	1	1	1	1	1	1	1		1	1	1	1	1	
25																		1					
26		2																1			3		
27		4																2		3			
28																		2		2			
29																		1					
30		3								1								1		1			
31																							
32																		1					
33																		2		3			
34																		1					
35		2		3														2		3	1		
36																							
37																		2		3			
38																		2		3			
39																		2		3			
40																		1					
41																		1					
42																		2					
43																		1					
44																		2		2			
45				1	1		1		1		1	1	1	1	1	1		1	1	1	1	1	

	24	25	26	27	28	29	30	31	32	33	34	35	36	37	38	39	40	41	42	43	44	45
1																						
2		1				1			1	1	1			1	1	1	1			1	1	0
3							1															
4									1		1											1
5																						1
6																						1
7																						1
8																						
9																						1
10																						1
11																						1
12																						1
13																						1
14																						1
15																						1
16																						1
17																						
18																						1
19																						1
20																						1
21																						1
22																						1
23		2	3		2	2	6	2	3	2	3	3				2	3	2		2	2	
24		1	1		1	1	1	1	1	1	1	1				1	1	1		1	1	
25																						1
26																						1
27																						
28																						1
29					3																	1
30			3									4				2						1
31																						1
32								7														1
33																						1
34										4												1
35		1							1		1											1
36																						0
37																						0
38																						0
39																						1
40																		4				1
41																			2			1
42																						
43																						1
44																						1
45		1	1		1	1	1	1	1	1	1	1				1	1	1		1	1	

APPENDIX D.6: TEST ORDERS OBTAINED WITH NUMBER OF BROKEN DEPENDENCIES AS COST FUNCTION (BCEL)

Test Orders									
1	2	3	4	5	6	7	8	9	10
45	45	45	45	45	45	45	45	45	45
36	36	36	36	36	36	36	42	36	36
6	6	6	6	31	31	6	36	6	6
20	8	8	8	6	6	20	6	20	8
8	17	10	15	20	20	8	8	8	13
13	16	15	10	8	8	15	15	15	15
15	20	20	20	10	10	33	10	13	20
10	13	18	13	13	15	13	13	31	18
18	31	33	18	15	18	10	20	18	17
33	11	17	5	18	33	18	18	33	42
31	10	14	33	33	42	2	31	17	5
2	15	12	2	17	2	28	5	9	7
32	18	11	43	19	43	22	28	14	10
5	33	9	42	9	25	39	17	10	33
44	42	42	28	28	5	34	11	11	1
25	41	5	31	41	28	29	19	42	31
41	5	1	25	12	34	44	33	19	14
34	28	19	16	11	32	17	14	12	28
22	12	13	39	14	17	9	16	7	11
43	2	31	17	2	12	14	12	2	19
28	40	7	41	1	19	25	7	39	2
37	34	41	38	43	11	19	41	37	37
4	29	28	34	29	9	12	9	44	43
17	19	2	32	44	7	11	2	28	9
9	9	34	14	25	14	43	38	29	25
39	43	37	11	38	1	31	32	41	32
7	7	39	7	40	13	21	34	34	34
11	44	32	9	21	44	5	21	21	29
12	14	25	44	32	22	42	22	5	38
38	22	44	40	5	21	37	40	43	12
16	25	43	12	39	41	32	25	25	41
14	32	22	22	34	27	1	43	32	21
19	4	40	19	4	16	38	44	1	4
40	38	21	29	27	4	4	39	38	27
21	21	29	21	16	35	27	4	4	35
29	39	16	1	42	26	35	37	27	40
42	37	4	27	37	39	16	1	35	16
1	1	27	4	22	40	26	27	40	39
27	27	26	26	7	30	41	26	26	26
35	35	35	30	35	29	40	35	30	30
26	26	38	35	26	3	30	29	22	3
30	30	30	3	30	37	7	30	16	22
24	3	3	24	23	38	24	24	3	44
23	23	23	37	24	24	3	3	23	23
3	24	24	23	3	23	23	23	24	24
Cost Function: Number of Dependencies	63	63	63	63	63	63	63	63	63
Attributes (min = 46)	128	128	125	125	128	125	127	127	125
Methods (min = 67)	73	70	72	75	73	72	73	72	72

**APPENDIX D.7: TEST ORDERS OBTAINED WHEN USING THE NUMBER OF
STUBBED ATTRIBUTES AS THE COST FUNCTION (BCEL)**

Test Orders									
1	2	3	4	5	6	7	8	9	10
36	36	36	36	36	36	36	36	36	36
6	6	31	6	6	6	6	6	6	31
31	20	6	20	20	20	20	16	20	6
18	16	20	18	18	18	31	20	18	20
20	18	16	28	33	33	16	18	28	18
17	31	18	17	2	1	18	33	42	33
9	33	33	5	42	8	17	42	33	42
16	28	28	14	31	42	33	17	17	8
42	2	17	42	41	31	28	12	8	41
7	25	11	33	34	2	2	5	41	2
14	17	9	31	38	10	9	41	1	39
8	44	2	41	25	34	11	28	31	10
13	34	34	1	39	44	39	7	15	34
12	32	29	7	32	32	14	2	16	38
15	42	8	12	8	15	7	14	2	17
28	22	25	9	15	13	1	34	32	12
11	38	32	2	5	17	12	39	34	7
33	1	37	29	28	9	32	44	22	14
19	43	1	44	37	11	38	31	39	16
41	30	42	43	4	19	42	29	30	5
2	41	13	32	17	28	41	43	40	22
39	37	38	25	7	22	19	38	5	44
43	5	44	19	19	5	40	32	7	32
29	8	22	39	12	16	21	25	10	28
25	13	19	22	16	43	22	19	44	13
22	10	12	34	43	38	25	37	9	37
21	29	43	11	1	25	44	22	14	29
44	12	41	40	11	14	5	4	11	25
34	11	39	37	30	12	27	1	43	4
37	19	7	4	27	21	34	30	12	19
32	9	5	30	40	29	43	40	13	1
38	7	27	16	22	4	26	9	25	30
5	14	10	21	44	41	4	11	19	27
1	39	21	45	13	27	37	21	4	9
30	4	30	8	14	40	29	27	27	43
40	40	40	15	10	39	8	35	29	11
4	21	15	10	29	30	13	26	37	40
10	15	4	27	9	35	35	3	38	21
45	45	14	13	21	7	10	45	21	45
27	27	45	26	35	26	30	8	45	15
35	26	26	35	26	3	45	15	35	26
26	35	35	23	3	45	15	10	26	35
24	23	3	24	45	24	23	13	3	3
23	3	23	3	23	23	3	24	24	23
3	24	24	38	24	37	24	23	23	24
Cost Function: Number of Attributes	47	47	47	46	46	46	46	46	46
Methods (min = 67)	86	87	85	84	85	84	76	83	84
Dependencies (min = 63)	75	75	74	72	74	72	71	73	73

**APPENDIX D.8: TEST ORDERS OBTAINED WHEN USING THE NUMBER OF
STUBBED METHODS AS THE COST FUNCTION (BCEL)**

Test Orders									
1	2	3	4	5	6	7	8	9	10
36	17	17	17	45	17	17	36	45	36
45	36	45	36	36	45	1	17	17	6
17	6	36	45	1	36	36	45	42	8
6	9	6	6	17	6	31	6	41	15
9	20	11	9	6	12	45	8	36	20
11	31	8	11	8	20	6	15	6	17
14	12	20	8	20	16	12	20	20	19
8	14	9	12	10	9	20	33	19	9
20	8	14	31	13	19	11	11	12	14
13	13	19	13	15	11	14	13	8	45
15	16	15	14	14	8	8	12	15	12
10	45	18	20	11	10	15	14	11	10
12	11	31	18	16	13	13	9	10	18
16	15	33	28	19	14	10	19	13	11
18	18	2	33	12	42	19	16	31	7
33	33	34	2	31	18	9	18	14	31
42	2	32	39	18	41	18	31	16	33
31	38	7	29	33	33	28	28	9	28
2	25	42	38	28	15	33	2	18	42
22	42	28	7	9	2	41	44	5	5
19	10	43	32	2	25	2	37	33	2
34	5	39	34	29	31	5	34	28	13
5	19	5	15	34	43	44	25	7	32
37	39	41	5	22	1	7	43	2	38
28	1	40	25	32	5	40	42	29	34
32	21	4	4	42	39	25	32	25	4
38	28	44	43	7	22	22	5	43	16
7	37	12	1	43	21	38	22	21	25
4	43	13	27	41	34	34	41	22	21
41	41	29	44	37	28	42	21	34	37
39	40	38	19	39	32	29	39	44	22
1	44	25	42	25	37	21	40	39	43
27	29	22	21	40	4	32	4	4	41
44	27	21	37	38	7	39	1	37	40
40	26	16	35	5	44	4	27	1	29
21	34	10	22	21	27	37	26	32	1
26	4	1	26	4	38	16	38	27	27
43	22	27	10	27	35	27	35	26	35
25	32	26	41	44	26	26	7	38	39
29	35	35	30	35	30	35	29	40	26
35	30	30	40	26	40	43	10	35	30
30	3	3	16	30	3	30	30	30	3
3	7	24	23	23	29	3	24	23	44
23	23	37	24	3	23	23	3	24	24
24	24	23	3	24	24	24	23	3	23
Cost Function: Number of Methods	67	70	70	70	67	69	69	69	70
Attributes (min = 46)	131	125	131	131	131	134	133	134	138
Dependencies (min = 63)	64	67	65	66	63	65	65	66	66

**APPENDIX D.9: TEST ORDERS OBTAINED USING A WEIGHTED
GEOMETRIC AVERAGE OF ATTRIBUTE AND METHOD COUPLING
MEASURES AS THE COST FUNCTION (BCEL)**

Test Orders									
1	2	3	4	5	6	7	8	9	10
36	42	1	36	17	36	36	17	17	17
6	17	36	6	42	6	6	36	36	36
8	36	6	20	36	16	17	31	6	6
16	6	20	17	6	8	11	6	20	9
20	14	8	12	11	10	20	8	7	11
15	8	15	9	9	20	9	12	8	12
18	13	17	11	20	13	8	20	9	20
28	15	19	33	7	18	10	15	14	19
13	20	10	16	14	5	12	16	12	31
33	31	31	14	8	28	19	18	11	14
5	19	11	19	16	42	18	5	19	18
42	12	14	8	10	41	31	33	18	33
41	10	9	13	15	31	33	28	33	2
17	18	12	18	13	17	28	9	31	34
12	33	18	31	12	12	5	2	2	21
19	2	33	10	18	19	2	38	37	44
14	44	7	28	41	14	34	14	34	28
31	1	42	1	5	9	39	39	42	16
11	41	28	2	31	1	42	42	10	42
9	5	41	37	33	11	43	22	15	1
7	39	5	22	2	15	22	19	25	39
2	28	2	25	40	33	25	44	32	7
32	11	34	34	25	2	29	10	13	37
22	9	44	21	32	37	4	37	28	43
34	40	32	32	38	32	41	1	1	25
40	21	39	29	37	29	16	34	41	32
39	32	37	27	34	34	14	32	40	22
4	37	21	26	21	44	40	4	21	5
25	25	4	7	4	4	21	11	27	27
29	34	22	42	28	38	15	41	44	4
37	4	27	43	39	43	1	21	22	8
44	43	26	5	1	21	27	27	38	13
21	16	25	41	27	25	26	29	5	10
38	27	13	38	26	27	32	25	26	35
10	35	35	39	43	26	37	35	4	41
43	26	30	44	44	39	13	43	39	40
1	38	3	4	29	35	35	26	35	29
27	7	16	35	35	22	30	40	29	26
26	29	29	30	30	40	3	30	16	38
35	30	43	3	45	30	45	7	30	15
30	22	38	45	3	45	7	45	43	30
3	3	45	40	19	3	44	13	45	3
45	45	40	15	22	7	38	24	24	45
23	23	23	24	24	24	24	3	23	23
24	24	24	23	23	23	23	23	3	24
Cost Function: OCplx ($W_M=W_A=0.5$)									
Attributes (min = 46)									
Methods (min = 67)									
Dependencies (min = 63)									
7.56	7.74	7.76	7.78	7.83	7.83	7.86	7.87	7.88	7.90
56	55	58	48	54	53	59	47	59	50
70	72	72	73	73	73	73	74	73	74
70	72	72	69	70	72	71	70	71	71

APPENDIX E.1: RELATIONSHIP MATRIX FOR DNS

ID	Class Name
1	A6Record
2	AAAARecord
3	Address
4	ARecord
5	BitString
6	CERTRecord
7	CNAMERecord
8	Compression
9	Credibility
10	DClass
11	DNAMERecord
12	dns
13	DNSSEC
14	FindServer
15	Flags
16	HINFORecord
17	Inet6Address
18	KEYRecord
19	LOCRecord
20	Master
21	Name
22	NameSet
23	NAPTRRecord
24	NSRecord
25	NS_CNAME_PTRRecord
26	NXTRecord
27	Opcode
28	Options
29	OPTRecord
30	PTRRecord
31	Rcode
32	Record
33	ResolveThread
34	RPRRecord
35	RRset
36	Section
37	SetResponse
38	SimpleResolver
39	SOARecord
40	SRVRecord
41	TKEYRecord
42	TSIG
43	TSIGRecord
44	TTL
45	TXTRecord

ID	Class Name
46	Type
47	TypeMap
48	UNKRecord
49	WireParseException
50	Zone
51	Cache
52	ExtendedResolver
53	Header
54	KXRecord
55	Message
56	MX_KXRecord
57	MXRecord
58	SIGRecord
59	Verifier
60	Resolver
61	ResolverListener

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
1								Us									As			
2								Us									As			
3				Us								Us								
4								Us												
5																				
6								Us												
7								Us												
8																				
9																				
10																				
11								Us												
12							Us		Us	Us	Us			Us						
13																				
14																				
15																				
16								Us												
17																				
18								Us					Us							
19								Us												
20										Us										
21					Us			Us			Us									
22																				
23								Us												
24								Us												
25								Us												
26								Us												
27																				
28																				
29								Us												
30								Us												
31																				
32								Us		Us										
33																				
34								Us												
35													Us							
36																				
37							Us				Us									
38														Us	Us					
39								Us												
40								Us												

	21	22	23	24	25	26	27	28	29	30	31	32	33	34	35	36	37	38	39	40
1	As											I								
2	As											I								
3					Us					Us		Us								
4	Us											I								
5								Us												
6	Us											I								
7	Us				I															
8	Cp																			
9																				
10																				
11	Us				I															
12	As				Us			Us			Us	Us			Us		Us			
13																				
14	As																			
15																				
16	Us											I								
17																				
18	Us							Us				I								
19	Us											I								
20	As				Us							As								
21								Us												
22	Cp																			
23	As							Us				I								
24	Us				I															
25	As											I								
26	As											I								
27																				
28																				
29	Us											I								
30	Us				I															
31																				
32	As							Us								Us				
33																		Us		
34	Cp											I								
35	Us				Us							Cp								
36																				
37															Us					
38	Us				Us			Us	Us		Us	Us	Us			Us			Us	
39	Cp											I								
40	Cp											I								

	41	42	43	44	45	46	47	48	49	50	51	52	53	54	55	56	57	58	59	60	61
1						Us															
2						Us															
3						Us															
4						Us															
5																					
6						Us															
7						Us															
8																					
9																					
10																					
11						Us															
12						Us				As				Us					As		
13																					
14																					
15																					
16						Us															
17																					
18						Us															
19						Us			Us												
20				Us		Us															
21									Us												
22						Us	Us														
23						Us															
24						Us															
25						Us															
26						Us															
27																					
28																					
29						Us															
30						Us															
31																					
32				Us		Us		Us										Us			
33												Us			As					As	As
34						Us															
35						Us															
36																					
37																					
38		As				Us		Us	Us						Us					I	Us
39				Us		Us															
40						Us															

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
41								Us												
42									Us											
43								Us												
44																				
45								Us												
46																				
47																				
48								Us												
49																				
50							Us		Us	Us	Us									Us
51							Us		Us		Us		Us		Us					Us
52														Us						
53															Us					
54								Us												
55								Us		Us					Us					
56								Us												
57								Us												
58								Us												
59																				
60																				
61																				

	21	22	23	24	25	26	27	28	29	30	31	32	33	34	35	36	37	38	39	40
41	Cp										Us	I								
42	Cp				Us			Us			Us	Us				Us				
43	Cp										Us	I								
44																				
45	Us											I								
46																				
47																				
48	Us											I								
49																				
50	As	I			Us			Us				Us			As	Us	Us		As	
51	Us	I									Us				Us	Us	Us		Us	
52	Us							Us			Us		Us					Us		
53							Us				Us					Us				
54	Us																			
55	Us				Us		Us		Us			Cp				Us				
56	As											I								
57	Us																			
58	Cp							Us				I								
59															Us					
60	Us																			
61																				

	41	42	43	44	45	46	47	48	49	50	51	52	53	54	55	56	57	58	59	60	61
41						Us												Us			
42			Us												Us						
43						Us															
44																					
45						Us															
46																					
47						Us															
48																					
49																					
50						Us	Us								Us					Us	
51						Us	Us								Us				As		
52															Us					I	Us
53																					
54						Us										I					
55			Us			Us							Cp								
56						Us															
57						Us										I					
58				Us		Us															
59																					
60															Us						Us
61															Us						

APPENDIX E.2: PRECEDENCE TABLE FOR DNS

ID	Precedent Classes															
1	8	17	21	32	46											
2	8	17	21	32	46											
3	4	12	25	30	32	46										
4	8	21	32	46												
5	28															
6	8	21	32	46												
7	8	21	25	46												
8	21															
9																
10																
11	25	46														
12	7	9	10	11	14	21	25	28	31	32	35	37	46	51	55	60
13																
14	21															
15																
16	8	21	32	46												
17																
18	8	13	21	28	32	46										
19	8	21	32	46	49											
20	10	21	25	32	44	46										
21	5	28	49													
22	21	46	47													
23	8	21	28	32	46											
24	8	21	25	46												
25	32	46														
26	8	21	32	46												
27																
28																
29	8	21	32	46												
30	8	21	25	46												
31																
32	10	28	36	44	46											
33	55	60	61													
34	8	21	32	46												
35	13	21	25	32	46											
36																
37	7	11	35													
38	14	15	28	29	31	36	39	42	46	49	55	60	61			
39	8	21	32	44	46											
40	8	21	32	46												

ID	Precedent Classes																		
41	8	21	31	32	46	58													
42	10	21	25	28	31	32	36	43	55										
43	8	21	31	32	46														
44																			
45	8	21	32	46															
46																			
47	46																		
48	32																		
49																			
50	7	9	10	11	20	21	22	25	28	32	35	36	37	39	46	47	55	60	
51	7	9	11	13	15	20	21	22	31	35	36	37	39	46	47	55	59		
52	14	28	31	55	60	61													
53	15	27	31	36															
54	8	21	46																
55	8	10	15	21	25	27	29	32	36	43	46	53							
56	8	21	32	46															
57	8	21	46	56															
58	28	32	44	46															
59	35																		
60	21	55	61																
61	55																		

APPENDIX E.3: DEPENDENCY MATRIX FOR DNS

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
1								1									1			
2								1									1			
3				1								1								
4								1												
5																				
6								1												
7								1												
8																				
9																				
10																				
11								1												
12							1		1	1	1			1						
13																				
14																				
15																				
16								1												
17																				
18								1						1						
19								1												
20										1										
21					1			1			1									
22																				
23								1												
24								1												
25								1												
26								1												
27																				
28																				
29								1												
30								1												
31																				
32								1		1										
33																				
34								1												
35														1						
36																				
37							1				1									
38															1	1				
39								1												
40								1												

	21	22	23	24	25	26	27	28	29	30	31	32	33	34	35	36	37	38	39	40
1	1											1								
2	1											1								
3					1					1		1								
4	1											1								
5								1												
6	1											1								
7	1				1															
8	1																			
9																				
10																				
11	1				1															
12	1				1			1			1	1			1		1			
13																				
14	1																			
15																				
16	1											1								
17																				
18	1							1				1								
19	1											1								
20	1				1							1								
21								1												
22	1																			
23	1							1				1								
24	1				1															
25	1											1								
26	1											1								
27																				
28																				
29	1											1								
30	1				1															
31																				
32	1							1								1				
33																		1		
34	1											1								
35	1				1							1								
36																				
37															1					
38	1				1			1	1		1	1	1			1			1	
39	1											1								
40	1											1								

	41	42	43	44	45	46	47	48	49	50	51	52	53	54	55	56	57	58	59	60	61
1						1															
2						1															
3						1															
4						1															
5																					
6						1															
7						1															
8																					
9																					
10																					
11						1															
12						1					1				1					1	
13																					
14																					
15																					
16						1															
17																					
18						1															
19						1			1												
20				1		1															
21									1												
22						1	1														
23						1															
24						1															
25						1															
26						1															
27																					
28																					
29						1															
30						1															
31																					
32				1		1		1										1			
33												1			1					1	1
34						1															
35						1															
36																					
37																					
38		1				1		1	1						1					1	1
39				1		1															
40						1															

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
41								1												
42										1										
43								1												
44																				
45								1												
46																				
47																				
48								1												
49																				
50							1		1	1	1									1
51							1		1		1		1		1					1
52														1						
53															1					
54								1												
55								1		1					1					
56								1												
57								1												
58								1												
59																				
60																				
61																				

	21	22	23	24	25	26	27	28	29	30	31	32	33	34	35	36	37	38	39	40
41	1										1	1								
42	1				1			1			1	1				1				
43	1										1	1								
44																				
45	1											1								
46																				
47																				
48	1											1								
49																				
50	1	1			1			1				1			1	1	1		1	
51	1	1									1				1	1	1		1	
52	1							1			1		1					1		
53							1				1					1				
54	1																			
55	1				1		1		1			1				1				
56	1											1								
57	1																			
58	1							1				1								
59															1					
60	1																			
61																				

	41	42	43	44	45	46	47	48	49	50	51	52	53	54	55	56	57	58	59	60	61
41						1												1			
42			1												1						
43						1															
44																					
45						1															
46																					
47						1															
48																					
49																					
50						1	1								1					1	1
51						1	1								1				1		
52															1					1	1
53																					
54						1										1					
55			1			1							1								
56						1															
57						1										1					
58				1		1															
59																					
60															1						1
61															1						

APPENDIX E.4: ATTRIBUTE COUPLING MATRIX FOR DNS

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
1								1									1			
2								1									1			
3				1								5								
4								1												
5																				
6								1												
7								1												
8																				
9																				
10																				
11								1												
12							0		2	1	0			0						
13																				
14																				
15																				
16								1												
17																				
18								1					1							
19								1												
20										1										
21					2			1			0									
22																				
23								1												
24								1												
25								1												
26								1												
27																				
28																				
29								1												
30								1												
31																				
32								1		1										
33																				
34								1												
35													1							
36																				
37							0				0									
38														0	1					
39								1												
40								1												

	21	22	23	24	25	26	27	28	29	30	31	32	33	34	35	36	37	38	39	40
1	12																			
2	12																			
3					1					0		8								
4	12																			
5								0												
6	12																			
7	12																			
8																				
9																				
10																				
11	12																			
12	12				1			0			2	8			4		9			
13																				
14	12																			
15																				
16	12																			
17																				
18	12							0												
19	12																			
20	12				1							8								
21								0												
22																				
23	12							0												
24	12																			
25	12																			
26	12																			
27																				
28																				
29	12																			
30	12																			
31																				
32	12							0								2				
33																		10		
34																				
35	12				1															
36																				
37															4					
38	12				1			0	1		1	8	4			4			7	
39																				
40																				

	41	42	43	44	45	46	47	48	49	50	51	52	53	54	55	56	57	58	59	60	61
1						1															
2						1															
3						2															
4						1															
5																					
6						1															
7						1															
8																					
9																					
10																					
11						1															
12						1				6				7					0		
13																					
14																					
15																					
16						1															
17																					
18						1															
19						1			0												
20				0		0															
21									0												
22						4	1														
23						1															
24						1															
25						1															
26						1															
27																					
28																					
29						1															
30						1															
31																					
32				0		1		1										9			
33												7			7					0	0
34						1															
35						1															
36																					
37																					
38		1				1		1	0						7						0
39				0		1															
40						1															

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
41								1												
42										1										
43								1												
44																				
45								1												
46																				
47																				
48								1												
49																				
50							0		1	1	0									6
51							0		8		0		1		1					6
52														0						
53															0					
54								1												
55								1		1					1					
56								1												
57								1												
58								1												
59																				
60																				
61																				

	21	22	23	24	25	26	27	28	29	30	31	32	33	34	35	36	37	38	39	40
41											0									
42					1			0			6	8				1				
43											1									
44																				
45	12																			
46																				
47																				
48	12																			
49																				
50	12				1			0				8			4	2	9		7	
51	12										2				4	3	9		7	
52	12							0			2		4					10		
53							0				0					0				
54	12																			
55	12				1		2		1							4				
56	12																			
57	12																			
58								0												
59															0					
60	0																			
61																				

	41	42	43	44	45	46	47	48	49	50	51	52	53	54	55	56	57	58	59	60	61
41						1												9			
42			7												7						
43						1															
44																					
45						1															
46																					
47						1															
48																					
49																					
50						6	1								7					0	
51						8	1								7				0		
52															7						0
53																					
54						1															
55			7			2															
56						1															
57						1															
58				0		1															
59																					
60															0						0
61															0						

APPENDIX E.5: METHOD COUPLING MATRIX FOR DNS

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
1								1									4			
2								1									4			
3				1								2								
4								1												
5																				
6								1												
7								1												
8																				
9																				
10																				
11								1												
12							1		0	0	1			1						
13																				
14																				
15																				
16								1												
17																				
18								1					0							
19								1												
20										1										
21				8				3			1									
22																				
23								1												
24								1												
25								1												
26								1												
27																				
28																				
29								1												
30								1												
31																				
32								1		1										
33																				
34								1												
35													0							
36																				
37							1				1									
38														1	0					
39								1												
40								1												

	21	22	23	24	25	26	27	28	29	30	31	32	33	34	35	36	37	38	39	40
1	2																			
2	1																			
3					1					1		1								
4	1																			
5								1												
6	5																			
7	1																			
8																				
9																				
10																				
11	1																			
12	6				1			1			0	2			2		9			
13																				
14	6																			
15																				
16	1																			
17																				
18	1							1												
19	1																			
20	2				1							4								
21								1												
22																				
23	2							1												
24	1																			
25	5																			
26	5																			
27																				
28																				
29	2																			
30	1																			
31																				
32	3							1								0				
33																		1		
34																				
35	1				1															
36																				
37															1					
38	1				1			1	2		1	1	1			0			1	
39																				
40																				

	41	42	43	44	45	46	47	48	49	50	51	52	53	54	55	56	57	58	59	60	61
1						0															
2						0															
3						0															
4						0															
5																					
6						0															
7						0															
8																					
9																					
10																					
11						0															
12						2				4					3					2	
13																					
14																					
15																					
16						0															
17																					
18						0															
19						0			1												
20				1		1															
21									1												
22						0	6														
23						0															
24						0															
25						0															
26						2															
27																					
28																					
29						1															
30						0															
31																					
32				1		1		3										2			
33												1			1					2	2
34						0															
35						0															
36																					
37																					
38		1				0		1	1						9						0
39				1		0															
40						0															

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
41								1												
42										0										
43								1												
44																				
45								1												
46																				
47																				
48								1												
49																				
50							1		0	0	1									2
51							1		0		1		0		0					2
52														1						
53															1					
54								1												
55								1		1					0					
56								1												
57								1												
58								1												
59																				
60																				
61																				

	21	22	23	24	25	26	27	28	29	30	31	32	33	34	35	36	37	3	39	40
41											1									
42					1			1			0	1				0				
43											1									
44																				
45	1																			
46																				
47																				
48	1																			
49																				
50	6				1			1				5				6	0	6		1
51	5										0					10	0	7		4
52	1							1			0		1						2	
53							1				1					1				
54	1																			
55	2				1		0		3							2				
56	5																			
57	1																			
58								1												
59															0					
60	0																			
61																				

	41	42	43	44	45	46	47	48	49	50	51	52	53	54	55	56	57	58	59	60	61
41						0												2			
42			9												4						
43						0															
44																					
45						0															
46																					
47						0															
48																					
49																					
50						0	2								3					2	
51						2	2								4				1		
52															3						2
53																					
54						0															
55			1			1															
56						0															
57						0															
58				1		2															
59																					
60															0						0
61															0						

**APPENDIX E.6: TEST ORDERS OBTAINED WITH NUMBER OF BROKEN
DEPENDENCIES AS COST FUNCTION (DNS)**

Test Orders																				
1		2		3		4		5		6		7		8		9		10		
36	29	17	57	13	39	27	11	49	55	44	22	44	54	10	39	9	2	49	25	
49	58	27	58	36	34	31	24	36	58	13	39	49	2	44	48	17	40	36	40	
17	41	31	41	28	19	36	16	10	19	17	34	17	34	9	45	46	45	10	39	
10	25	10	39	5	16	9	23	27	2	10	23	28	29	15	20	47	43	46	20	
13	24	49	20	49	48	17	39	15	23	15	25	10	47	31	29	15	39	47	19	
28	30	9	13	44	43	28	35	17	6	31	11	5	22	28	53	36	55	15	7	
5	11	28	18	15	25	13	37	9	1	28	30	21	23	5	55	31	61	13	30	
27	7	15	35	9	11	49	59	31	40	5	58	14	43	36	61	49	60	9	35	
31	4	36	37	31	35	5	20	53	4	49	41	9	55	27	60	28	33	44	59	
21	48	53	59	21	59	21	34	46	41	27	16	36	42	49	33	5	48	17	11	
46	39	46	29	14	30	8	1	47	11	9	24	27	61	21	13	21	41	28	37	
47	35	47	55	8	24	10	19	44	56	21	35	15	60	8	18	8	56	5	24	
14	20	5	51	46	45	14	45	28	57	14	20	13	38	14	23	14	57	21	2	
22	45	21	61	47	20	44	54	5	42	8	7	8	33	46	54	10	6	8	41	
8	37	22	42	22	23	15	47	21	61	36	59	31	52	47	42	22	4	22	26	
44	53	14	54	27	53	53	22	22	60	46	53	53	6	22	38	13	42	14	27	
32	55	8	45	17	55	46	43	14	13	32	48	46	4	17	52	27	38	31	53	
15	61	44	40	10	42	32	55	8	35	26	37	32	20	32	24	53	26	32	55	
19	60	32	2	32	54	56	61	32	59	1	55	25	50	58	35	44	20	56	42	
18	42	16	34	29	61	57	60	25	18	19	51	11	51	25	59	32	54	1	51	
26	38	23	19	2	60	58	42	24	38	43	42	7	12	30	37	1	52	45	61	
1	33	1	6	6	33	41	51	30	33	29	61	56	30	7	50	58	34	4	60	
23	59	25	60	1	38	2	12	7	52	56	60	48	3	11	51	18	35	34	12	
2	54	24	33	56	52	26	40	26	20	57	38	19	58	4	12	25	37	23	38	
56	52	7	38	57	7	4	3	48	37	40	6	1	41	56	3	7	50	48	50	
57	9	11	12	58	37	6	38	16	50	2	50	26	57	57	41	30	59	43	52	
43	51	26	52	41	51	48	50	34	51	4	52	39	18	1	34	11	51	58	33	
34	12	48	50	40	12	29	52	29	54	45	12	24	40	19	2	24	12	29	54	
16	3	4	30	18	3	25	33	43	12	18	3	35	16	6	16	23	3	16	57	
40	50	43	3	4	50	7	18	45	3	54	33	37	45	40	26	29	19	6	3	
6		56		26		30		39		47		59		43		16		18		
Cost Function: Number of Dependencies		6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6	
Attributes (min = 19)		22	28	28	19	22	19	22	19	22	28	28	28	28	28	28	28	19	19	
Methods (min = 11)		11	11	11	11	11	11	11	11	11	11	11	11	11	11	11	11	11	11	

APPENDIX E.7: TEST ORDERS OBTAINED WHEN USING THE NUMBER OF STUBBED ATTRIBUTES AS COST FUNCTION (DNS)

Test Orders																				
1		2		3		4		5		6		7		8		9		10		
31	20	31	30	17	35	17	58	17	56	9	2	28	16	36	57	31	34	10	57	
10	30	36	41	13	37	49	4	15	57	36	6	5	1	49	29	36	26	15	34	
44	24	10	29	31	59	9	16	27	26	17	43	27	6	27	6	49	45	28	29	
13	7	13	23	44	48	28	25	31	25	15	48	17	29	17	1	9	56	31	2	
49	11	17	19	49	20	5	20	13	24	10	1	46	23	31	43	15	57	9	26	
28	37	49	43	36	26	46	35	9	55	46	58	47	48	44	26	13	43	5	16	
46	29	15	39	46	9	13	59	36	42	47	41	31	25	28	25	17	39	36	20	
47	58	44	14	47	24	15	30	53	20	27	39	49	20	5	11	27	41	13	40	
27	4	46	45	15	16	47	11	10	35	49	25	9	7	10	7	53	25	17	1	
15	23	47	34	28	54	10	54	28	59	31	11	15	24	13	24	28	11	46	18	
5	43	28	20	5	22	21	7	5	7	28	30	21	30	9	20	5	7	47	48	
21	56	5	16	21	23	14	37	46	30	5	20	14	35	21	35	21	24	49	43	
22	57	21	35	14	27	8	31	47	11	21	7	8	59	14	59	8	35	21	55	
8	53	22	59	8	43	36	43	49	37	22	24	54	11	8	37	14	37	8	42	
54	55	8	48	10	53	44	41	21	1	8	35	10	37	46	30	44	59	14	61	
14	42	54	37	32	55	32	24	14	2	54	59	13	53	54	15	46	30	54	60	
36	61	32	27	29	42	56	45	22	16	13	53	44	55	47	53	47	48	27	38	
32	60	26	53	34	56	23	27	8	34	14	55	22	42	32	55	22	55	22	52	
39	38	1	55	18	57	2	53	54	48	44	42	36	61	16	51	54	61	53	33	
45	52	56	42	19	45	29	55	44	18	32	61	32	60	45	42	10	42	44	35	
40	33	57	61	4	51	19	42	32	51	4	60	43	38	40	61	32	60	32	37	
16	17	58	60	39	61	40	61	45	61	34	38	18	52	48	60	29	38	25	59	
19	1	6	38	1	60	26	60	6	60	45	52	39	51	39	38	16	52	11	50	
34	2	18	52	6	38	22	38	29	12	56	33	19	12	22	12	4	33	24	45	
6	9	40	33	2	12	57	50	19	38	57	23	40	3	23	50	1	20	30	58	
48	51	4	9	58	52	34	52	58	52	40	37	26	50	56	52	6	50	7	41	
18	50	2	50	41	50	48	33	41	33	19	51	2	34	18	33	23	51	39	51	
26	12	25	51	40	30	6	51	23	4	26	50	4	33	2	4	40	12	4	12	
25	3	7	12	25	3	1	12	40	3	16	12	56	58	58	3	18	3	19	3	
35	41	11	3	11	33	18	3	43	50	18	3	57	41	41	19	19	2	23	6	
59		24		7		39		39		29		45		34		58		56		
Cost Function:		19		19		19		19		19		19		19		19		19		
Number of Attributes		19		19		19		19		19		19		19		19		19		
Methods (min = 11)		11		11		11		11		11		11		11		11		11		
Dependencies (min = 6)		7		7		7		6		7		7		7		7		7		

APPENDIX E.8: TEST ORDERS OBTAINED WHEN USING THE NUMBER OF STUBBED METHODS AS COST FUNCTION (DNS)

Test Orders																					
1		2		3		4		5		6		7		8		9		10			
9	56	9	4	27	1	44	45	27	4	28	57	36	57	31	26	17	26	49	30		
49	57	10	6	31	26	10	34	10	2	46	19	13	23	49	56	36	2	15	24		
44	34	49	19	9	43	27	25	49	25	36	26	15	39	28	57	10	40	46	11		
17	2	13	29	17	4	31	20	46	24	9	29	44	20	15	4	27	30	44	45		
28	25	17	26	49	41	49	7	28	11	31	40	31	18	5	48	9	58	17	43		
46	11	15	41	15	29	28	11	5	7	13	45	46	48	36	25	15	1	13	19		
10	24	27	40	13	48	15	35	21	30	49	23	28	40	10	24	31	6	27	48		
47	20	36	48	36	45	13	59	8	43	15	18	5	35	46	35	28	18	47	2		
31	35	28	39	44	19	9	37	54	48	47	2	47	37	47	59	5	23	28	39		
5	59	5	35	10	30	17	16	31	1	5	48	49	59	17	30	46	16	5	35		
21	7	21	59	28	22	36	30	36	29	21	25	21	27	27	7	47	56	21	59		
8	37	8	20	5	40	53	24	47	23	44	55	22	53	53	11	49	57	8	20		
22	45	14	1	21	56	5	40	13	18	27	42	14	55	9	37	21	48	54	55		
14	58	44	43	14	57	21	55	22	39	14	61	8	51	13	6	22	41	14	42		
54	41	31	55	8	39	14	61	9	35	22	60	54	42	21	1	8	39	9	61		
27	40	53	61	53	55	8	42	17	37	8	38	10	61	8	58	54	20	36	60		
13	30	46	42	46	42	46	60	15	59	17	52	32	60	54	41	13	55	22	38		
15	48	47	34	54	61	32	38	53	55	54	33	26	38	22	20	14	42	31	33		
36	29	22	60	47	60	23	52	14	42	53	20	25	33	14	43	44	61	53	52		
53	55	54	33	32	38	18	33	44	61	10	35	7	50	44	55	53	60	10	6		
32	42	32	16	34	52	2	56	32	60	32	59	11	12	32	51	32	33	32	56		
43	51	45	2	16	33	4	57	19	38	16	11	9	41	19	61	34	38	26	57		
18	61	18	38	2	35	48	58	58	52	34	7	24	52	34	60	25	52	23	58		
16	60	56	11	6	37	54	41	41	33	58	37	43	4	45	12	24	35	34	41		
4	12	57	37	58	59	39	47	6	20	39	50	30	3	16	3	7	37	1	7		
23	3	23	51	23	20	43	22	56	50	41	51	6	45	29	33	11	50	40	37		
39	33	58	50	18	51	6	51	57	51	43	12	16	19	18	42	45	59	18	50		
6	38	25	12	25	12	26	12	40	12	6	30	56	17	2	38	19	51	16	51		
26	50	24	3	7	3	19	3	45	3	4	3	29	1	40	50	43	12	29	12		
19	52	7	52	24	50	29	50	34	16	1	24	34	2	23	52	4	3	4	3		
1		30		11		1		26		56		58		39		29		25			
Cost function: Methods (min = 11)		11	11	11	11	11	11	11	11	11	11	11	11	11	11	11	11	11	11		
Attributes (min = 19)		28	28	19	19	19	19	19	19	19	22	28	28	28	28	28	28	28	22		
Dependencies (min = 6)		7	7	7	7	7	7	7	7	7	7	7	7	7	7	7	7	7	7		

**APPENDIX E.9: TEST ORDERS OBTAINED USING A WEIGHTED
GEOMETRIC AVERAGE OF ATTRIBUTE AND METHOD COUPLING
MEASURES AS THE COST FUNCTION (DNS)**

Test Orders																			
1	2	3	4	5	6	7	8	9	10										
27	48	36	25	10	24	44	1	49	4	46	23	15	20	28	24	36	43	27	35
15	29	44	35	44	45	49	56	27	1	31	4	10	6	5	30	49	18	46	59
17	6	17	59	17	26	10	57	44	23	27	48	44	43	36	7	31	29	47	30
31	34	49	24	49	23	28	16	17	39	10	58	49	40	10	43	46	19	49	2
13	4	13	30	31	18	46	39	10	35	28	41	46	55	9	16	27	6	13	26
9	19	28	20	46	1	47	35	13	59	44	22	47	42	49	19	9	23	10	58
36	56	5	11	9	20	27	37	28	7	5	45	36	16	15	1	13	2	17	41
49	57	21	58	13	58	13	59	5	30	49	25	31	1	27	39	15	39	36	20
46	58	14	29	15	41	15	20	21	20	9	7	27	58	13	35	10	25	15	45
47	41	10	15	28	40	9	19	14	41	21	30	28	2	21	37	47	20	44	34
10	25	27	53	27	29	17	23	15	47	8	20	5	4	8	59	28	24	31	9
28	20	31	55	5	19	5	29	31	22	54	24	21	41	17	55	5	35	28	11
5	55	8	61	21	16	21	34	9	18	15	11	14	34	46	42	53	59	5	37
21	42	46	60	8	35	22	6	36	11	14	53	22	61	47	61	21	11	21	56
22	61	54	42	54	37	8	45	8	37	17	55	8	60	22	60	8	55	14	57
8	60	47	38	14	53	54	58	46	43	36	42	54	38	14	20	22	42	22	53
54	38	9	52	36	55	14	41	54	40	32	61	53	52	54	48	14	61	8	55
14	52	32	33	32	42	31	48	32	2	26	60	17	45	31	23	54	60	32	51
53	33	4	7	56	61	36	53	45	53	1	38	32	9	53	58	17	38	18	42
44	11	1	37	57	60	32	55	29	55	19	52	19	23	44	18	44	52	39	61
32	7	43	26	48	38	25	42	25	51	16	33	29	13	32	38	32	33	6	60
39	30	45	40	43	52	7	61	56	42	47	13	25	18	2	52	1	30	48	12
18	35	23	16	6	33	11	60	48	61	34	18	30	35	45	50	16	58	29	38
45	59	19	22	39	59	24	38	26	60	43	35	7	37	26	33	48	41	23	50
43	37	34	51	2	47	30	52	57	12	2	37	24	50	29	57	26	7	1	52
1	51	39	12	34	22	26	50	24	3	29	59	11	59	4	51	45	37	43	54
16	12	56	3	4	51	40	33	19	38	39	51	39	33	40	12	56	51	19	4
26	3	57	50	25	12	18	51	16	50	56	12	26	51	6	3	57	12	16	3
2	50	2	41	7	3	4	12	34	52	57	3	48	12	56	34	40	50	25	40
40	24	18	6	11	50	43	3	58	33	40	50	56	3	25	41	4	3	24	33
23		48		30		2		6		6		57		11		34		7	
Cost Function: OCplx ($W_M=W_A=0.5$)										1.47	1.47	1.47	1.47	1.47	1.47	1.47	1.47	1.47	1.47
Attributes (min = 19)										19	19	19	19	19	19	19	19	19	19
Methods (min = 11)										11	11	11	11	11	11	11	11	11	11
Dependencies (min = 6)										7	7	7	7	7	7	7	7	7	6

APPENDIX E.7: TEST ORDERS OBTAINED WHEN USING THE NUMBER OF STUBBED ATTRIBUTES AS COST FUNCTION (DNS)

Test Orders																			
1	2	3	4	5	6	7	8	9	10										
31	20	31	30	17	35	17	58	17	56	9	2	28	16	36	57	31	34	10	57
10	30	36	41	13	37	49	4	15	57	36	6	5	1	49	29	36	26	15	34
44	24	10	29	31	59	9	16	27	26	17	43	27	6	27	6	49	45	28	29
13	7	13	23	44	48	28	25	31	25	15	48	17	29	17	1	9	56	31	2
49	11	17	19	49	20	5	20	13	24	10	1	46	23	31	43	15	57	9	26
28	37	49	43	36	26	46	35	9	55	46	58	47	48	44	26	13	43	5	16
46	29	15	39	46	9	13	59	36	42	47	41	31	25	28	25	17	39	36	20
47	58	44	14	47	24	15	30	53	20	27	39	49	20	5	11	27	41	13	40
27	4	46	45	15	16	47	11	10	35	49	25	9	7	10	7	53	25	17	1
15	23	47	34	28	54	10	54	28	59	31	11	15	24	13	24	28	11	46	18
5	43	28	20	5	22	21	7	5	7	28	30	21	30	9	20	5	7	47	48
21	56	5	16	21	23	14	37	46	30	5	20	14	35	21	35	21	24	49	43
22	57	21	35	14	27	8	31	47	11	21	7	8	59	14	59	8	35	21	55
8	53	22	59	8	43	36	43	49	37	22	24	54	11	8	37	14	37	8	42
54	55	8	48	10	53	44	41	21	1	8	35	10	37	46	30	44	59	14	61
14	42	54	37	32	55	32	24	14	2	54	59	13	53	54	15	46	30	54	60
36	61	32	27	29	42	56	45	22	16	13	53	44	55	47	53	47	48	27	38
32	60	26	53	34	56	23	27	8	34	14	55	22	42	32	55	22	55	22	52
39	38	1	55	18	57	2	53	54	48	44	42	36	61	16	51	54	61	53	33
45	52	56	42	19	45	29	55	44	18	32	61	32	60	45	42	10	42	44	35
40	33	57	61	4	51	19	42	32	51	4	60	43	38	40	61	32	60	32	37
16	17	58	60	39	61	40	61	45	61	34	38	18	52	48	60	29	38	25	59
19	1	6	38	1	60	26	60	6	60	45	52	39	51	39	38	16	52	11	50
34	2	18	52	6	38	22	38	29	12	56	33	19	12	22	12	4	33	24	45
6	9	40	33	2	12	57	50	19	38	57	23	40	3	23	50	1	20	30	58
48	51	4	9	58	52	34	52	58	52	40	37	26	50	56	52	6	50	7	41
18	50	2	50	41	50	48	33	41	33	19	51	2	34	18	33	23	51	39	51
26	12	25	51	40	30	6	51	23	4	26	50	4	33	2	4	40	12	4	12
25	3	7	12	25	3	1	12	40	3	16	12	56	58	58	3	18	3	19	3
35	41	11	3	11	33	18	3	43	50	18	3	57	41	41	19	19	2	23	6
59		24		7		39		39		29		45		34		58		56	
Cost Function:																			
Number of Attributes										19	19	19	19	19	19	19	19	19	19
Methods (min = 11)										11	11	11	11	11	11	11	11	11	11
Dependencies (min = 6)										7	7	7	6	7	7	7	7	7	7

APPENDIX E.8: TEST ORDERS OBTAINED WHEN USING THE NUMBER OF STUBBED METHODS AS COST FUNCTION (DNS)

Test Orders																					
1		2		3		4		5		6		7		8		9		10			
9	56	9	4	27	1	44	45	27	4	28	57	36	57	31	26	17	26	49	30		
49	57	10	6	31	26	10	34	10	2	46	19	13	23	49	56	36	2	15	24		
44	34	49	19	9	43	27	25	49	25	36	26	15	39	28	57	10	40	46	11		
17	2	13	29	17	4	31	20	46	24	9	29	44	20	15	4	27	30	44	45		
28	25	17	26	49	41	49	7	28	11	31	40	31	18	5	48	9	58	17	43		
46	11	15	41	15	29	28	11	5	7	13	45	46	48	36	25	15	1	13	19		
10	24	27	40	13	48	15	35	21	30	49	23	28	40	10	24	31	6	27	48		
47	20	36	48	36	45	13	59	8	43	15	18	5	35	46	35	28	18	47	2		
31	35	28	39	44	19	9	37	54	48	47	2	47	37	47	59	5	23	28	39		
5	59	5	35	10	30	17	16	31	1	5	48	49	59	17	30	46	16	5	35		
21	7	21	59	28	22	36	30	36	29	21	25	21	27	27	7	47	56	21	59		
8	37	8	20	5	40	53	24	47	23	44	55	22	53	53	11	49	57	8	20		
22	45	14	1	21	56	5	40	13	18	27	42	14	55	9	37	21	48	54	55		
14	58	44	43	14	57	21	55	22	39	14	61	8	51	13	6	22	41	14	42		
54	41	31	55	8	39	14	61	9	35	22	60	54	42	21	1	8	39	9	61		
27	40	53	61	53	55	8	42	17	37	8	38	10	61	8	58	54	20	36	60		
13	30	46	42	46	42	46	60	15	59	17	52	32	60	54	41	13	55	22	38		
15	48	47	34	54	61	32	38	53	55	54	33	26	38	22	20	14	42	31	33		
36	29	22	60	47	60	23	52	14	42	53	20	25	33	14	43	44	61	53	52		
53	55	54	33	32	38	18	33	44	61	10	35	7	50	44	55	53	60	10	6		
32	42	32	16	34	52	2	56	32	60	32	59	11	12	32	51	32	33	32	56		
43	51	45	2	16	33	4	57	19	38	16	11	9	41	19	61	34	38	26	57		
18	61	18	38	2	35	48	58	58	52	34	7	24	52	34	60	25	52	23	58		
16	60	56	11	6	37	54	41	41	33	58	37	43	4	45	12	24	35	34	41		
4	12	57	37	58	59	39	47	6	20	39	50	30	3	16	3	7	37	1	7		
23	3	23	51	23	20	43	22	56	50	41	51	6	45	29	33	11	50	40	37		
39	33	58	50	18	51	6	51	57	51	43	12	16	19	18	42	45	59	18	50		
6	38	25	12	25	12	26	12	40	12	6	30	56	17	2	38	19	51	16	51		
26	50	24	3	7	3	19	3	45	3	4	3	29	1	40	50	43	12	29	12		
19	52	7	52	24	50	29	50	34	16	1	24	34	2	23	52	4	3	4	3		
1		30		11		1		26		56		58		39		29		25			
Cost function: Methods (min = 11)		11		11		11		11		11		11		11		11		11			
Attributes (min = 19)		28		28		19		19		19		19		22		28		28		22	
Dependencies (min = 6)		7		7		7		7		7		7		7		7		7		7	

**APPENDIX E.9: TEST ORDERS OBTAINED USING A WEIGHTED
GEOMETRIC AVERAGE OF ATTRIBUTE AND METHOD COUPLING
MEASURES AS THE COST FUNCTION (DNS)**

Test Orders																				
1	2	3	4	5	6	7	8	9	10											
27	48	36	25	10	24	44	1	49	4	46	23	15	20	28	24	36	43	27	35	
15	29	44	35	44	45	49	56	27	1	31	4	10	6	5	30	49	18	46	59	
17	6	17	59	17	26	10	57	44	23	27	48	44	43	36	7	31	29	47	30	
31	34	49	24	49	23	28	16	17	39	10	58	49	40	10	43	46	19	49	2	
13	4	13	30	31	18	46	39	10	35	28	41	46	55	9	16	27	6	13	26	
9	19	28	20	46	1	47	35	13	59	44	22	47	42	49	19	9	23	10	58	
36	56	5	11	9	20	27	37	28	7	5	45	36	16	15	1	13	2	17	41	
49	57	21	58	13	58	13	59	5	30	49	25	31	1	27	39	15	39	36	20	
46	58	14	29	15	41	15	20	21	20	9	7	27	58	13	35	10	25	15	45	
47	41	10	15	28	40	9	19	14	41	21	30	28	2	21	37	47	20	44	34	
10	25	27	53	27	29	17	23	15	47	8	20	5	4	8	59	28	24	31	9	
28	20	31	55	5	19	5	29	31	22	54	24	21	41	17	55	5	35	28	11	
5	55	8	61	21	16	21	34	9	18	15	11	14	34	46	42	53	59	5	37	
21	42	46	60	8	35	22	6	36	11	14	53	22	61	47	61	21	11	21	56	
22	61	54	42	54	37	8	45	8	37	17	55	8	60	22	60	8	55	14	57	
8	60	47	38	14	53	54	58	46	43	36	42	54	38	14	20	22	42	22	53	
54	38	9	52	36	55	14	41	54	40	32	61	53	52	54	48	14	61	8	55	
14	52	32	33	32	42	31	48	32	2	26	60	17	45	31	23	54	60	32	51	
53	33	4	7	56	61	36	53	45	53	1	38	32	9	53	58	17	38	18	42	
44	11	1	37	57	60	32	55	29	55	19	52	19	23	44	18	44	52	39	61	
32	7	43	26	48	38	25	42	25	51	16	33	29	13	32	38	32	33	6	60	
39	30	45	40	43	52	7	61	56	42	47	13	25	18	2	52	1	30	48	12	
18	35	23	16	6	33	11	60	48	61	34	18	30	35	45	50	16	58	29	38	
45	59	19	22	39	59	24	38	26	60	43	35	7	37	26	33	48	41	23	50	
43	37	34	51	2	47	30	52	57	12	2	37	24	50	29	57	26	7	1	52	
1	51	39	12	34	22	26	50	24	3	29	59	11	59	4	51	45	37	43	54	
16	12	56	3	4	51	40	33	19	38	39	51	39	33	40	12	56	51	19	4	
26	3	57	50	25	12	18	51	16	50	56	12	26	51	6	3	57	12	16	3	
2	50	2	41	7	3	4	12	34	52	57	3	48	12	56	34	40	50	25	40	
40	24	18	6	11	50	43	3	58	33	40	50	56	3	25	41	4	3	24	33	
23		48		30		2		6		6		57		11		34		7		
Cost Function: OCplx ($W_M=W_A=0.5$)											1.47	1.47	1.47	1.47	1.47	1.47	1.47	1.47	1.47	
Attributes (min = 19)											19	19	19	19	19	19	19	19	19	
Methods (min = 11)											11	11	11	11	11	11	11	11	11	
Dependencies (min = 6)											7	7	7	7	7	7	7	7	6	