LAPPEENRANTA UNIVERSITY OF TECHNOLOGY
DEPARTMENT OF INFORMATION TECHNOLOGY
LABORATORY OF INFORMATION PROCESSING
SOFTWARE ENGINEERING RESEARCH GROUP

THE IMPACT OF ARCHITECTURAL DESIGN ON SOFTWARE DEVELOPMENT

The subject for this thesis for the degree of Master of Science in Engineering was accepted by the Council of the Department of Information Technology on 11th September, 2002.

Examiners:	Prof. D.Sc. (Econ.) Jouni Lampinen, LUT
	Lic.Sc. (Tech.) Päivi Ovaska, LUT
Supervisors:	Prof. D.Sc. (Econ.) Jouni Lampinen, LUT
	Lic.Sc. (Tech.) Päivi Ovaska, LUT

Alexandre Bern LAPPEENRANTA, November 25, 2002

Teknologiapuistonkatu 4 E 4
53850 LAPPEENRANTA
GSM: +358 50 5233869
E-Mail: bern@lut.fi

ABSTRACT

Author:	Alexandre Bern
Department:	Department of Information Technology
Place:	Lappeenranta University of Technology
Subject:	**The Impact of Architectural Design on Software Development**
Master's Thesis:	78 pages, 7 figures, 32 tables, 8 appendices
Year:	2002
Examiner:	Prof. D.Sc. (Econ.) Jouni Lampinen, LUT Lic.Sc. (Tech.) Päivi Ovaska, LUT
Supervisors:	Prof. D.Sc. (Econ.) Jouni Lampinen, LUT Lic.Sc. (Tech.) Päivi Ovaska, LUT
Keywords:	architectural metric, coupling, differential evolution, evolutionary algorithm, soft computing, software architecture, software component

This thesis studies the impact of software architectural design properties on the development effort of a mobile service application that has a client-server architecture. The data applied is based on a real-life software project in which, during the qualitative analysis, it was observed that the coupling between the architectural components had a strong influence on the development effort. The main objective of this research was to quantitatively investigate the correctness of the above observation. To accomplish this task, an architectural design metrics suite was created to describe the subsystems, a client and a server, of the system studied, and two models that use the suite, a linear and non-linear model, were selected to estimate the development effort (the sum of the design, implementation and testing times of a component). Using a non-linear global optimisation method, a differential evolution algorithm, the free parameters of the models were defined, or optimized, using first all the architectural design properties, also known as attributes, and then leaving them out one by one in such a way that the models corresponded as accurately as possible with the measured development effort. When leaving out coupling, which is defined as the number of components to which a component being studied refers, the error between the measured and estimated development effort increased in some cases by 367 %, meaning that the model did not fit the data well without coupling. This was the highest increase in the error for all the attributes excluded. Based on these results, it was concluded that the development effort of the system under study was clearly dependent on coupling and that coupling was probably the most important architectural design property with respect to the development effort of the system.

TIIVISTELMÄ

Tekijä:	Alexandre Bern
Osasto:	Tietotekniikan osasto
Paikka:	Lappeenrannan teknillinen korkeakoulu
Nimi:	**Arkkitehtuurisuunnittelun vaikutus ohjelmiston toteutukseen**
Diplomityö:	78 lehteä, 7 kuvaa, 32 taulukkoa, 8 liitettä
Vuosi:	2002
Tarkastaja:	Prof. KTT Jouni Lampinen, LTKK TkL Päivi Ovaska, LTKK
Ohjaajat:	Prof. KTT Jouni Lampinen, LTKK TkL Päivi Ovaska, LTKK
Hakusanat:	architectural metric, coupling, differential evolution, evolutionary algorithm, soft computing, software architecture, software component

Tässä työssä tutkitaan ohjelmistoarkkitehtuurisuunnitteluominaisuuksien vaikutusta erään client-server –arkkitehtuuriin perustuvan mobiilipalvelusovelluksen suunnittelu- ja toteutusaikaan. Kyseinen tutkimus perustuu reaalielämän projektiin, jonka kvalitatiivinen analyysi paljasti arkkitehtuurikompponenttien välisten kytkentöjen merkittävästi vaikuttavan projektin työmäärään. Työn päätavoite oli kvantitatiivisesti tutkia yllä mainitun havainnon oikeellisuus. Tavoitteen saavuttamiseksi suunniteltiin ohjelmistoarkkitehtuurisuunnittelun mittaristo kuvaamaan kyseisen järjestelmän alijärjestelmien arkkitehtuuria ja luotiin kaksi suunniteltua mittaristoa käyttävää, työmäärää (komponentin suunnittelu-, toteutus- ja testausaikojen summa) arvioivaa mallia, joista toinen on lineaarinen ja toinen epälineaarinen. Näiden mallien kertoimet sovitettiin optimoimalla niiden arvot epälineaarista gloobaalioptimointimenetelmää, differentiaalievoluutioalgoritmia, käyttäen, niin että mallien antamat arvot vastasivat parhaiten mitattua työmäärää sekä kaikilla ominaisuuksilla eli attribuuteilla että vain osalla niistä (yksi jätettiin vuorotellen pois). Kun arkkitehtuurikompenttien väliset kytkennät jätettiin malleista pois, mitattujen ja arvoitujen työmäärien välinen ero (ilmaistuna virheenä) kasvoi eräässä tapauksessa 367 % entisestä tarkoittaen sitä, että näin muodostettu malli vastasi toteutusaikoja huonosti annetulla ainestolla. Tämä oli suurin havaitu virhe kaikkien poisjätettyjen ominaisuuksien kesken. Saadun tuloksen perusteella päätettiin, että kyseisen järjestelmän toteutusajat ovat vahvasti riippuvaisia kytkentöjen määrästä, ja näin ollen kytkentöjen määrä oli mitä todennäköisemmin kaikista tärkein työmäärään vaikuttava tekijä tutkitun järjestelmän arkkitehtuurisuunnittelussa.

Acknowledgements

This Master's thesis was one of my primary objectives and is the result of many years of hard work. I would like to thank all the professors and lecturers who made it possible for me to complete my studies in such a short time. I have learnt a lot from them all. I especially wish to thank my supervisors, Prof. Jouni Lampinen and Mrs. Päivi Ovaska for their valuable advice and cooperation in this project. They have made a tremendous contribution to this work.

I dedicate this work to my parents, Victor and Valentina Bern, and am greatly thankful to them for the total freedom that they gave me with respect to my studies. For the last 15 years I have felt no pressure from them in my studies, which made it possible for me to attain such a high degree in a relatively short time.

Lappeenranta, November 25, 2002

Alexandre Bern

TABLE OF CONTENTS

SYMBOLS AND ABBREVIATIONS

AI	**A**rtificial **I**ntelligence
ANN	**A**rtificial **N**eural **N**etwork
ANOVA	**AN**alysis **O**f **VA**riance
COCOMO	**CO**nstructive **CO**st **MO**del
CORBA	**C**ommon **O**bject **R**equest **B**roker **A**rchitecture
DE	**D**ifferential **E**volution
EA	**E**volutionary **A**lgorithm
HHM	**H**idden **M**arkov **M**odels
GA	**G**enetic **A**lgorithm
KLOC	**K**ilo **L**ines **O**f **C**ode
LUT	**L**appeenranta **U**niversity of **T**echnology
MLP	**M**ulti-**L**ayer **P**erceptron
N/A	**N**ot **A**vailable
OLS	**O**rdinary **L**east **S**quares
PCAP	**P**rogrammer **CAP**ability
TSP	**T**raveling **S**alesman **P**roblem
UML	**U**nified **M**odelling **L**anguage

1 INTRODUCTION

It is common sense that building a house without an architectural design is impossible. Before laying the foundation of the house, or sometimes even reserving a site for it, it is very important to design the house and elaborate an architectural design that satisfies the customer's needs (requirements). A properly and carefully designed architectural design will enable a sufficiently accurate time schedule to be prepared for the house and can ensure the house's stability and comfort. A poor architectural design, for its part, can dramatically influence the progress of the construction of the house; the time schedule may suffer, the final result could be terrible, and other unwanted consequences may arise. Once an architectural design has been prepared, it has to be followed and referred to throughout the whole construction period. This means that the architectural design plays the most essential role during the whole construction process, and its role is indisputable.

Even thought software engineering is a relatively young field when compared with construction, it still has a lot of similarities with the latter. Before creating a solid or less solid software system, the architecture must first be created by software architects and validated by the customer. Once the architecture has been carefully design and no mistakes have been found, the software engineers and developers are ready to start building the system. As in the first case, the role of a software architect is one of great importance during the whole software construction (development) process. It is the architect's further responsibility to ensure that the software engineers understand the whole architecture and follow it. The architecture must be sufficiently detailed in order for the developers to understand their own duties and fluently interact with each other when putting the architectural components together.

A poorly designed architecture will leave very bad prints on the software and can easily destroy the whole business process of a company as well as the company's reputation. The general principle is that the later a mistake (e.g. a software bug) is

found, the more expensive its removal becomes. Again, if the architecture has been designed properly and carefully, with the customer's satisfaction (having thus a low fault risk), it is very likely that the software will be released on time, which will be mutually satisfactory.

Just like projects in other fields, every software project (like projects of other branches) requires timetables for its implementation. There are deadlines for the releases and nobody (neither managers nor engineers) likes deadlines to get closer. This is the reason why it would be very nice and essential to be able to define to a sufficient degree of accuracy the development effort (or, simply, the effort) based on the software architecture or even before the architecture is designed. Here, the *development effort* is defined as being the sum of the design, implementation and testing times of a component.

It would be of great value to extract the architectural design properties (also the architectural attributes from this perspective) that have the most crucial impact on the development effort. Once these attributes are known, it is easy to manipulate the effort right from the beginning by completely or partially avoiding them.

This research focuses on a mobile service application that has a client-server architecture. The research was performed to support a doctoral thesis that partially focused on the same results from the qualitative point of view. Qualitative analysis was used to study human behaviour in application development and indicated that coupling between architectural components had a critical influence on the development effort of the system. More information on qualitative analysis (in general) may be found in [38].

The emphasis of this work was on obtaining and analysing the quantitative results that describe in numbers the structure of the software architecture, and to compare these results with qualitative ones. The main objective was to study the correctness of the following hypothesis:

Coupling between the architectural components of the system plays an important role in the development effort of the system.

Based on the architecture of the system, which is composed of two separate subsystems, a server and client, the most effective attributes have been defined, and it was noticed that according to the importance rule (see chapter 5.2.2), coupling, which is defined as the number of components to which the component under study refers, did indeed play an important role in the development effort of the system. This conclusion was reached by defining an architectural design metrics suite, which describes the architectures of the subsystems, and by creating two development effort estimating models, a linear and a non-linear model that both use this suite. The free parameters (or simply the parameters) of the models were defined by applying a non-linear global optimisation approach and, in particular, by minimising the objective functions that involve the models and the measured development effort using a novel soft computing method, that is, an evolutionary optimisation algorithm called a differential evolution (DE) algorithm. The above-mentioned algorithm employed two different strategies: *DE/rand/*1 and *DE/best/*1 (see chapter 5.3.4). The importance of a parameter was defined by leaving out the corresponding attribute from the model and by investigating the increase in the error between the measured and estimated effort.

No generalisations have been made on the basis of the results. It is very probable that the results apply to the system studied here only; however, the approach used for obtaining the results is generalisable with a high level of probability. All the decisions made apply to this project and the system studied. The suggested metrics suite is also assumed to be satisfactory only for the system studied here.

In the next session, related work is discussed. Chapter 3 discusses the theory of software architecture in general, giving a concrete example of software architecture and presenting architectural styles, and contains an overview of software metrics. Chapter 4 discusses the architecture of the mobile service

application as well as the architectural design metrics suite used in this project. Section 5 presents the data applied and method used as well as the selected models. The results are presented in chapter 6 and discussed in chapter 7. The last section (Chapter 7) also discusses the performance of the selected approach for achieving the results and offers some suggestions for further research.

2 RELATED WORK

With the proper metrics suite (see chapter 3.4), software development can be evaluated for its cost, quality, fault tolerance and maintenance. A lot of research has been carried out in this field. Probably one of the most famous papers in this field is [5], in which L. C. Briand and J. Wüst present the results of their studies on the impact of coupling, cohesion and complexity on the development cost of object-oriented systems. L. C. Briand and J. Wüst obtained acceptable results using traditional statistical methods such as Poisson regression and regression trees.

In addition to the work done by Briand and Wüst, a lot of related work has been done. In [6], L. C. Briand, K. El Emam and F. Bomarius present a hybrid method for estimating software cost, benchmarking and for assessing risk. Their method is based on a productivity estimation model consisting of two components: the cost overhead and a productivity model. R. Jeffery, M. Ruhe and I. Wieczorek estimate the software development effort using public domain metrics [7]. In their work, they use Ordinary Least Squares regression (OLS regression), stepwise Analysis of Variance (stepwise ANOVA), regression trees (CART) and analogy. In [8], K. Pillai and V.S. Sukumaran Nair describe Putnam's SLIM model that offers a method for estimating the cost and effort of software development. In [9], S. H. Zweben, S. H. Edwards, B. W. Weide and J. E. Hollingsworth study how layering and encapsulating impacts on the cost and quality of software development. They start by assuming that the layering approach should result in reduced development costs and the increased quality of the new components through the increased reuse of existing ones.

In addition to object-oriented systems, function-based systems have been studied. In [9], J. E. Matson, B. E. Barrett and J. M. Mellichamp estimate the cost of software development by using function point analysis, a method for quantifying the size and complexity of a software system. In [10], Y. Yokoyama and M.

Kodaira use the multiple regression analysis method to evaluate the cost and quality of software.

Studies have also been carried out on the quality of software only. In [11], J. Bansiya and C. G. Davis present a hierarchical model for assessing the quality of object-oriented design. They use a suite of object-oriented design metrics and the model relates design properties such as encapsulation, modularity, coupling and cohesion to high-level quality attributes, which are reusability, flexibility and complexity.

3 SOFTWARE ARCHITECTURE

As has already been mentioned, software architecture plays an extremely important role in software production. But unlike architecture in traditional fields (real estate and machine construction), software architectures did not appear as a well-defined area in software engineering. Rather, they have passed through a series of evolutionary cycles, which is the result of the desire of software engineers to improve the process of building ever more complex and demanding software systems ([16], pp.1). As a result, many different architectural styles and paradigms have been created.

Section 3.1 discusses software architectures in general; section 3.2 gives an illustrative example of software architecture; sections 3.3 presents the main architectural styles; section 3.4 of this chapter discusses software metrics, dividing them into *high-* and *low-level* software metrics and providing an overview of both of types of software metrics.

3.1 Software Architecture in General

The architecture of software can be compared with that of a building, which describes the main components of the building. These components can be the building blocks, floors, rooms, doors and windows (to communicate with the external world), the pipes that connect the building blocks etc. M. Shaw and D. Garlan give the following definition for software architecture in [16], pp. 1: *"Abstractly, software architecture involves the description of elements from which systems are built, interaction among those elements, patterns that guide their composition, and constraints on these patterns. In general, a particular system is defined in terms of a collection of components and interactions among those components. Such a system may in turn be used as a (composite) element in a larger system design."* From the arguments presented above, many similarities can easily be found between the architecture of a building and that of software. In

any case, the definition given by Shaw and Garlan is not the only one. For example, the following definition for architecture can be found in [17], pp. 27: *"Architecture is the structure of the components of a program or system, their interrelationships, and principles and guidelines governing their design and evolution over time."* Using slightly different words, Bass, Clements and Kazman present the same concept as Shaw and Garlan. Generally, no common definition exists for software architecture.

To better understand the meaning of a software component, [39] gives the following definition: *"A software component is a unit of composition with contractually specified interfaces and explicit context dependencies only. A software component can be deployed independently and is subject to composition by third parties."* The definition given here is one of the many ways of describing a software component.

3.2 An Example of Software Architecture

Let us study the architecture of a hypothetical system presented in Figure 1. The system consists of three subsystems: *A*, *B* and *C* (e.g. a server and two clients). The high-level architecture of each subsystem, as well as the architecture of the whole system, is presented.

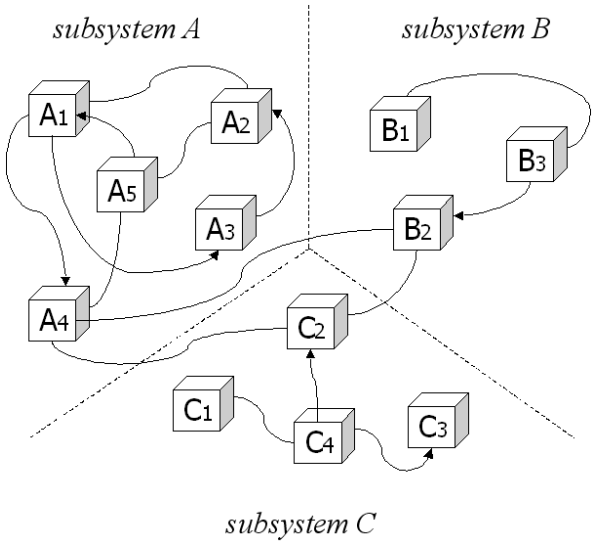

Figure 1: One way of illustrating the architecture of a system composed of three subsystems.

Subsystem *A* consists of five components (or *architectural elements*) shown in the form of cubes, $A_1,\ldots, A_5$; 7 internal links, 4 of which are unidirectional and 3 bi-directional; subsystem *A* furthermore consists of two external links that connect it to the two other subsystems. Subsystem *B* has only three components, B_1, B_2 and B_3, and two internal links that interconnect components B_1 and B_3 and connect component B_3 to component B_2. Additionally, this subsystem is interconnected with the two other subsystems. The last subsystem, *C*, consists of four components, $C_1,\ldots, C_4$, that have two unidirectional and one bi-directional internal links. The system is also interconnected with the two other subsystems. Obviously, from the architectural point of view, the most difficult subsystem to implement would be subsystem *A* due to the fact that it has the most complex structure; subsystem *B* would be the easiest subsystem to implement.

Let us go bit further, according to the definition given by Shaw and Garlan, and study an architectural element i.e. a component. One way to illustrate a component is given in Figure 2.

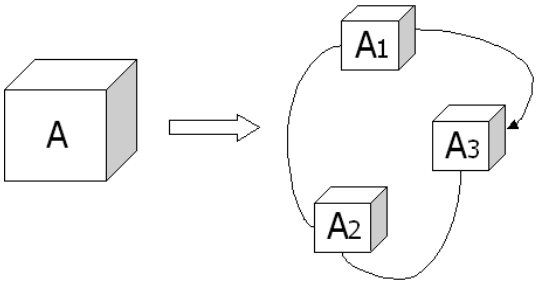

Figure 2: An architectural element i.e. component.

Thereby, a component, also known as a module, can be a composition of subcomponents interconnected by one- or bi-directional connectors (links). Each subcomponent can be independent or can depend on some other subcomponent. Having many interconnections between subcomponents makes it difficult to implement and maintain a component.

In general, a software system should be designed in such a way that its components are as independent of each other as possible and that their interconnections (interfaces) are easy to maintain.

3.3 Architectural Styles

When discussing architectural design, it makes sense to also touch on architectural styles. An architectural style refers to a pattern that is followed in architectural design. [17], pp. 25 gives the following definition for architectural style: *"An architectural style is a description of component types and a pattern of their runtime control and/or data manipulation."*

So far, many architectural styles have been created for different needs. [16], pp. 20, presents a list of common architectural styles. The main style groups are (1)

Dataflow systems, (2) Call-and-return systems, (3) Independent components, (4) Virtual machines, and (5) Data-centred systems (also called repositories).

Each of the architectural styles presented above has its own advantages and disadvantages, and all the styles differ from each other; no general architectural style exists for all software. The current trend of software houses is to create their own architectural styles or even a common architecture, although it should be remembered that the development units of these companies are restricted to some specific region.

3.4 Software Metrics

3.4.1 Software Metrics in General

Metrics are crucial in the evaluation of software; without a proper metrics suite, it is not possible to evaluate software to an acceptable degree of accuracy. This is why it is especially important to choose metrics that describe the system to be evaluated in the best possible manner.

Many books have been published and a lot of research carried out on software metrics. N. E. Fenton and S. L. Pfleeger have published a comprehensive book on the literature on software metrics [18].

Software metrics are closely related to the software measurement needed to evaluate the status of projects, products and resources ([18], pp. 11). They help in controlling the drift of a project and can indicate what is going wrong and when.

Software metrics may involve different attributes such as usability, integrity, efficiency, testability, reusability, portability and interoperability (external attributes) as well as size, effort and cost (internal attributes) ([18], pp. 78).

Software design metrics can be divided into two main groups: *high-level* and *low-level design metrics*. High-level design metrics comprise the *architectural design metrics* described in chapter 3.4.1. Low-level design metrics are, for example, *object-oriented metrics* and *function-based metrics*.

3.4.1 High-Level Design Metrics

Architectural design metrics are the software metrics used to evaluate software as early as possible, which is during the architectural design phase. Architectural design metrics are high-level software metrics. Architectural design metrics may be used, for instance, for estimating the cost (development effort), maintainability, fault tolerance or risk prediction of software. They are applied when the architecture of the software is being created, i.e. before coding has begun.

In [3], A. Avritzer and E. J. Weyuker present a *risk prediction metric*, a metric for architectural assessment, and give detailed information on its use. [4] presents the construction of information coupling and cohesion metrics at a sufficiently high level of abstraction. In [19], F. Xia discusses module coupling and suggests a complicated formula for computing the coupling complexity of modules.

When creating architectural design metrics, different aspects should be taken into account. First at all, there must be knowledge of what is to be measured. For example, a cost estimating metrics suite may be of no use in evaluating software for its maintainability. Another important factor is the adequacy of a metric. For instance, in what way could a defined size be taken as an architectural design metric? It is probably not possible to tell the size of a component by the number of classes (for the object-oriented paradigm) in its high-level design stage. This information is given, because the creation of an architectural design metrics suite was part of this research.

3.4.2 Low-Level Design Metrics

Object-oriented metrics and function-based metrics belong to the group of low-level design metrics. Unlike high-level design metrics, low-level design metrics are applied once the code has been created.

Object-oriented metrics are suitable for studying the interactions between and within classes. Based on the information extracted from the code of the system under study, they can tell in numbers, the strength of the coupling that exists between the classes or the tightness (reflecting good cohesion) of the classes, but have no direct use in the case of entire architectural entities (i.e. components). Examples of object-oriented metrics are Chidamber and Kemerer metrics, Lorenz and Kidd metrics and Abreu metrics [1, 2].

Function-based metrics are used for studying the interactions between methods and their internal behaviours. For example, in [9], J. E. Matson, B. E. Barrett and J. M. Mellichamp use a function-based metrics suite for software development cost estimation.

4 ARCHITECTURAL DESIGN METRICS SUITE

4.1 Architecture of the System

The system being studied here was implemented by a Finnish telecommunications company and consists of two subsystems, a CORBA–based (Common Object Request Broker Architecture), highly distributed server (let us call it subsystem *A*) and a centralised client (let us call it subsystem *B*). That is, the system has a client-server architecture. The server is responsible for mining data and transmitting it world-wide. The client is responsible for offering user interfaces and establishing Internet connections. Both subsystems consist of six components that are responsible for different tasks. The architecture of the system is shown in Figure 3.

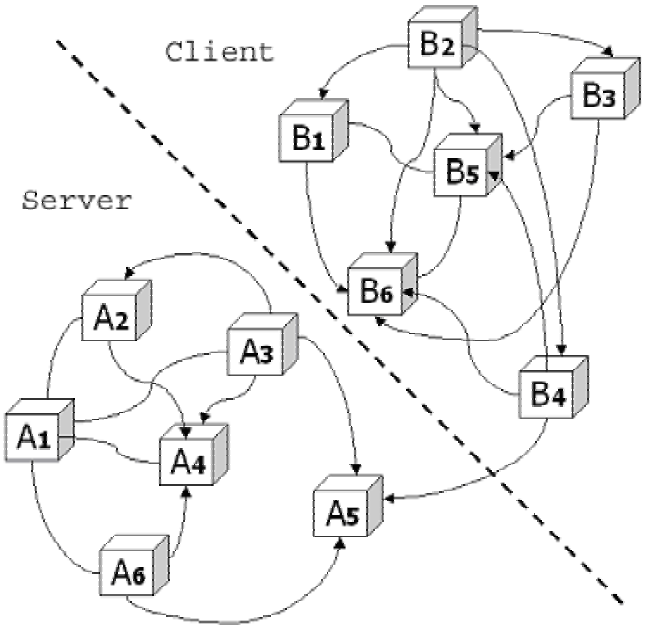

Figure 3: The architecture of the system being studied.

As was mentioned, the server is based on CORBA; however, the links between the components of the server are only those that are of the physical significance. That is, the logical interconnections provided by CORBA are not taken into

account. Only the component references explicitly implemented in the code are regarded. The same holds for the components; only implemented components are regarded. The components provided by CORBA are not taken into account in the architecture of the server. The interconnections are summarised in Table 1.

Table 1: A summary of the interconnections between the modules (components) of the subsystems.

Subsystem A				***Subsystem B***		
Module	*Refers to*	*Referred by*		*Module*	*Refers to*	*Referred by*
A_1	A_2, A_3, A_4, A_6	A_2, A_3, A_4, A_6		B_1	B_5, B_6	B_2, B_5
A_2	A_1, A_4	A_1, A_3		B_2	B_1, B_3, B_4, B_5, B_6	-
A_3	A_1, A_2, A_4, A_5	A_1		B_3	B_5, B_6	B_2
A_4	A_1	A_1, A_2, A_6		B_4	B_5, B_6, A_5	B_2
A_5	-	A_3, A_6, B_4		B_5	B_1, B_6	B_1, B_2, B_3, B_4, B_6
A_6	A_1, A_4, A_5	A_1		B_6	B_5	B_1, B_2, B_4, B_5

The components of both subsystems are coupled to each other relatively tightly, which probably made their development difficult. Once again, when a component is coupled with many other components, even a little change made to it may have a dramatic influence on the functionality of the other components involved. This is especially dangerous when the rate of messaging between interconnected components is high. In a properly design software system, the rate of messaging

between different components should be kept as low as possible. In this way, the components can be thought of being independent of each other. [41]

The subsystems are physically connected to each other only through one unidirectional link from component B_4 to component A_5, which means that the coupling between the subsystems is low. This makes the server almost independent of the client.

The system was implemented in a purely object-oriented way. Each component consists of a set of classes implemented in Java. That is, object-orientation is one of the system's properties. Java, which has ready packages of different communication protocols, is of enormous value when programming client-server applications.

The main difference between the server and the client is the high distribution of the former provided by CORBA. For example, CORBA makes it possible for intelligent components to discover each other and interoperate on an object bus. CORBA has many other properties that are highly valuable in server-client applications (for more information, please refer to [40]). These facts give reason to assume that the subsystems may probably have different architectural properties. Since CORBA puts immense pressure on distributiveness and interoperability, it may be assumed that coupling is extremely important in the architecture of the server.

4.2 The Metrics Suite

After careful discussion, the members of the software research group (Mrs. Päivi Ovaska, Mr. Kari Smolander and Mr. Alexandre Bern) agreed upon an architectural design metrics suite that includes size, coupling, cohesion and complexity. The main emphasis in developing the metrics suite was on the creation of a set of metrics that satisfy the needs of the project (in which the main

objective was to study the influence of coupling). The other goal of the development of the metrics suite was to extract the architectural attributes that would possibly be independent of each other.

Since many other properties have an impact on the architectural design, other attributes, which are summarised in Table 2 and described later on, have been suggested and accepted.

Table 2: A summary of the metrics suite.

Attribute (architectural property)	***Metric name***	***Description***
a_1	Size	Size of a component in KLOC (Kilo Lines Of Code)
a_2	Size	Size of a component in number of classes
a_3	Coupling	Number of components referring to this component
a_4	Coupling	Number of components this component refers to
a_5	Cohesion	Number of aggregations, compositions and relations among the classes of a component
a_6	Complexity	Number of use cases of a component
a_7	Complexity	Number of subcomponents that form a component
a_8	Complexity	Number of databases connected to a component

All these attributes are directed to single components and not to entire subsystems. The subsystems are evaluated based on the values of the attributes and the corresponding free parameters.

4.2.1 Size

It is reasonable to assume that the size of a system influences its development effort. The larger a system is, the more effort is required to implement it. In this work, two different size attributes are used: a_1 referring to the component size in KLOC and a_2 referring to the number of classes composing the component.

4.2.2 Coupling

According to [12], pp. 375, coupling can be defined as follows: *"Coupling is a measure of interconnection among modules in a program structure… Coupling depends on the interface complexity between modules, the point at which entry or reference is made to a module, and what data pass across the interface."* In this work, coupling simply measures the amount of interconnections (references) between components.

Here, two different attributes are used for coupling. Attribute a_3 refers to the number of components to which the component being studied refers, whereas attribute a_4 defines the number of components that refer to the component being studied. An example is shown in Figure 4. Table 3 shows the values of the corresponding attributes.

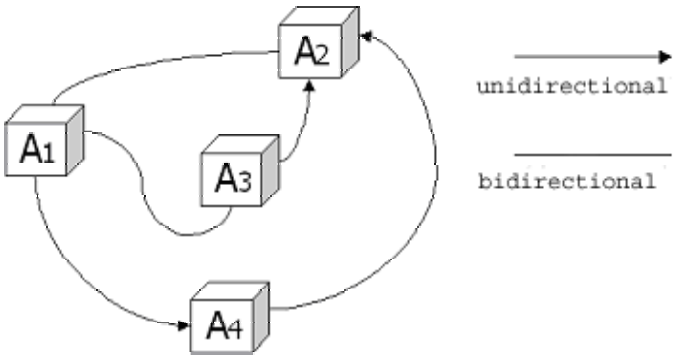

Figure 4: An example of coupling between four components.

Two different definitions of coupling have been used for the reason that the interconnections between modules can be both unidirectional and bi-directional as shown in the above diagram. Some information might be lost if coupling were to be defined as simply the number of relations between the component being studied and the other components.

Table 3: The values of the corresponding attributes according to Figure 4.

Component	**Value of a_3**	**Value of a_4**
A_1	*3* (refers to A_2, A_3 and A_4)	*2* (referred by A_2 and A_3)
A_2	*1* (refers only to A_1)	*2* (referred by A_1 and A_4)
A_3	*2* (refers to A_1 and A_2)	*1* (referred only by A_1)
A_4	*1* (refers only to A_2)	*1* (referred only by A_1)

4.2.3 Cohesion

According to [12], pp. 374, cohesion is *"a measure of the relative functional strength of a module."* Within the limits of this project, cohesion (attribute a_5) is defined as a number of aggregations, compositions and relations in the class diagram of a component. The higher the number is, the more difficult it is to implement the component since the number of connections between classes increases. On the other hand, stronger cohesion should be achieved in order to implement an internally strong module.

Let us consider Figure 5 as an example of cohesion. Class *C* is composed of three other classes, *D*, *E* and *F*. Class C also has a unidirectional association to class *H* and a bi-directional association to class *G*. That is, the value of attribute a_5 is 5 (inheritance is not considered).

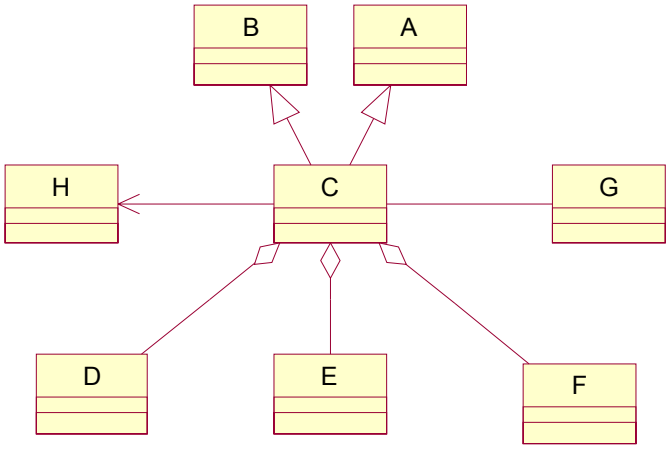

Figure 5: An example of cohesion.

4.2.4 Complexity

In this project, three attributes are related to complexity. The first complexity attribute, a_6, refers to the number of use-cases of a component being studied. The second attribute, a_7, tells the amount of subcomponents that make up the actual component. The last one, a_8, refers to the number of databases related to the component.

4.2.5 Comments on a_2 and a_5

It is reasonable to assume that neither the size of a system expressed as a number of classes nor its cohesion expressed in the form of classes belong directly to architectural design metrics, which is true. These attributes have been adopted for the purpose of studying their precise influence on the development effort. In other situations, they would be unnecessary.

5 DATA AND METHODS

The first section of this chapter discusses the data used in this project to estimate the development effort. As well as presenting the data itself, the fist section describes how it was obtained and edited. The second section presents the selected models and the objective functions that were created. The final part of this chapter (section 5.3) describes the method (DE algorithm) used in this project to define the parameters of the models by minimising the objective function given by equations (3) and (4). It also discusses other potentially competitive methods that were under consideration but not adopted; the reasons for this decision will also be explained in this chapter.

5.1 Data Acquisition

The specification documents of the components and their implementation code were used as the raw data. The specification documents were reviewed and all the useful information was extracted. Based on these documents, complexity numbers (the numbers of subcomponents and data bases), as well as the coupling and cohesion information for some of the components of both subsystems, were successfully extracted. The rest of the information was extracted from the implementation code.

As usual, some problems were encountered during data acquisition. Some of the specification documents were not up-to-date, which made it necessary to study the implementation code more carefully. For example, the information on the subsystem architecture (by this information, we refer to the diagrams) varied according to the specification documents, which thus rendered it unreliable. As a result, the architectures of the subsystems (shown in Figure 3, chapter 4.1) were reconstructed using the implementation code.

Since the UML (Unified Modelling Language) diagrams also turned out to be unreliable for some components, they were reconstructed (through re-engineering) using the Together 5.5 development tool for application modelling and round-trip engineering for Java and C++ [20].

The numbers of lines of code were obtained using an application for counting lines of code which had been downloaded from the Web [21]. When counting the numbers of lines, comments were left out.

The extracted values of the attributes are shown in Table 4 for the server and in Table 5 for the client, respectively. The values that describe the development effort were taken from the project management software (Niku Workpage).

Table 4: The values of the attributes of the server.

Attribute	A_1	A_2	A_3	A_4	A_5	A_6
a_1	1	7	3	4	1	1
a_2	9	53	43	47	23	10
a_3	4	2	1	4	3	1
a_4	4	2	4	1	0	3
a_5	5	65	30	21	9	10
a_6	10	7	12	3	13	7
a_7	1	2	1	1	1	2
a_8	0	4	1	1	0	0
Uncorrected development effort (h)	540.5	634.5	889.5	712	417	579
Correction coefficient	1.0	0.76	1.0	1.0	1.0	1.0
Corrected development effort (h)	540.5	835	889.5	712	417	579

Table 5: The values of the attributes of the client.

Attribute	B_1	B_2	B_3	B_4	B_5	B_6
a_1	1	6	1	2	10	3
a_2	20	13	3	8	118	14
a_3	2	0	1	1	5	5
a_4	2	5	2	3	2	1
a_5	6	9	0	0	10	9
a_6	19	6	8	7	17	3
a_7	1	3	1	1	4	1
a_8	0	0	0	0	1	1
Uncorrected development effort (h)	1220.5	1488	934	950	966	1141.5
Correction coefficient	1.15	1.15	1.15	1.15	0.76	1.15
Corrected development effort (h)	1061	1294	812	826	1271	993

The above tables contain rows with the correction coefficients and the corresponding development efforts. In the case of the server (Table 4), only the development effort for module A_2 is corrected (by dividing the effort by 0.76), whereas for the client (Table 5), the development efforts are corrected for all six components. The corrections (that reflect the programmer's capability) were made based on the experience of component developers. Coefficients below 1.0 and above 1.0 reflect above-ordinary and below-ordinary capability, respectively. The coefficients were obtained from the PCAP Cost Driver table (PCAP, Programmer Capability) in [37], pp.48.

The idea to make these corrections was proposed by the departmental manager of the company that had implemented the system. Since the manager knew the developers of the system well and was capable of evaluating their professional skills, she estimated the proper correction coefficients. The idea came up for the

reason that the results (see chapter 6) obtained using the initial (uncorrected) development effort were poor and the professional skills of the developers non-homogenous.

The values presented in Table 4 and Table 5 represent the only information used in the models presented below.

5.2 Models for Estimating the Development Effort

The models presented here were used to evaluate the development effort of the subsystems through the mapping of the models onto the objective functions and the minimisation of these functions using the method presented in section 5.3. The mapping procedure is described in section 5.2.1, while section 5.2.2 offers hints as to how to interpret the obtained models.

5.2.1 From Models to Objective Functions

In order to model the development effort using architectural properties, two different functions have been tried: (1) a linear function, and (2) a non-linear function.

$$F(x_1, x_2, ..., x_8) = b_1 x_1 + b_2 x_2 + ... + b_8 x_8 \qquad (1)$$

The linear function was selected because it is one of the simplest functions yet that is able to easily demonstrate the significance of its variables (through the values of their parameters $b_1, ..., b_n$): the higher the value of a parameter (or coefficient), the greater the influence of the corresponding variable on the result

will be when it is increased. That is, having obtained the coefficients of a linear model, it is trivial to extract the most significant variables.

The non-linear function given in equation (2) was selected to compare the results (significance and importance of attributes, see chapter 5.2.2) with those produced by the linear function given by equation (1). Since it has the same basic properties (the higher a power coefficient is, the greater its influence on the result will be), it is also easily interpretable. Another reason for the selection of this function to be one of the models was its suitability for the construction of the estimating function for the development effort when combined with the linear function, which thus produced a function that was better able to fit different data. The function is given by equation (3).

$$G(x_1, x_2, ..., x_8) = x_1^{b_1} + x_2^{b_2} + ... + x_8^{b_8} \quad (2)$$

In both functions ((1) and (2)), x_n refers to the value of attribute a_n and b_n is a (linear or power) coefficient or parameter (n = 1…8). Since there are, at most, eight attributes that describe a component, the equations have the same number of variables. With respect to these functions, the following assumption is made: there are no mutual dependencies between the attributes.

$$H(x_1, x_2, ..., x_n) = a_1 x_1^{b_1} + a_2 x_2^{b_2} + ... + a_n x_n^{b_n} \quad (3)$$

Thus, equations (1) and (2) are used to estimate development effort of the mobile service application. In order to define b_n, the following functions are minimised:

$$W_1(b_1,b_2,...,b_n)=\frac{1}{m}\sum_{m=1}^{6}\left(100\frac{|H_m-h_m|}{h_m}\right)=\frac{10^2}{m}\sum_{m=1}^{6}\frac{|H_m-h_m|}{h_m} \quad (4)$$

$$W_2(b_1,b_2,...,b_n)=\frac{1}{m}\sum_{m=1}^{6}\left(100\frac{H_m-h_m}{h_m}\right)^2=\frac{10^4}{m}\sum_{m=1}^{6}\left(\frac{H_m-h_m}{h_m}\right)^2 \quad (5)$$

In the above equations, number 100 is taken to indicate the percentile error and h_m (m = 1…6) the measured development effort value of component m. H_m is defined as follows:

$$H_m=H_m(b_1,b_2,...,b_n)=\begin{bmatrix}F_m(b_1,b_2,...,b_n)=b_1x_1+b_2x_2+...+b_nx_n\\ G_m(x_1,x_2,...,x_n)=x_1^{b_1}+x_2^{b_2}+...+x_n^{b_n}\end{bmatrix} \quad (6)$$

This means that when defining the coefficients, the values of $\vec{X}$ $(x_1,x_2,...,x_8)$ remain fixed throughout the whole optimisation process. The optimisation process is subject to the following constrains: when minimising W_k ($k=\{1,\ 2\}$), which is formed by F_m, b_n is forced to take only non-negative real values; and when minimising W_k, which is formed by G_m, b_n is forced to belong to the interval $[-1,\ \infty[$, allowing an attribute to be made insignificant by negative values.

Equation (4) represents the mean error of all the six components. Equation (5), for its part, represents the mean quadratic error, thus, equalising the mean errors of the development efforts of the components. These functions are called objective functions or functions to be minimised.

Two different models ((1) and (2)) and two different objective functions ((4) and (5)) were taken into use in order to compare the results and examine the possible differences between them.

5.2.2 Interpreting the Models

The main goal of this project was not to create an exact model that depicts the development effort based on architectural attributes but rather to extract the attributes that have the greatest and most significant influence on the development effort. In the following two sections, rules of significance and importance are introduced to study the attributes.

Rule of Significance

Firstly, all the coefficients (b_n, $n = 1...8$) of the models are defined. In the case of the linear model, an attribute is significant only if the value of the corresponding coefficient (or parameter) is non-negative. Otherwise, the attribute would then have a negative influence (or no influence whatsoever) on the development effort.

In the case of the non-linear model, an attribute is significant only if the value of the corresponding parameter is at least one. This restriction is applied for the reason that when a number is raised to a power of less than one, its value decreases; thus, this kind of an attribute would have no increasing influence on the development effort.

Rule of Importance

When all the coefficients (b_n, $n = 1...8$) are defined for both models, the development efforts are once again estimated by leaving a significant attribute out of the model, thus causing an increase in the error (meaning the value of the objective function). The greater the error is, the greater the influence of the attribute excluded is on the development effort and, thus, the more important the attribute in question is. This conclusion is drawn on the basis of the fact that without the excluded attribute, the error in the model increases, which means that the model does not fit the data well.

5.3 The Method

Traditional optimisation methods, such as linear and quadratic optimisation, non-linear and discrete optimisation, which still a few years ago held a strong position in optimisation, are slowly loosing ground to soft computing methods. Instead of traditional optimisation methods, new methods such as evolutionary algorithms (non-linear global optimisation methods) and simulated annealing (SA) are strongly taking their place in optimisation problems. More information on evolutionary algorithms (e.g. their application domain) is presented in chapters 5.3.1 – 5.3.3.

In addition to the methods described above, artificial neural networks (ANN) and fuzzy logic (FL) have been tried for the creation of development effort and cost estimation models. In [22], A. Adri, T. M. Khoshgoftaar and A. Abran study how easily artificial neural networks can be interpreted in software cost estimation by mapping the neural network to a system based on a fuzzy rule. A few years earlier, in 1996, G. R. Finnie and G. E. Wittig proposed AI (Artificial Intelligence) tools for software development effort estimation [23]. In their work, they examined the potential of two intelligence approaches: ANNs, and case-based reasoning for creating development effort estimation models. Even earlier,

in 1993, A.R. Venkatachalam used an ANN to model software cost estimation expertise [24]. Another interesting study was performed in 1999 by W. Pedrycz, J.F. Peters and S. Ramanna [25] who used a fuzzy set approach to estimate the cost of software projects.

For this project, two potential methods were considered: a non-linear global optimisation method i.e. a differential evolution algorithm, and a modelling method based on ANN. Due to the insufficient amount of data (only six modules per subsystem), the latter proposal was left out.

Traditional optimisation methods were not considered because of the trickiness of the objective functions, which is based on their difficult structure caused by the combination of several equations (each module has its own equation depicting the model as presented in equation (6)) to form the objective functions, as given in equations (4) and (5). Another reason for not considering traditional optimisation methods is that the optimisation involved restrictions (in intervals). Instead, it was assumed that DE would perform well in this case, because of the possibilities it offers and the experience from its use so far (the arguments are presented in chapters 5.3.1 and 5.3.2).

5.3.1 Possibilities of Evolutionary Algorithms

Being non-linear global optimisation methods, evolutionary algorithms (such as genetic algorithms (GAs) and differential evolution algorithms) can be used to optimise functions of different types (e.g. linear, non-linear, discrete and integer-value functions). In any case, these algorithms are especially valuable in problems described by non-continuous functions that have difficult reliefs (noise, flatness, multiple local minimums and maximums), a high level of dimensionality and that allow for parameter interaction, non-differentiability and possibly multiple, non-trivial and non-linear constrains limiting the feasible solutions to a small subset of

the whole search space, as well as for penalty functions. As a result, EAs produce a satisfactorily precise result that may or may not be the global optimum. [26, 27]

In engineering, many tasks fall into the category of mixed integer-discrete-continuous problems. For example, the size of some details (nails, screws, etc.) is defined according to some commercially available standard and is, thus, a discrete value. The number of teeth on a gear may be given only as an integer value and the amount of raw material (for example, in kilograms) needed to produce these details as a continuous value. It is obvious that traditional optimisation methods are not capable of solving this kind of a problem. [27]

When discussing traditional approaches we refer to methods, such as exhaustive search, analytical optimisation, the Simplex method (and variations of it) and optimisation based on line minimisation. Instead, evolutionary algorithms, as well as simulated annealing, belong to a group of optimisation methods inspired by natural approaches that imitate real-life processes.

The drawback of the exhaustive search is its slowness, because it attempts all the possible solutions. In any case, this method returns, as its result, the optimal solution. This method is also known as the brute-force approach. The principle of the work of analytical optimisation lies in finding the extreme value of a function (of two or more parameters) by taking the gradient of the function and setting it at equal to zero. The next step is to solve the obtained equations and obtain a family of lines, the intersection of which is the extreme value. The drawback of this method is that it does not provide any information as to the optimality of the solution. In optimisation based on the Simplex method, the most elementary geometric figure, which has $n + 1$ sides in an n–dimensional space, is used to reach the minimum by generating a new vertex for the simplex at each iteration step. The disadvantage of this method lies in its slowness and need for the function to be assumed to be continuous. If the assumption does not hold, the method becomes ineffective. It may also stick to a local minimum. Methods based on line minimisation choose a direction in which to move after selecting a random

point and move in that direction until the function being processed begins to increase. These methods are also known for their slowness and can stick to a local minimum. [28]

By modelling a biological process to optimise highly complex cost functions, EAs are able to overcome problems that are fatal for traditional methods as well as to outperform traditional methods in speed and robustness. That is, evolutionary algorithms should be attempted whenever a problem is known to be difficult to solve (for instance, slow) using a traditional method.

5.3.2 Evolutionary Algorithms Application Domains

So far, evolutionary algorithms have been tried in many different areas that vary from scientifically intriguing problems such as the travelling salesman (TSP) and knapsack problem (both are combinatorial tasks) to tasks in the field of mechanical engineering.

On the basis of current trends, it seems that EAs are being used to an ever increasing extent for different optimisation tasks. If still a few years ago, evolutionary algorithms were used to solve such scientifically interesting problems as the zero/one multiple knapsack problem [29], they are now used to optimise the weights of neurons of ANNs [30], and to solve industrial and biological problems. In [31], Fayech, Hammadi, Maouche and Borne propose methods for regulating urban bus traffic using evolutionary algorithms. In [32], Seong-Joo Han and Se-Young Oh combine an evolutionary algorithm with an ANN to optimise autonomous mobile robot navigation using ultrasonic sensors. In [33], Watts, Major and Tate describe how they optimised an MLP neural network using an evolutionary algorithm to experimentally model a determined protein synthesis termination signal strength. D. H. Milone, J. J. Merelo and H. L. Rufiner propose, in [34], a new technique based on an evolutionary algorithm. Using this method, they permit the segmentation of speech without the need for a previous

training process by classical methods. In [27], J. Lampinen and I. Zelinka present numerical examples of the use of differential evolution algorithms in the design of a gear train, a pressure vessel and a coil spring, which are examples of optimisation in mechanical engineering.

5.3.3 Differential Evolution Algorithm

A differential evolution algorithm is a very simple but nevertheless powerful stochastic function minimiser based on the generation of a new population from an existing one. It was created as a result of Ken Price's attempts to solve the Chebychev polynomial fitting problem proposed to him by Rainer Storn. Mr. Price solved the problem by coming up with the idea of using vector differences to perturb a vector population, which happened in 1994. Since those days, many substantial improvements have been made to the algorithm, enhancing its robustness and performance. [35]

DEA in a Nutshell

Basically, a differential evolution algorithm generates a trial vector (a vector to be compared with the target vector) by adding the weighted difference between two vectors from the current population to a third vector. If the trial vector has a lower cost that the target vector, the newly generated vector replaces it. [15]

Usually, the *objective function* (the function to be optimised) can be given as follows:

$$f\left(\vec{X}\right): R^D \rightarrow R \tag{7}$$

In the above equation, a mapping from a *D*-dimensional space of real values (R^D) is made onto a 1-dimensional space of real values (*R*). $\vec{X}$ is a vector consisting of *D* elements and is defined as follows:

$$\vec{X} = \begin{bmatrix} x_1 \\ x_2 \\ \dots \\ x_D \end{bmatrix}, \quad \vec{X} \in R^D \tag{8}$$

The objective of optimisation is to minimise $f\left(\vec{X}\right)$ by applying an optimisation method to vector $\vec{X}$.

Very often, the parameters of the object function may be forced to belong to a specific interval, having thus upper and lower boundary constrains $\vec{X}^{(L)}$ and $\vec{X}^{(U)}$, as given in the equation below:

$$x_j^{(L)} \le x_j \le x_j^{(U)}, \quad j = 1,\dots, D \tag{9}$$

As in the case of all other evolutionary algorithms, DEA operates on a *population*, P_G, of candidate solutions also known as the *individuals* of the population. DEA maintains a population of a constant size, consisting of *NP* real-valued vectors, $\vec{X}_{i,G}$, where i refers to the population member and *G* to the *generation* to which the population belongs. Thus, the population can be given as follows:

$$P_G = \left[\vec{X}_{1,G}, \vec{X}_{2,G}, ..., \vec{X}_{NP,G} \right], \quad G = 0, ..., G_{\max} \tag{10}$$

P_G can be thought of as being a matrix consisting of a set of vectors or individuals. Thereby, the individuals belonging to the population of generation *G* can be given in the following form:

$$\vec{X}_{i,G} = \begin{bmatrix} x_{1,i,G} \\ x_{2,i,G} \\ \cdots \\ x_{D,i,G} \end{bmatrix}, \quad i = 1, ..., NP, \quad G = 0, ..., G_{\max} \tag{11}$$

That is, index i indicates i[th] individual in the population of generation *G*.

The first step of the algorithm is to randomly create the initial population. Taking the boundary constrains into account, the individuals are created according to the scheme shown below:

$$x_{j,i,0} = rand_j[0,1] \cdot \left(x_j^{(U)} - x_j^{(L)} \right) + x_j^{(L)}, \quad i = 1, ..., NP, \quad j = 1, ..., D \tag{12}$$

In the equation, $rand_j[0,1]$ denotes a uniformly distributed random value from the interval [0, 1] (scheme *DE/rand/1/bin*, see chapter 5.3.4). A new value is generated for each individual.

Starting from the first generation, vectors in the current population (P_G) are randomly selected and combined to create candidate vectors for the next

generation (P_{G+1}). The population of the candidate vectors, also known as *trial vectors* and denoted by $\vec{U}_{i,G+1} = u_{j,i,G+1}$ (i = 1,…,*NP* and j = 1,…, *D*), is created according to the following scheme:

$$u_{j,i,G+1} = \begin{cases} v_{j,i,G+1}, & \text{if } rand_j[0,1) \le CR \vee j = k \\ x_{j,i,G}, & \text{otherwise} \end{cases} \qquad (13)$$

In the equation,

$v_{j,i,G+1} = x_{j,r3,G} + F \cdot (x_{j,r1,G} - x_{j,r2,G})$ forms a *noisy* or *mutated vector*

$i = 1,\ldots,NP,\ j = 1,\ldots,D$

$k_i \in \{1,...,D\}$ is a random parameter index chosen once for each i

$r_1, r_2, r_3 \in \{1,...,NP\},\ r_1 \neq r_2 \neq r_3 \neq i$ are randomly selected indexes

$CR \in [0,1];\quad F \in (0,1+]$

As is shown above, the trial vector is created from three different individuals of the current population (with requirement $r_1 \neq r_2 \neq r_3 \neq i$). Index k refers to a randomly selected chromosome used to ensure that each trial vector differs from its counterpart in the previous generation and is updated for each value of index i.

Parameters F and CR (including NP and G) are the *control parameters* of DE. Both F and CR remain unchanged during the whole optimisation process. F, also known as the *mutation constant*, is a real-valued factor from the interval $(0,1+]$. CR is called the *crossover factor* and belongs to the interval $[0,1]$ that defines the probability with which a trial vector's chromosome will be selected from the

mutated vector (formed by $v_{j,i,G+1}$) instead of from the target vector (formed by $x_{j,i,G}$). Both *F* and *CR* influence the robustness and convergence velocity of the optimisation process. Their optimal values depend on the characteristics of the object function and the population size.

If there are boundary constrains, it is important that the values of the newly created genes (vector parameters) lie inside them. One simple way to ensure this is to substitute any gene that violates the boundary constrain rule with a randomly generated value from the feasible interval as follows:

$$u_{j,i,G+1} = \begin{cases} rand_j[0,1] \cdot \left(x_j^{(U)} - x_j^{(L)}\right) + x_j^{(L)}, & \text{if } u_{j,i,G+1} < x_j^{(L)} \vee u_{j,i,G+1} > x_j^{(U)} \\ u_{j,i,G+1}, & \text{otherwise} \end{cases} \quad (13)$$

Finally, the population of the next generation, P_{G+1}, is selected from the current population, P_G, and the child population according to the following scheme:

$$\vec{X}_{i,G+1} = \begin{cases} \vec{U}_{i,G+1}, & \text{if } f\left(\vec{U}_{i,G+1}\right) \le f\left(\vec{X}_{i,G}\right) \\ \vec{X}_{i,G}, & \text{otherwise} \end{cases} \quad (14)$$

Figure 6 graphically illustrates differential evolution and shows how a noisy vector is created. Generating the individual in the manner presented refers to the *DE/rand/1/bin* scheme. More differential evolution schemes are discussed in chapter 5.3.4. [26]

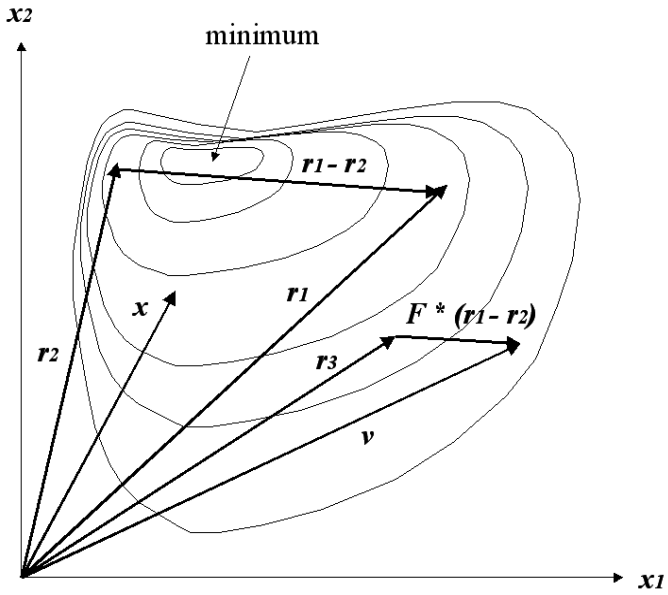

Figure 6: An example of a two-dimensional objective function with its contour lines an the process of generating a noisy vector *v*.

An Example of an Iteration of DEA

Let us consider applying DE to the minimisation of a simple objective function given by equation (15). A DE iteration is shown in Figure 7. The explanations are presented immediately after the figure.

$$f(x_1, x_2, ..., x_5) = x_1 + x_2 + ... + x_5 \qquad (15)$$

First, a target vector is chosen. Then two randomly selected different vectors (other than the target vector) are used to form a difference vector that is later multiplied by the mutation constant *F*. The result is then added to the a third randomly selected vector, forming thus a noisy vector. After applying the

crossover operation to the noisy and target vectors, the trial vector is created. If the cost of the trial vector is lower than one that of the target vector, the latter vector is substituted by the former one. In the case presented illustrated in Figure 7, the target vector has a lower cost, which means that it remains in the new population.

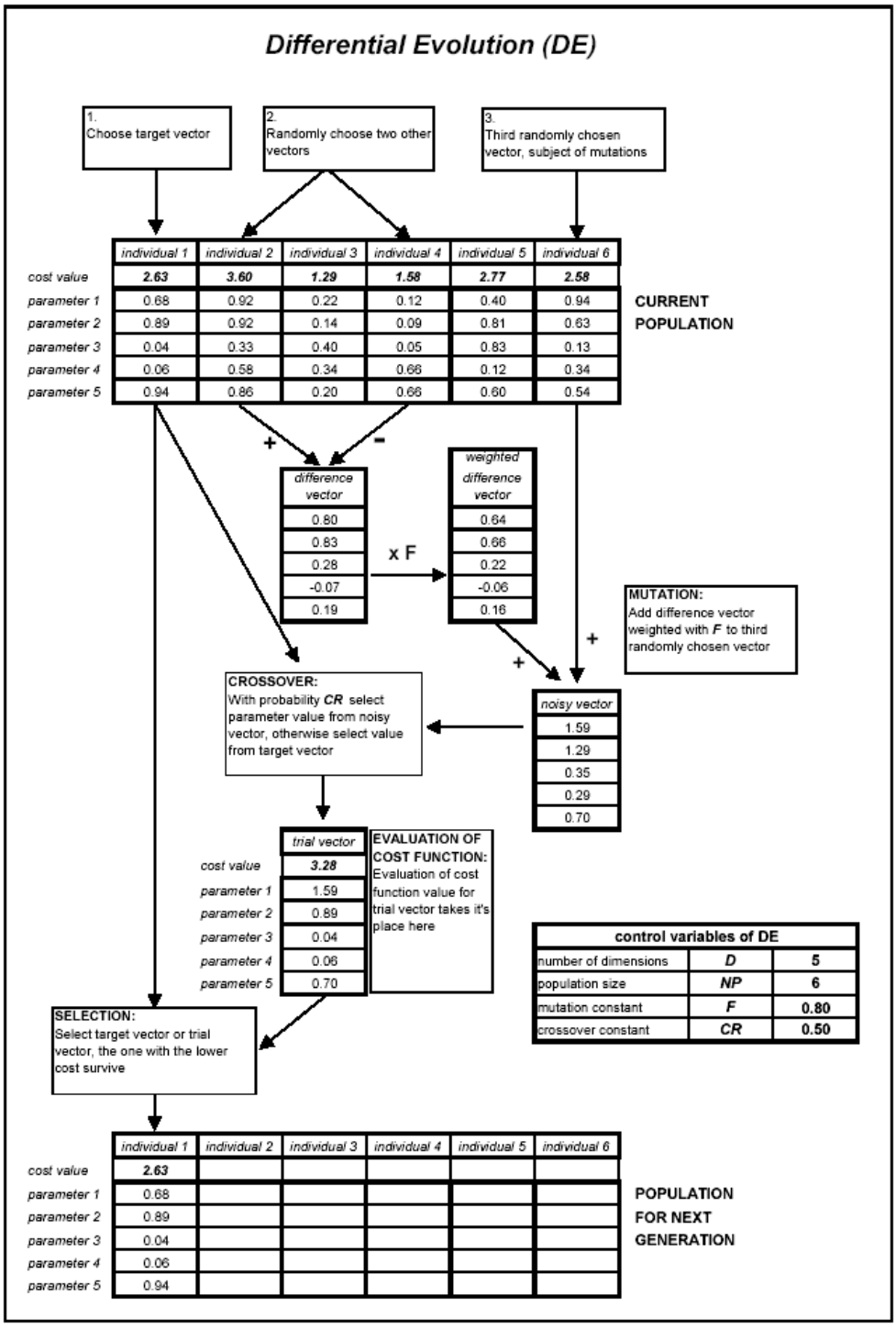

Figure 7: One step of a differential evolution. The diagram is shown here with the permission of Prof. Jouni Lampinen, and the original image can be found in [36].

5.3.4 Differential Evolution Schemes

In differential evolution, there are many different schemes for generating noisy (or mutated) vectors. This chapter discusses four widely used schemes that are summarised according to [15] in Table 6.

Table 6: Four different differential evolution perturbation schemes.

#	Scheme	Rule
1	*DE/rand/1*	$\bar{v}_{i,G+1} = \bar{x}_{r1,G} + F \cdot \left(\bar{x}_{r2,G} - \bar{x}_{r3,G}\right)$
2	*DE/best/1*	$\bar{v}_{i,G+1} = \bar{x}_{best,G} + F \cdot \left(\bar{x}_{r1,G} - \bar{x}_{r2,G}\right)$
3	*DE/best/2*	$\bar{v}_{i,G+1} = \bar{x}_{best,G} + F \cdot \left(\bar{x}_{r1,G} + \bar{x}_{r2,G} - \bar{x}_{r3,G} - \bar{x}_{r4,G}\right)$
4	*DE/rand-to-best/1*	$\bar{v}_{i,G+1} = \bar{x}_{r1,G} + \lambda \cdot \left(\bar{x}_{best,G} - \bar{x}_{r1,G}\right) + F \cdot \left(\bar{x}_{r2,G} - \bar{x}_{r3,G}\right)$

The randomly chosen indices in scheme *DE/rand/1* (r_1, r_2, r_3) are mutually different in addition to being different from the running index *i*. In the second scheme (*DE/best/1*), the vector to be perturbed is the best performing vector of the current generation. The same constrains also hold in this scheme. The third scheme uses two difference vectors as the perturbation. Here, all the randomly selected vectors are mutually different and also differ from the best vector. In the last scheme, the perturbation is placed in a location between a randomly selected population member and the best population member. In the scheme, λ controls the *greediness* of the evolution. Here, again, all the vectors are different.

6 RESULTS

The client and server were evaluated using both (linear and the non-linear) models. The models were created for both corrected and uncorrected development efforts. The essential results are presented in the following chapters, whereas all the other results can be found in the appendices. The results of essential importance are presented in bold type.

6.1 The Server

6.1.1 The Linear Model

Table 7 shows the values of the parameters (coefficients) of the linear model that estimates the development effort for the server. The first four rows of the table (from "*F*" to "*G*") refer to the control parameters of the differential evolution algorithm: *F* is the mutation factor, *CR* the crossover factor, *NP* for the size of the population, and *G* the maximum number of generations to be created. The values of *F*, *CR* and *NP* were selected in order to yield the best result after different combinations were tried.

Row five of the table shows the strategy (the DE scheme, see chapter 5.3.4) followed. Two strategies were tried here (in order to ensure the correctness of the results): 6 refers to *DE/best/1/bin* and 7 to *DE/rand/1/bin*. The next rows ($b_1 ... b_8$) represent the parameters (coefficients) of the model in such a way that, for instance, parameter b_1 is a coefficient of attribute a_1. The table is filled with the values of the coefficients and the results of the objective functions, W_1 (runs I and II) and W_2 (runs III and IV), given by equations (4) and (5).

Table 7: The values of the coefficients of the linear model that estimates the development effort for the server (the measured effort has been corrected).

Parameters	***Run I***	***Run II***		***Run III***	***Run IV***
F	0.8	0.8		0.8	0.8
CR	0.8	0.8		0.8	0.8
NP	30	30		30	30
G	5000	5000		5000	5000
Strategy	6	7		6	7
Range	[0, Inf]	[0, Inf]		[0, Inf]	[0, Inf]
b_1	0.0	0.0		0.0	0.0
b_2	10.5	10.5		9.5	9.5
b_3	4.8	4.8		9.8	9.8
b_4	68.8	68.8		69.9	69.9
b_5	0.0	0.0		0.0	0.0
b_6	3.1	3.1		4.9	4.9
b_7	120.6	120.6		103.2	103.2
b_8	0.0	0.0		0.0	0.0
	W_1			W_2	
Value	2.6	2.6		22.3	22.3

Table 8 shows the percentile error distribution between the measured and the estimated development effort among the components of the server that correspond to the above table.

Table 8: The percentile error distribution of the estimated development effort for the server (the linear model, the measured effort has been corrected).

F	A_1	A_2	A_3	A_4	A_5	A_6	***Mean error***
W_1	0	15.8144	0	0	0	0	2.6
W_2	3.0083	8.4470	3.1713	5.1501	0.4165	4.0532	4.0

Using the information given in Table 7, the following functions for the estimation of the development effort can be defined for the server:

$$F_1(x_1, x_2, ..., x_8) = 10.5x_2 + 4.8x_3 + 68.8x_4 + 3.1x_6 + 120.6x_7 \qquad (16)$$

$$F_2(x_1, x_2, ..., x_8) = 9.5x_2 + 9.8x_3 + 70.0x_4 + 4.9x_6 + 103.2x_7 \qquad (17)$$

The coefficient values of equation (16) are obtained by minimising W_1 and the coefficient values of equation (17) by minimising W_2. In these equations, variables $x_1, x_2, ..., x_8$ refer to the values of attributes $a_1, a_2, ..., a_8$, as presented in Table 4 and Table 5. For example, if we now want to define the estimated development effort for component A_1 (belonging to the server), we simply substitute the corresponding values from Table 4 into equation (16) as follows: $F_1(1, 9, \ldots, 0) = 10.5 \cdot 9 + 4.8 \cdot 4 + ... + 120.6 \cdot 1 = 540.5$ (the same as the measured value). The estimation can be performed in the same way using equation (17).

Using the same idea, Table 9 presents the coefficients of the linear model for estimating the development effort for the server defined by the uncorrected measured effort, and Table 10 contains the distribution of the corresponding errors

between the measured and the predicted development effort for the server's components.

Table 9: The values of the coefficients of the linear model that estimates the development effort for the server (the measured effort has not been corrected).

Parameters	***Run I***	***Run II***		***Run III***	***Run IV***
F	0.8	0.8		0.8	0.8
CR	0.8	0.8		0.8	0.8
NP	30	30		30	30
G	5000	5000		5000	5000
Strategy	6	7		6	7
Range	[0, Inf]	[0, Inf]		[0, Inf]	[0, Inf]
b_1	0.0	0.0		0.0	0.0
b_2	10.5	10.5		7.3	7.3
b_3	4.8	4.8		21.0	21.0
b_4	68.8	68.8		72.6	72.6
b_5	0.0	0.0		0.0	0.0
b_6	3.1	3.1		8.9	8.9
b_7	120.6	120.6		64.2	64.2
b_8	0.0	0.0		0.0	0.0
	W_1			W_2	
Value	8.7	8.7		182.4	182.4

Table 10: The percentile error distribution of the server's estimated development effort (the linear model, the measured effort has not been corrected).

F	A_1	A_2	A_3	A_4	A_5	A_6	***Mean error***
W_1	0	52.4114	0	0	0	0	8.7
W_2	9.7845	20.8768	10.3147	16.7507	1.3548	13.1830	12.0

Based on the information presented in Table 9, the following linear models for estimating the development effort for the server can be conducted:

$$F_1(x_1, x_2, ..., x_8) = 10.5x_2 + 4.8x_3 + 68.8x_4 + 3.1x_6 + 120.6x_7 \quad (18)$$

$$F_2(x_1, x_2, ..., x_8) = 7.3x_2 + 21.0x_3 + 72.6x_4 + 8.9x_6 + 64.2x_7 \quad (19)$$

Table 11 presents the summary of the errors for the linear model when significant attributes (see "Rule of Significance" in chapter 5.2.2) are excluded one by one. The values of the most critical attributes are highlighted. The number given in percent illustrates the increase in the error (from the original error).

Table 11: A summary of the errors for the linear model with the percentile increase in parentheses for the server.

	The measured effort corrected		***The measured effort not corrected***	
Leaving out	W_1	W_2	W_1	W_2
None	*2.6*	*4.0*	*8.7*	*12.0*
a_2	**10.5 (299 %)**	**11.9 (195 %)**	**14.4 (65 %)**	17.5 (45 %)
a_3	3.0 (15 %)	3.7 (-)	8.9 (2 %)	11.2 (-)
a_4	**12.3 (367 %)**	**15.5 (283 %)**	**17.1 (96 %)**	**21.0 (74 %)**
a_6	3.8 (45 %)	4.8 (19 %)	9.7 (11 %)	13.8 (14 %)
a_7	**5.3 (101 %)**	**8.2 (102 %)**	10.6 (22 %)	12.3 (2 %)

6.1.2 The Non-Linear Model

The parameters (power coefficients) of the non-linear model were defined using the same principle as for the linear model. Their values were obtained by minimising the objective functions that contain the corrected measured development efforts are presented in Table 12, and Table 13 shows the corresponding error distribution. The strategies used to minimise W_1 produced slightly different results (not noticeable in Table 12). For this reason, the corresponding percentile error distribution differs insignificantly between the two strategies (case A_2); for this reason the results for both strategies can be observed.

Table 12: The values of the coefficients of the non-linear model that estimates the development effort for the server (the measured effort has been corrected).

Parameters	***Run I***	***Run II***		***Run III***	***Run IV***
F	0.8	0.8		0.8	0.8
CR	0.8	0.8		0.8	0.8
NP	30	30		30	30
G	5000	5000		5000	5000
Strategy	6	7		6	7
Range	[-1, Inf]	[-1, Inf]		[-1, Inf]	[-1, Inf]
b_1	-0.2	-0.2		-0.2	-0.2
b_2	1.6	1.6		1.6	1.6
b_3	3.6	3.6		3.7	3.7
b_4	4.0	4.0		4.0	4.0
b_5	0.2	0.2		0.2	0.2
b_6	2.0	2.0		2.1	2.1
b_7	8.6	8.6		8.4	8.4
b_8	0.0	0.0		0.0	0.0
	W_1			W_2	
Value	6.4	6.4		99.9	99.9

Table 13: The percentile error distribution of the estimated development effort for the server (the non-linear model, the measured effort has been corrected).

F	A_1	A_2	A_3	A_4	A_5	A_6	***Mean error***
W_1 *(strat. 6)*	0	38.6850	0	0	0	0	6.4
W_1 *(strat. 7)*	0	38.6849	0	0	0	0	6.4
W_2	7.4823	15.5239	8.6540	10.5597	0.4529	10.7645	8.9

Using the information shown in Table 12, the following non-linear models for estimating the development effort for the server can be derived:

$$G_1(x_1, x_2, ..., x_8) = x_2^{1.6} + x_3^{3.6} + x_4^{4.0} + x_6^{2.0} + x_7^{8.6} \quad (20)$$

$$G_2(x_1, x_2, ..., x_8) = x_2^{1.6} + x_3^{3.6} + x_4^{4.0} + x_6^{2.0} + x_7^{8.4} \quad (21)$$

From the above equations, it should be noticed that the variables, which are increased to a power of less than one, are not taken, since they do not have an increasing influence on the (estimated) development effort (see "Rule of Significance", chapter 5.2.2).

The same operations are performed to obtain the values of the parameters of the non-linear model based on the uncorrected measured development effort. The values are shown in Table 14, whereas Table 15 gives an overview of the corresponding error distribution.

Table 14: The values of the coefficients of the non-linear model estimating the server's development effort (the measured effort has not been corrected).

Parameter	***Run I***	***Run II***		***Run III***	***Run IV***
F	0.8	0.8		0.8	0.8
CR	0.8	0.8		0.8	0.8
NP	30	30		30	30
G	8000	8000		5000	5000
Strategy	6	7		6	7
Range	[-1, Inf]	[-1, Inf]		[-1, Inf]	[-1, Inf]
b_1	-0.2	-0.2		-0.2	-0.2
b_2	1.6	1.6		1.5	1.5
b_3	3.6	3.6		3.9	3.9
b_4	4.0	4.0		4.0	4.0
b_5	0.2	0.2		0.2	0.2
b_6	2.0	2.0		2.1	2.1
b_7	8.6	8.6		8.1	8.1
b_8	0.0	0.0		0.0	0.0
	W_1			W_2	
Value	13.8	13.8		317.4	317.4

Table 15: The percentile error distribution of the server's estimated development effort (the non-linear model, the measured effort has not been corrected).

F	A_1	A_2	A_3	A_4	A_5	A_6	***Mean error***
W_1	0	82.5090	0	0	0	0	13.8
W_2	14.5977	23.2006	16.8209	20.5159	0.8832	21.1712	16.2

According to values shown in Table 14, the non-linear model that estimates the development effort for the server can be conducted as follows:

$$G_1(x_1, x_2, ..., x_8) = x_2^{1.6} + x_3^{3.6} + x_4^{4.0} + x_6^{2.0} + x_7^{8.6} \quad (22)$$

$$G_2(x_1, x_2, ..., x_8) = x_2^{1.5} + x_3^{3.9} + x_4^{4.0} + x_6^{2.1} + x_7^{8.1} \quad (23)$$

The table below (Table 16) summarises the errors that occurred when a significant attribute was excluded from the model. The values of the attributes that have a crucial influence on the development effort are highlighted.

Table 16: A summary of the errors for the non-linear model with the percentile increase in the error in parentheses for the server.

	The measured effort corrected		***The measured effort not corrected***	
Leaving out	W_1	W_2	W_1	W_2
None	*6.4*	*8.9*	*13.8*	*16.2*
a_2	**16.6 (157 %)**	**21.2 (138 %)**	18.7 (36 %)	23.8 (46 %)
a_3	9.0 (40 %)	12.8 (44 %)	14.5 (5 %)	17.0 (5 %)
a_4	**10.4 (61 %)**	**14.9 (67 %)**	16.2 (18 %)	19.2 (19 %)
a_6	**14.2 (120 %)**	**17.6 (97 %)**	**21.6 (57 %)**	**24.3 (50 %)**
a_7	**11.3 (75 %)**	**16.0 (79 %)**	14.8 (7 %)	17.9 (11 %)

6.2 The Client

6.2.1 The Linear Model

The same procedure was carried out with the client evaluation. Using the same idea, Table 17 presents the coefficients of the linear model for estimating the client's development effort defined using the corrected measured effort, and Table 18 illustrates the distribution of the corresponding errors between the measured and predicted development effort for the client's components.

Table 17: The values of the linear model's coefficients that linear model estimate the development effort for the client (the measured effort has been corrected).

Parameter	***Run I***	***Run II***		***Run III***	***Run IV***
F	0.8	0.8		0.8	0.8
CR	0.8	0.8		0.8	0.8
NP	30	30		30	30
G	5000	5000		5000	5000
Strategy	6	7		6	7
Range	[0, Inf]	[0, Inf]		[0, Inf]	[0, Inf]
b_1	0.0	0.0		0.0	0.0
b_2	0.0	0.0		0.0	0.0
b_3	110.0	110.0		122.5	122.5
b_4	200.5	200.5		225.0	225.0
b_5	21.5	21.5		6.1	6.1
b_6	16.4	16.4		15.8	15.8
b_7	0.0	0.0		0.0	0.0
b_8	0.0	0.0		0.0	0.0
	W_1			W_2	
Value	5.8	5.8		70.3	70.3

Table 18: The percentile error distribution of the estimated development effort for the server (the linear model, the measured effort has been corrected).

F	B_1	B_2	B_3	B_4	B_5	B_6	***Mean error***
W_1	0	0	20.9464	0	13.6276	0	5.8
W_2	2.8062	1.5121	13.9678	9.9005	9.5027	5.3378	7.2

Using the information shown in Table 17, the following functions for estimating the linear development effort can be derived for the client (based on the corrected measured efforts):

$$F_1(x_1, x_2, ..., x_8) = 110.0x_3 + 200.5x_4 + 21.5x_5 + 16.4x_6 \quad (24)$$

$$F_1(x_1, x_2, ..., x_8) = 122.0x_3 + 225.0x_4 + 6.1x_5 + 15.8x_6 \quad (25)$$

The same operations were performed to obtain the values of the parameters of the linear model for the client on the basis of the uncorrected measured development effort. The values are given in Table 19, whereas Table 20 shows the overview of the corresponding error distribution.

Table 19: The values of the linear model's coefficients that linear model estimate the development effort for the client (the measured effort has not been corrected).

Parameter	***Run I***	***Run II***		***Run III***	***Run IV***
F	0.8	0.8		0.8	0.8
CR	0.8	0.8		0.8	0.8
NP	30	30		30	30
G	5000	5000		5000	5000
Strategy	6	7		6	7
Range	[0, Inf]	[0, Inf]		[0, Inf]	[0, Inf]
b_1	0.0	0.0		0.0	0.0
b_2	0.0	0.0		0.0	0.0
b_3	126.4	126.4		111.7	111.7
b_4	230.5	230.5		298.5	298.5
b_5	24.7	24.7		0.0	0.0
b_6	18.9	18.9		5.6	5.6
b_7	0.0	0.0		0.0	0.0
b_8	0.0	0.0		0.0	0.0
	W_1			W_2	
Value	15.5	15.5		70.3	70.3

Table 20: The percentile error distribution of the estimated development effort for the server (the linear model, the measured effort has not been corrected).

F	B_1	B_2	B_3	B_4	B_5	B_6	***Mean error***
W_1	0	0	20.9463	0	71.9285	0	15.5
W_2	24.1125	2.5543	19.3490	10.1305	29.4226	23.4523	18.2

The values of the coefficients presented in Table 19 suggest the following linear models (defined by using the uncorrected measured efforts) for estimating the development effort for the client:

$$F_1(x_1, x_2, ..., x_8) = 126.4x_3 + 230.5x_4 + 24.7x_5 + 18.9x_6 \quad (26)$$

$$F_1(x_1, x_2, ..., x_8) = 111.7x_3 + 298.5x_4 + 5.6x_6 \quad (27)$$

The table below summarises the errors between the measured and estimated development efforts when a significant attribute is excluded from the model. N/A (Not Available) means that the corresponding attribute had no significance in the case in question, for which reason it was not taken into consideration.

Table 21: A summary of the errors for the linear model with the percentile increase for the client in parentheses.

	The measured effort corrected		***The measured effort not corrected***	
Leaving out	W_1	W_2	W_1	W_2
None	*5.8*	*7.2*	*15.5*	*18.2*
a_3	8.1 (41 %)	10.3 (42 %)	17.3 (12 %)	21.2 (16 %)
a_4	**21.8 (279 %)**	**28.5 (297 %)**	**38.5 (149 %)**	**43.7 (141 %)**
a_5	6.7 (15 %)	7.9 (10 %)	16.4 (6 %)	N/A
a_6	8.5 (47 %)	10.7 (49 %)	16.5 (6 %)	18.7 (3 %)

6.2.2 The Non-Linear Model

The parameters (power coefficients) of the non-linear model for the client were defined using the same principle as they were for the server. The difference between the linear and non-linear model is that attribute a_8 is not taken into account here, since it is only assigned the values zero and one (see Table 5) and thus has little influence on the development effort. The parameters of the model obtained on the basis of the corrected efforts are shown in Table 22, and Table 23 illustrates the corresponding error distribution.

Table 22: The values of the coefficients of the non-linear model that estimates the development effort for the client (the measured effort has been corrected).

Parameter	***Run I***	***Run II***		***Run III***	***Run IV***
F	0.8	0.8		0.8	0.8
CR	0.8	0.8		0.8	0.8
NP	30	30		30	30
G	5000	5000		5000	5000
Strategy	6	7		6	7
Range	[-1, Inf]	[-1, Inf]		[-1, Inf]	[-1, Inf]
b_1	-0.1	-0.1		0.3	0.3
b_2	0.2	0.2		0.4	0.4
b_3	4.3	4.3		4.1	4.1
b_4	4.4	4.4		4.5	4.5
b_5	0.9	0.9		0.0	0.0
b_6	2.4	2.4		2.4	2.4
b_7	-1.0	-1.0		-1.0	-1.0
b_8	N/A	N/A		N/A	N/A
	W_1			W_2	
Value	32.1	32.1		2102.0	2102.0

Table 23: The percentile error distribution of the estimated development effort for the client (the non-linear model, the measured effort has been corrected).

F	B_1	B_2	B_3	B_4	B_5	B_6	***Mean error***
W_1	0	0	80.5349	72.2838	40.0317	0	32.1
W_2	0.2136	8.4967	19.6113	71.1636	25.1042	19.6113	34.2

Based on the information presented in Table 22, the following non-linear models for estimating the development effort for the client can be conducted (when defining the model's parameters, the corrected measured development effort was used):

$$G_1(x_1, x_2, ..., x_8) = x_3^{4.3} + x_4^{4.4} + x_6^{2.4} \quad (28)$$

$$G_2(x_1, x_2, ..., x_8) = x_3^{4.2} + x_4^{4.5} + x_6^{2.4} \quad (29)$$

As was done on the basis of the corrected development effort, the power coefficients of the non-linear model were obtained on the basis of the uncorrected development effort, and their values are shown in Table 24. The corresponding error distributions can be found in Table 25, respectively.

Table 24: The values of the coefficients of the non-linear model that estimates the development effort for the client (the measured effort has not been corrected).

Parameter	***Run I***	***Run II***		***Run III***	***Run IV***
F	0.8	0.8		0.8	0.8
CR	0.8	0.8		0.8	0.8
NP	30	30		30	30
G	5000	5000		5000	5000
Strategy	6	7		6	7
Range	[-1, Inf]	[-1, Inf]		[-1, Inf]	[-1, Inf]
b_1	-0.2	-0.2		-0.0	-0.0
b_2	0.2	0.2		0.3	0.3
b_3	1.9	1.9		4.0	4.0
b_4	4.5	4.5		4.6	4.6
b_5	1.1	1.1		0.6	0.6
b_6	2.4	2.4		2.3	2.3
b_7	-1.0	-1.0		-1.0	-1.0
b_8	N/A	N/A		N/A	N/A
	W_1			W_2	
Value	41.7	41.7		2768.7	2768.7

Table 25: The percentile error distribution of the client's estimated development effort (the non-linear model, the measured effort has not been corrected).

F	B_1	B_2	B_3	B_4	B_5	B_6	***Mean error***
W_1	0	0	81.2721	73.4436	0	95.5076	41.7
W_2	8.3527	8.353	83.6637	74.0859	39.1138	46.0057	45.2

Using the information illustrated in Table 24, the non-linear models for the estimation of the client's development effort can be derived as:

$$G_1(x_1, x_2, ..., x_8) = x_3^{1.9} + x_4^{4.5} + x_5^{1.1} + x_6^{2.4} \quad (30)$$

$$G_2(x_1, x_2, ..., x_8) = x_3^{4.0} + x_4^{4.6} + x_6^{2.3} \quad (31)$$

The last table (Table 26) of this chapter presents a summary of the errors for the non-linear model for the client.

Table 26: A summary of the errors for the non-linear model with the percentile increase for the client in parentheses.

	The measured effort corrected		***The measured effort not corrected***	
Leaving out	W_1	W_2	W_1	W_2
None	*32.1*	*34.2*	*41.7*	*45.2*
a_3	36.0 (12 %)	38.7 (13 %)	41.8 (0 %)	46.7 (3 %)
a_4	41.7 (30 %)	44.0 (29 %)	55.4 (33 %)	58.2 (29 %)
a_5	N/A	N/A	41.7 (0 %)	N/A
a_6	44.0 (37 %)	48.5 (42 %)	49.0 (16 %)	52.9 (17 %)

7 SUMMARY AND DISCUSSION

7.1 Summarizing and Discussing Results

The first thing we noticed when we obtained the results was that they showed that the architectural properties of the subsystems differ slightly from each other. Tables 27 and 29 through 31 summarise the values of the coefficients (parameters) of the attributes (the reader is requested to refer to Table 28 to recall the attributes and the corresponding coefficients). For both subsystems, these values differ significantly from each other. Some architectural properties that influence the development effort for the server have no influence on the development effort for the client (these attributes are the size of the subsystem expressed in the number of classes, a_2, and the number of subcomponents, a_7) and vice versa; the module strength (complexity, a_6, that specifies the number of use cases of a component) has no influence on the development effort for the client, while it does have influence in the case of the server (Table 27 and parts of Table 29 and Table 30). Once again, attributes, such as a_1 and a_8, have no influence whatsoever on the development effort for both subsystems.

Table 27: The values of the coefficients (parameters) of the linear model F (equation (1)) defined by minimising W_1 (equation (3)).

	Server		***Client***	
Coef.	*Value (Effort corrected)*	*Value (Effort not corrected)*	*Value (Effort corrected)*	*Value (Effort not corrected)*
b_2	10.5	10.5	-	-
b_3	4.8	4.8	110.0	126.4
b_4	**68.8**	**68.8**	**200.5**	**230.5**
b_5	-	-	21.5	24.7
b_6	3.1	3.1	16.4	18.9
b_7	120.6	120.6	-	-

Table 28: A summary of the metrics suite and related to it parameters.

Attribute (architectural property)	***Related coef. (param.)***	***Metric name***	***Description***
a_1	b_1	Size	Size of a component in KLOC (**K**ilo **L**ines **O**f **C**ode)
a_2	b_2	Size	Size of a component in number of classes
a_3	b_3	Coupling	Number of components referring to this component
a_4	b_4	Coupling	Number of components this component refers to
a_5	b_5	Cohesion	Number of aggregations, compositions and relations among the classes of a component
a_6	b_6	Complexity	Number of use cases of a component
a_7	b_7	Complexity	Number of subcomponents that form a component
a_8	b_8	Complexity	Number of databases connected to a component

As shown in Appendix 1 (the section "a_2 excluded**"**), when excluding attribute a_2, the coefficient of attribute a_1 takes on a significant value in both the linear and non-linear models for the server, while still having no influence on the client. On the basis of this fact, the following decision could possibly be made: the number of classes hides information on the size (in KLOC) of a component (considering the client only).

Table 29: The values of the coefficients (parameters) of the linear model F (equation (1)) defined by minimising W_2 (equation (4)).

	Server		***Client***	
Coef.	*Value (Effort corrected)*	*Value (Effort not corrected)*	*Value (Effort corrected)*	*Value (Effort not corrected)*
b_2	9.5	7.3	-	-
b_3	9.8	21.0	122.0	111.7
b_4	**70.0**	**72.6**	**225.0**	**298.5**
b_5	-	-	6.1	-
b_6	4.9	8.9	15.8	5.6
b_7	103.2	64.2	-	-

According to the results, the size of a component expressed in the number of classes (attribute a_2) is significant and important for the architecture of the server but not for that of the client. When considering the server, the coefficient of that attribute has a value much lower than, for instance, the coefficient of a_4, but it should be remembered that the average value of a_2 is much higher (up to fifteen times) than that of a_4 (Table 32). In the architecture of the client, a_2 plays an important role: when it was removed from the models, the error increased to 299 % in the case of the linear model (corrected efforts, W_1; see Table 11), and by up to 157 % in the case of the non-linear model (corrected efforts, W_1; see Table 16).

For the server, the coefficient of attribute a_3 gets relatively insignificant values (Table 27 and Table 29), when taking into account that the attribute has a low average value (Table 32). Once again, for the client in most cases, the corresponding value is high for both models (Table 30 and Table 31).). In any case, leaving out this attribute increases the error insignificantly: at most by only 44 % in the case of the server and 41 % in the case of the client (Table 16 and Table 21). Based on these numbers, it may be decided that attribute a_3 is not too important in the architectural design of both subsystems. The explanation for this

might be that attribute a_4 probably hides some information about a_3: when a_4 is excluded, the value of the parameter of a_3 increases significantly (Appendix 1, section " a_4 excluded").

When excluded from both models in both subsystems, attribute a_4 causes an increase in the error to 367 % in the linear model for the server (Table 11) and to 297 % in the linear model for the client (Table 21). The coefficient, as well as the exponent, of the variable that corresponds to attribute a_4 gets significant values in both models in both subsystems (Table 27 to Table 31). Due to the above-mentioned behaviour, this attribute probably becomes the most important architectural property (according to the importance rule presented in chapter 5.2.2) for the development effort of both subsystems if their architectures are evaluated by the linear model (the non-linear model gives lower errors: Table 16 and Table 26). On the basis of these observations, the hypothesis presented in the introductory part to this thesis can be assumed proven.

In software engineering, it is well known that cohesion (attribute a_5) should be high in the modules of a correctly implemented software system. According to the results (coefficient b_5), the parameters (linear and power coefficients) of both models are insignificant for the server and have some significance for the client. One reason for this could be the improper selection of the definition of cohesion. Even though cohesion is present in some models for the client, the value of its coefficient (or exponent) is relatively low compared with those of coupling (attribute a_4). When leaving cohesion out from the models, the increase in the error is quite insignificant; at most 15 % (Table 21 and Table 26), which means that this architectural attribute is not too important in the architecture of the client. And again, in the case of the server, cohesion has no influence on its architecture.

Architectural property a_6 (complexity), which denotes the number of use-cases of a model, is present in both models of both subsystems. Furthermore, the behaviour of the value of its coefficient (or exponent) is more or less stable even

though the values of the attribute are not too reliable. This is because the values of the use-cases were defined based on the technical specifications and each developer had his/her own view on what the use-cases are. In some cases, the number of use-cases of a component is important in the architecture of the subsystems. For example, excluding a_6 from the non-linear model optimised by W_1 with the corrected development efforts for the server increases the error by 120 % (Table 16). In the case of the client, the increases are not that significant (the biggest is 49 % for the linear model optimised by W_2 with corrected development efforts, see Table 21).

Table 30: The values of the power coefficients (parameters) of the non-linear model G (equation (2)) defined by minimising W_1 (equation (3)).

	Server		***Client***	
Coef.	*Value (Effort corrected)*	*Value (Effort not corrected)*	*Value (Effort corrected)*	*Value (Effort not corrected)*
b_2	1.6	1.6	-	-
b_3	3.6	3.6	4.3	1.9
b_4	**4.0**	**4.0**	**4.4**	**4.5**
b_5	-	-	-	1.1
b_6	2.0	2.0	2.4	2.4
b_7	8.6	8.6	-	-

According to the results, a_7 (which refers to the number of subcomponents forming a component) is the most significant attribute in the server's architecture described by both models. Coefficients with such high values could be explained by the relatively low values of the attribute: for the server, the average value is 1 and for the client 2, whereas, for instance, the average value of attribute a_4 (coupling) for the server is 2 and for the client 3, respectively (Table 32). In any case, high significance did not make a_7 the most important attribute. For example, in the case of the linear model optimised by W_1 with the corrected

development effort, leaving a_7 out increases the error by only 101 %, whereas the corresponding increase in the case of attribute a_4 is 367 % (Table 11). Again, in the case of the client's architecture, a_7 has no influence at all.

As was later noticed, the databases of the components were implemented as Java classes. This offers a reasonable explanation for the insignificance of attribute a_8: the information on the databases of a component is probably contained within the number of classes corresponding to attribute a_2.

Table 31: The values of the power coefficients (parameters) of the non-linear model G (equation (2)) defined by minimising W_2 (equation (4)).

	Server		***Client***	
Coef.	*Value (Effort corrected)*	*Value (Effort not corrected)*	*Value (Effort corrected)*	*Value (Effort not corrected)*
b_2	1.6	1.5	-	-
b_3	3.6	3.9	4.2	4.0
b_4	**4.0**	**4.0**	**4.5**	**4.6**
b_5	-	-	-	-
b_6	2.0	2.1	2.4	2.3
b_7	8.4	8.1	-	-

There may be two possible explanations for the differences in the architectural properties of the subsystems. The fact is that the server was implemented within the same site by experienced developers and there was a prototype that described the interfaces between the components. The client was developed in different sites by less experienced software engineers (which was the reason why the equalisation of the development effort presented in chapter 5.1 was performed). The client had to be developed before the server but was delayed. No prototype was created, and the developers confronted sizeable problems when integrating the components, because the interfaces between them had not been properly

designed. This can be seen in the significance of attributes a_3 and a_4 which have very high values, b_3 and b_4, (according to the above tables), and from their importance (especially a_4): when excluding attribute a_4 from the linear model, the error produced by W_1 increased from 5.8 % to 21.8 %, which is a change of approximately four-fold (Table 21). The values of b_4 for the server are significantly lower than the corresponding values for the client, especially in the case of the linear model (tables above), but the exclusion of b_4 causes an increase in the error from 2.6 % to 12.3 % (about five-fold, see Table 11). These numbers depict the high importance of coupling.

The other fact is that the server was implemented using CORBA, which might influence the difference between the architectural properties of the server and the client. The purpose of this work is not to discuss CORBA in depth, and therefore, readers who wish to know more about CORBA are requested to refer to [40].

Table 32: The average values of the attributes of both subsystems. The values were defined based on the information presented in Table 4 and Table 5.

Attribute	*Average value (the server)*	*Average value (the client)*
a_1	**3** ($2\frac{5}{6}$)	**4** ($3\frac{5}{6}$)
a_2	**31** ($30\frac{5}{6}$)	**29** ($29\frac{1}{3}$)
a_3	**3** ($2\frac{1}{2}$)	**2** ($2\frac{1}{3}$)
a_4	**2** ($2\frac{1}{3}$)	**3** ($2\frac{1}{2}$)
a_5	**23** ($23\frac{1}{3}$)	**6** ($5\frac{2}{3}$)
a_6	**9** ($8\frac{2}{3}$)	**10**
a_7	**1** ($1\frac{1}{3}$)	**2** ($1\frac{5}{6}$)
a_8	**1**	**0** ($\frac{1}{3}$)

A difference can be noticed in the results of the models. In all the cases, the linear model fitted the data better than the non-linear model. The explanation for this could be the very high sensitivity of the non-linear function because it consists of variables raised to powers. It is well known that even a small increase in an exponent may produce very large changes in the result, whereas increases in the variables (or coefficients) of a linear model are not that dramatic. The final conclusions in the scope of this project are that (1) the number of components, to which the component being studied refers, is very probably the most important architectural property of the architectures of both subsystems, and (2) a linear model is more suitable for estimating the development effort than a non-linear model.

7.2 Performance of the Selected Approach

From the theoretical point of view, it is possible to utilise soft computing methods in software engineering when modelling software processes. As this work shows, a differential evolution algorithm, which is a soft computing approach, is suitable for defining the parameters (the values of the coefficients and exponents) of a model for estimating the software development effort on the basis of architectural design properties. The quantitative results support the observation made during the qualitative analysis. This means the method used (DE) also works from the practical perspective, which encourages the use of DE in other suitable software engineering tasks.

7.3 Suggestions for the Future Work

In the scope of this research, it seems that the attributes of the architectural design metrics employed are not completely independent. Future research could focus on studying the interdependencies between architectural attributes. From the results presented in the appendices, it can be seen that excluding specific attributes influenced the values of the coefficients and exponents of other attributes as well as the result. These phenomena are to be interpreted. A local sensitivity analysis on the attributes could also be performed by increasing the value of a specific attribute by, for instance, one percent and studying how this change affects other significant attributes. Finally, the exact models for estimating the development effort on the basis of architectural properties could be created by, for example, combining both linear and non-linear models. Another topic of interest would be investigating whether estimation models should include the (non-linear) interdependencies between the studied attributes.

DE is a unique optimisation approach and performed well within the scope of this project, which makes it reasonable to continue with its usage in future work as well.

REFERENCES

1. R. Harrison, S. Counsell, R. Nithi, 1997: An Overview of Object-Oriented Design Metrics (Editors: Budgen, D., Hoffnagle, G., Trienekens, J. Dept. of Electron. & Comput. Sci., Southampton Univ., UK). Proceedings of Software Technology and Engineering Practice. 8th IEEE International Workshop on Incorporating Computer Aided Software Engineering. London, UK. 14-18 July 1997. Pages: 230 – 235. ISBN: 0-8186-7840-2.

2. Shyam R. Chidamber and Chris F. Kemerer, 1994: A Metrics Suite for Object Oriented Design. IEEE Transactions on Software Engineering. June 1994. Volume: 20, Issue: 6, Pages: 476 –493. ISSN: 0098-5589.

3. Alberto Avritzer, Elaine J. Weyuker, 1998: Investigating Metrics for Architectural Assessment. Software Metrics Symposium, 1998, 5th International Proceedings on Software Metrics. Bethesda, MD, USA. 20-21 Nov. 1998. Pages 4-10. ISBN: 0-8186-9201-4.

4. M. Shereshevsky, H. Ammari, N. Gradetsky, A. Mili, H. H. Ammar, 2001: Information Theoretic Metrics for Software Architectures. 25th Annual International Conference on Computer Software and Applications, 2001, COMPSAC 2001. Chicago, IL, USA. 8-12 Oct. 2001. Pages: 151 – 157. ISBN: 0-7695-1372-7.

5. Lionel C. Briand, Jürgen Wüst, 2001: The Impact of Design Properties on Development Cost in Object-Oriented Systems. Software Metrics Symposium, 2001. METRICS 2001. 7th International Proceedings on Software Metrics. London, UK. 4-6 April 2001. Pages: 260 – 271. ISBN: 0-7695-1043-4.

6. Lionel C. Briand, Khaled El Emam, Frank Bomarius, 1998: CORBA: A Hybrid Method for Software Cost Estimation, Benchmarking, and Risk

Assessment. Proceedings of the 1998 International Conference on Software Engineering. Kyoto, Japan. 19-25 April 1998. Pages: 390 – 399. ISBN: 0-8186-8368-6.

7. Ross Jeffery, Melanie Ruhe, Isabella Wieczorek, 2000: Using Public Domain Metrics to Estimate Software Development Effort. Software Metrics Symposium, 2001. METRICS 2001. 7th International Proceedings. London, UK. 4-6 April 2001. Pages: 16 – 27. ISBN: 0-7695-1043-4.

8. Krishnakumar Pillai, V.S. Sukumaran Nair, 1997: A Model for Software Development Effort and Cost Estimation. IEEE Transactions on Software Engineering. Aug. 1997. Volume: 23, Issue: 8; Pages: 485 – 497. ISSN: 0098-5589.

9. Jack E. Matson, Bruce E. Barrett, and Joseph M. Mellichamp, 1997: Software Development Cost Estimation Using Function Points. IEEE Transactions on Software Engineering. April 1994. Volume: 20, Issue: 4; Pages: 275 – 287. ISSN: 0098-5589.

10. Yooichi Yokoyama and Mitsuhiko Kodaira, 1998: Software Cost and Quality Analysis by Statistical Approaches. Proceedings of the 1998 International Conference on Software Engineering. Kyoto, Japan. 19-25 April 1998. Pages: 465 – 467. ISBN: 0-8186-8368-6.

11. Jagdish Bansiya, Carl G. Davis, 2002: A Hierarchical Model for Object-Oriented Design Quality Assessment. IEEE Transactions on Software Engineering. Jan. 2002. Volume: 28, Issue: 1, Pages: 4 – 17. ISSN: 0098-5589.

12. Roger S. Pressman: Software Engineering, A Practitioner's Approach. 4th edition. The McGraw-Hill Companies, Inc.. ISBN: 0077094115. p. 129.

13. Orsolya Dobán, András Pataricza, 2001: Cost Estimation Driven Software Development Process. 27th Proceedings of Euromicro Conference, 2001. Warsaw, Poland. 4-6 Sept. 2001. Pages: 208 – 213. ISBN: 0-7695-1236-4.

14. Sunita Devnani-Chulani, Bradford Clark, Barry Boehm, 1998: Calibrating the COCOMO II Post-Architectural Model. Proceedings of the 1998 International Conference on Software Engineering. Kyoto, Japan. 19-25 April 1998. Pages: 477 – 480. ISBN: 0-8186-8368-6.

15. Rainer Storn, 1996: On the Usage of Differential Evolution for Function Optimization (Editors: Smith, M.H., Lee, M.A., Keller, J., Yen). 1996 Biennial Conference of the North American. Fuzzy Information Processing Society, 1996. NAFIPS. Berkeley, CA, USA. 19-22 June 1996. Pages: 519 – 523. ISBN: 0-7803-3225-3.

16. Mary Shaw, David Garlan: Software Architecture: Perspectives on an Emerging Discipline. Prentice-Hall, Inc., 1996. ISBN: 0-13-182957-2.

17. Len Bass, Paul Clements, Rick Kazman, 1998: Software Architecture in Practice. Addison Wesley Longman, Inc., 1998. ISBN: 0-201-19930-0.

18. Norman E. Fenton, Shari Lawrence Pfleeger, 1997: Software Metrics: A Rigorous & Practical Approach. 2nd edition. PWS Publishing Company, 1997. ISBN: 0-534-95425-1.

19. Franck Xia, 1996: Module Coupling: A Design Metric. Proceedings of the 1996 Asia-Pacific Conference on Software Engineering. Seoul, South Korea. 4-7 Dec. 1996. Pages: 44 – 54. ISBN: 0-8186-7638-8.

20. The official Web-site of the TogetherSoft™ Corporation: TogetherSoft, a Development Tool for an Application Modeling and Round Trip

Engineering for Java and C++. Web-document, URL: http://www.togethersoft.com/. [Referred 22.09.2002].

21. Lines of Code Counter, Dr. John Dalbey's Web-page on Web-server of Dep. of Comp. Sc., California Polytechnic State University. Web-document, URL: http://www.csc.calpoly.edu/~jdalbey/SWE/PSP/LOChelp.html; [Referred 22.09.2002].

22. Ali Idri, Taghi M. Khoshgoftaar, Alain Abran, 2002: Can neural Networks be easily Interpreted in Software Cost Estimation? Proceedings of the 2002 IEEE International Conference on Fuzzy Systems. FUZZ-IEEE'02. Honolulu, HI, USA. 12-17 May 2002. Volume: 2, Pages: 1162 – 1167. ISBN: 0-7803-7280-8.

23. Gavin R. Finnie and Gerhard E. Wittig, 1996: AI Tools for Software Development Effort Estimation (Editors: Purvis, M. Bond Univ., Gold Coast, Qld., Australia). Proceedings of the 1996 International Conference on Software Engineering: Education and Practice. Dunedin, New Zealand. 24-27 Jan. 1996. Pages: 346 – 353; ISBN: 0-8186-7379-6.

24. A.R. Venkatachalam, 1993: Software Cost Estimation Using Artificial Neural Networks. Proceedings of the 1993 International Joint Conference on Neural Networks. IJCNN '93-Nagoya. October 25-29, 1993. Volume 1, Pages: 987 – 990. ISBN: 0-7803-1421-2.

25. W. Pedrycz, J.F. Peters, S. Ramanna, 1999: A Fuzzy Set Approach to Cost Estimation of Software Project (Editor: Meng, M.). IEEE Canadian Conference on Electrical and Computer Engineering, 1999. Edmonton, Alta, Canada. 9-12 May 1999. Volume: 2, Pages: 1068 – 1073. ISBN: 0-7803-5579-2.

26. Jouni Lampinen: Multi-Constrained Nonlinear Optimization by the Differential Evolution Algorithm. A Document Proposing an Extension for the Differential Evolution Algorithm (DEA) for Handling Non-linear Constraint Functions. Web-document, URL: http://www.it.lut.fi/opetus/01-02/010778000/DECONSTR.PDF. [Referred 29.09.2002].

27. Jouni Lampinen and Ivan Zelinka, 1999: Mechanical Engineering Design Optimization by Differential Evolution. In: David Corne, Marco Dorigo and Fred Glover: New Ideas in Optimization; McGraw-Hill, London (UK), ISBN: 007-709506-5. Pages: 127-146.

28. Randy L. Haupt, Sue Ellen Haupt, 1998: Practical Genetic Algorithms, John Wiley & Sons, Inc., 1998, ISBN: 0-471-18873-5.

29. Sami Khuri, Thomas Black and Jörg Heitkötter, 1993: The Zero/One Multiple Knapsack Problem and Genetic Algorithms. The ACM Symposium of Applied Computation (SAC'94) proceedings. 1993 ACM Press.

30. Velazco, M.I. and Lyra, C., 2002: Optimization with Neural Networks Trained by Evolutionary Algorithms. Proceedings of the 2002 International Joint Conference on Neural Networks. IJCNN'02. Honolulu, HI, USA. 12-17 May 2002. Volume: 2, Pages: 1516 – 1521. ISBN: 0-7803-7278-6.

31. B. Fayech, S. Hammadi, S. Maouche and P. Borne, 2001: Urban Bus Traffic Regulation by Evolutionary Algorithms. 2001 IEEE International Conference on Systems, Man, and Cybernetics. Tucson, AZ, USA. 7-10 Oct. 2001. Volume: 2, Page: 1316 – 1322. ISBN: 0-7803-7087-2.

32. Seong-Joo Han and Se-Young Oh, 2001: Evolutionary Algorithm Based Neural Network Controller Optimization for Autonomous Mobile Robot Navigation. Proceedings of the 2001 International Joint Conference on Neural Networks. IJCNN'01. Washington, DC, USA. 15-19 July 2001. Volume: 3, Pages: 2194 – 2199. ISBN: 0-7803-7044-9.

33. Michael Watts, Louise Major and Warren Tate, 2002: Evolutionary Optimisation of MLP for Modelling Protein Synthesis Termination Signal Efficiency. Proceedings of the 2002 Congress on Evolutionary Computation, CEC '02. Honolulu, HI, USA. 12-17 May 2002. Volume: 1, Pages: 193 – 198. ISBN: 0-7803-7282-4.

34. D. H. Milone, J. J. Merelo and H. L. Rufiner, 2002: Evolutionary Algorithm for Speech Segmentation. Proceedings of the 2002 Congress on Evolutionary Computation, CEC '02. Honolulu, HI, USA. 12-17 May 2002. Volume: 2, Pages: 1115 – 1120. ISBN: 0-7803-7282-4.

35. A Web-server of the International Computer Science Institute ICSI, Berkley, a Home Page of Rainer Storn, Reference to Differential Evolution (DE). Web-document, URL: http://www.icsi.berkeley.edu/~storn/code.html. [Referred 27.09.2002].

36. Jouni Lampinen: Global Optimization by Differential Evolution; An Introduction to the Differential Evolution Algorithm (DEA) for Global Optimization Over Continuous Spaces. Web-document, URL: http://www.it.lut.fi/opetus/01-02/010778000/DE.pdf. [Referred 28.09.2002].

37. Barry W. Boehm, Chris Abts, A. Winsor Brown, Sunita Chulani, Bradford K. Clark, Ellis Horowitz, Ray Madachy, Donald Reifer and Bert Steece: Software Cost Estimation with COCOMO II, Prentice-Hall, Inc., 2000, ISBN: 0-13-02-6692-2.

38. Päivi Ovaska, 2002: Designing a Qualitative Study for Analyzing Relationships between Software Architecture and Software Project Outcome. Information Systems Research in Scandinavia, a Workshop in Denmark. August 10-13, 2002.

39. EE290A: Advanced Topics in CAD. Component-Based Design of Electronic Systems. A Course Given in the University of California, Berkley by Mr. Kurtz Keutzer and Mr. Richard Newton. The Lecture Notes. Lectures on Spring Semester 1999. Web-document, URL: http://www-cad.eecs.berkeley.edu/~newton/Classes/EE290sp99/lectures/ee290aSp994_1/tsld009.htm. [Referred 6.11.2002].

40. Robert Orfali and Dan Harkey: Client/Server Programming with Java and CORBA, 2nd Edition. John Wiley & Sons, Inc., 1998. ISBN: 0-471-24578-X.

41. B. Bruegge & A. H. Dutoit: Object-Oriented Software Engineering: Conquering Complex and Changing Systems. Prentice Hall, 2000. ISBN: 0-13-489725, pp. 174-178.

Appendix 1. Server Application, Linear Model, Effort Corrected

a_2 excluded

Table 33: The values of the parameters (coefficients).

Parameter	***Run I***	***Run II***		***Run III***	***Run IV***
F	0.8	0.8		0.8	0.8
CR	0.8	0.8		0.8	0.8
NP	30	30		30	30
G	5000	5000		5000	5000
Strategy	6	7		6	7
Range	[0, Inf]	[0, Inf]		[0, Inf]	[0, Inf]
b_1	50.0	50.0		80.0	80.0
b_2	-	-		-	-
b_3	0.0	0.0		18.5	18.5
b_4	44.2	44.2		53.6	53.6
b_5	0.0	0.0		0.0	0.0
b_6	17.8	17.8		14.7	14.7
b_7	136.0	136.0		91.9	91.9
b_8	0.0	0.0		0.0	0.0
	W_1			W_2	
Value	10.5	10.5		189.3	189.3

Table 34: The percentile error distribution among the components.

F	A_1	A_2	A_3	A_4	A_5	A_6	***Mean error***
W_1	0	0	23.9910	39.0964	0	0	10.5
W_2	12.2078	17.7958	17.0091	18.6141	0.1166	5.8551	11.9

(continues on next page)

(continuation of Appendix 1)

a_3 **excluded**

Table 35: The values of the parameters (coefficients).

Parameter	***Run I***	***Run II***		***Run III***	***Run IV***
F	0.8	0.8		0.8	0.8
CR	0.8	0.8		0.8	0.8
NP	30	30		30	30
G	5000	5000		5000	5000
Strategy	6	7		6	7
Range	[0, Inf]	[0, Inf]		[0, Inf]	[0, Inf]
b_1	0.0	0.0		0.0	0.0
b_2	9.9	9.9		9.9	9.9
b_3	-	-		-	-
b_4	69.9	69.9		71.8	71.8
b_5	0.0	0.0		0.0	0.0
b_6	5.6	5.6		6.8	6.8
b_7	115.3	115.3		99.6	99.6
b_8	0.0	0.0		0.0	0.0
	W_1			W_2	
Value	3.0	3.0		26.9	26.9

Table 36: The percentile error distribution among the components.

F	A_1	A_2	A_3	A_4	A_5	A_6	***Mean error***
W_1	0	12.1285	0	6.0454	0	0	3.0
W_2	0.5212	9.3372	0.3028	7.9622	0.6475	3.1653	3.7

(continues on next page)

(continuation of Appendix 1)

a_4 excluded

Table 37: The values of the parameters (coefficients).

Parameter	***Run I***	***Run II***		***Run III***	***Run IV***
F	0.8	0.8		0.8	0.8
CR	0.8	0.8		0.8	0.8
NP	30	30		30	30
G	5000	5000		5000	5000
Strategy	6	7		6	7
Range	[0, Inf]	[0, Inf]		[0, Inf]	[0, Inf]
b_1	0.0	0.0		13.7	13.7
b_2	4.5	4.5		6.4	6.4
b_3	66.6	66.6		28.0	28.0
b_4	-	-		-	-
b_5	0.0	0.0		0.0	0.0
b_6	0.0	0.0		4.5	4.5
b_7	233.7	233.7		211.2	211.2
b_8	0.0	0.0		0.0	0.0
	W_1			W_2	
Value	12.3	12.3		346.7	346.7

Table 38: The percentile error distribution among the components.

F	A_1	A_2	A_3	A_4	A_5	A_6	***Mean error***
W_1	0	0.5536	44.4319	0	28.8076	0	12.3
W_2	18.8191	12.9658	31.6612	2.9495	23.1237	3.4605	15.5

(continues on next page)

(continuation of Appendix 1)

a_6 **excluded**

Table 39: The values of the parameters (coefficients).

Parameter	***Run I***	***Run II***		***Run III***	***Run IV***
F	0.8	0.8		0.8	0.8
CR	0.8	0.8		0.8	0.8
NP	30	30		30	30
G	5000	5000		5000	5000
Strategy	6	7		6	7
Range	[0, Inf]	[0, Inf]		[0, Inf]	[0, Inf]
b_1	0.0	0.0		0.0	0.0
b_2	9.1	9.1		9.4	9.4
b_3	20.3	20.3		19.4	19.4
b_4	57.5	57.5		68.8	68.8
b_5	0.0	0.0		0.0	0.0
b_6	-	-		-	-
b_7	147.9	147.9		121.1	121.1
b_8	0.0	0.0		0.0	0.0
	W_1			W_2	
Value	3.8	3.8		33.9	33.9

Table 40: The percentile error distribution among the components.

F	A_1	A_2	A_3	A_4	A_5	A_6	***Mean error***
W_1	0	11.4987	11.4950	0	0	0	3.8
W_2	3.4130	10.0567	7.6322	0.0908	4.9052	2.8751	4.8

(continues on next page)

(continuation of Appendix 1)

a_7 **excluded**

Table 41: The values of the parameters (coefficients).

Parameter	***Run I***	***Run II***		***Run III***	***Run IV***
F	0.8	0.8		0.8	0.8
CR	0.8	0.8		0.8	0.8
NP	30	30		30	30
G	5000	5000		5000	5000
Strategy	6	7		6	7
Range	[0, Inf]	[0, Inf]		[0, Inf]	[0, Inf]
b_1	0.0	0.0		0.0	0.0
b_2	11.5	11.5		11.3	11.3
b_3	19.1	19.1		3.7	3.7
b_4	71.9	71.9		90.0	90.0
b_5	0.0	0.0		0.0	0.0
b_6	7.3	7.3		10.8	10.8
b_7	-	-		-	-
b_8	0.0	0.0		0.0	0.0
	W_1			W_2	
Value	5.3	5.3		104.0	104.0

Table 42: The percentile error distribution among the components.

F	A_1	A_2	A_3	A_4	A_5	A_6	***Mean error***
W_1	0	1.0558	0	0	0	30.7574	5.3
W_2	8.1137	3.2407	10.0736	6.0931	1.3972	20.1782	8.1

Appendix 2. Server Application, Linear Model, Effort Not Corrected

a_2 excluded

Table 43: The values of the parameters (coefficients).

Parameter	***Run I***	***Run II***		***Run III***	***Run IV***
F	0.8	0.8		0.8	0.8
CR	0.8	0.8		0.8	0.8
NP	30	30		30	30
G	5000	5000		5000	5000
Strategy	6	7		6	7
Range	[0, Inf]	[0, Inf]		[0, Inf]	[0, Inf]
b_1	16.9	16.9		44.1	44.1
b_2	-	-		-	-
b_3	0.0	0.0		35.3	35.3
b_4	45.7	45.7		58.0	58.0
b_5	0.0	0.0		0.0	0.0
b_6	19.7	19.7		13.8	13.8
b_7	143.4	143.4		88.0	88.0
b_8	0.0	0.0		0.0	0.0
	W_1			W_2	
Value	14.4	14.4		406.4	406.4

Table 44: The percentile error distribution among the components.

F	A_1	A_2	A_3	A_4	A_5	A_6	***Mean error***
W_1	0	0	31.0037	55.6449	0	0	14.4
W_2	19.0445	21.0958	26.5347	29.0385	0.1819	9.1342	17.5

(continues on next page)

(continuation of Appendix 2)

a_3 **excluded**

Table 45: The values of the parameters (coefficients).

Parameter	***Run I***	***Run II***		***Run III***	***Run IV***
F	0.8	0.8		0.8	0.8
CR	0.8	0.8		0.8	0.8
NP	30	30		30	30
G	5000	5000		5000	5000
Strategy	6	7		6	7
Range	[0, Inf]	[0, Inf]		[0, Inf]	[0, Inf]
b_1	0.0	0.0		0.0	0.0
b_2	9.9	9.9		8.0	8.0
b_3	-	-		-	-
b_4	69.7	69.7		76.8	76.8
b_5	0.0	0.0		0.0	0.0
b_6	5.6	5.6		13.1	13.1
b_7	115.3	115.3		55.1	55.1
b_8	0.0	0.0		0.0	0.0
	W_1			W_2	
Value	8.9	8.9		203.9	203.9

Table 46: The percentile error distribution among the components.

F	A_1	A_2	A_3	A_4	A_5	A_6	***Mean error***
W_1	0	47.5608	0	6.0454	0	0	8.9
W_2	4.5768	22.7767	2.9716	23.2101	1.8850	11.5228	11.2

(continues on next page)

(continuation of Appendix 2)

a_4 excluded

Table 47: The values of the parameters (coefficients).

Parameter	***Run I***	***Run II***		***Run III***	***Run IV***
F	0.8	0.8		0.8	0.8
CR	0.8	0.8		0.8	0.8
NP	30	30		30	30
G	5000	5000		5000	5000
Strategy	6	7		6	7
Range	[0, Inf]	[0, Inf]		[0, Inf]	[0, Inf]
b_1	0.0	0.0		0.0	0.0
b_2	0.0	0.0		5.3	5.3
b_3	71.7	71.7		40.9	40.9
b_4	-	-		-	-
b_5	0.0	0.0		0.0	0.0
b_6	0.0	0.0		6.6	6.6
b_7	253.6	253.6		185.8	185.8
b_8	0.0	0.0		0.0	0.0
	W_1			W_2	
Value	17.1	17.1		534.6	534.6

Table 48: The percentile error distribution among the components.

F	A_1	A_2	A_3	A_4	A_5	A_6	***Mean error***
W_1	0	2.5554	63.4225	24.0871	12.4186	0	17.1
W_2	14.2411	23.0068	39.9748	13.1684	23.8792	11.5800	21.0

(continues on next page)

(continuation of Appendix 2)

a_6 **excluded**

Table 49: The values of the parameters (coefficients).

Parameter	***Run I***	***Run II***		***Run III***	***Run IV***
F	0.8	0.8		0.8	0.8
CR	0.8	0.8		0.8	0.8
NP	30	30		30	30
G	5000	5000		5000	5000
Strategy	6	7		6	7
Range	[0, Inf]	[0, Inf]		[0, Inf]	[0, Inf]
b_1	0.0	0.0		0.0	0.0
b_2	9.1	9.1		7.1	7.1
b_3	20.3	20.3		39.4	39.4
b_4	57.5	57.5		70.6	70.6
b_5	0.0	0.0		0.0	0.0
b_6	-	-		-	-
b_7	147.9	147.9		95.6	95.6
b_8	0.0	0.0		0.0	0.0
	W_1			W_2	
Value	9.7	9.7		221.8	221.8

Table 50: The percentile error distribution among the components.

F	A_1	A_2	A_3	A_4	A_5	A_6	***Mean error***
W_1	0	46.7320	11.4950	0	0	0	9.7
W_2	10.8476	23.8141	18.9364	7.8758	9.7437	11.4112	13.8

(continues on next page)

(continuation of Appendix 2)

a_7 **excluded**

Table 51: The values of the parameters (coefficients).

Parameter	***Run I***	***Run II***		***Run III***	***Run IV***
F	0.8	0.8		0.8	0.8
CR	0.8	0.8		0.8	0.8
NP	30	30		30	30
G	5000	5000		5000	5000
Strategy	6	7		6	7
Range	[0, Inf]	[0, Inf]		[0, Inf]	[0, Inf]
b_1	0.0	0.0		0.0	0.0
b_2	11.5	11.5		8.6	8.6
b_3	19.1	19.1		16.6	16.6
b_4	71.9	71.9		85.3	85.3
b_5	0.0	0.0		0.0	0.0
b_6	7.3	7.3		12.4	12.4
b_7	-	-		-	-
b_8	0.0	0.0		0.0	0.0
	W_1			W_2	
Value	10.6	10.6		215.0	215.0

Table 52: The percentile error distribution among the components.

F	A_1	A_2	A_3	A_4	A_5	A_6	***Mean error***
W_1	0	32.9891	0	0	0	30.7574	10.6
W_2	12.7115	17.6006	1.4565	16.7620	1.9356	23.0556	12.3

Appendix 3. Server Application, Non-Linear Model, Effort Corrected

a_2 excluded

Table 53: The values of the parameters (coefficients).

Parameter	***Run I***	***Run II***		***Run III***	***Run IV***
F	0.8	0.8		0.8	0.8
CR	0.8	0.8		0.8	0.8
NP	30	30		30	30
G	8000	5000		5000	5000
Strategy	6	7		6	7
Range	[-1, Inf]	[-1, Inf]		[-1, Inf]	[-1, Inf]
b_1	1.6	1.6		1.8	1.8
b_2	-	-		-	-
b_3	4.3	4.3		4.2	4.2
b_4	0.8	0.8		3.6	3.6
b_5	1.3	1.3		1.5	1.5
b_6	2.2	2.2		2.2	2.2
b_7	8.9	8.9		8.5	8.5
b_8	0.0	0.0		0.0	0.0
	W_1			W_2	
Value	16.6	16.6		559.4	559.4

Table 54: The percentile error distribution among the components.

F	A_1	A_2	A_3	A_4	A_5	A_6	***Mean error***
W_1 *(strat. 6)*	0	0	61.7215	37.8596	0	0	16.6
W_2	24.9536	14.8073	37.2677	35.1665	4.8693	10.2676	21.2

(continues on next page)

(continuation of Appendix 3)

a_3 **excluded**

Table 55: The values of the parameters (coefficients).

Parameter	***Run I***	***Run II***		***Run III***	***Run IV***
F	0.8	0.8		0.8	0.8
CR	0.8	0.8		0.8	0.8
NP	30	30		30	30
G	10000	5000		5000	5000
Strategy	6	7		6	7
Range	[-1, Inf]	[-1, Inf]		[-1, Inf]	[-1, Inf]
b_1	-0.3	-0.3		-0.4	-0.4
b_2	1.5	1.5		1.6	1.6
b_3	-	-		-	-
b_4	4.2	4.2		4.2	4.2
b_5	0.2	0.2		0.2	0.2
b_6	2.2	2.2		2.1	2.1
b_7	8.5	8.5		8.2	8.2
b_8	0.0	0.0		0.0	0.0
	W_1			W_2	
Value	9.0	9.0		232.7	232.7

Table 56: The percentile error distribution among the components.

F	A_1	A_2	A_3	A_4	A_5	A_6	***Mean error***
W_1	0	3.5008	0	50.7840	0	0	9.0
W_2	4.2877	19.3218	11.3640	26.2681	2.3702	13.3980	12.8

(continues on next page)

(continuation of Appendix 3)

a_4 **excluded**

Table 57: The values of the parameters (coefficients).

Parameter	***Run I***	***Run II***		***Run III***	***Run IV***
F	0.8	0.8		0.8	0.8
CR	0.8	0.8		0.8	0.8
NP	30	30		30	30
G	10000	5000		5000	5000
Strategy	6	7		6	7
Range	[-1, Inf]	[-1, Inf]		[-1, Inf]	[-1, Inf]
b_1	-0.1	-0.1		0.2	0.2
b_2	1.5	1.5		1.5	1.5
b_3	4.3	4.3		4.2	4.2
b_4	-	-		-	-
b_5	0.3	0.3		0.5	0.5
b_6	2.1	2.1		2.2	2.2
b_7	8.9	8.9		8.7	8.7
b_8	0.0	0.0		0.0	0.0
	W_1			W_2	
Value	10.4	10.4		344.3	344.3

Table 58: The percentile error distribution among the components.

F	A_1	A_2	A_3	A_4	A_5	A_6	***Mean error***
W_1	0	11.1007	51.2142	0	0	0	10.4
W_2	6.9205	15.7558	36.9341	1.8874	16.8036	10.9253	14.9

(continues on next page)

(continuation of Appendix 3)

a_6 **excluded**

Table 59: The values of the parameters (coefficients).

Parameter	***Run I***	***Run II***		***Run III***	***Run IV***
F	0.8	0.8		0.8	0.8
CR	0.8	0.8		0.8	0.8
NP	30	30		30	30
G	8000	5000		5000	5000
Strategy	6	7		6	7
Range	[-1, Inf]	[-1, Inf]		[-1, Inf]	[-1, Inf]
b_1	-0.9	-0.9		-1.0	-1.0
b_2	1.7	1.7		1.6	1.6
b_3	3.5	3.5		3.9	3.9
b_4	4.3	4.3		4.2	4.2
b_5	0.5	0.5		0.6	0.6
b_6	-	-		-	-
b_7	8.7	8.7		8.4	8.4
b_8	0.0	0.0		0.0	0.0
	W_1			W_2	
Value	14.2	14.2		459.3	459.3

Table 60: The percentile error distribution among the components.

F	A_1	A_2	A_3	A_4	A_5	A_6	***Mean error***
W_1	0	40.0327	0	0	45.2216	0	14.2
W_2	10.2616	21.1635	12.1453	4.6118	42.6441	14.6751	17.6

(continues on next page)

(continuation of Appendix 3)

a_7 **excluded**

Table 61: The values of the parameters (coefficients).

Parameter	***Run I***	***Run II***		***Run III***	***Run IV***
F	0.8	0.8		0.8	0.8
CR	0.8	0.8		0.8	0.8
NP	30	30		30	30
G	10000	5000		5000	5000
Strategy	6	7		6	7
Range	[-1, Inf]	[-1, Inf]		[-1, Inf]	[-1, Inf]
b_1	-1.0	-1.0		-1.0	-1.0
b_2	1.6	1.6		1.6	1.6
b_3	3.6	3.6		3.2	3.2
b_4	3.9	3.9		4.2	4.2
b_5	1.2	1.2		1.2	1.2
b_6	2.0	2.0		2.1	2.1
b_7	-	-		-	-
b_8	0.0	0.0		0.0	0.0
	W_1			W_2	
Value	11.3	11.3		708.6	708.6

Table 62: The percentile error distribution among the components.

F	A_1	A_2	A_3	A_4	A_5	A_6	***Mean error***
W_1	0	0	0	0	0	67.7632	11.3
W_2	6.1765	2.9878	12.8720	9.2243	1.7786	62.8525	16.0

Appendix 4. Server Application, Non-Linear Model, Effort Not Corrected

a_2 excluded

Table 63: The values of the parameters (coefficients).

Parameter	***Run I***	***Run II***		***Run III***	***Run IV***
F	0.8	0.8		0.8	0.8
CR	0.8	0.8		0.8	0.8
NP	30	30		30	30
G	8000	8000		5000	5000
Strategy	6	7		6	7
Range	[0, Inf]	[0, Inf]		[0, Inf]	[0, Inf]
b_1	1.2	1.2		1.6	1.6
b_2	-	-		-	-
b_3	4.0	4.0		4.2	4.2
b_4	3.4	3.4		3.7	3.7
b_5	1.0	1.0		1.3	1.3
b_6	2.3	2.3		2.2	2.2
b_7	8.8	8.8		8.4	8.4
b_8	0.0	0.0		0.0	0.0
	W_1			W_2	
Value	18.7	18.7		753.0	753.0

Table 64: The percentile error distribution among the components.

F	A_1	A_2	A_3	A_4	A_5	A_6	***Mean error***
W_1 *(strat. 6)*	0	0	53.1194	59.3000	0	0	18.7
W_2	28.2105	14.4666	41.6121	39.7302	5.3180	13.2012	23.8

(continues on next page)

(continuation of Appendix 4)

a_3 **excluded**

Table 65: The values of the parameters (coefficients).

Parameter	***Run I***	***Run II***		***Run III***	***Run IV***
F	0.8	0.8		0.8	0.8
CR	0.8	0.8		0.8	0.8
NP	30	30		30	30
G	10000	10000		5000	5000
Strategy	6	7		6	7
Range	[-1, Inf]	[-1, Inf]		[-1, Inf]	[-1, Inf]
b_1	-0.3	-0.3		-0.4	-0.4
b_2	1.5	1.5		1.6	1.6
b_3	-	-		-	-
b_4	4.2	4.2		4.2	4.2
b_5	0.2	0.2		0.2	0.2
b_6	2.2	2.2		2.2	2.2
b_7	8.5	8.5		7.8	7.8
b_8	0.0	0.0		0.0	0.0
	W_1			W_2	
Value	14.5	14.5		502.9	502.9

Table 66: The percentile error distribution among the components.

F	A_1	A_2	A_3	A_4	A_5	A_6	***Mean error***
W_1	0	36.2030	0	50.7872	0	0	14.5
W_2	0.8828	27.3659	6.5020	39.9497	2.4157	24.9722	17.0

(continues on next page)

(continuation of Appendix 4)

a_4 excluded

Table 67: The values of the parameters (coefficients).

Parameter	***Run I***	***Run II***		***Run III***	***Run IV***
F	0.8	0.8		0.8	0.8
CR	0.8	0.8		0.8	0.8
NP	30	30		30	30
G	10000	10000		5000	5000
Strategy	6	7		6	7
Range	[-1, Inf]	[-1, Inf]		[-1, Inf]	[-1, Inf]
b_1	-0.2	-0.2		0.0	0.0
b_2	1.5	1.5		1.5	1.5
b_3	4.3	4.3		4.2	4.2
b_4	-	-		-	-
b_5	0.2	0.2		2.2	2.2
b_6	2.1	2.1		0.3	0.3
b_7	8.9	8.9		8.5	8.5
b_8	0.0	0.0		0.0	0.0
	W_1			W_2	
Value	16.2	16.2		563.9	563.9

Table 68: The percentile error distribution among the components.

F	A_1	A_2	A_3	A_4	A_5	A_6	***Mean error***
W_1	0	46.1980	51.2230	0	0	0	16.2
W_2	0.3963	23.3466	45.0318	8.0426	17.0805	21.3045	19.2

(continues on next page)

(continuation of Appendix 4)

a_6 **excluded**

Table 69: The values of the parameters (coefficients).

Parameter	***Run I***	***Run II***		***Run III***	***Run IV***
F	0.8	0.8		0.8	0.8
CR	0.8	0.8		0.8	0.8
NP	30	30		30	30
G	8000	8000		5000	5000
Strategy	6	7		6	7
Range	[-1, Inf]	[-1, Inf]		[-1, Inf]	[-1, Inf]
b_1	-0.7	-0.7		-1.0	-1.0
b_2	1.7	1.7		1.6	1.6
b_3	3.5	3.5		4.0	4.0
b_4	4.3	4.3		4.2	4.2
b_5	0.4	0.4		0.5	0.5
b_6	-	-		-	-
b_7	8.7	8.7		8.1	8.1
b_8	0.0	0.0		0.0	0.0
	W_1			W_2	
Value	21.6	21.6		754.4	754.4

Table 70: The percentile error distribution among the components.

F	A_1	A_2	A_3	A_4	A_5	A_6	***Mean error***
W_1	0	84.2455	0	0	45.2537	0	21.6
W_2	17.8304	29.1796	20.7801	4.5366	46.8596	26.6272	24.3

(continues on next page)

(continuation of Appendix 4)

a_7 **excluded**

Table 71: The values of the parameters (coefficients).

Parameter	***Run I***	***Run II***		***Run III***	***Run IV***
F	0.8	0.8		0.8	0.8
CR	0.8	0.8		0.8	0.8
NP	30	30		30	30
G	8000	8000		5000	5000
Strategy	6	7		6	7
Range	[-1, Inf]	[-1, Inf]		[-1, Inf]	[-1, Inf]
b_1	-0.2	-0.2		-0.7	-0.7
b_2	1.6	1.6		1.6	1.6
b_3	3.6	3.6		3.4	3.4
b_4	4.0	4.0		4.2	4.2
b_5	0.2	0.2		0.8	0.8
b_6	2.0	2.0		2.1	2.1
b_7	-	-		-	-
b_8	0.0	0.0		0.0	0.0
	W_1			W_2	
Value	14.8	14.8		766.7	766.7

Table 72: The percentile error distribution among the components.

F	A_1	A_2	A_3	A_4	A_5	A_6	***Mean error***
W_1	0	19.6070	0	0	0	69.0231	14.8
W_2	10.3555	9.6516	6.1698	15.1335	1.8709	64.2596	17.9

Appendix 5. Client Application, Linear Model, Effort Corrected

a_3 excluded

Table 73: The values of the parameters (coefficients).

Parameter	***Run I***	***Run II***		***Run III***	***Run IV***
F	0.8	0.8		0.8	0.8
CR	0.8	0.8		0.8	0.8
NP	30	30		30	30
G	5000	5000		5000	5000
Strategy	6	7		6	7
Range	[0, Inf]	[0, Inf]		[0, Inf]	[0, Inf]
b_1	0.0	0.0		0.0	0.0
b_2	0.0	0.0		0.0	0.0
b_3	-	-		-	-
b_4	202.5	202.5		229.7	229.7
b_5	10.5	10.5		1.5	1.5
b_6	31.2	31.2		28.0	28.0
b_7	0.0	0.0		0.0	0.0
b_8	602.6	602.6		535.2	535.2
	W_1			W_2	
Value	8.1	8.1		133.4	133.4

Table 74: The percentile error distribution among the components.

F	B_1	B_2	B_3	B_4	B_5	B_6	***Mean error***
W_1	0	0	19.3675	0	29.2722	0	8.1
W_2	5.7415	2.7614	15.8516	7.1404	16.8566	13.1696	10.3

(continues on next page)

(continuation of Appendix 5)

a_4 excluded

Table 75: The values of the parameters (coefficients).

Parameter	***Run I***	***Run II***		***Run III***	***Run IV***
F	0.8	0.8		0.8	0.8
CR	0.8	0.8		0.8	0.8
NP	30	30		30	30
G	5000	5000		5000	5000
Strategy	6	7		6	7
Range	[0, Inf]	[0, Inf]		[0, Inf]	[0, Inf]
b_1	0.0	0.0		0.0	0.0
b_2	0.0	0.0		0.0	0.0
b_3	107.5	107.5		65.9	65.9
b_4	-	-		-	-
b_5	0.0	0.0		37.8	7.8
b_6	24.4	24.4		7.8	37.8
b_7	382.5	382.5		216.5	216.5
b_8	0.0	0.0		0.0	0.0
	W_1			W_2	
Value	21.8	21.8		991.5	991.5

Table 76: The percentile error distribution among the components.

F	B_1	B_2	B_3	B_4	B_5	B_6	***Mean error***
W_1	0	0	15.6191	20.0030	95.2935	0	21.8
W_2	4.9170	26.8500	27.9962	33.7914	50.7461	26.5154	28.5

(continues on next page)

(continuation of Appendix 5)

a_5 **excluded**

Table 77: The values of the parameters (coefficients).

Parameter	***Run I***	***Run II***		***Run III***	***Run IV***
F	0.8	0.8		0.8	0.8
CR	0.8	0.8		0.8	0.8
NP	30	30		30	30
G	5000	5000		5000	5000
Strategy	6	7		6	7
Range	[0, Inf]	[0, Inf]		[0, Inf]	[0, Inf]
b_1	0.0	0.0		0.0	0.0
b_2	0.0	0.0		0.0	0.0
b_3	141.2	141.2		132.3	132.3
b_4	239.9	239.9		228.9	228.9
b_5	-	-		-	-
b_6	15.7	15.7		15.6	15.6
b_7	0.0	0.0		0.0	0.0
b_8	0.0	0.0		0.0	0.0
	W_1			W_2	
Value	6.7	6.7		73.6	73.6

Table 78: The percentile error distribution among the components.

F	B_1	B_2	B_3	B_4	B_5	B_6	***Mean error***
W_1	0	0	8.0245	17.5600	14.3262	0	6.7
W_2	3.9365	4.3136	11.9359	12.3916	8.9702	5.5946	7.9

(continues on next page)

(continuation of Appendix 5)

a_6 **excluded**

Table 79: The values of the parameters (coefficients).

Parameter	***Run I***	***Run II***		***Run III***	***Run IV***
F	0.8	0.8		0.8	0.8
CR	0.8	0.8		0.8	0.8
NP	30	30		30	30
G	5000	5000		5000	5000
Strategy	6	7		6	7
Range	[0, Inf]	[0, Inf]		[0, Inf]	[0, Inf]
b_1	0.0	0.0		0.0	0.0
b_2	0.0	0.0		0.0	0.0
b_3	127.7	127.7		148.6	148.6
b_4	228.7	228.7		263.0	263.0
b_5	12.6	12.6		4.7	4.7
b_6	-	-		-	-
b_7	12.2	12.2		0.0	0.0
b_8	0.0	0.0		0.0	0.0
	W_1			W_2	
Value	8.5	8.5		154.4	154.4

Table 80: The percentile error distribution among the components.

F	B_1	B_2	B_3	B_4	B_5	B_6	***Mean error***
W_1	24.5228	0	26.4422	0	0	0	8.5
W_2	19.7720	4.8671	16.9280	13.5010	3.5193	5.5466	10.7

Appendix 6. Client Application, Linear Model, Effort Not Corrected

a_3 excluded

Table 81: The values of the parameters (coefficients).

Parameter	***Run I***	***Run II***		***Run III***	***Run IV***
F	0.8	0.8		0.8	0.8
CR	0.8	0.8		0.8	0.8
NP	30	30		30	30
G	5000	5000		5000	5000
Strategy	6	7		6	7
Range	[0, Inf]	[0, Inf]		[0, Inf]	[0, Inf]
b_1	0.0	0.0		0.0	0.0
b_2	0.0	0.0		0.0	0.0
b_3	-	-		-	-
b_4	232.8	232.8		291.3	291.3
b_5	12.0	12.0		11.6	11.6
b_6	35.9	35.9		16.6	16.6
b_7	0.0	0.0		0.0	0.0
b_8	0.0	0.0		281.9	281.9
	W_1			W_2	
Value	17.3	17.3		564.8	564.8

Table 82: The percentile error distribution among the components.

F	B_1	B_2	B_3	B_4	B_5	B_6	***Mean error***
W_1	0	0	19.3685	0	23.8882	60.6757	17.3
W_2	20.7163	11.5816	23.3983	4.2251	30.7105	36.2899	21.2

(continues on next page)

(continuation of Appendix 6)

a_4 **excluded**

Table 83: The values of the parameters (coefficients).

Parameter	***Run I***	***Run II***		***Run III***	***Run IV***
F	0.8	0.8		0.8	0.8
CR	0.8	0.8		0.8	0.8
NP	30	30		30	30
G	5000	5000		5000	5000
Strategy	6	7		6	7
Range	[0, Inf]	[0, Inf]		[0, Inf]	[0, Inf]
b_1	0.0	0.0		0.0	0.0
b_2	0.0	0.0		0.0	0.0
b_3	123.5	123.5		9.6	9.6
b_4	-	-		-	-
b_5	28.1	28.1		42.4	42.4
b_6	439.8	439.8		53.5	53.5
b_7	0.0	0.0		34.3	34.3
b_8	0.0	0.0		0.0	0.0
	W_1			W_2	
Value	38.5	38.5		2180.2	2180.2

Table 84: The percentile error distribution among the components.

F	B_1	B_2	B_3	B_4	B_5	B_6	***Mean error***
W_1	0	0	15.6291	20.0070	195.4682	0	21.8
W_2	8.5213	45.8583	49.4714	55.9541	57.2388	45.2866	28.5

(continues on next page)

(continuation of Appendix 6)

a_5 **excluded**

Table 85: The values of the parameters (coefficients).

Parameter	***Run I***	***Run II***		***Run III***	***Run IV***
F	0.8	0.8		0.8	0.8
CR	0.8	0.8		0.8	0.8
NP	30	30		30	30
G	5000	5000		5000	5000
Strategy	6	7		6	7
Range	[0, Inf]	[0, Inf]		[0, Inf]	[0, Inf]
b_1	0.0	0.0		N/A	N/A
b_2	0.0	0.0		N/A	N/A
b_3	162.3	162.3		N/A	N/A
b_4	275.9	275.9		N/A	N/A
b_5	-	-		N/A	N/A
b_6	18.1	18.1		N/A	N/A
b_7	0.0	0.0		N/A	N/A
b_8	0.0	0.0		N/A	N/A
	W_1			W_2	
Value	16.4	16.4		N/A	N/A

Table 86: The percentile error distribution among the components.

F	B_1	B_2	B_3	B_4	B_5	B_6	***Mean error***
W_1	0	0	8.0378	17.5436	72.9846	0	16.4
W_2	N/A	N/A	N/A	N/A	N/A	N/A	N/A

(continues on next page)

(continuation of Appendix 6)

a_6 **excluded**

Table 87: The values of the parameters (coefficients).

Parameter	***Run I***	***Run II***		***Run III***	***Run IV***
F	0.8	0.8		0.8	0.8
CR	0.8	0.8		0.8	0.8
NP	30	30		30	30
G	5000	5000		5000	5000
Strategy	6	7		6	7
Range	[0, Inf]	[0, Inf]		[0, Inf]	[0, Inf]
b_1	0.0	0.0		0.0	0.0
b_2	0.0	0.0		0.0	0.0
b_3	145.3	145.3		121.8	121.8
b_4	268.2	268.2		311.5	311.5
b_5	16.3	16.3		0.0	0.0
b_6	-	-		-	-
b_7	0.0	0.0		0.0	0.0
b_8	0.0	0.0		0.0	0.0
	W_1			W_2	
Value	16.5	16.5		421.9	421.9

Table 88: The percentile error distribution among the components.

F	B_1	B_2	B_3	B_4	B_5	B_6	***Mean error***
W_1	24.2171	0	27.0061	0	47.6242	0	16.5
W_2	29.0080	4.6630	20.2671	11.1771	27.5064	19.3837	18.7

Appendix 7. Client Application, Non-Linear Model, Effort Corrected

a_3 excluded

Table 89: The values of the parameters (coefficients).

Parameter	***Run I***	***Run II***		***Run III***	***Run IV***
F	0.8	0.8		0.8	0.8
CR	0.8	0.8		0.8	0.8
NP	30	30		30	30
G	8000	5000		5000	5000
Strategy	6	7		6	7
Range	[-1, Inf]	[-1, Inf]		[-1, Inf]	[-1, Inf]
b_1	0.1	0.1		0.5	0.5
b_2	0.3	0.3		0.5	0.5
b_3	-	-		-	-
b_4	4.2	4.2		4.1	4.1
b_5	2.7	2.7		2.9	2.9
b_6	2.3	2.3		2.3	2.3
b_7	-1.0	-1.0		-1.0	-1.0
b_8	N/A	N/A		N/A	N/A
	W_1			W_2	
Value	36.0	36.0		2457.3	2457.3

Table 90: The percentile error distribution among the components.

F	B_1	B_2	B_3	B_4	B_5	B_6	***Mean error***
W_1	0	0	82.3097	76.8826	0	56.7661	36.0
W_2	2.5970	11.2352	82.7005	77.6684	22.9517	34.8172	39.0

(continues on next page)

(continuation of Appendix 7)

a_4 excluded

Table 91: The values of the parameters (coefficients).

Parameter	***Run I***	***Run II***		***Run III***	***Run IV***
F	0.8	0.8		0.8	0.8
CR	0.8	0.8		0.8	0.8
NP	30	30		30	30
G	8000	5000		5000	5000
Strategy	6	7		6	7
Range	[-1, Inf]	[-1, Inf]		[-1, Inf]	[-1, Inf]
b_1	0.5	0.5		0.8	0.8
b_2	0.3	0.3		0.4	0.4
b_3	0.0	0.0		0.0	0.0
b_4	-	-		-	-
b_5	3.1	3.1		3.0	3.0
b_6	2.3	2.3		2.3	2.3
b_7	-0.9	-0.9		-1.0	-1.0
b_8	-	-		N/A	N/A
	W_1			W_2	
Value	41.7	41.7		3007.0	3007.0

Table 92: The percentile error distribution among the components.

F	B_1	B_2	B_3	B_4	B_5	B_6	***Mean error***
W_1	0	19.8120	85.8592	89.4836	55.1893	0	41.7
W_2	2.8857	32.3825	85.7485	89.3131	37.1924	16.4916	44.0

(continues on next page)

(continuation of Appendix 7)

a_6 **excluded**

Table 93: The values of the parameters (coefficients).

Parameter	***Run I***	***Run II***		***Run III***	***Run IV***
F	0.8	0.8		0.8	0.8
CR	0.8	0.8		0.8	0.8
NP	30	30		30	30
G	8000	5000		5000	5000
Strategy	6	7		6	7
Range	[-1, Inf]	[-1, Inf]		[-1, Inf]	[-1, Inf]
b_1	0.3	0.3		0.6	0.6
b_2	0.8	0.8		1.1	1.1
b_3	3.7	3.7		3.1	3.1
b_4	4.0	4.0		4.1	4.1
b_5	2.9	2.9		3.0	3.0
b_6	-	-		-	-
b_7	-1.0	-1.0		-1.0	-1.0
b_8	N/A	N/A		N/A	N/A
	W_1			W_2	
Value	44.0	44.0		3824.5	3824.5

Table 94: The percentile error distribution among the components.

F	B_1	B_2	B_3	B_4	B_5	B_6	***Mean error***
W_1	78.2018	0	97.3049	88.7116	0	0	44.0
W_2	73.6407	14.2102	97.1531	87.8355	11.0418	6.8284	48.5

Appendix 8. Client Application, Non-Linear Model, Effort Not Corrected

a_3 excluded

Table 95: The values of the parameters (coefficients).

Parameter	***Run I***	***Run II***		***Run III***	***Run IV***
F	0.8	0.8		0.8	0.8
CR	0.8	0.8		0.8	0.8
NP	30	30		30	30
G	5000	5000		5000	5000
Strategy	6	7		6	7
Range	[-1, Inf]	[-1, Inf]		[-1, Inf]	[-1, Inf]
b_1	-0.1	-0.1		0.2	0.2
b_2	0.2	0.2		0.4	0.4
b_3	-	-		-	-
b_4	4.5	4.5		4.4	4.4
b_5	1.6	1.6		2.8	2.8
b_6	2.4	2.4		2.3	2.3
b_7	-1.0	-1.0		-1.0	-1.0
b_8	N/A	N/A		N/A	N/A
	W_1			W_2	
Value	41.8	41.8		3035.9	3035.9

Table 96: The percentile error distribution among the components.

F	B_1	B_2	B_3	B_4	B_5	B_6	***Mean error***
W_1	0	0	81.4518	73.7291	0	95.5169	41.8
W_2	14.5527	9.7016	84.6191	77.5425	33.8364	59.9281	46.7

(continues on next page)

(continuation of Appendix 8)

a_4 **excluded**

Table 97: The values of the parameters (coefficients).

Parameter	***Run I***	***Run II***		***Run III***	***Run IV***
F	0.8	0.8		0.8	0.8
CR	0.8	0.8		0.8	0.8
NP	30	30		30	30
G	5000	5000		5000	5000
Strategy	6	7		6	7
Range	[-1, Inf]	[-1, Inf]		[-1, Inf]	[-1, Inf]
b_1	0.2	0.2		0.4	0.4
b_2	0.2	0.2		0.3	0.3
b_3	0.0	0.0		0.0	0.0
b_4	-	-		-	-
b_5	3.2	3.2		3.0	3.0
b_6	2.3	2.3		2.2	2.2
b_7	-1.0	-1.0		-1.0	-1.0
b_8	N/A	N/A		N/A	N/A
	W_1			W_2	
Value	55.4	55.4		3963.3	3963.3

Table 98: The percentile error distribution among the components.

F	B_1	B_2	B_3	B_4	B_5	B_6	***Mean error***
W_1	0	19.9997	86.3950	90.0067	136.2008	0	55.4
W_2	25.7546	51.6427	88.5774	91.4532	50.9314	40.5686	58.2

(continues on next page)

(continuation of Appendix 8)

a_5 **excluded**

Table 99: The values of the parameters (coefficients).

Parameter	***Run I***	***Run II***		***Run III***	***Run IV***
F	0.8	0.8		0.8	0.8
CR	0.8	0.8		0.8	0.8
NP	30	30		30	30
G	5000	5000		5000	5000
Strategy	6	7		6	7
Range	[-1, Inf]	[-1, Inf]		[-1, Inf]	[-1, Inf]
b_1	-0.2	-0.2		N/A	N/A
b_2	0.2	0.2		N/A	N/A
b_3	2.1	2.1		N/A	N/A
b_4	4.5	4.5		N/A	N/A
b_5	-	-		N/A	N/A
b_6	2.4	2.4		N/A	N/A
b_7	-1.0	-1.0		N/A	N/A
b_8	N/A	N/A		N/A	N/A
	W_1			W_2	
Value	41.7	41.7		N/A	N/A

Table 100: The percentile error distribution among the components.

F	B_1	B_2	B_3	B_4	B_5	B_6	***Mean error***
W_1	0	0	81.2056	73.3304	0	95.8046	41.7
W_2	N/A	N/A	N/A	N/A	N/A	N/A	N/A

(continues on next page)

(continuation of Appendix 8)

a_6 **excluded**

Table 101: Values of the parameters (coefficients).

Parameter	***Run I***	***Run II***		***Run III***	***Run IV***
F	0.8	0.8		0.8	0.8
CR	0.8	0.8		0.8	0.8
NP	30	30		30	30
G	5000	5000		5000	5000
Strategy	6	7		6	7
Range	[-1, Inf]	[-1, Inf]		[-1, Inf]	[-1, Inf]
b_1	-0.2	-0.2		0.3	0.3
b_2	0.3	0.3		0.7	0.7
b_3	4.2	4.2		3.6	3.6
b_4	4.5	4.5		4.4	4.4
b_5	1.8	1.8		2.9	2.9
b_6	-	-		-	-
b_7	-1.0	-1.0		-1.0	-1.0
b_8	N/A	N/A		N/A	N/A
	W_1			W_2	
Value	49.0	49.0		4107.9	4107.9

Table 102: The percentile error distribution among the components.

F	B_1	B_2	B_3	B_4	B_5	B_6	***Mean error***
W_1	94.1965	0	97.0863	84.4919	0	18.3093	49.0
W_2	82.2240	11.8383	97.2500	86.6185	18.6994	20.8926	52.9