Architecture framework in support of effort estimation of legacy systems modernization towards a SOA environment

Towards a SOA Modernization Impact Analysis

Final Version

Zdravko Anguelov

Architecture framework in support of effort estimation of legacy systems modernization towards a SOA environment

THESIS

submitted in partial fulfillment of the
requirements for the degree of

MASTER OF SCIENCE

in

COMPUTER SCIENCE

by

Zdravko Anguelov

Software Engineering Research Group
Department of Software Technology
Faculty EEMCS, Delft University of Technology
Delft, the Netherlands
www.ewi.tudelft.nl

Global Business Services
IBM Nederland B.V.
Johan Huizingalaan 765
Amsterdam, The Netherlands
www.ibm.com

Architecture framework in support of effort estimation of legacy systems modernization towards a SOA environment

Author: Zdravko Anguelov
Student id: 1232436
Email: angelov.zdravko@nl.ibm.com

Abstract

Because of their poor Business/IT alignment, many legacy systems lack the flexibility to support rapid changes to the business processes they implement, required by today's enterprises. Furthermore, after many years of maintenance, there is a need to manage their resulting increased complexity and maximize asset utilization through reuse. The third complicating circumstance is that these legacy systems cannot simply be replaced as it is too expensive and risky. For these three reasons, legacy systems are modernized towards a Service Oriented Architecture.

This thesis presents a framework for performing an impact analysis of such a modernization. It supports the trade-off analysis, needed in the planning phase, for finding the optimal selection of modernization strategies and judging their yield. The impact is expressed through the estimation of, on the one side, the effort and, on the other side, the gain of the changes these modernization strategies entail. The thesis concentrates on one of the many types of changes in modernization – the architectural and design changes to the software system.

The presented framework structures current approaches to modernization in a set of class definitions, system model relationships and a process description. This is done according to the effort they produce, preparing them for its estimation. For this effort estimation, this thesis introduces a Rating Model for quantifying the modernization effort using the system models of the framework. This quantification is done through the identification of so-called Points of Modernization, a categorization of the modernization strategies and a set of effort indicator metrics.

Based on this framework, this thesis also presents an experiment. For a subject legacy system, concrete approaches are shown for the instantiation of the framework models and the subsequent effort estimation is done using the indicator of Scattering.

The analysis of the resulting effort and its relation to the gain show the optimal solutions for the modernization of the subject system. Concluding, this thesis discusses the feasibility of the approach and the future work such as more quantitative research on the rest of the effort indicators.

Thesis Committee:

Chair:	Prof. Dr. A. van Deursen, Faculty EEMCS, TU Delft
University supervisor:	Dr. Hans-Gerhard Gross, Faculty EEMCS, TU Delft
Company supervisor:	Dr. Raymond van Diessen, IBM
Committee Member:	Ir. Bernard Sodoyer, Faculty EEMCS, TU Delft

Preface

This thesis is the final work of my Masters study Computer Science at the Delft University of Technology. It was conducted between December 2008 and December 2009 in collaboration with IBM Amsterdam. The subject domain of the research is the modernization of legacy systems towards a Service-Oriented environment. The existing modernization strategies in this domain are analyzed for the changes they require to the architecture and design of legacy systems. This has served as basis for the construction of a framework for evaluating the overall impact of legacy system modernization and in particular the estimation of effort required for performing the modernization strategies.

The contribution of this research is the organization of the modernization changes in such a way that their effort impact can be estimated following a suggested process. Next to this theoretical contribution, the thesis suggests concrete automated approaches for the instantiation of the architectural models prescribed in the framework. These approaches are applied for the purpose of effort estimation on a contemplated modernization of a subject legacy system. The outcome of this case study is analyzed in the context of impact analysis and trade-off analysis. This work concludes with an evaluation of the presented framework and with the outlines for further development of this field through future work.

The reason for performing this research is the need in IBM for a quantitative model of the origin and propagation of impact for the domain of SOA modernization. In particular, an estimation is needed of the formation of effort due to modernization. Such a model would have the predictive power needed for a trade-off analysis of future modernization initiatives in enterprises.

This thesis had two complementing goals on the road to the needed impact analysis. In the first place, to do a broad investigation of the field of modernization towards SOA and lay the grounds for impact analysis, which encompasses amongst others the estimation of effort, gain and risk of the modernization. To do this, the important issues had to be identified that have impact on the modernization process and results. After which, an inventory had to be made of the existing approaches to modernization considering the issues they belong to and the way they produce effort. As there was no such previous work bringing together these multiple domains, the literature study phase of the research was a real challenge.

On the other hand, this thesis also had as a goal to contribute quantitative results to the field. For this purpose, we have concentrated on the issue of effort estimation. In this field,

we investigate into detail a concrete instance of legacy system modernization and evaluate the effort estimation results in the context of the broad goal of SOA modernization. Many of the issues in constructing the framework were cleared through the literature research. However the experimental part helped me to get a more clear view on issues and make the whole more precise. I hope that I have been able to find the balance between the two goals and produce useful results to both theory and practice.

Finally, I would like to express my gratitude and give thanks to everybody that made this thesis possible. In the first place, these are my supervisors Raymond van Diessen from IBM and Hans-Gerhard Gross from the Software Evolution Research Lab at TU Delft. I thank them for their support and feedback throughout all the research phases. I would also like to thank my people-managers at IBM, Jochem Harteveld and Schelto van Heemstra, who welcomed me into the Business Application Modernization team and helped me feel at home there. Of course, I would also like to thank all the team members themselves for having me and especially David Stern who helped me with the technical realization of the experiment of this thesis. For this, I would also like to thank Marcel Wulterkens and his team at IBM for making the experiment possible with their readiness to provide me with the code of the subject legacy system. And finally the many others at IBM I talked to about projects and issues, which opened my eyes also for the practical issues and considerations in real modernization projects.

Zdravko Anguelov
Delft, the Netherlands
January 20, 2010

Contents

List of Figures

Listings

Chapter 1

Problem Statement

Many enterprises have legacy systems that are up to several decades old. Such systems are characterized by the fact that they have been changed and enhanced, through maintenance, during the long period of time of their existence. The reasons for these changes, are changes in the business requirements(environment), which have to be reflected in the supporting IT system. This continuous evolution, with 5 to 7% functionality modification a year [51], has made these systems large, complex and heterogeneous. Due to the inevitable maintenance, the system's design has deteriorated, which makes further changes even more difficult and less isolated (with greater impact on the rest of the system). Such legacy systems have become essential to the operation of the enterprise and cannot just be discarded and rebuilt from scratch. This is often too risky (functionality may get lost, too complex functionality, undocumented functionality), too expensive and takes too much time.

At the same time, enterprises have goals that require changes of their systems to be made, ranging from technology driven maintenance to the incorporation of new business requirements. Two main goals are: 1) becoming agile and 2) reducing the cost of technology. Enterprises want to become more competitive by being able to react fast to market changes [30]. This means that it must be possible for them to change business processes fast and thus reduce the time-to-market of their products. Reducing the cost of technology can be achieved by improving system interoperability and increasing reuse. Interoperability needs to be improved internally, between the enterprise's own applications, but also externally, with other parties such as suppliers and customers. There is thus a need for creating a dynamic and easily reconfigurable IT environment, which corresponds to the dynamic business environment of today. Furthermore, increasing reuse will increase efficiency and give the enterprise a competitive edge on the market. All the above-mentioned characteristics, that will improve the agility of the system and reduce the cost of technology, are not easily achievable in most legacy systems, because of the systems' rigid design and heterogeneity.

Legacy System Modernization

This description of the current state of legacy systems and of the desired business organization exposes a mismatch. This is the mismatch between modern business requirements and

the supporting legacy software systems. It is manifested through the fact that the structure of legacy IT systems is not optimal for a good IT-to-Business alignment. At the same time, industry case studies and research indicate that an architectural change of legacy systems, towards Service Oriented Architecture, can remedy this problem [8, 74, 66]. Thus, so far we have established two facts:

1. *Legacy systems do not satisfy the business needs of the enterprises using them:* They are not easy to change at the pace of change of the business processes they support. They are also not economically efficient as there is little reuse possible due to their design.
2. *SOA addresses many of theses issues:* SOA-based systems offer a good solution for supporting a fast changing environment, because of better IT-to-Business alignment. SOA also structures the system in such a way that reuse is much easier.

Therefore, it is obvious that the need exists for achieving the potential solution(2.) to the problem(1.). The process to do that is a special case of the general approach from the field of system evolution, described in section 2.3. The fundamental problem in this area is, thus, concerned with the evolution of an as-is legacy system to a to-be system (Figure 1.1), where the to-be system has to comply with the architectural principles of Service-Orientation (section 2.2).

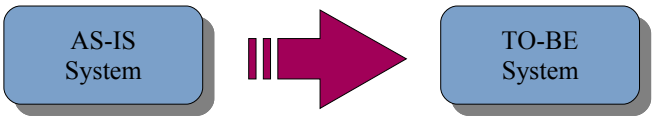


Figure 1.1: Simple model of system modernization

Impact Analysis and Effort Estimation

The above-mentioned modernization is carried out according to modernization strategies that can have far reaching consequences [8, 56, 60, 23]. The modernization impact can vary significantly in its effort, cost and duration, depending on the extensiveness of the changes needed to reach the target system organization. This means that there is a significant investment involved for the enterprise with a high risk of loss of legacy functionality, performance etc. [91, 27]. An enterprise will find the investment needed, justified only up to a certain level, depending on the benefits it offers. Therefore, a feasibility analysis of such modernization is needed prior to its execution.

Such a feasibility analysis has to weight the positive against the negative impacts of such a modernization project. In its most basic form, these would be the benefits against the needed effort. Therefore, an important prerequisite to impact analysis is the establishing of an estimation mechanism for the effort involved. Furthermore, since it is only a preliminary analysis, it must not require much resources itself to perform, which means relying heavily on automation and at hand resources such as source code. For these reasons an automatable estimation mechanism is most suitable.

Research Question Formulation

Modernization towards SOA has many aspects, each having some form of impact on the eventual effort needed. Hence the subtitle of this thesis – "Towards SOA modernization impact analysis", as we will concentrate only on part of the issues relevant to a complete impact analysis . As an initial division, the literature study has identified the organizational, business and technology (information and systems) aspects. From these, the aspect we are most interested in is the technology and specifically the software architecture organization. Concentrating on the software architecture description of the as-is and to-be systems is a good choice as an indicator of the changes the modernization will require for three reasons.

In the first place, the architectural view is an abstraction from the "how?" [53, p.42]. Thus an approach based on it will give a more generic approach, applicable to a broader set of subject systems and will not be influenced by low level implementation dependent differences.

Second, as is shown in section 2.2 on Service Oriented Architecture, many of the decisions in such systems are done on a high-abstraction level – architectural and design level. There they are more easily relatable to the business domain. The software architectural level is, thus, where the link can be more easily made between rather different system organizations such as legacy and SOA.

Finally, architectural views have as an advantage that they usually can be derived in an automated way from the legacy source code. Finding an automated approach is of more interest compared to expert-driven approaches, because legacy systems display two characteristics, which can be most efficiently handled through automation:

1. *size and complexity:* due to the size and complexity of legacy systems the sought after approach by the research has to be repeatable and be able to handle a large volume of input information. This will make it applicable to projects in different contexts, based on reuse with the same initial development investment.
2. *poor understanding and documentation:* due to the limited availability of other system design sources, the approach has to mainly rely on hard assets such as source code and deployment configurations.

So in summary, this research is focused on how to use this architectural information about a legacy system to analyze the impact of modernizing towards an Service Oriented Architecture by estimating the needed effort. This is formulated as the following research question:

> **The modernization process of legacy systems to a SOA environment imposes system architectural/design changes. How to perform an automated overall (software) change impact analysis for these changes?**

Thesis Research Goals

The literature study has identified the following existing work relevant to the research problem described above:

- generic modernization frameworks such as ADM, RMM, Renaissance and SMART (automated vs expert-driven)
- legacy system portfolio analysis for identification and prioritization of sub-system modernization needs
- SOA target environment specification guidelines
- set of modernization strategies on the architectural level

From the survey of current approaches (section 2.3), it becomes apparent there is no method that binds them together. Such a method would make it possible to relate the application of the known modernization strategies in the context of legacy system modernization towards SOA. In doing so, such a methodology can be used to evaluate the impact and extent of the architectural changes that are the consequence of the modernization strategies. This would give the basis for performing a SOA change impact analysis and modernization effort estimation. So the broad setting for the problem stated above is the identification and targeting of the legacy system weak points, specification of target SOA architecture, and application of SOA modernization strategies as part of a broader process of system modernization. Thus, the sub-problems this thesis research focuses on are:

- Which selection of elementary modernization changes on architectural/design level should be used to describe the SOA modernization strategies? These strategies as defined in section sec:strategies can be described through the software architecture modeling entities described in section sec:architecture. A selection of these elementary changes will form the basis for describing the modernization plan towards SOA of the legacy system.
- How to organize these modernization changes? Are there groups of related changes? Is there impact propagation (between groups) based on these relationships?
- How to relate the changes in this organization to each other, so that the total impact of the modernization can be quantified? What scale of measurement is most appropriate and feasible to rate this impact–nominal, ordinal, interval or ratio scale?

This report began in this chapter with a clear statement of the research problem to be solved. The purpose of the rest of the report is to describe the findings of the thesis research. In chapter 2, a selection of existing approaches is described that were identified during the literature study and that together form the context for the further thesis research. These include software architecture modeling concepts, SOA specification concepts and modernization strategies. Chapter 3 describes the concrete approach suggested for tackling the problem stated in the research question. It consists of a framework specification together with its scope and the assumptions made. This approach is then applied in practice. The experimental setting for that and the results are presented in chapter 4. The chapter consists of the description of a case study legacy system and a concrete implementation of the suggested framework used to analyze it. In the last section of that chapter the results of the experiment are analyzed. In chapter 5 the whole research is evaluated as to whether it can be considered successful and what its added value is. Finally in chapter 6 the conclusions of the research as a whole and the possibilities for future work are presented.

Chapter 2

Theoretical context

This chapter describes the theoretical context of the research problem. It summarizes the fields of research relevant to the problem statement. These established facts, theories and notions will function as premises for the research. They form the theoretical context that will be extensively referenced later on. It is needed in order to make a clear distinction between background knowledge, which will not be put in question throughout the rest of the research, and the sought after knowledge. This will help in the identification of the accomplishments of the performed research. The purpose of the theoretical framework is also to mark the boundaries of the research project.

The research problem is at the intersection of four research areas: **Software Architecture**, **Service-Oriented Architectures**, **System Evolution**, **Impact Analysis**. They are described in the following sections with the artifacts from these domains and the methods to manipulate them. First is the the domain of Software Architecture in section 2.1, listing the concepts used to describe the structure of software systems. Based on this foundation, section 2.2 describes the characteristics of the specific class of Service-Oriented Systems. In section 2.3 we present the existing approaches for overall system evolution and modernization. We conclude in section 2.4 with field of Impact Analysis and its application in legacy modernization focusing on the specific issues in transforming the architectural aspects of legacy systems towards SOA.

2.1 Software Architecture

Complex systems can benefit from a high-level design, guiding its construction and maintenance [49]. Architecturing involves the specification on a high, abstract level, of a system, its elements and the way they interact with each other.

Definition 2.1 ***Architecture:*** *The fundamental organization of a system, embodied in its components, their relationships to each other and the environment, and the principles governing its design and evolution, IEEE standard P1471–2000 [49]*

Based on this definition, we can conclude that using software architecture is useful for two reasons. First, architectural descriptions manage to stay focused on the fundamental

software system characteristics. This is possible through the use of abstraction and hiding of detail. One such example of filtering out information and concentrating on a particular system aspect is through system *views* [49], such as a **development view**, **process view** etc. [61].

Second, it is possible to distinguish between different levels of abstraction through software architecture. This can be summarized in a hierarchy of structural abstraction paradigms [53], shown in figure C.1. This demonstrates the difference in abstraction level between software architecture, software design and implementation structures. The systems architecture consists of early design decisions, on a higher abstraction level, meant to guide the further more detailed system design. This view has also been adopted by the IEEE [49].

These two characteristics of software architectural descriptions make them very suitable as models for reasoning about system modernization. Through software architecture, a level of abstraction can be found where it is possible to describe different systems or different states of the same system using the same architectural concepts. This enables the comparison of these instances even though they may be implemented very differently. In the following subsections we describe concepts on three levels of abstraction that are used to capture software architectural views – components, patterns and frameworks.

Components and Connectors

One of the main viewpoints, to document the software architecture, is in terms of **components** and **connectors** [36, 2, 53]. These are two concepts on a medium level of abstraction (Figure C.1). There is no exact definition of a component [53, p.30], only a general understanding about its characteristics. A component is considered the basic building block of an application. It is an autonomous, encapsulated computational entity which accomplishes pre-defined tasks through internal computation and external communication. There are no limitations on the granularity of a component. Its size can vary from a GUI button through a collection of classes to a complex application service. Procedures, packages, objects and clusters can all be viewed as instances of the same abstraction - the software component. Components define their contract, to be used for external communication, in the form of interfaces. An **interface** thus defines the services the implementing component supplies to its requesters. Two components can communicate with each other through a connector, that adheres to the communication definitions of the interfaces of both components. A connector is thus the collection of rules and constraints that define the relationship between components [69]. The purpose of this representation is to decouple the functionality of a system in independent entities - components and connectors. This way different compositions of these entities are made possible, through the pre-defined interfaces, and ultimately support reusability. This decoupling makes it also possible to focus on select groups of software components in isolation when designing or analyzing a software architecture.

Definition 2.2 ***Component:*** *an autonomous, encapsulated computational entity which accomplishes pre-defined tasks through internal computation and external communication*

Definition 2.3 ***Connector:*** *the collection of rules and constraints that define the relationship between components through their interfaces*

Component attributes are used to describe the components. Put together, they form a **Component Model** [53, p.113-120]. Sample elements of this model are the specification of the externally visible properties of the component, constraints on these properties and an event model. Based on this component view of a system's architecture and the component models one can reason about the software architecture as a collection of components. Garlan et al. [37] have introduced the term *architectural mismatch* to denote interoperability problems between components due to assumptions about:

- the nature of components: functionality supply, infrastructure, control model, data manipulation
- the nature of connectors: protocols and data models
- the architecture of the assemblies: constraints on interactions
- the runtime construction process: order of instantiation

Further component attributes that can have influence on component ensembles interoperability are given by Bhuta et al. [7]. In their work the authors describe an automated method to evaluate the compatibility of components given their component descriptions. These description consists of four four categories of attributes:

- General attributes
- Interface attributes: control inputs, data inputs, data outputs
- Internal assumption attributes: concurrency, encapsulation, layering, preemption
- Dependency attributes: communication dependency, execution language support

In system modernization, architectural transformations are part of the process (section 2.4). Mismatches between the resulting subsystems and components will have to be overcome, leading to additional effort. In an effort estimation framework, it is thus necessary to be able to estimate the compatibility of resulting subsystems and components. The above-mentioned Component Models offer a quantifying approach for this purpose.

Patterns

A further concept for describing the software architecture of a system is a **pattern**. Hess [45] emphasizes the application of patterns in the modernization of legacy systems. Experience has shown that well designed systems in the same problem domain, display similar architecture [42]. Groups of components and connectors display characteristic structures, that are typical for certain commonly encountered problems. These typical groups of components are instances of the abstract solution to the common problem which is called a pattern. The definition of a pattern we are going to use is given by Buschamann [18] as follows:

Definition 2.4 ***Pattern:*** *A pattern for software architecture describes a particular recurring design problem that arises in specific design contexts, and represents a well-proven generic scheme for its solution. The solution scheme is specified by describing its constituent components, their responsibilities and the ways in which they collaborate.*

Patterns can be found on each of the abstraction levels of figure C.1. Kaisler [53, p.29] and Buschmann [18] make the distinction between architectural patterns, design patterns

and idioms in order of decreasing abstraction. The definition of architectural patterns according to Buschmann [18, p.12] as a means of capturing the overall structuring principles of a system is:

Definition 2.5 ***Architectural pattern:*** *An architectural pattern expresses a fundamental structural organization schema for software systems. It provides a set of predefined subsystems, specifies their responsibilities, and includes rules and guidelines for organizing the relationship between them.*

The most widely spread architectural patterns are first documented by Garlan and Shaw in 1994 [38] who use the term architectural styles. These architectural styles have been also categorized in taxonomies [83, 58], [53, p.218-327] based on similarities and differences in their properties and application. Architectural patterns help achieve a specific global system property, such as the adaptability of the user interface of a system. An overview of the patterns on the design level is given by Gamma et al. [35]. They are categorized as creational, structural and behavioral. The most important four from their work are also known as the Gang Of Four. Design patterns are, as architectural patterns, a proven solution to a common problem in a generic form [53]. They must be instantiated for the particular context they are going to be used in. The difference with architectural patterns is that their solution space is much more fine-grained. The definition given by Buschmann:

Definition 2.6 ***Design Pattern:*** *A design pattern provides a scheme for refining the subsystems or components of a software system, or the relationship between them. It describes a commonly-recurring structure of communicating components that solves a general design problem within a particular context.*

Patterns can be nested in each other to combine their advantages and solve the software engineering problem gradually at different levels. A single system may use multiple architectural styles. They can be combined in multiple ways, for example hierarchically, to form a heterogeneous architecture [38].

Frameworks

Using components and patterns, a **framework** goes even further and specifies a partial realization of a system. Frameworks are above the level of software architecture in the abstraction hierarchy (Figure C.1). The main goal of frameworks is to enable *reuse* of the entire system designs and application structures, found in these lower abstraction levels [53, p.344]. This is done by making only the fundamental choices/assumptions on architectural and design level for a particular domain and packaging these as a ready-to-use infrastructure. This supplies the guidelines for the structure of the class of solutions in the particular domain. At the same time, it leaves open the implementation of the details to each concrete application using the framework. As a result, the framework determines the overall structure of the applications derived from it, but also allows for flexibility and customization.

Definition 2.7 ***Framework:*** *a generic architecture complemented by an extensible set of components [53, p.343]*

The specification of a framework is built up from the following three parts:

- **Class definitions:** collection of concept definitions [53, p.405]
- **Relationships:** set of structural and behavioral relationships between the classes of the framework. Together the classes and relationships capture the structure of a framework [53, p.405]
- **Control Model:** this interactions' specification [53, p.345, Taligent Corp.] "ties the parts of the framework together to provide a skeletal solution for a class of problems" [53, p.410]. Usually this results in an inverted role of control between the application and the infrastructure – the infrastructure calls/invokes application routines

By definition, a framework supports **Instantiation**(of the framework), **Refinement**(for a specific domain) and **Extension**(for a more general domain) [53, p.406-407]. These capabilities and the above-mentioned structure of frameworks offer the following advantages [53]:

- *improved maintainability*, because of the consistent reuse of the framework facilities within each application. But also because of the consistency between applications using the same framework.
- *improved quality and reliability*, because all concrete applications based on a framework use proven design/architecture and best practices.
- *shorter design-time*, because the set of common facilities and solution structures are supplied by the infrastructure of the framework and are reused for each new application.

Frameworks are relevant to this research for three reasons:

I. frameworks complete the set of abstraction concepts for architecture description
II. our definition of Service Oriented Architecture (Definition 2.8) is based on the concept of a framework
III. frameworks are useful for the structuring of our own approach to effort estimation

Knowledge about the presence of a framework in the architectural description of a system is useful for modernization effort estimation. In the first place, just as patterns, frameworks indicate a degree of cohesion between the components they consist of. This cohesion is the source of resistance to change of only part of the structure. Changes due to modernization of only one part, may propagate to the rest of the structure and result in additional effort. Furthermore, frameworks are bound to contain a lot of architectural assumptions leading to interoperability problems between different frameworks: architectural mismatches between the control models, mismatch between the provided and required services by the different frameworks [53, p.410-411].

A framework arrangement is also useful for the specification of the impact analysis and effort estimation suggested in this research. Frameworks are also used outside the domain of software architecture. There, they can specify any generic structure together with the provision of facilities for extension and customization. Also in this broader sense, the advantages and capabilities of using a framework apply. Because this research is just

an initial attempt in the field, we want it to have the advantages of frameworks described above to enable future improvements. The construction of the framework will be done in stages. The set of assumptions will be separated from the set of configuration parameters. The challenge of framework design is in finding the balance between reuse and flexibility. On the one hand, the goal is to increase reusability by packaging components so that they can be reused in as many application domains as possible. But at the same time, an equally important goal is to design architectures that are easily adapted to the requirements of the target application domains, thus, achieving flexibility [53, p.404].

2.2 Service Oriented Architecture

This section gives a description of Service Oriented Architecture (SOA). It describes the features of SOA that distinguish it from the typical legacy system architecture. The modernization process this thesis considers has SOA as the target environment. Achieving every characterizing feature of this target environment impacts the amount of effort required. This is why we need to define what has to be established through modernization in order to quantify the resulting effort.

Enterprise Total Architecture

Businesses need to employ a holistic view of their enterprise in order to be successful. In this view the enterprise consists of four interrelated elements: **business processes**, **people**, **information** and **systems** brought together by a purpose and described in the so-called *Enterprise Total Architecture* [17]. The success of this total architecture is directly related to its ability to enable the achievement of the enterprise purpose. The purpose of the architecture is, thus, to support the business processes that make the enterprise work. This means that understanding these business processes is an essential prerequisite to creating that architecture. **Service Oriented Architecture** is a paradigm on how to arrange the four above-mentioned elements of the Total Architecture of the Enterprise in such a way that the enterprise can reach its goal. Through its guidelines on all four aspects it helps build a coordinated whole, that can evolve and still keep up a good overall quality. Designing a SOA aids, for example, in building an environment that integrates the business process architecture and the software architecture of the enterprise. There are more aspects to designing a SOA that the software and the applications. This holistic view of the enterprise by the SOA paradigm distinguishes it from the approaches used to engineer legacy IT systems.

Service Oriented Architecture challenge

A fundamental idea of the Service Oriented Architecture paradigm, as we have seen above, is the case for any ICT infrastructure, is that the business processes do not simply depend on the information systems - they define the services these systems should provide [16]. In other words SOA is a **Business-Driven** approach [32, p.52]. The business processes inevitably change and this leads to changes in the supporting systems. Here lies the major challenge for SOA - designing systems in such a way that accommodating most business

process changes can be achieved by simply rearranging existing business services. Thus, the main aim is to achieve **flexibility** [52]. But this flexibility is determined by each of the above-mentioned aspects of the Total Architecture. This means that, besides system flexibility, a SOA must ensure a clear and flexible organization, roles and processes. Achieving flexibility is hindered by the challenges of existing distributed systems in the enterprise, different owners and system heterogeneity [52, p.13].

SOA definition

In the following sections we give the characteristics of the Service Oriented Architecture paradigm, by giving definitions of its most important concepts, including its own design paradigm and design principles, design patterns, a distinct architectural model, and related concepts and technologies. Three of the most important of these concepts are **services**, **interoperability** and **loose coupling** [52, p.16]. Service Oriented Architecture is still an evolving field of research and there is still no single industry-wide definition. We give the definition, that captures best all the aspects in a single formulation.

Definition 2.8 ***Service-Oriented Architecture:*** *A service-oriented architecture is a framework for integrating business processes and supporting IT infrastructure as secure, standardized components services - that can be reused and combined to address changing business priorities [9]*

Definition 2.9 ***Capability:*** *A capability is a resource that may be used by a service provider to achieve a real world effect on behalf of a service consumer [67]*

Definition 2.10 ***Business Process:*** *A business process is a description of the tasks, participants' roles and information needed to fulfill a business objective [67]*

"So an SOA is not simply an approach to designing computer systems; it is an approach to designing the enterprise. Shared **capabilities** are utilized in different contexts to achieve economies of scale and consistency of operations and control. The computer system should be designed to align with the design of the business" [26, p.32]. The technological foundation of SOA is the design paradigm of Service-Orientation. It is guided by a set of 8 design principles that will be described later on. The **service** is here the fundamental unit through which solution logic is represented [32]. However designing a SOA is not just making sure that a system displays characteristics of integration, loose coupling and is built out of services. For a successful SOA the following four ingredients are important - **Infrastructure**, **Architecture**, **Processes**, **Governance** [52, p.18].

Services

As already mentioned, SOA is a Business-Driven approach, so the main goal is to structure the supporting software system following the structure of the business processes. The ideal end-situation is an easy and direct 1-to-1 mapping, between business and software solution logic, which results in improved flexibility. This results in flexibility, because the cause

for change is often initiated from the business. Thus, being able to follow the changes in the business processes by only reconfiguring(reconnecting components) and not functionality redistribution results in flexibility. This is why the concepts of *services* and *Service-Orientation* are introduced. The definition of a service is not yet settled upon [16, 30, 82]. We follow:

Definition 2.11 ***Service:*** *A service is an implementation of a well-defined piece of business functionality, with a published interface that is discoverable and can be used by service consumers when building different applications and business processes [82, Ch1.5].*

Services have been mentioned in the section on software architecture as general concepts for functionality that a component can supply to a requesting party. This is mainly a fine-grained concept on the level of function calls. In SOA, a service has a more course-grained meaning. Services form a Service Layer [30, p.23] as an abstraction above components, functioning as a convenient way of encapsulating functionality and offering it as a capability. Ideally "services map to the business functions that are identified during business process analysis"[30, p.28].

In spite of the absence of concrete definitions there is a common set of characteristics most associated with service orientation [75]. These are further distiled, by Erl [31], to the 8 Design Principles, mentioned in the SOA definition subsection:

Contracts: Services share a formal contract defining the terms of information exchange in their interaction

Reusability: Services are designed to support potential reuse

Coupling: Services are loosely coupled

Abstraction: Services abstract underlying logic. The only part of a service that is visible to the outside world is what is exposed via the services description and formal contract. The underlying logic is invisible and irrelevant to service requestors.

Composability: Services may compose other services. This possibility allows logic to be represented at different levels of granularity and promotes reusability and the creation of abstraction layers

Autonomy: Services are autonomous. The logic governed by a service resides within an explicit boundary. The service has complete autonomy within this boundary and is not dependent on other services for the execution of this governance

Statelessness: Services should be designed to maximize statelessness even if that means deferring state management elsewhere

Discoverability: Services are discoverable. They should allow their descriptions to be discovered and understood by humans and service users who may be able to make use of the services logic

From the principles mentioned above autonomy, loose coupling, abstraction and the need for a formal contract can be considered the core that forms the foundation for SOA. From this point of view, SOA is nothing more than a paradigm aimed at improving system flexibility. As such, it is appropriate for large distributed systems with different owners and a high degree of heterogeneity [52].

SOA Software Architecture Modeling

Two architectural views of the software architecture of Service-Oriented systems are central to identifying modernization changes and thus effort. The first view captures the aspects of **Process View** described by Kruchten [61] and is given by Lewis et al. [64] (Figure C.5). In this view, central is the interoperation between Service Providers and Services Consumers.

Definition 2.12 ***Service Provider:*** *a participant that offers a service that permits some capability to be used by other participants [67]*

Definition 2.13 ***Service Consumer:*** *a participant that interacts with a service in order to access a capability to address a need [67]*

Besides these entities, **Infrastructure** is also an important enabling concept. The SOA Infrastructure connects service consumers to service providers. It usually implements a loosely coupled, synchronous or asynchronous, message-based communication model, but other mechanisms are possible. The infrastructure contains elements to provide support for cross-cutting concerns in achieving system integration such as service discovery, security, transaction management and logging. A common SOA infrastructure is an Enterprise Service Bus (ESB) to support web service environments [64]. The Process View of SOA illustrates the present separation of concerns on an architectural level between *Service Providers*, *Service Consumers* and *Infrastructure*.

The second view of the software architecture of Service-Oriented systems is from the work of Gimnich [39] (Figure C.6). It demonstrates the difference in architecture between Service-Oriented systems and legacy systems through the **Development View** described in [61]. In legacy systems the functionality from the layers 1, 2, and 4 – *Operational Systems*, *Service Components* and *Business Processes* – can be found, albeit not always so neatly separated. This results in direct coupling between the business processes and the supporting software systems. SOA introduces the additional abstraction of the Service Layer [30] between the Business and the Components. It enables that important IT-to-Business alignment mentioned in the problem description in chapter 1 by decoupling the business process from the implementing components.

There are two standards for modeling the Development View. One standard is concerned with modeling the application logic. The other, models the data used by the applications. The first is the **Service Components Architecture (SCA)**[1,2], used to model the link between the elements from each of the four layers of the Development View. The SCA specification (Figure C.2) defines how the system components are built and how to combine those components into complete applications. The *Service Components* layer of the Development View is organized according to SCA through the concepts of **Components** and **Composites**. By adding **Services** and **References** to the them, the mapping is done from the *Service Components* layer to the *Service* layer. The components are linked to the services they supply. Linking Services and References to each other through the concept of **Wires**, the further link upwards is done to the *Business Processes* layer. The wired services

[1] `http://osoa.org`

[2] `http://www.oasis-opencsa.org/`

are aggregated into business processes. Finally this resulting implementation independent model can also be linked through **Bindings** to a specific implementation from the bottom layer of the Development View – *Operational Systems*.

The second standard complementing SCA is the **Service Data Objects (SDO)**[3]. It specifies an architecture for handling the business data in an implementation independent way. The goal is unified data access to heterogeneous data sources. The SDO representation of the data is exchanged through the wires of the SCA model (Figure C.3). The SDO architecture is based on the concept of disconnected *Data Graphs* built out of *Data Objects*. Information about the structure is contained as a *Metadata*(Figure C.4). Access to data sources is provided by a class of components called **Data Access Services (DAS)**. The DAS is responsible for the marshaling and unmarshaling of the implementation specific and possibly heterogeneous data sources into the SDO representation as Data Graphs (Figure C.4).

Both architectural views, mentioned in this subsection, are needed for the estimation of modernization effort. They model architectural characteristics that are impacted by modernization. These are the structure of components and their interoperation of both application logic and data (section 2.4). To manage the complexity of the effort estimation problem, the "divide and conquer" approach has to be applied. Using the above-mentioned views enables the division of the target SOA architecture in pieces for which the effort can be estimated in relative isolation and in respect to different modernization concerns.

2.3 Existing approaches to system evolution

This section describes three existing approaches for performing system evolution and modernization – **Renaissance**, **ADM** and **SMART**. Only when the steps of modernizing a system are clearly defined, can an estimation be made of the effort this process will require. The definition of these steps is what these approaches supply. They share some common features, but also complement each other in other areas. Each one emphasizes on different aspects of the modernization process.

Renaissance has the most oversight of the three. It defines all phases of the modernization process. This gives a good starting point and orientation of where the effort of modernizing a legacy system lies. Renaissance focuses on possible modernization strategies, cost factors and their categorization. On the other hand, ADM and the accompanying RMM approach focus on the modernization planning phase. They model the changes involved on the architectural level and the distinction between affected domains. This is an important extension to Renaissance as modernization towards SOA consists for an important part of architectural transformations. The reason for this is that the differences that need to be overcome between legacy and SOA systems ares mainly in the software architecture.

Both Renaissance and ADM are approaches that lend themselves for automation. That is not the case with SMART. It is an expert-driven approach in a questionnaire form. However, it has to its advantage that it is specifically developed for evaluating the modernization towards SOA. Therefore, parts of these three approaches will be combined for the construction of the overall effort estimation framework in chapter 3. The rest is mentioned as an

[3] `http://osoa.org/display/Main/Service+Data+Objects+Home`

indicator of how much is left out of the scope of this research and will be mentioned in the chapter on future work, chapter 6.

2.3.1 Renaissance

Renaissance is a generic incremental approach for performing legacy system modernization. "Renaissance provides and end-to-end guidance, from project conception through to system deployment, for managing evolution projects" [98, p.17]. The method defines:

1. a generic process that guides practitioners through the activities of evolution projects
2. practical advice and techniques on how to perform the activities of the process
3. an information repository defining the information to gather during the process

The process of modernization iterates through four phases: *Plan Evolution*, *Implement*, *Deliver*, *Deploy* (Appendix C.7). In the Planning phase is decided whether to decommission the system, continue to maintain it, improve through reengineering or replace it. This **Portfolio Analysis** is based on an assessment of the legacy system and its operational context. The system's **"technical quality"** and **"business value"** are estimated by quantifying a corresponding set of parameters. For the technical quality, attributes such as age of the system, failure rate, complexity and size are important [98, Table3.2]. The choice of further action depends on the cost/risk trade-off decisions taken for the project. Once the technical quality and business value have been quantified, Renaissance prescribes a recommendation about further action, depending on the combination (Figure C.8).

Evolution strategies

In the case of further action, the Renaissance method identifies six evolution strategies to evolve to a new desired state, given in figure C.10. Each of them involves certain effort an has its benefits, given in the description [98, p.50-54].

Evolution modeling

To support the modernization process, Renaissance gives the evolutionary steps that have to be taken, together with the information they require (Appendix C.9). Two main aspects are modeled: the **Legacy System** and the **Target System**. Each of these models consist of two parts **Context Model** and **Technical Model** [98, Figure4.1].

Context modeling is done from four viewpoints on the system [98, Figure4.4]. To represent these viewpoints a set of documents [98, Table4.1] and capturing techniques [98, Table4.2] is recommended for each of them.

- **Business:** captures the system's support for the business processes
- **Functional:** describes the system functionality which implements the business processes captured in the business view
- **Structural:** overview of the system's software architecture and data structures model
- **Environmental:** describes the physical system structure such as its network organization and communication model

Technical modeling is done by further decomposing the documents from the structural and environment view for both the legacy and target system. The objectives for the technical modeling are three: *a*) identify components which can be worked on independently *b*) provide a base to extract, adopt, and reuse components *c*) provide sufficient detail to subsequently implement the target system.

Information Management

Renaissance defines six categories of information needed for the evolution process:

- **Business**: contains Business goals, Business process description, Problem statement
- **Legacy system**: contains the Assessment report, System documentation, Context model, technical model
- **Target System**: contains the Context model and Technical model
- **Evolution strategy**: contains Possible evolution strategies, Cost Benefit Risk analysis, System evolution strategy
- **Project Management**: contains Project plan and Deployment plan
- **Test**: contains the Test strategy plan, Test data, Test report

The first four are needed in order to plan the modernization of the system (Appendix C.11). The target system specification does not need to created into detail as it is adjusted in the Implementation and Delivery phases. The last two information sources, Project Management and Test, contribute to the effort and costs required, but are dependent on the first four and are only made after these are ready.

Integration Models

Renaissance presents six *Integration Models* for the components of the legacy system [98, Fig5.7]. Each integration model is appropriate for a particular reengineering strategy [98, Table5.2]. It describes a logical structure and how to integrate parts of the legacy system through it into a target system. Here we list the models and the way to achieve them suggested by Renaissance:

- Data Integration - Direct access, Meta-data encapsulation
- Service Integration - Remote Procedure Call, Message-oriented middleware
- Presentation Integration - Screen scraping
- Distributed Object Integration - Direct Access, Encapsulation

Cost Drivers

Renaissance gives a process for estimating the risks and costs associated with each evolution strategy [98, Figure3.14]. The costs are divided in three categories:

I. **RIC** – Reengineering Investment Costs
II. **OSRC** – Operations and Support costs for Reengineered Components
III. **OSLC** – Operations and Support costs for Legacy Components

The costs are a consequence of effort required for modernization activities such as [98, p.25, Table3.7]:

- Reverse engineering
- Software development
- Data migration
- Incremental evolution
- Target system deployment
- Legacy system integration
- Integration testing
- Licensing costs
- Hardware
- Training
- Operational site activation

The possible estimation techniques for the above-mentioned costs are: *a*) *Expert judgment b*) *Analogy-based c*) *Top-down d*) *Bottom-up e*) *Parametric* [98, Table3.8].

2.3.2 Architecture Driven Modernization and RMM

The Risk-Management Modernization (RMM) approach gives a more detailed process description (Appendix C.12) of the first two steps in a modernization track, identified by the Renaissance method – *Plan Evolution* and *Implement*. The initial portfolio analysis is similar to the one performed according to the Renaissance method [82, Figure3-2]. Once the parts of the system, that need to be modernized, are identified – the modernization candidates, the process continues by identifying the stakeholders and their requirements. These are then used in a cost-benefit analysis to "help decide which approaches, if any, should be evaluated further" [82, p.31].

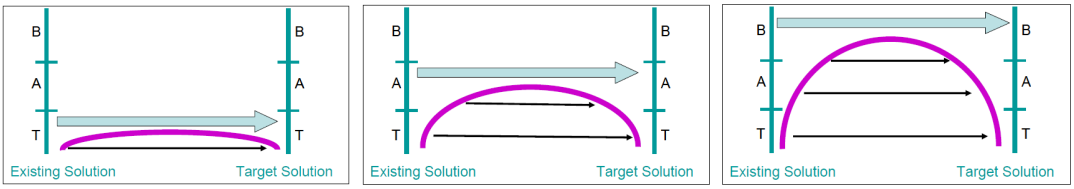
ADM Horseshoe model

The actions part of the modernization are performed in the context of the Architecture Driven Modernization(ADM) model [60, 59]. According to ADM, the existing legacy system as well as the target solution can be represented by models on three levels of abstraction: **Business architecture** domain, **Application architecture** domain and **Technology architecture** domain. The modernization effort is then represented by a path starting from the Technology models of the legacy system and ending at the Technology models of the target solution. This path can go through the higher abstraction levels or remain limited to the lower levels. This leads to three basic types of modernization approaches, Figure 2.1. They range from less invasive and at the same time with less modernization impact(Figure 2.1a), to more invasive and more difficult to perform, but with much more modernizing impact(Figure 2.1c).

Going between the models of the different layers is done by transforming them. There are three types of such transformations: **Formal** transformations, **Abstraction** transformations and **Enhancing** transformations (Appendix C.13). The path that is thus traversed has the form of a horseshoe and thus the name of this model – the ADM Horseshoe Model.

RMM Program understanding

In order to evaluate the modernization in terms of risk and effort, RMM includes the phase of *Program Understanding*. It consists of two parallel activities of *Legacy System Un-*



(a) Technical Architecture-Driven Modernization (b) Application/Data Architecture-Driven Modernization (c) Business Architecture-Driven Modernization

Figure 2.1: ADM modernization paths

derstanding and *Target Technology Understanding*. The Legacy System Understanding is where the first part of the ADM modernization path is done, going from the Technology architecture models to the Business architecture models – reconstruction. On the code-structure level reconstruction techniques include artifact extraction, static and dynamic analysis, and slicing techniques. On the function-level, the used techniques include semantic and behavioral pattern matching, and aggregation and clustering.

Once the models of the legacy system have been constructed to the desired level of abstraction follows the Target Technology Understanding. This is a contingency-based evaluation of multiple technology alternatives in parallel, targeted at a sample model problem. Following this initial investigation, an evaluation is also done at the architectural level following the ATAM approach [57]. For this purpose, use cases are used to evaluate how well each architectural alternative supports the desired system requirements and quality attributes. After target architecture has been chosen, a modernization strategy is developed. This strategy targets issues about the scheduling of the phases of the modernization activities, such as code migration, data migration and intermediate deployment states [82, Figure 13-3]. Seacord et al. recommend the usage of **Data adapters** and **Logic adapters** as building blocks for the realization of such a modernization strategy. The alternatives are analyzed again and a choice is made about the global ordering of activities. The concrete details of the transformations are developed in a **Code Migration Plan** and a **Data Migration Plan**. The identification of the needed transformations in the Code Migration Plan is based on a componentization analysis. The componentization can be based on two different cohesion criteria, according to Seacord et al. [82, p.256]:

- **user-transaction based** – identifies, through structural analysis on the call-graph of the source code, four categories of program elements: root-, node-, leaf- and isolated program elements. The root elements and the call tree underneath them map directly to user transactions. This functionality is then clustered together.
- **business object based** – closely linked data entities, extracted from the database technical design are clustered together. For each such data record set, the functionality around it that manipulates data entities from the set, is clustered together.

The Code Migration Plan then schedules the migration of the clusters over sequential iterations. The goal are as few as possible adapters, necessary to accommodate for the interoperability between already migrated and not yet migrated clusters, during the intermediate states. Finally the complete modernization strategy if reconciled with the stakeholder re-

quirements and the resources needed to realize the modernization strategy are estimated, using the estimated size of the system and cost models such as COCOMO II [13].

2.3.3 SMART

The Service Migration and Reuse Technique(SMART) [63, 64] is a modernization approach developed by the SEI specifically for the migration towards SOA. The method analyzes, through a questionnaire of more than 60 categories, the viability of reusing legacy components as the basis for services by answering questions like: *What services* make sense to develop? *What components* can be mined to derive these services? *What changes* are needed to accomplish the migration? *What migration strategies* are most appropriate? What are the preliminary estimates of cost and risk? The method contains three important elements:

SMART process a systematic means to gather information about the legacy components, the candidate services, and the target SOA environment (Appendix C.14).

Service Migration Interview Guide (SMIG) contains the questions that guide the information gathering. Its goal is to cover all known issues that could influence the migration in cost and effort.

Artifact Templates output of the SMART process – Stakeholder List, Characteristics List, Migration Issues List, Business Process-Service Mapping, Service Table, Component Table, Notional Service-Oriented System Architecture, Service-Component Alternatives, Migration Strategy

SMART process activities

Establish Context Understand the business and technical context for migration. Identify stakeholders, Understand the legacy system and target SOA environment at a high level, Identify a set of candidate services for migration

Define Candidate Services select a small number of services from the initial list of candidate services, that perform concrete functions, have clear inputs and outputs, and can be reused across a variety of potential applications

Describe Existing Capability technical personnel are questioned to gather information about the legacy system components that contain the functionality meeting the needs of the services selected in the Define Candidate Services activity.

Describe Target SOA Environment gather information about major components of the SOA environment, impact of specific technologies and standards used in the environment, guidelines for service implementation, state of target environment, interaction patterns between services and the environment and execution environment for services. Build the so-called Notional Service-Oriented System Architecture to describe the system in terms of its components – service consumers, infrastructure, services, legacy components – and how they interact with each other, resulting in a representation similar to the SOA Process View (section 2.2).

Analyze the Gap provide preliminary estimates of the effort, risk, and cost to convert the candidate legacy components into services, given the candidate service requirements and target SOA characteristics. The discussion of the changes that are necessary for

each component is used as the input to calculate these preliminary estimates. The Service-Component Alternatives artifact is created during this activity to illustrate the potential sources for functionality to satisfy service requirements.

Develop Strategy generate a strategy to address the migration issues generated by the previous steps

Overall, although the SMART method has a few disadvantages, it is an important addition to Renaissance and ADM. Its first disadvantage is that it is a proprietary approach and the concrete content of the questionnaires is not publicly available. But even then, the approach is not based on automated data collection or artifacts. It does not rely on scanning and analyzing the source code of the legacy system. This means that the SMART Process must be performed by trained experts, collecting information through interviews or manual inspection. In spite of these two disadvantages, SMART does provide indicators about the specifics of modeling the modernization towards SOA, which the previous methods lack – Artifact Templates, Define Candidate Services, Describe Existing Capabilities. For this reason, these will be used to extend the approaches of Renaissance and ADM.

2.4 Modernization towards Service Oriented Architectures

Previously it was described how to model legacy system architectures, target SOA environments and the overall process of modernization. There is a fourth and final element needed to address the research question. This is the issue of quantifying the impact and effort involved in performing the modernization towards SOA in the context of the approaches presented earlier. Specifically, we will concentrate on the use of software architecture information – the concrete models and changes on the architectural level, that we consider to influence the modernization effort.

The section begins with the fundamentals of evaluating the impact of modernization – *Impact Analysis*. It gives an overview of the aspects where the impact is manifested – Global, Organizational, Business and Technological. In this context, the *Software Impact* on legacy systems is positioned relative to the impact on the organization and business. This positioning of the impact on the software aspect in the *Global Impact* of modernization is necessary for the definition of the research scope and the relation to future work.

The remaining two sections get into the details of the modernization towards SOA at the software architecture level. They present the ways to organize the known modernization strategies and their architectural and design changes looking at the effort they require. Their known impact on the modernization effort will be used as a base for the overall effort estimation of this research's framework.

2.4.1 Impact Analysis

System evolution and modernization involve performing changes to the system. The estimation of the consequences and extent of these changes is done through the process of *Impact Analysis*. We distinguish three areas that are impacted in the case of legacy system modernization towards SOA. We present them all here in order to make clear what is the extend of the scope of the research specified later in chapter 3.

Definition 2.14 ***Impact Analysis:*** *the activity of identifying what needs to be modified in order to make a change, or to determine the consequences on the system if the change is implemented [50]*

Global Impact

The SOA modernization effort is concerned with all aspects of the *Enterprise Total Architecture* – people, business processes, information, and systems (section 2.2). A similar holistic view is adopted by Renaissance (Appendix C.16). In addition, Josuttis [52] describes four areas of concern specific to developing a Service Oriented Architecture: *Architecture*, *Infrastructure*, *Processes*, and *Governance*. In the case of SOA as target architecture, modernization changes are not only limited to the software system, but have a more globally reaching impact.

Each of these aspects needs to be addressed in the modernization and initiates its own series of changes. However they are all interrelated and changes in one impact the others. One example is the relationship of the technology aspect to the business. The architectural choices made, limit the possible options and/or create opportunities for the business [78]. A second example is the relation of the technology aspect to the organization of the enterprise. Once the Service Oriented Solution is in place it has to be maintained and the facilities for this SOA Governance have to be put in place [16, ch13], [76, §2.5], [52, p.261].

For a complete Impact Analysis of the modernization all of these aspects have to be involved. But we will further concentrate on the change impact in the software system aspect. We cannot consider it in isolation and will have to make assumptions about the rest and the enterprise purpose.

Impact organization aspect

The organization must be set up to enable the anticipated changes to the legacy system and at the same time adjust to them. This is characterized by Tilley in [93, §5.6] as **organizational readiness**. The ripple-effects of the change impact will create the need for also modernizing the organizational structures [52, ch.8]. Part of assessing the severity of this impact is the *capability assessment*, an inventory of the *training needs*, an *application usage survey*, an *operational deployment considerations* and the available capabilities in *managing organizational change*. Furthermore, specifically for Service-Oriented systems, it has been shown that a different set of roles is needed for their development and maintenance [54],[16, ch7]. The overall degree of impact of the modernization towards SOA is, thus, also related to its organizational SOA readiness – **SOA maturity** [16, ch6],[5],[44, p.532, ch13]. The same changes in a different organization may lead to a different impact and effort.

Impact business aspect

The modernization towards SOA has also an impact on the business aspect of the legacy architecture [17, ch.9]. The design principles of Service-Orientation have clear guidelines about participants, roles and responsibilities in the enterprise business processes. Thus the modernization of the legacy system calls for the introduction of their clear definitions and

possible restructuring of orchestration and responsibilities [52, ch.7]. There exist a number of approaches for the identification of business legacy affected by the modernization. This Business Domain Analysis helps identify the problematic points in the business process landscape that need modernization and thus deduce the impact such as the IBM Component Business Model (CBM) [33].

Impact technology aspect

The emphasis of this research lies with the impact analysis for the technology and software systems aspect. SOA is a Business-Driven paradigm so the input of the impact analysis process for the technology aspects must be the result of the Business Domain Analysis. The impact on software systems manifests itself in two different ways: architecture/design and system quality attributes [65, 74, 76]. There are two aspects to their quantification:

I. difference/similarity analysis
   - between architectural models of as-is and to-be system: data [22] or functionality [2, 69]
   - based on system quality attributes characteristic for Service-Oriented Architectures: interoperability [55, 7], compatibility [2], service-orientation [46]
   - reuse potential: identification reusable components [40]

II. change propagation [101]

2.4.2 Classification of modernization strategies

Besides the differences between as-is and to-be system architectures, the modernization effort is also determined by the *modernization strategies* chosen to overcome these differences [48]. In order to make an estimation of the overall effort, we need to quantify the difference between these modernization strategies in terms of effort. Existing work makes it possible to do this on an ordinal scale in two dimensions. First, we classify them according to the issues they addressed in the modernization towards SOA. Second, we use a classification of modernization strategies according to their impact and required effort.

We use the following identified issues in modernizing towards SOA to make a classification of the modernization strategies according to which one they target:

	O'Brien [76]
	Identification/Mining of services
	integration of common shared services
	development SOA Infrastructure
	Development service providers/consumers
	SOA Governance
	Architectural Trade-off Analysis

	Sneed [89, 90, 86]
	degree of dependency on the environment(none, partially, totally)
	identify business rules ("code striping")
	achieving statelessness of services
	n:m relationship between business rules and code blocks
	datamodel mapping to SOA technology(eg.XML)

Bierhoff - Architectural mismatch [10]
the nature of services
- functionality supply
- infrastructure expectations
- control model
- data manipulation
communication between services
- asynchronous communication
- message data model
global architecture structure
construction process

Depending on their impact, modernization strategies can be classified in two different categories: **Black-Box** modernization and **White-Box** modernization [82, p.9].

Black-box modernization involves examining only the external behavior of the legacy system through its inputs and outputs. The system's internal working mechanisms are ignored. There are two common motives for this approach. Either there is simply no description of the internals available (no source code) or the internals are considered too complex and are abstracted away by considering only their effect on the environment. A common black-box method is wrapping. Unfortunately this approach is not always practical and often still requires knowledge about the system component's internals through white-box techniques [80].

White-box modernization is much more extensive and complex than the black-box approach. It requires the expertise of software engineers on a much more deeper level and is thus also known as *software reengineering*.

Definition 2.15 ***Software reengineering:*** *Reengineering is the systematic transformation of an existing system into a new form to realize quality improvements in operation, system capability, functionality, performance, or evolvability at a lower cost, schedule or risk to the customer [82].*

The reengineering process involves three phases. First, it gathers information about the internals of the legacy system through *program understanding* [24]: modeling the domain, extracting information from the code, creating abstractions that describe the underling system structure. This process is also known as reverse engineering:

Definition 2.16 ***Reverse engineering:*** *The process of analyzing a subject system to identify the system's components and their interrelationships and create representations of the system in another form or at a higher level of abstraction [24].*

Then, using the knowledge from this analysis, follows *software restructuring*. After which the third and last phase of forward engineering is performed. This view on system reengineering is also advocated by the Architecture Driven Modernization approach.

Definition 2.17 ***Software restructuring:*** *the transformation from one representation form to another at the same relative abstraction level, while preserving the subject system's external behavior(functionality and semantics) [24].*

Reengineering is often based on graphs as a representation of the system. This has also been done by Cremer et al. [25] for a system written in Cobol similar to the one used in the experiment of this research.

Within the white-box and black-box categories, we distinguish specifically for the modernization towards SOA three main non-mutually exclusive categories: **wrapping approaches**, **componentization**, **service extraction**. They are aimed at achieving the changes needed for Service-Orientation (Appendix C.15 [39]).

Wrapping approaches

Wrapping is a way of overcoming the mismatches between the interface offered by a software component and the interface required by its environment. This approach corresponds to the replacement of a connector between two components with a new component with a connector to each of the components.

Definition 2.18 ***Wrapping:*** *surrounding the legacy system with a software layer that hides the unwanted complexity of the old system and exports a modern interface [82]*

Wrapping can be performed for different parts of the system and with different degree of added functionality in the wrapper. In all cases, the application of this approach aimed at satisfying the SOA design principle of Contracts. Wrapping approaches have been extensively studied [15, 23, 47, 85, 88, 89, 12, 1, 19, 21] with the following most significant examples of different types:

- Simple wrapper for encapsulation and integration of services [89]
- Wrapper containing additional functionality: Logic adapter for the request-response pattern [19, 21], Session Based, Transaction Based, Data Based [1]
- Replacement with COTS components [7, 47] such as an ESB [8] complying with the same contract
- Componentization combined with wrapping towards a MVC design pattern [12]

Componentization

Componentization is a group of restructuring approaches. They are based on clustering/-grouping together functionality into components according to some criteria they share, such as the use of the same data source or the implementation of the same concern. By doing that, the cohesion of the code is increased according to the clustering criteria and the coupling to the rest of the functionality is decreased. This approach introduces autonomous and manageable units of reuse – the components – and defines clear separating lines between them. The result of such restructuring makes possible the fulfillment of the SOA design principles of coupling, autonomy and reusability. The following work describes the important characteristics of the method and the variations in the approach:

- Componentization according to a domain model: use domain knowledge to identify *"features"* supported by the legacy system and identifying the functionality belonging to each feature. This code is then clustered per feature [68]

- Componentization according to existing code structure (coupling, dependency matrix): automatic identification of components [40]
- Componentization for a concern: clustering for client-server architecture [20]
- Componentization for a quality: component mining for reusability [84]

Service extraction

The group of service extraction approaches is a special case of componentization. Here, extra effort is spent on making sure that the resulting components are also suitable to be used as services in the infrastructure of a Service-Oriented architecture. This means that the SOA design principles of contracts, composability, statelessness, and discoverability are central. The clustering criteria are chosen accordingly in the following approaches:

- Combining black-box and white-box approaches: legacy system decomposition into components and exposing them as services to be integrated into a service oriented architecture [103]
- Service extraction as part of ADM: service extraction by following datamodel definitions [41]
- Business process and service extraction [104]

2.4.3 Architectural and design features

The modernization strategies described above are targeted at transforming the legacy architecture into a Service-Oriented organization. In this transformation process, most entities from the legacy architecture have their analog in to Service Oriented Architecture. This correspondence can be one-to-many, many-to-one or many-to-many. For example a set of legacy components can be chosen to form one SCA Composite exposing a set of Services or one legacy component can be split and each part wrapped and exposed as a Service.

However, not every entity from the legacy architecture has its SOA counterpart and the other way around. We can illustrate this by saying that the entities described in the above transformations form the intersection between the source and target architectures. There are however, entities that belong to only one of them. This means that on the one hand, the addition may be required of architectural features not present in the legacy architecture. On the other hand, it may be necessary to remove architectural features that are obsolete in the target architecture. Here we describe these two categories according to their root cause.

Target Architecture

The first category is formed by the features that are required in the target SOA architecture, but that don't have their analogue in the legacy system. Papazoglou [79] gives an overview of the basic architectural implications of introducing a Service-Oriented Architecture. He specially emphasizes on the concept of the Enterprise Service Bus (ESB). Together with it, a set of SOA capabilities are describes such as:

- Transaction capabilities
- Reliable messaging capabilities
- Dynamic connectivity

These are features that may not b present in the legacy architecture and thus have to be added from scratch.

Next to these features, Stal [92] points out the use of certain patterns as a result of enforcing SOA design principles. This relationship is valid both ways: the presence of the particular pattern leads to the enforcement of the corresponding design principle and on the other hand the pursuit of a particular design principle leads to the use of the corresponding pattern. Some of the correspondences Stal mentions are:

- Loose coupling ⇔ Bridge- , Observer- , Reactor- , Store-and-Forward patterns
- Statelessness ⇔ Resource Lifecycle Manager, Activator- , Evictor patterns
- Composability ⇔ Coordinator pattern, Strategy pattern

Legacy Architecture

Modernization may also necessitate the removal of entities from the legacy architecture. These can be components, connectors or whole patterns these form. To estimate the impact of this, we need to know what the functionality represents that is being removed. We quantify how much of it is taken over by either replacement in another form or by dropping it all together. For this, a classification is very suitable of connectors, components and structural relationships.

The classification of connectors is given by Mehta et al. [69] through an extensive taxonomy. Removing or replacing connectors can be a way of decreasing coupling, one of the main goals of SOA. This taxonomy makes is possible to identify what capability supported by the connector is dropped that may need (partial)replacement and which other connectors should be considered that are close to it in the taxonomy.

A similar approach for components is based on architectural mismatches [10, 7]. The set of properties describing the component (Component Model, section 2.1) are used to identify mismatches and similarities between it and its replacement or between the left over components. Thus, when components are removed, these mismatches can be used to quantify the effort needed to make the whole work together again.

Removal can also affect a whole group of components and connectors following a design pattern. In such cases, it is also important to classify the removed functionality. There are two types of information that have influence on the effort for modernization. The first is what feature is removed described by the pattern's rationale. The second is which further components may be affected captured by the set of components involved in the pattern. Arcelli [3] describes a set of patterns that are useful for this purpose as they are most relevant for the case of modernization towards SOA.

Chapter 3

Effort Estimation Framework – Research description

In this chapter, we present the theoretical results of the conducted research. These results are twofold. In the first place, this is the combination of the described existing approaches most suitable for the goal of modernization effort estimation. Secondly, that is the contribution of our own concepts and argumentation about how those approaches should be extended to use in the form of an Effort Estimation Framework. This chapter consists of two parts. The first two sections specify the preconditions of the research. The following two sections contain the above-mentioned original contribution of this thesis.

The preconditions of the research begin with the presentation of the scope in section 3.1. Within this scope, the goal, as described in the problem statement (chapter 1), is to construct a framework that structures the process of impact analysis of the modernization towards Service Oriented Architectures of legacy systems. Continuing the research preconditions, section 3.2 presents the assumptions on which the framework is built. They form the invariable part that is the basis for the framework.

The following two sections present the original work of this research. First, section 3.3 presents the configuration of the Effort Estimation Framework. In this process the issues in modernization effort estimation with their multiple possible solutions are considered. For each of them, the preferred choice is presented and how it fits in the framework. Finally, all of these configurations are combined and presented in the form of the resulting Effort Estimation Framework. Section 3.4 presents its three components *Class definitions*, *Relationships* and *Control Model*.

3.1 Targeting the Effort in the Software Application domain

This section presents the initial scoping of the research, done by narrowing down the application domain of the framework. Three aspects are used to limit the relevant domains:

1. type of impact
2. modernization impact aspect
3. abstraction domains

Impact Types

Extending the definition of Impact Analysis given in Definition 2.14 on page 21, we define two types of impact in the case of legacy system modernization:

Definition 3.1 ***Gain:*** *the potential value the changes deliver [78] (Positive impact from the point of view of the enterprise). Characterized by an individually defined value function* $GAIN()$

Definition 3.2 ***Effort:*** *the cost of the changes needed to modernize (Negative impact from the point of view of the enterprise). Characterized by an individually defined cost function* $EFFORT()$

The problem statement of this research (chapter 1) described the needed feasibility analysis of a modernization project. Such an analysis would weight the benefits against the cost of such a project. These two trade-off factors will be represented by the two functions defined above – $Benefit = GAIN()$, $Cost = EFFORT()$. The valuations of these depend on multiple variables, some of which we will try to identify in this research as so called *Indicators*. The effort estimation approach of this research will be based on the balancing of the above-mentioned two Impact Types with an overall envisioned dependency of the Effort to the desired Gain as shown in figure 3.1. In this research we will concentrate on the effort cost function and only monitor the resulting gain.

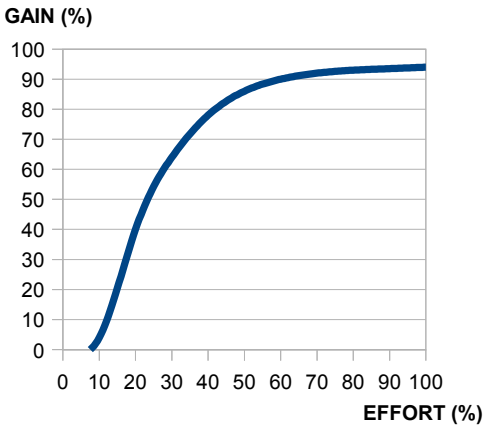


Figure 3.1: Graph of the expected relationship between Effort and Gain

Here we need to make a distinction between the Business Gain and the Gain we are considering in this research. We explain why we make this distinction and how the two are related. According to [81] an enterprise has a Mission from which follow its Business Goals. These in turn serve as a basis for the construction of a Business Strategy for achieving them. Based on this we define the overall Business Gain from a modernization project as the degree it contributes to the achievement of the set Mission, Business Goals and Business Strategy of an enterprise.
At this point the mentioned enterprise goals and strategy are still only defined in the business domain and are as Bleistein [11] shows only soft goals. Examples of such goals are

an increase in *Return On Investment*, decrease in *Time To Market* and increased *flexibility*. In order to translate them into measurable hard goals an implementation plan needs to be defined containing the concrete Objectives to achieve. The realization of this implementation plan depends on the enabling factors such as people, processes, structures and culture, and IT. We speak of alignment when these are ready for the execution of the strategy [81]. Part of achieving alignment is the IT-Business alignment [71]. Bleistein suggests an approach for achieving such an alignment through the translation of the Business Goals to IT requirements [11]. For the realization of these requirements and the implementation of the business strategy there are multiple choices for enabling technologies amongst which SOA. So a decision needs to be made whether the elicited requirements can be best satisfied by the Service-Oriented approach and the advantages it offers such as flexibility and reuse or whether there are other more suitable technologies. The Gain considered in this research is thus on the level of implementing the above-mentioned Objectives and is defined as the degree of gained conformance to the principles of Service Orientation. Increase in this Gain will lead to an increased Business Gain only and only if SOA and its advantages fit the business requirements resulting from the Business Goals [71]. In [81], [71] and [70] are mentioned classifications of business strategies. Based on these we give an indication of those that will most likely experience the greatest Business Gain from the modernization towards a Service Oriented Architecture. These include the adaptive firm in a very challenging environment, the entrepreneurial conglomerate, the innovator and companies that have as main strategy the entering of new markets through process innovation or product differentiation.

Concluding, we assume for the further research that the process of Business Strategy determination has resulted in the decision for SOA as enabling technology. Based on this goal, the subsequent gain is determined by the degree of achievement of Service-Orientation expressed through the SOA design principles such as loose coupling, reusability and autonomy described in section 2.2.

Modernization Impact Aspects

As mentioned in the context of this research (§2.4, §2.3), there are many aspects of the legacy system where the impact of modernization is visible. This research is concentrated on the Application Software sub-part of the Technical aspect (Appendix C.16). Within this Application Software sub-part, we recognize three factors that influence the effort estimation: (1) as-is system state, (2) to-be target state, (3) modernization strategies (known from figure 1.1).

Modernization Domains

All the above-mentioned three factors can be viewed on three levels of abstraction as described in the Architecture Driven Modernization (§2.3.2). The impact of the modernization on both gain and effort relative to the domain is shown in figure C.17 [60]. The research is concentrated on the Application/Data domain together with its links to the Business domain. The reason for this choice is that these domains are concerned with abstraction and

enhancing transformations, where the link to the Technology domain merely involves formal transformations [59].

3.2 Framework foundation on the existing approaches

In this section, we give the basic assumptions of this research within the scope given in the previous section. These are based on the existing approaches presented in sections 2.3 and 2.4. These assumptions further define the outlines of the proposed solution and the invariable parts in its application.

First and foremost, we assume a framework structure of the research end-result. The advantages of such an organization (§ 2.1) are needed for two reasons. First, the modernization process can be viewed as a transformation function between a *Domain* – as-is legacy systems and a *Range* – to-be SOA systems. Both the domain and the range of the modernization represent large sets of systems. With a framework we can specify the high-level guidelines for the effort estimation common to the modernization between different instances. Thus allowing its extension with additional modernization effort aspects and more information sources. Second, a framework offers a clear division between the components of modernization effort estimation. This makes it possible to concentrate on a subpart in isolation and specialize it for a particular set of instance systems. This will leave the rest unaffected and will still produce overall results.

The Effort Estimation Framework as such has to define:

Class definitions: Meta-model presented in section 3.4.1 + Rating Model
Relationships: Relationships, part of Meta-model presented in section 3.4.1
Control Model: Process + Rating Model + Relationships

This research takes into account the challenge in finding the balance between reusability and flexibility (§2.1) of the resulting framework. In order to make the flexibility level of the framework clear, we divide its theoretical base in two parts. First, we present the rigid set of assumptions in the rest of this section that define the frameworks broad reuse domain. Then, in section 3.3, we present a set of configuration issues that give it flexibility. These offer choices for concrete instantiation of issues from the framework. A different set of choices could be made there to influence the estimation outcome, but that would still preserve the overall relationships within the framework. For example, for each step in the *Control Model* a concrete approach can be chosen. In the framework instantiation process there is still one last level of implementational choices left, which we will present in chapter 4.

The assumptions of the framework that follow have two dimensions:

1. the subject they are about: Legacy system, SOA system or Modernization strategies
2. the framework part they belong to: Class definitions, Relationships or Control Model

We will present them grouped together according to the first one and indicate where necessary which framework part they are relevant for.

Overall

O-1. use models on each level expressed in (abstraction)concepts in order to make the framework possible. These models can be replaced or extended and instantiated for a particular system. As long as they supply the required information to fit in the framework. One can add indicators, concepts etc.
O-2. the framework contains 3 Domains – Business, Application, Technology – based on ADM figure C.17[Class definitions, Control Model]
O-3. concepts are linked top-down to other domains' concepts (relation "is-implemented-by") – makes sequential causal impact propagation analysis possible
O-4. Concepts are linked to the concrete data sources they originate from and are represented in Architectural Views [Class Definitions]
O-5. portfolio analysis(fig.C.8) is performed prior to effort estimation. See also Future Work in chapter 6
O-6. cost estimation is the phase following this estimation output. Assume cost division as given by Renaissance on page 16

Legacy System

L-1. the software system legacy code is the only input source (no documentation, no expert knowledge) [Class Definitions]
L-2. the considered programming paradigm is that of procedural languages (no OO and its advantages of embedded semantics). There is different research done to model legacy systems and each is tailored at a specific paradigm. Only the OMG has so far produced a standard (KDM) that attempts to bring all paradigms together. Currently it is only a standard and there are no implementations of it [Class Definitions]

Modernization

M-1. both the legacy system and the SOA target can be abstracted to the architectural level, expressed with the same concepts, where the only difference will be in the structure
M-2. 4 Phases based on Renaissance(fig. C.9) and RMM(fig. C.12) [Entities]. See also Future Work in chapter 6
M-3. changes and their impact propagate top-down starting from the Business Process level(see fig.C.17)
M-4. model overall steps of SOA modernization based on SMART (see fig.C.14). Steps: *Establish context*, *Define Candidate Services*, *Describe Existing Capability*, *Describe target SOA environment*, *Analyze the Gap*
M-5. using Renaissance modernization strategies shown in fig.C.10 as basis
M-6. using Data Adapters and Logic Adapters as strategy types (see RMM on page 18
M-7. set of available approaches to be: wrapping, componentization, service extraction (see section 2.4.2)
M-8. although there are also other factors, the effort is largely determined by the architectural changes needed to the legacy system architecture

M-9. rating is to be done based on gap analysis between as-is and to-be, fig.C.9 [Process Model - Stages]
M-10. a hierarchical rating approach with modernization issues on the top and concrete measurable indicators on the bottom

SOA

S-1. two categories of entities – infrastructure/services, see definition 2.8 [Entities, Rating Model]
S-2. definition 2.11 "well-defined piece of business functionality" for service identification [Service extraction methods]
S-3. SOA Design Principles(sec.2.2) are the guiding force behind architectural decisions about the target environment
S-4. possible SOA capabilities are the ones listed by Papazoglou in [79] [Rating Model]
S-5. 2 architectural views on SOA systems – Development View (fig.C.5) and Process View (fig.C.6) [Rating model]
S-6. it is possible to describing the target Service-Oriented system in terms of SCA concepts (fig.C.2)

3.3 Framework configuration

Starting in this section, we present the original work of this thesis. Here, the configuration is presented of each of the three components of the Effort Estimation Framework – *Class definitions* (§3.3.1), *Relationships* (§3.3.2) and *Control Model* (§3.3.3). The configuration is done by first presenting the possible options for addressing an issue and then giving the choices we have made. The resulting framework is then presented in the next section 3.4.

The contribution in all three parts is two types. In the first place, concepts from existing approaches are selected and combined through categorization. This is the case for the entities selection in the Class Definitions, the model selection in the Relationships and phases and domains in the Control Model. Secondly, this section introduces a Rating Model built on top of the selection of existing approaches. It uses the information gathered with them to produce an effort estimation of the modeled system modernization.

3.3.1 Class definitions

Configuration categorization

C-1. we organize the class definitions as a meta-model consisting of 3 sets of modeling concepts: Legacy System, SOA system and Modernization strategies
C-2. choose for the modeling concepts on the architectural level: component, connector, interfaces (sections 2.1)
C-3. the differences between legacy systems and SOA-based ones, that cause the need for evolution are the Service Abstraction Layer and the resulting grouping of func-

tionality on the implementational level, mapping business functionality to software functionality (1:1)

C-4. architectural and design patterns (see section 2.1) are considered important concepts for the modernization towards SOA as noted by Arcelli [3] and Stal [92]. Their work shows that certain patterns can be used to achieve SOA design principles as described in section 2.4.3. However, in the literature study there is no concrete indicator found for the estimation of the modernization effort needed given the presence/absence of a pattern. Based on this, the use of architectural and design patterns in the framework is configured to be limited. On the one side, the place of patterns in the meta-model is defined by the correspondence between their rationale and the SOA design principles. However, they cannot be included in the Rating Model until further research is done on the relation between patterns and modernization effort such as: *a*) the effort needed to transform a set of legacy entities into a pattern *b*) the effort that is saved by reusing an existing pattern binding a set of legacy entities

C-5. use a Rating Model to organize the aggregation of the different factors to a effort estimation output

Configuration Rating Model

The Rating Model is base on the Goal/Question/Metric method [97]. Two issues that stand at the basis of the configuration of the Rating Model are the choice of an effort quantification method and the choice of a rating scale for the estimation output.

Quantification method – an effort estimation analysis could be based on:

1. quality attributes and general legacy system characteristics such as size, complexity, etc. These are easily measurable, but would produce too inaccurate results. The reason for that is that they do not provide enough information, and only give an indication of the *resistance* of the system to change. What is equally important is measuring the *need*, the spreading of the changes. Such an approach cannot, thus, produce an accurate estimation of the *effort*, which is the product of the *resistance* and the *need*
2. a second and much more accurate approach is to try and approximate the modernization process that would take place. In doing so, giving an estimation for both *resistance* and *need*. This though means modeling the modernization changes following the ADM modernization scenario from section 2.3.2, a more extensive process

Rating scale – the effort estimation can be expressed on one of four measurement scales:

1. Nominal scale: division in sets through labels
2. Ordinal scale: relative order, but no magnitude of the difference
3. Interval scale: difference magnitude, but no absolute zero point
4. Ratio scale: rating with an absolute zero point

Points of Modernization as basis for effort estimation

In order to apply the second, more accurate, above-mentioned quantification method we need to model the modernization process in the framework and use this approximation for effort estimation. The base for this model is provided by the concept of a **Point of Modernization(PoM)** in the Rating Model. A PoM signifies a gap between the architectural models of the as-is and to-be systems. It is identified by checking the as-is architectural views for conformance with the SOA design principles(see section 2.2). A Point of Modernization uniquely identifies the relationship between a set of legacy entities, a set of SCA to-be entities and the the set of strategies that will be applied to transform the one into the other. For a set of legacy components this could mean the application of multiple strategies such as re-componentization and wrapping to achieve the desired end-system organization. Each such Point of Modernization is, thus, the source of a certain amount of effort in the modernization process. The way this effort is calculated from the identified PoMs is specified in the Rating Model.

In this version of the Rating Model we limit ourselves to the change identification through difference analysis (§2.4.1) for quantifying the impact of modernization. Although the remaining issue of change propagation is ignored, we emphasize that there are dependencies between the separate Points of Modernization and their corresponding legacy and SOA entities. This will lead to certain kinds of impact propagation, such as the so-called indirect impact [14]. For example, the application of a modernization strategy on one set of legacy components can affect other related components as well, initially not included in any PoM. This will result in the need for creating an additional Point of Modernization covering them, resulting in more overall effort. At the same time the coupling of a component to data structures and routines that are to be changed as well can also influence the overall effort. Isolated changes may cost more effort than when they are combined as is pointed out by Bayer [6]. These effects of impact propagation have to be considered before extending the Rating Model.

Modernization effort classification

We choose to organize the Rating Model by grouping the modernization strategies and their corresponding effort according to the following prominent modernization concerns: **identification**, **componentization** and **integration**. This categorization is based on the modernization strategies presented in section 2.4.2 and the modernization issues from section 2.4.2.

Strategy effort indicators and rating aggregation

As was mentioned earlier in this section, we have chosen to base the production of an effort estimation, in the Rating Model, on the discovered set of Points of Modernization. This effort valuation of each PoM is determined by two factors. First, a set of **indicators** is used to rate the impact of the modernization strategy chosen for the PoM within its category. Second, the category is of importance to which the modernization strategy belongs, from the ones described above.

The values produced by the indicators are aggregated following the Rating Model categories to produce an effort estimation on the ordinal scale. The ordinal scale is the result of the choice to order the impact of the modernization strategies only on an ordinal scale

(as opposed to interval and ratio scales). For instance, white-box strategies such as restructuring have a higher effort weight than a black-box strategy such as wrapping. With the current knowledge only an ordinal scale is feasible. In order to increase the accuracy by giving an interval scale grading of categories and their strategies more research has to be done on the relation between them. This issue is further explained in the chapter on future work, chapter 6. The set of useful indicators for the componentization concern is given in table 3.1 and for the integration concern in table 3.2.

PROPERTY	**INDICATOR**
Autonomy	set of data used by given function [89] function to (global)variable ratio [68]
Separation of concerns [63]	degree of scattering [102] decomposability [20] feature-to-function ratio [68] multiplicity business rule-to-code block [89] presence of design patterns [3]
Flexibility [4]	Service granularity [89] code similarity dendrogram based on features=identifier names [103] impact domain (call-graph subfunction propagation) [90] dominance tree from call graph [6]
Statelessness [89]	data-usage [89, 77] number of stateless/stateful functions [68] static data containment [90]
Dependency on environment [89]	Commercial software dependency [63] dependencies on operating systems, databases, filesystems [63]
Coupling	number of outgoing GO-TO statements from the mined component [90] number of global/local vars [68] coupling to functionality outside of the component [103] number of cross-cutting points [102] use of dynamic code collaboration patterns [100]

Table 3.1: Componentization indicators

Among the indicators, we make a distinction between two categories – **Level1** and **Level2**. A Level1 indicator can only give a relative information about the effort on an ordinal scale. While a Level2 indicator makes it more absolute and improve the estimation to an interval scale. This is the case, for example, with the indicators of "scattering" and "impact domain size", mentioned later on. Used on its own, a high scattering value gives a relative idea about the effort compared to a low scattering value. However, only when this scattering is combined with the absolute size of the respective impact domain (e.g. lines of code), can both estimations be compared on an interval scale. The reason for this is that combined the two indicators make it possible to order a single complex change (high scattering and small impact domain) can be compared to a simpler but extensive change (low scattering and large impact domain). It is furthermore obvious that using only the categorization of the strategies without their indicator based evaluation, also only an ordinal scale of estimation can be achieved.

PROPERTY	**INDICATOR**
Coupling	interfacing styles(socket, RPC, signal, pipe, file I/O) [23] interaction model type (session based vs. request/response) [21]
Abstraction	interface-to-implementation coupling [23]/design patterns [3]
Contracts	feature composition and relations [68] interface complexity (number of used data types) [90] interface definition [103] the input/output interface parameters of component [data-flow analysis] [12] unidirectional/bidirectional integration [1, 23]
Architectural mismatches [10]: = infrastructure expectations = data manipulation = communication model = message data model	 connector taxonomy [69] Business Object Model differences [43] communication direct/hub-and-spoke [8], protocols [7] logical data model, common message model [4]

Table 3.2: Integration indicators

Rating Model input configuration

We configure the Rating Model to use two sources of input. One with information about the legacy system and one with information about the target system. These correspond respectively to the domain and range of the modernization transformation function as described in the assumptions of section 3.2. In the same section, we also noted the fact that the domain and range encompass very large spaces of instance systems. Which exact instance from that space is the target design depends on the system requirements. Narrowing them down makes it possible to produce a more accurate estimation for the specific situation at hand. We make this possible by enabling the use of **input parameters** for specifying the desired end-system architecture.

We have already described the type of input information being gathered about the legacy system in the form of Points of Modernization. But in order to be able to calibrate the Rating Model also for a more specific target Service Oriented Architecture we configure it to be able to receive input about that system's configuration. We do this by grouping the above-mentioned effort indicators according to the SOA **property** they promote, see tables 3.1 and 3.2. The choice for these properties is based on the SOA design principles described in section 2.2. The input about the target SOA system consists, thus, of a set of desired properties and the degree to which each should be achieved. The choice of property translates into a choice for one of the corresponding indicators to be included in the Rating Model for the evaluation of the modernization effort. The desired degree can then be used together with the resulting effort estimation for an optimal choice in the cost/benefit analysis described in the problem statement (chapter 1) and in the scoping of the research in section 3.1 by comparing the effort/gain predictions for different sets of input parameters.

To give an example for the concrete case of the *Coupling* property, we distinguish between three degrees, ordered from high to low, given bellow. For each of them, we also

show how it manifests itself on the application level through the coupling classification by Myers [72]. The rating scale for this input is also ordinal, because a higher accuracy is superfluous as it would just be lost when used in the ordinal Rating Model.

High: high coupling between components exists both in the application logic and in the data models. High coupling in the application logic manifests itself through the use of global variables (Common coupling) and direct calls between components (Control/Content coupling). The coupling between data models is demonstrated by Data and Stamp coupling, where components share data type definitions, which makes them dependent on each other.

Medium: only high coupling between the data models of different components exists, while their application logic is loosely coupled. A loose coupling between the functionality of components is introduced through the use of contracts (interfaces) or through design patterns such as the Mediator.

Low: the components are loosely coupled both in their application logic and their data definitions. The additional decoupling between data models in this level is demonstrated through the use of a canonical data model to which each component can make its own mapping for internal use. An possible architectural organization for this level of coupling, where both the data and logic of components are decoupled, is an environment based on messaging with a canonical data model of the for the communication messages.

Concluding the input configuration, it can differ which indicator will be included in the Rating Model or will be evaluated as part of the effort estimation. It depends on the SOA property requirements and the degree they should be present in the end-system. The selection of these is part of the configuration and calibration process of the framework.

Rating Model effort estimation variables

Here we configure the internal workings of the Rating Model valuation. We presented the functions $EFFORT()$ and $GAIN()$ in definitions 3.1 and 3.2, which are evaluated in the Rating Model. The variables in these functions are the input variables of the Rating Model and through them also the effort indicators. The cost/benefit analysis is performed by evaluating these two functions for a given set of input variables and weighting them against each other. By changing the set of used variables and their values different situations can be considered for finding an optimal choice.

The variables of the gain function $GAIN()$ follow from the scope definition in section 3.1 and the configuration of the Rating Model input. It is a function of the SOA properties. For the concrete valuation function we assume a simple weighted sum of their achieved degree, where the weights are specific to the modernization goals and proportional to their relative importance. Thus, we write:

$$GAIN(flexibility, coupling, ..) = w_f * flexibility + w_c * coupling + ... \tag{3.1}$$

The effort function $EFFORT()$, however, is the one this framework concentrates on and is thus more elaborate. It is based in the Rating Model and it depends on the effort indicators

and the effort classification categories presented earlier in the configuration section. We use the following definitions:

$$EFFORT(S_i,N,C,I) = N(S_i) \oplus C(Nb_i,Ni_i) \oplus I(C_i) \oplus S(Ni_i) \quad (3.2)$$

$$N(S_i) = k * P_{unclassified} * |S_i| + m * P_{infra} * |Nb_i| + n * (1 - P_{infra}) * |S_i| \quad (3.3)$$

$$C(Nb_i,Ni_i) = f(|Nb_i|,|Ni_i|,scattering,|impact\_domain|,coupling) \quad (3.4)$$

$$I(C_i) = f(|C_i|,implementation\_coupling,interface,mismatch) \quad (3.5)$$

$$S(Ni_i) = f(|CCP_{Ni_i}|,\{SOA\_capability\}) \quad (3.6)$$

where

S_i = set of legacy entities considered for modernization

Nb_i = set of legacy entities identified as business logic for set S_i

Ni_i = set of legacy entities identified as infrastructural logic for set S_i

C_i = set of components resulting from the componentization for set S_i

CCP_{Ni_i} = set of cross-cutting points for set of entities Ni_i

$N()$ = effort estimation function for the identification concern group

$C()$ = effort estimation function for the componentization concern group

$I()$ = effort estimation function for the integration concern group

$S()$ = effort estimation function for the satisfaction of SOA concerns group

$P_{unclassified}$ = percentage of unclassified legacy entities

P_{infra} = percentage entities identified as responsible for supplying infrastructural services

The following paragraphs clarify the relations chosen above with a few important remarks. For the $EFFORT()$ function, equal importance is assumed for the three concern groups and thus weights of 1 are chosen in equation (3.2). In addition to that, the operator $\oplus$ is used to indicated the aggregation of the effort of the different concern groups. As was already mentioned in this section, the relation between the estimated effort for the different groups can not be specified quantitatively yet and only an ordinal scale can be used. This is the reason that results of the separate groups cannot be added up directly yet.

The dependency specified in equation (3.3) is the result of assumption S-1. We recognize two subsequent levels of identification of system entities. In the first place there is a division between classified and unclassified entities. The ratio between these is signified by the percentage $P_{unclassified}$. Classified entities are those for which it is possible to establish their role in the system as belonging to either one of the categories on the second level – infrastructural- or business logic. The ratio between these two is given by P_{infra}. This division creates three categories each of which will be treated differently in the subsequent modernization and thus generate different effort:

- $P_{unclassified} * |S_i|$ – proportional to the number of unclassified entities and source of effort for the identification of their role, possibly manually

- $P_{infra} * |Nb_i|$ – proportional to amount of business logic the infrastructure part interacts with
- $(1 - P_{infra}) * |S_i|$ – proportional to the number of business logic related entities

For each of these components, a weight coefficient should be chosen such that it fits experimental data. In equation 3.3 these are signified by k,m, and n, but the relation between the three components does not need to be linear. The same holds for the function specifications of $C(b,i)$ and $I(c)$, which only give a general indication of the dependency. We only want to indicate that there is a direct correlation to the variables. The concrete choice of weight factors is left for the instantiation and calibration phase. We give an example of such an instantiation in the next chapter with a concrete case study legacy system.

Another important notice about the effort quantification equations is that they are all dependent on the set of legacy entities considered for modernization S_i. This means that $EFFORT()$ can be used to investigate the best phasing of the project as well. By considering first a small initial set and monitoring and how the effort and gain change as this set is extended with more legacy entities. This way an analysis similar to the one given in figure 3.2 can be produced.

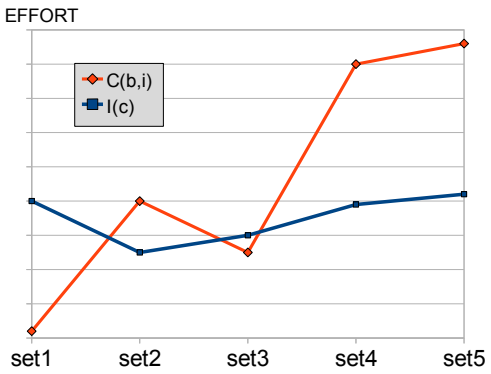


Figure 3.2: Componentization- and Integration effort over increasingly larger legacy entity sets

The figure also shows the difference in effort needed for the modernization of the application logic supplying infrastructural services such as communication and logging and business specific logic. The effort for the former has a high initial value, but once modernized these facilities requires few extra effort with the extension of the legacy entities set. The latter is characterized by a steady growth in effort.

3.3.2 Relationships

In this subsection we describe the configuration of the relations between the different models of the framework. These are a selection from the information sources used to capture the characteristics of a modernization case by the different existing approaches. As was described in the scope section 3.1, the framework is concentrated around the Business and Application domains. So these are the domains for which models are defined as follows:

- Business Model

- Architectural Model, Integration Model
- Service Model
- Development View, Process View
- Rating Model

We keep the relation between the Business domain and the Application domain as is shown in figure C.17. All the information from the Business domain is contained in a single Business model. However, the Application domain is captured in two separate models containing information about two separate concerns – Architecture model concerned with Componentization and Integration model concerned with Integration. Together they form the as-is model of the legacy application.

To these as-is models we relate two models of the to-be system - the Service model and the Development View/Process View. We introduce the additional Service Model based on our assumption C-3. It is positioned between the Business and Application models and has a correspondence relation to both in a similar way to the relation depicted in figure C.6. The Development and Process views capture again application domain information, but about the SOA to-be system. They are also split based on the concerns they model – Componentization and Integration respectively. Both are expressed through Service Component Architecture entities from figure C.2.

Finally we have the Rating Model. It consists of two parts. First is the rating aggregation part, which contains the strategy and indicator entities. Second is the part containing the Points of Modernization mentioned in the previous subsection. These are also related to the rest of the models, specifically the Development/Processes views and the Architectural/Integration models, and thus function as an interface to the rating part with input about the legacy system. The complete overview of the resulting relationships configuration is shown as part of the framework in figure C.21.

3.3.3 Control Model

In the configuration of the Control Model we decide on the workflow to be followed when using the framework. This is why in this subsection we present the two major parts that define it – the order of model construction and the choices for approaches to achieving particular steps.

We decide for the analysis process to follow a simple pipe-and-filter configuration. This means that the order of model construction is sequential, with each step using the results of the previous one. This is motivated by both ADM where each model has its roots in a lower/higher level model and the steps described in the modernization methodologies Renaissance and SMART, section 2.3. The configuration of the steps of the Control Model consists of the following four phases, which apply for all the three domains of the framework:

Reverse Engineering First identify the available sources: code, documentation, deployment configuration, maintenance history. Then gain structural(as opposed to semantic) knowledge of the legacy system architecture by constructing its static(as opposed to dynamic) model with the concepts specified in the Meta-model(table C.18).

The result of this phase for each of the three domains is an as-is Model, Business-, Application- and Technology respectively

Restructuring Taking the as-is models, result of the Reverse Engineering phase, in each of the domains as a starting point, restructuring is done to identify the desired end-system organization. This is done according to the SOA design principles and using the known requirements for the end Service Oriented Architecture

Gap Analysis The analysis consists of three important parts. First make an inventory of the differences by comparing as-is to to-be models in each domain in the form of Points of Modernization. Second, assign a set of possible solutions to each PoM and propagate the impact of the change within the same or to a lower domain if necessary. And last, give a valuation to the end-result by using the decision variables in the Rating Model

Forward Engineering This phase considers the bridging options, result of the gap analysis phase. The end-system requirements and the limitations they impose are percolated up the domains starting from the Technical domain. There, first SOA environment restrictions are taken into consideration and their impact on the Application domain is evaluated. In this way a refinement of the effort estimation made in the previous phase is produced

Approaches for each of the above-mentioned phases tend to divide in two groups – automated vs expert-driven – as is the case with ADM and SMART described in section 2.3. We choose to use techniques that are simpler and produce more crude results, but are automatable.

Business Service identification can be done top-down or bottom-up [31, 76]:

1. Top-down: identify needed business services based on a business domain analysis and map them to existing components, by searching which components capability could satisfy the need
2. Bottom-up: identify the business needs by looking at the capabilities offered by the components

The business domain analysis part of the top-down approach involves domain expert participation. Although it has the disadvantage to be less accurate, the bottom-up approach is preferred in the further configuration of the framework because of its suitability for automation and less supervision.

Constraint percolation
TO-BE system requirement based constraints percolate in the direction opposite to the impact(see RMM and fig.C.12). This is recognized and part of the overall process model in figure C.22, but will not be further explored in this research.

Automated analysis type
We have chosen for static structural analysis. This is the most basic type of analysis, but at the same time the one most prone to automation. The analysis could be extended to dynamic and even semantic, but they require an increasing degree of expert supervision.

3.4 Framework specification

This section presents the complete framework for modernization effort estimation based on the scope, assumptions and configuration from the previous sections of this chapter. It consists of a set of deliverables forming together the framework that can be instantiated, refined and extended. The contribution of the framework is twofold. In the first place, it combines and integrates existing approaches and concepts. Secondly. these are then used as a base for the new effort estimation Rating Model.

First we describe the meta model of the framework in subsection 3.4.1, containing the class definitions and relationships. Following, in subsection 3.4.2, is the description of the process of instantiating the framework and its control model. Finally, we present the Rating Model result of the framework configuration.

3.4.1 Framework Meta Model

This meta model contains the Class Definitions and Relationships part of the framework. The Class Definitions consist of a categorization of entities from the existing approaches divided over the three effort factors described in the scope: Legacy system (Appendix C.18), SOA target system (Appendix C.19) and Modernization strategies (Appendix C.20).

Secondly, the relationships are presented that the framework defines between the different models and the entities they contain. These were initially configured in subsection 3.3.2. Here we present them in detail in UML notation in figure C.21. For each model, we also show the major entities from the meta model that it contains. Through these entities the relationships between the models are concretely defined. In the figure is also indicated the distribution of the models over the framework domains. The layering is similar to the ADM approach, with the Business domain on top.

3.4.2 Framework instantiation process

The instantiation process of the framework, that prepares it for the effort estimation analysis, is depicted in two figures, C.21 and C.22. Figure C.22 shows the global order in which the models of the framework should be created. It is based on the configuration of the Control Model and shows its 4 Phases split over the 3 domains of modernization described in the scope. This defines twelve different activities in their intersection. These are shown as numbered green rectangles in figure C.22. Each of these steps identifies and separates out a particular goal for a particular domain. For example, step#1 is concerned with extracting or reverse engineering the as-is legacy system in terms of the business domain entities specified in table C.18 of the meta-model. The figure also specifies the set of input and output models for each of the above-mentioned steps. A central role is played by the **PoM Model**. For each of the domains, this model contains the set of Points of Modernization identified through the Gap Analysis comparing the as-is to the to-be models.

It is further also important to note the indicated impact propagation direction and the requirements percolation in the opposite direction. This is based on the assumption that the Business Domain models are implemented through the Application Domain models. And

this means that any changes in the former may propagate and impact the later. The same also holds for the relationship between the Application Domain models and the Technology Domain models. This is indicated by the assumption M-3 in section 3.2.

Compared to the higher-level specification of figure C.22, figure C.21 is a concrete instantiation resulting from the framework configuration of section 3.3. Based on the configuration of the Rating Model, it shows the concrete models to be used to supply it with the needed input. The numbers next to the inter-model relationships indicate the order in which the models are to be produced.

3.4.3 Effort estimation Rating Model

The Rating Model configuration was presented in section 3.3. The resulting model is shown here in figure C.23. It is based on aggregating the effort estimated for the modernization strategy attached to each Point of Modernization. This aggregation is first done in the three concern groups – **identification**, **componentization** and **integration** – with their strategies indicated in the meta model table C.20. Next to these groups, aimed at the achievement of separation of concerns, there is a forth group concerned with the satisfaction of the end-result system requirements that may not have a correspondence in the legacy system. These are the SOA capabilities also mentioned in the meta-model in table C.19. The Rating Model thus contains three levels: at the highest level are the concerns, at the middle level the modernization strategies that belong to that concern, and at the lowest level the effort indicators. These were already described in the framework configuration section. For this version of the Rating Model we have kept the choice of indicators limited as there is not yet enough empirical evidence to indicate the exact relation of indicators to modernization strategies to be able to distribute them correctly over the categories. Both effort and gain can be aggregated according to the same scheme presented here.

The configuration section of the Rating Model (3.3, *"Strategy effort indicators and rating aggregation"*) explained the choice for an ordinal scale of strategy rating. This order is also depicted here in the diagram of the Rating Model by the order of the concern groups they belong to. In addition to their relative effort, this order also indicates the succession in which the groups are evaluated for an increasing accuracy of the effort estimation. Starting from the left with "Identification", the strategies require increasingly less effort. This is based on the way these strategies are implemented as was presented in section 2.4. Here we state the main points:

- Identification: may require a large amount of manual effort
- Componentization: great deal of restructuring, which requires also more insight into the implementation
- Integration: merely mismatch alignment through wrapping and adapter addition
- Concern satisfaction: merely plugging in or removal of functionality similar to the COTS strategy and to some degree wrapping

Following, we will go into the rating for each of these groups and their strategies. We will conclude with the role design patterns can play in the rating process.

Identification

The result in this concern group is an estimation of the effort needed for the Reverse Engineering activities in the modernization. As presented earlier in figure C.22, the activities of this phase are present in all three domains – Business, Application and Technology and thus, identification effort is required in each of them. This is why the rating model reserves a place for strategies for each of the domains. The business category from figure C.23 is about identifying Business Model entities and Technology about building a technology model of the system including such as operating systems, etc (see Meta-model entities figure C.18). We acknowledge these aspects and give their place in the rating model. However we concentrate on the Application aspect, as was described in the scope in section 3.1 and give its buildup of strategies and indicators.

The rating for the Application Domain is based on strategies for the classification of the legacy entities according to two criteria. The first has to do with the role of the entity – component (**COMPONENT**)or connector (**CONNECTOR**). While the second makes a distinction based on the purpose – business application logic (**APP_LOGIC**) or infrastructural functionality (**INFRASTRUCTURE**). This is expressed in assumption S-1 (page 32), stating these two types of functionality. The combination of these two criteria with two alternatives each, creates four categories for the identification of legacy entities. Once it has been identified for every legacy entity in which category it belongs, the rating can be done using the two indicators shown in the figure – the percentage classified legacy entities and the percentage that implements infrastructural functionality.

The reason for choosing this division is that specific business logic and more general functionality can benefit best from different modernization strategies. Thus, investing effort early on in identifying them correctly, will pay off in less effort in later phases as it would be possible to apply the most appropriate strategy. The APP_LOGIC entities are more likely to be kept during the modernization and bring about effort in the componentization and integration concern groups. Their functionality will most likely be encapsulated into services and integrated to work together to perform the same business processes. On the other hand, the infrastructural facilities such as communication and resource discovery are more likely to be replaced as a whole in favor of a new infrastructure/framework that supplies the same services. This, as opposed to reengineering it. The former will result in effort in the integration and satisfaction of concerns groups, while the later will result in effort for componentization.

Concluding this concern group, we need to make an important remark about its effort output and how it will be influenced in case of an extension of the framework. Currently, it has been setup for the ordinal scale. As is shown in equation 3.3, this means using the enumeration of PoMs to be identified, a qualitative relationship between them such as infrastructure entities require more effort to identify than business logic and giving an effort estimation based on the size of this set and its composition. However, if the rating is extended to an interval or ratio scale, the degree of automation that can be applied for the extraction of the models in the Reverse Engineering phase will start influencing the effort. This is a consequence of the difference in realization effort needed between automated and manual expert-driven approaches. This was discussed in section 3.3.3 and throughout

chapter 2. Once a unit of measure is introduced on an absolute scale such as costs, different weights will be used when translating to it for automated and expert-driven identification. For the expert-driven part this weight factor will be much greater for two reasons: greater time investment per PoM and greater costs per time unit.

Componentization

The componentization rating is concerned with delivering an estimation of the effort based on only local subsystem restructuring where needed. The modernization strategies that fall in this group are the more invasive white-box modernization strategies. The effort estimation resulting from this category gives a local estimate about restructuring subsystems in isolation. This means that different subsets can be considered for modernization together by only summing their corresponding efforts.

The goal of the strategies in this category is to achieve the rough architectural description of the application as required by the Service-Oriented paradigm. This means, separation of concerns, Business-to-IT alignment and SOA design principles. The selection of indicators from table 3.1 was thus made such that it would measure the effort of the changes the modernization strategies would involve.

Integration

Through this category the effort and gain for the system as a whole are estimated. This is done by rating the impact of the integration issues between the component subsets chosen after the componentization. The strategies here represent the possible actions that may be needed for Points of Modernization. They are all aimed at achieving a seamless whole by considering the separate components as a black-box. This is why these strategies are considered to have less impact than the componentization strategies. Internally there is also an order in the estimated amount of effort required by each strategy. In the diagram, the **Wrapper** strategy is the one requiring the least amount and the replacement of a components with a **COTS** (Commercial Of The Shelve) solution has the greatest estimated effort. A selection was made from the possible indicators from table 3.2, which can be used with each strategy.

SOA concern satisfaction

All the concern groups and their strategies described so far rate the effort in transforming existing assets through restructuring or integration. This covers the intersection of functionality present in both the legacy and the SOA system. The category of SOA concern satisfaction, on the other hand, is responsible for estimating the effort needed for handling entities that don't have an equivalent in either the as-is or the to-be system. This means that there are two strategies that could be applied – **ADD** or **REMOVE**. The former is necessary in case one of the SOA capabilities mentioned in table C.20 is required in the target system, but has no equivalent in the legacy system. For example the addition of a messaging capability to a system based on direct communication through API's. The latter strategy – REMOVE – is applied to legacy entities that are obsolete in the SOA environment. For

both strategies holds that their effort depends on the number of places they interact with the rest of the application as they represent cross-cutting concerns, described in section 2.2 in the subsection on "SOA Software Architecture". Thus, a quantifiable indicators for these strategies is the so called number of cross-cutting points (CCP) [102], used in equation (3.6) of the Rating Model $EFFORT()$ function. It is also mentioned in table 3.1 as an useful indicator for the *Componentization* concerns of which *SOA concern satisfaction* is a special case.

Rating through the use of design patterns
Design patterns play an important role in the forward engineering step of the target SOA [3]. In the reverse engineering part though, they are less useful. Their use is based on the fact that the presence of a design pattern can help to induce semantic knowledge from the structure only by assuming that the pattern structure was used with the intent of its rationale. So the presence of a pattern in the legacy system means two things:

- there is a clear structure, so restructuring will be easier
- the structure has a rationale(pattern forces). This can then be compared to the to-be situation forces.

The presence of a pattern and thus the certain degree of separation of concerns it achieves does not automatically mean that the involved components can be reused as is. It still might be the case that the whole composite would have to be rewritten from scratch. In such a case the pattern's only added value is that it aids in the system understanding. A pattern could also indicate a less optimal solution, but one that is easier to achieve by following the patterns structure. For example including in the component all other classes participating in the pattern, although they do not directly have any relationship with the core functionality.

Chapter 4

Legacy Logistics System – effort estimation experiment

This chapter has two goals. The first is to demonstrate the instantiation of the Effort Estimation Framework for a concrete legacy system. The second goal is to evaluate the resulting effort estimation and through it the effectiveness of the framework that delivered it.

The chapter begins in section 4.1 with the definition of the experiment performed to reach the above-mentioned goals. This experiment is structured according to the guidelines by Wohlin [99] on experimentation in software engineering. Based on the experiment definition, section 4.2 describes the planning for performing the experiment. This includes the hypothesis and design of the experiment, the description of the legacy system used as a subject and the instrumentation. The execution of this planning is then presented in section 4.3 Experiment operation. The results of the experiment are presented in section Data Analysis (4.4) noting the important facts about them. An interpretation of these results is given in section 4.5. We end the chapter with a discussion and the conclusions from the experiment in section 4.6. This last section also containing ideas about future work for extending the experiment setting.

4.1 Experiment definition

The rest of the chapter describes the experiment with object of study the *Effort Estimation Framework* from chapter 3. The purpose is to evaluate the effectiveness of the suggested framework and the effort estimation results it produces. This experiment is a single object study done from the perspective of the researcher. The single subject of the study is the Legacy Logistics System undergoing the experiment through a prototype software system.

4.2 Experiment planning

4.2.1 Context selection

The experiment is executed by the research student himself with no test subjects. Professional software engineers were involved, but their contribution was not related to the test

data itself. The tests are performed in an off-line situation parallel to a real life modernization project. Although the examined problem is from a real situation, it is constrained in size due to time restrictions. These constraints are both on the study object as well as on the study subject. For the experiment only part of the framework is considered and only part of the subject legacy system.

4.2.2 Hypothesis

To evaluate the effectiveness of the framework, the following null hypothesis is formulated: "According to the Effort Estimation Framework, the specified input Gain is unrelated to the framework output Effort". The experiment is designed with the goal to deliver empirical data to reject this null hypothesis in favor of the alternative hypothesis. That alternative is formulated as "The Gain is related to the Effort following the Effort Estimation Framework in an exponential relationship". This expectation was mentioned in figure 3.1, section 3.1.

4.2.3 Variable selection

In order to disprove the null hypothesis, the relevant parameters from the framework are selected as independent and dependent variables to monitor during the experiment.

Notation

Due to the restricted setup, not all framework variables will be involved in the experiment. We introduce a notation to emphasize that only a limited subset of the parameters included in the framework are being evaluated. The two main functions are $GAIN()$ and $EFFORT()$, defined in section 3.3.1. When these are only evaluated in relation to only some of their specified parameters, we respectively denote them by $Gain()$ and $Effort()$.

Independent variables

The controlled environment is created by the set of selected independent variables. The first of theses is the input factor of the experiment – **Granularity**. It is chosen, because it is a measurable entity proportional to the frameworks input parameters of Flexibility [8] and Composability (section 2.2). The Effort Estimation Framework defines these on their turn to be linearly proportional to the Gain – $GAIN(flexibility, coupling, ...)$ (equation 3.1). So for the experiment we have $Gain(flexibility)$ as theoretical cause to be examined through the treatment of the independent variable granularity, denoted with g (see Appendix C.28). For each treatment of g a test will be done executing the Effort Estimation Framework with it as input. The granularity is measured as the total number of components to partition the legacy system in and has a range of [0..5386] on a ratio scale.

The remaining set of independent variables will, however, be kept constant. These variables together with their value and measurement scale are:

- **Legacy components subset (S_i)**: value = *all*; scale = *Ordinal*
- **Rating Model concern**: value = *Componentization*; scale = *Ordinal*
- **Modernization strategy**: value = *Restructure*; scale = *Ordinal*

- **Model extraction methods**: described in the experiment design; scale = *Nominal*

Dependent variables
The experiment will monitor the development of two dependent variables:
- **Scattering**: indicator from the framework Rating Model; scale = *Ratio*
- **Effort**: indirect measurement based on the scattering; scale = *Ratio*

The function $EFFORT()$ is defined in equation 3.2. Because this experiment evaluates only the *Componentization* concern, we use only the $C(Nb_i, Ni_i)$ part of the equation. Furthermore, assumption iii on page 50 about the supplied LLS code means that $Ni_i = \emptyset$ from which follows that $Nb_i = S_i$. So for the effort estimation in this case study holds that:

$$EFFORT() \approx C(Nb_i, Ni_i) = Effort(scattering) \tag{4.1}$$

Because the Componentization concern is considered in isolation, no aggregation of effort estimations is done. This makes it possible to do the experiment on the ratio scale. The score however is an index internal for the indicator/modernization strategy combination. It cannot be related to an index produced for a different strategy with different indicator. The relation between these two is defined only on the ordinal scale by the Rating Model. Currently, when merging multiple indicators or categories the score has to be converted to the Ordinal scale by a custom scaling that divides the scores in a couple of groups. With the 100% being the maximal total effort.

4.2.4 Subject system description

The subject legacy system for the experiment was selected through stratified random sampling. A set of criteria were important for the characterization of the group from which a system was randomly chosen. These criteria are typical for the problem domain of the Effort Estimation Framework: legacy application, implemented in a procedural language, complex interdependencies, in need of modernization indicated by the maintenance team.

The chosen system is the **Logistics Legacy System – LLS**. It contains many subsystems with heterogeneous implementations and with many interdependencies. The devision between them is based on their responsibility domains. Three of these subsystems are given bellow with their corresponding implementation language and runtime environment. They share the following characteristics: *a*) transaction oriented, *b*) work on a common data model (in DB2) and *c*) rely on some types of infrastructure support (CICS vs. IMS-TM).
- **Request Handling** – CICS/Cobol
- **Warehouse Management** – IMS/PL1
- **Location Planning** – IMS/PL1

We concentrate on the Request Handling sub-system, which will be further referred to as **RH**. It is concerned with the management of incoming requests for parts and, thus, the implementation code includes request process flows and inventory management. The code considered as belonging to this sub-system was exported from the LLS codebase and supplied by the maintaining team.

A brief examination of it gives the following points of consideration:

i. the relation of compilation to run-time units is one-on-one. One PROGRAM/PROCEDURE per file. This means that the units in a fine grain representation of the code can be seen as both files and programs.
ii. No built in environment facility for program address passing. This has had to be custom implemented as part of the code base. This code falls into a different category when it comes to modernizing possibly to other platform/language with different capabilities. It contains no business functionality but utility
iii. we assumed that the code contains application logic and no infrastructural facilities

4.2.5 Experiment design

For each treatment of the granularity factor, the experiment instantiates four parts of the Effort Estimation Framework: the meta-model entities (section 3.4.1), the meta-model relationships (section 3.4.1), the analysis process steps and models (section 3.4.2) and the Rating Model (section 3.4.3). Here follows a list of the parts used for each of them.

Meta-model entities:

- Legacy – table C.18, columns *Business* and *Application*
- SOA – table C.19, column *Application*
- Strategies – table C.20, column *Componentization*

Meta-model relationships (figure C.21):

1. Business Model: processes/activities
2. Architectural Model, Integration Model: Legacy Components and Connectors
3. map Business Model to Architectural Model: which ”Legacy Components” implement each ”Business Process”
4. Service Model: information(cluster around data types), business(logic), general(rest) [96]
5. Rating Model

Analysis process (figure C.22):

- steps: 1, 2, 3, 4, 5, 6
- models: AS-IS Business Model, AS-IS Software System Model, TO-BE Business Model, TO-BE Software System Model, Business PoM Model, Software PoM Model

For each test run in the experiment, three steps are executed to build the models described above. First is constructed the AS-IS Business Model. Followed by the extraction and abstraction of the AS-IS and TO-BE Architectural models. In the third step, the two are combined giving the PoM Model. This is used to calculate the dependent variables of *Scattering* and *Effort*. These three steps are described below.

Business Model

The Business Model of the RH subsystem is constructed through extraction and abstraction of the source code. The end result is a representation of the business process flows in the Business Process layer from the *Development View* of figure C.6. The approach is a simplified version of the one described by Zou [104]. While extracting this view we keep

the traceability to the code level, the importance and purpose of which is described by Xiao [101]. This means that the link between each business process and the PROGRAM entities implementing them is kept (figure C.24).

In the variable selection is described that the subset of legacy components considered for the experiment will be constant for all treatments and include all the available components. This results in a constant **number of business processes** that will be extracted in this step of the experiment.

Architectural Model

The Architectural Model of the RH subsystem is constructed by abstracting legacy components based on the coupling between programs. The end result is a representation of the legacy components in the Service Components layer from the *Development View* of figure C.6. While extracting this view we also keep the traceability to the code level (figure C.25).

Complete as-is model and PoM Model

In this step we first build a complete as-is model of the system by linking the processes from the Business Process model to the components implementing them in the Architectural Model. Figure C.26 shows conceptually how this is done. We use the traceability links of both models to the code level to relate the entities in them. This mapping between the two models is used to identify the Points of Modernization. For each business process there is one Point of Modernization with an attached strategy of Restructuring. For each of these PoMs we measure the effort indicator of *Scattering* that belongs to the *Separation of concerns* property. Scattering is defined as the number of components that implement a given business process and thus, the scattering of business functionality over legacy components. We use the following notation:

$$
\begin{aligned}
D_g &= \text{distribution of the scattering over the business processes for granularity } g \\
D_g &= \left[\ \overline{bp}\ \ \bar{s}\ \right] \\
\overline{bp}_g &= \text{business process distribution component of } D_g \\
\bar{s}_g &= \text{scattering component of } D_g
\end{aligned}
$$

For a given granularity level g, we use the scattering distribution D_g and use it in an aggregation function to calculate the effort for the *Componentization* concern. The concrete specification of the effort as a function of the granularity is as follows:

$$Effort(g) = \sum_{i=1}^{n} bp_{g,i} * s_{g,i}^2 \tag{4.2}$$

here, n is the size of the distribution matrix D_g, $bp_{g,i}$ - the number of business processes from that distribution, $s_{g,i}$ - the corresponding scattering index. For the construction of this valuation function a choice has to be made about the second part of the product, $s_{g,i}^2$. It represents the **scattering weight**, which indicated the effort relation between different scattering indexes. In other words, how much more(or less) effort does a PoM with a scattering

indicator value of s_n require than one with scattering of s_{n-1}.
The choice of scattering weight relation we have made here, is based on two assumptions:

- same scattering index means same scattering weight. However two PoMs with the same scattering index do not have to valuate to the same scattering weight. Think of other factors such as size, structure etc that could influence the relation.
- we assume a fully connected graph between all the implementing programs of one business process. This means that splitting them into $n+1$ components would lead to $\frac{n*(n-1)}{2}$ connections. This is of order $O(n^2)$ so we also assume a quadratic relation for the scattering weights.

4.2.6 Instrumentation

The above-mentioned experiment steps will be fully automated with a prototype software system. The process of model instantiation and evaluation is implemented in this prototype in the following phases:

1. Extraction+Abstraction ->AS-IS Models:
   - Business Model
   - Application Model
     - Architectural Model
     - Integration Model
2. Restructure ->TO-BE Models:
   - Service Model+SOA Infrastructure
   - Development View
   - Process View
3. Analysis
   - Rating Model

The flow between these phases and the data that is saved is shown in figure C.27. Minor preprocessing of the source code files was needed before they could be parsed in the Extraction phase. This was done with a custom written Java processor and involved the removal of invalid characters from the source such as EOF. For the Extraction phase itself, the Relativity Modernization Workbench®[1] was used. The extracted information was then exported in Excel format. A second custom Java application was used to transform this data into a format that could be imported in the tool used for the following phase of Abstraction. For it as well as for the Restructuring and Rating Model analysis, MATLAB®[2] was used.

4.2.7 Validity evaluation

Here are presented the considered threats to the validity of the experiment, mainly internal and external. The concepts under study in this experiment are shown according to [99] in Appendix C.28.

[1] http://www.microfocus.com/products/modernizationworkbench/index.asp
[2] http://www.mathworks.com/products/matlab/

None of the main threats to the internal validity apply to the designed experiment, because it is automated through the software prototype. This means that the application of the treatments does not change with time, they do not have any side effects on the subject system and there are no human factors influencing the test results. Thus, the experiment offers a high degree of certainty that the applied treatment is the cause of the observed effects. The single exception is the instrumentation. It forms however no big threat as the tools used are considered reliable. Relativity is a commercial workbench of high quality. The main parts of the custom MATLAB code are to be found in appendix B for inspection.

The external validity of the experiment is also not jeopardized, which increases the possibility to generalize the results. Again, because of the automated process performed by the prototype, the test data is not susceptible to the moment in time the tests are run or any characteristics of the population of subject participants. In addition, the subject legacy system has been chosen so that is representative for the problem domain targeted by the Effort Estimation Framework (section 4.2.4). This makes any generalizations based on the test data valid for the domain of similar legacy systems.

4.3 Experiment operation

As an input Application Model was used the call-graph of the RH subsystem. The Relativity Modernization Workbench® was used to parse the legacy Cobol code and produce a caller/callee map. That was exported in the form of a report, an excerpt of which is shown in appendix C.29. This callmap was converted to a directed graph represented as an adjency matrix in MATLAB, which we will further refer to under as **M**.

The unique id of each node in this graph representation is shown in column 2 in figure C.29. Each such node corresponds to a PROGRAM entity from the code. The edges represent the calls between the programs. The graph contains in total 5386 nodes and 8349 edges. It is important to note that the caller/callee map produced by Relativity contains only information about which program calls/is called by other programs. It does not extract the order in which these are called. So the nesting part of the flow is preserved, but the order within a program is not. This means there is no particular order (pre-, in- or post-) in which the tree nodes should be read, but for our purposes this is no limitation.

A second observation worth noting is that M was found to be an acyclic graph (DAG). This fact simplifies some of the algorithms used in the Business Model (4.3.1) and Architectural Model (4.3.2) sections, but does not invalidate their generality.

4.3.1 Business Model

As input for this step we used the adjency matrix M. To extract an approximation of the business process flows in the RH subsystem, the code from Listing B.1 was used. According to it, we first identify the set of **entry-nodes**. These are programs that are not called by any other program but only invoke others. These root nodes are considered the starting point of business processes, which can be invoked from outside or through the system's front-end. In total we identified 1026 such entry-point nodes out of a total of 5386. Once the entry-point nodes are identified, we extract the subtree of nodes from the call-graph with

root each of the entry-nodes. This produces the end-result **subtrees**. This is a set of size 1026 containing the sets with the nodes belonging to each business process. This result is only an approximation of the real business processes of the RH subsystem, because of the call-graph limitations described in the introduction. This means that the extracted business processes do not completely depict the order of activity execution, but that for our purposes this limitation does not influence the end result. The traceability to the code level is kept by preserving the ids of the nodes in the subtrees, which uniquely identify the implementing programs.

Visualization of the extracted process flows was done with the code from Listing B.2. Three such flows with increasing size are shown in figures C.30, C.31 and C.32. The one shown in figure C.32 contains the most nodes in the whole RH subsystem. For the resulting process trees, it is common to share subtrees. This could be used as an indicator for the choice of granularity level (see subsection 4.3.2) so as to optimize reuse.

4.3.2 Architectural Model

Here as well, the adjacency matrix M was used as starting point. It was process with the code from listing B.3 with the goal of identifying clusters of program nodes to form components. First the distances between the different programs in the call-graph was calculated, forming the **DIST_MATRIX**. Then this information was used to calculate their hierarchical clustering structure, similar to the approach of Fan-Chao [34], which clusters the nodes closest to each other first. For the concrete implementation we used weighted clustering (WPGMA).

The result of the clustering is a *dendrogram* showing the way the program nodes can be gradually clustered together forming larger components. This dendrogram, which we will further refer to as **TREE** is shown in figure C.34. From this, clusters can be defined by deciding on the level of granularity at which the tree should be cut up into components. Such a sample devision is shown in figure C.35, where the different resulting components are colored differently.

4.3.3 Complete as-is model and Rating Model

The Rating Model is instantiated and used to estimate the effort for modernizing towards achieving the SOA property of *Composability*. This is in our case the input parameter for the Rating Model about the to-be system as specified in 3.3. Figure C.26 shows how this step was performed. On the left side, we have the set of **subtrees** and on the right we have the **TREE** clustering structure. From these, we calculate the scattering using the code from listing B.4.

4.4 Data Analysis

4.4.1 Rating Model indicator of Scattering

Descriptive statistics

- Relative **frequency distribution** for five of the treatments for the factor *g*: D_{300} (Appendix C.36), D_{500} (Appendix C.37), D_{1000} (Appendix C.38), D_{3000} (Appendix C.39), D_{5386} (Appendix C.40)
- **box plots** of the same five distributions (Appendix C.41)
- plot of the **median** for all distributions of the treatment of *g* (Appendix C.42)
- pot of the **mean** for all distributions of the treatment of *g* (Appendix C.43)
- plot of the **standard deviation** for all distributions of the treatment of *g* (Appendix C.44)

The distribution plots show the relative frequency (y axis) of the scattering index (x axis) in the set of business processes. All distributions display a logarithmic decay - high concentration of low scattering and increasingly less occurrences of higher scattering indexes. When we compare the distributions with an increasing granularity (D_{300}, D_{500} etc.), we also see that the scattering is very concentrated for low granularities, but that it spreads out to higher values with the increasing granularity. How this happens exactly is seen for the scattering indexes 1 to 4 in Appendix C.45, where the areas S2, S3, S4 display a ripple effect from low to high scattering.

Furthermore, from the frequency distribution plot for the extreme case of D_{5386}, it is noticeable that there are no more business processes that have a scattering index of 1. This means that in case of such a partitioning, every business process is implemented by two or more separate components. How this situation arises can be seen in Appendix C.45. At point P_1 the scattering index 1 starts dropping and for the same granularity treatment at point P_2 the scattering index 2 starts increasing with similar rate.

The most important fact to note about the scattering test data is based on the mean and median plots. Both plots over all treatments of the granularity show a clear increasing pattern. This indicates that there is a relationship between the treatment *g* and the dependent variable of the experiment *Scattering*. We will test this statement as part of the hypothesis testing section (4.5.1).

Outliers

The box plots and the plot of the standard deviation show that there is an increasing number of outliers, with an increasing deviation from the mean scattering value. Furthermore, although the mean value for each g increases, the scattering outliers increase at an even faster rate. They seem to form groups with similarly large scattering. This is best seen in the box plot for g=5386. There, we can see four groups (A, B, C, D) with similar very large scattering deviating significantly from the mean value.

Based on this observation, we decide not to dismiss these scattering outliers. They display a clear structure and are not sporadic. Furthermore, they are not the result of any threat to the experiment validity such as measurement faults (see section 4.2.5). This means

that they supply important information. An interpretation of their presence is given in section 4.5.

4.4.2 Resulting Effort

Based on the test results for the scattering, we calculate the effort for all the treatments of g according to equation 4.2. The graph of the produced effort estimation is shown in Appendix C.46.

The Effort graph shows an increasing growth for an increasing granularity. It has two important features. The first is the initial flat effort up to a granularity of 275 components. This is due to the fact that up to that point the total number of implementing components is so low that the implementation of every business process fits into exactly one component. This changes as the components become more fine-grained.

The second interesting feature of the effort graph is the kink around g=4250. To analyze it further, we divide the business processes in five groups. These groups are formed based on the scattering at g=3000. In the box plot for g=3000 (Appendix C.41) we showed that there are four groups with similar scattering indexes. We call them A, B, C and D from highest to lowest scattering. All the rest of the processes with lower scattering we put in the fifth group – Z. The mean scattering in each of these groups of business processes is followed (AppendixC.47). The graph shows a sudden and sharp increase at $g = 4250$ in the average scattering for groups A,B,C and D, the ones with highest scattering.

Regression Analysis

Because we are interested in the Gain as a function of the Effort, we will analyze the inverse of the Effort output. We investigate possible approximations of the relation between the granularity and the Effort. We do this for two data sets: **ALL** containing all the data points from the Effort function, **LIMIT** containing all the data points up to the kink phenomenon described earlier starting at g=3250. The regression approximations we consider are evaluate based on their RMSE (Root Mean Squared Error). We consider three models for the approximation:

- Polynomials: $a_1 * x^n + .. + a_n * x + c$ for $n = 1..8$
- Sum of sin functions: $a_1 * sin(b_1 * x + c_1) + .. + a_n * sin(b_n * x + c_n)$ for $n = 1..8$
- Power function: $a * x^b + c$

The data about the approximations from table 4.1 shows three things:

- the best approximation models on average for LIMIT and ALL are both based on periodic functions
- for ALL, the polynomial approximation delivers reasonably good results ($f = -4.065e^{-10} * x^2 + 0.002961 * x + 454.3$), but an increasing degree does not improve the fit significantly
- for LIMIT, the power approximation is a also very good ($f = 0.2066 * x^{0.6855} + 210$)

	ALL			LIMIT		
n	**Polynomial**	**Sum of sin**	**Power**	**Polynomial**	**Sum of sin**	**Power**
1	239.9	124.8	135.2	112	68.2	37.3
2	122.6	127.0	-	59.0	34.2	-
3	122.0	92.6	-	42.5	33.1	-
4	116.9	75.4	-	32.5	26.6	-
5	95.0	58.0	-	32.0	25.2	-
6	94.3	83.1	-	31.4	30.2	-
7	93.0	66.7	-	29.5	28.1	-
8	91.7	35.0	-	28.9	24.4	-
Mean	121.9	82.8	135.2	46.0	33.8	37.3

Table 4.1: RMSE of Regression models

4.5 Interpretation of results

In this section we interpret the effort estimation results from the experiment using the data analysis from section 4.4.

4.5.1 Effort development and relationship to Gain

To definitively disprove the null hypothesis, we assume that Effort and Gain are not related to each other. This means that the independent variable of the experiment - granularity and the dependent variable - scattering are assumed to have a normal distribution. The ANOVA analysis of all scattering distributions D_g for all the treatments of g shows a p-value of 0. From this we conclude that the scattering is related to the granularity variable. This was already suggested by the increasing mean value of the scattering (Appendix C.43).

By construction in this experiment, the Effort is related to the scattering ($Effort \sim scattering$, equation 4.2) and the Gain is related to the granularity ($Gain \sim flexibility \sim g$, section 4.2.3). When we use these two relationships together with the fact that the scattering is related to the granularity, we can conclude that the Effort is related to the Gain following the Effort Estimation Framework ($Effort \sim scattering \sim g \sim Gain$). This rejects the null hypothesis.

4.5.2 Trade-off analysis – Gain/Effort

The main application of the proven relationship is to support trade-off analysis in the planning phase of modernization. This becomes clear from the examination of the relationship between Effort and Gain, using the regression analysis made earlier. There, we analyzed the Gain as a function of the Effort, which means working on the inverted data points of the experiment results.

For the LIMIT data subset the trend is modeled well enough by the power function. On the other hand, for the ALL set of data, the overall trend is captured by the polynomial of second degree. It captures the trend in the data well and a further increase of the polynomial

degree does not significantly improve the fit. The difference between the two is that the approximation of LIMIT has no maximum and the one of ALL does. The limit of the derivative of the power function approaches 0, This means that the approximation never approaches a maximum. On the other hand the quadratic function does have a maximum, because its derivative is linear decreasing function. Both approximation can be seen in the graph in Appendix C.48.

The phenomenon at $g = 4250$ explains why although LIMIT has a power approximation it turns into a quadratic one when all points are considered. At that granularity, the complex processes with high scattering start disintegrating very fast with the increase in granularity. These processes have a high weight in the overall effort estimation and thus increase it sharply. This stops the growth that LIMIT displays and turns the overall relationship into a quadratic function. The power function for the whole domain would mean that an ever increasing effort would deliver unlimited increase in gain. The quadratic overall relationship however indicates that there is a maximum to the growth.

So the overall trend is quadratic, but for the more exact trade-off analysis the periodic approximation is more useful. It models the local fluctuations in effort, which appear to be periodic. This approximation by an eighth degree sum of sin functions is shown in Appendix C.48.

The trade-off analysis helps determine the best granularity level for the componentization of the legacy system. For this purpose, the optimal points have to be identified. These optimal points lie on the Pareto frontier (Appendix C.49) and are defined by the optimal combination of Effort and Gain. In this case, this means minimizing the effort and maximizing the gain. So the utility function to minimize is $\sum(1 - Gain) + Effort(g)$. The resulting graph is shown in figure C.50. It contains one global optimum (GO) and several local optima ordered in utility (LO_1, LO_2, etc). The global optimum is at the same granularity as the kink described earlier caused by the scattering of the more complex processes exploding. This means that beyond that point splitting the complex business processes up to increase the granularity is not worth the effort. So unless it is a requirement to reach a certain higher level of granularity, it is recommended to stay at the global optimum point.

The growth in gain per unit effort can also be used in the decision to increase or decrease the chosen level of granularity. This growth is displayed by the derivative of the regression approximation (Appendix C.51).

4.6 Discussion and conclusion

Some unexpected results were observed in this experiment. This discussion presents their possible interpretation and consequences. Still, the results of the experiment also confirmed the hypothesis about the effort/gain relationship and support a number of conclusions. These will be presented before concluding with suggestions about future work and extension of the experiment setup.

In the first place, the unusual outliers were observed in the box plots of the scattering distributions D_g (Appendix C.41). They form groups with similar scattering much higher than the average. These groups also have different growth rate of increasing scattering.

This seams to indicate that there is an additional factor that is related to the scattering - the complexity of the business process. This complexity is defined as the number of steps (executing Cobol PROGRAMs). More complex processes react differently to the increase in granularity than more simple ones. This indicates than an additional experiment parameter (independent variable) is in its place - the percentage of complex processes in the legacy system. It might be possible to use it as an effort indicator.

The second unexpected observation is in the discovered dependency between the Gain and the Effort. It does resemble to a certain degree the overall expected dependency from figure 3.1. However, the growth of the effort is not exponential as expected, but polynomial. One explanation for this could be that the effort growth is a system dependent characteristic related to the percentage of complex processes in it. As the experiment examined only one subject system there is not enough evidence to give a definitive answer. An extended research comparing different systems could clarify this issue.

A second reason for the limited polynomial growth of the Effort could be the constrained setup of this experiment. It considers only one factor for the Gain and only one effort indicator. The absence of certain factors in the experiment that are are inversely related to the number of system components and are more dominant for higher granularities such as performance (or maintenance of the resulting components) might be the reason for the limited growth of the effort. Generalizing this idea, effort indicators could be classified according to the degree they stimulate (negative factors) or limit (positive factors) the growth in effort. This would make it possible to estimate how balanced and thus how representative the estimation is. Too many indicators from one category could make the results skewed.

Finally, the third interesting observation is related to the effort regression analysis. The relation of the effort and thus also the scattering to the granularity seams to be periodic. The best approximation on average of the effort is the sum of sin functions. It has an error lower than polynomial and power approximations even for low degrees. An explanation for this might be the ripple effect in the scattering development that was shown in appendix C.45. Business processes gradually ascent into higher degrees of scattering with the increasing granularity following a periodic motion.

We can conclude that with this experiment the relationship between scattering and granularity was established. Through this, also the Gain was related to the Effort in the context of the Effort Estimation Framework. It was also shown that these scattering and effort estimations can be used for trade-off analysis. It was shown that there is a point where splitting further the processes is not lucrative any more looking at the effort/gain ratio. However, an extension of the experiment is necessary with more independent variables in order to clarify two of the observations mentioned in the discussion above. In the first place, what is the significance of the suspected new parameter for the percentage of complex business process. Secondly, are there any other parameters that would make the effort estimation display exponential growth.

4.6.1 Future work – Increase accuracy with Service Model

This subsection presents the suggested approach for increasing the accuracy of the effort estimation in the experiment. It has not been tested in practice due to time constraints. The increased accuracy is based on the fact that also the service layer is being taken into consideration for the modernization. This more closely resembles the desired SOA end-organization. The service layer is the intermediate abstraction layer between the invoking business processes and implementing components in the Development View (figure C.6.

For producing a model of this layer different identification techniques can be used. Formal Concept Analysis(FCA) [28, 62, 96] for the abstracting the Business Object Model, aspect mining for the identification of infrastructural services [102] and program slicing [20, 73] for the extraction of the three basic service type defined by the estimation framework – *Information Services*, *Business Services*, *General Services*. For the extraction of the services the following encapsulation model can be used, suggested by Sneed [89] shown in figure C.55. Specifically for Information services, the approach should be to componentize based on coupling to data types (cluster functionality around data types) and encapsulation as DAS as shown in figure C.4

Incorporation of this approach in the structure of the performed experiment is similar to the existing model extraction steps. The goal here is also to keep the traceability to the code level. This is possible because the above-mentioned techniques are bottom-up, generating abstractions from the code. This, opposed to the top-down approaches using a domain model put together by domain experts. This relation of the service model to the code level entities can then be used to establish their corresponding legacy components and business processes. This process is shown in figures C.52 and C.53. Figure C.54 depicts the reorganization that would have to take place in order to reach a 1-to-1 mapping between the implementing components and the business process activities.

Chapter 5

Research evaluation

This research was targeted at the problem stated in chapter 1, in the field of system modernization towards SOA. That problem is the need for effort estimation in support of decision making in such modernization projects. Now that this thesis has presented the Effort Estimation Framework in chapter 3 together with an empirical experiment in chapter 4, we can evaluate how they contributes to achieving the goal expressed in the problem statement. First, the contribution of this research is evaluated based on the applications of the delivered Effort Estimation Framework (§5.1). Despite its broad applicability, the framework still has its limitations, which are evaluated in section 5.2. In conclusion, both the framework's contribution and its limitations are taken into consideration in an evaluation of the overall quality (§5.3).

5.1 Contribution and applications of research results

The main contribution of this thesis is the structure it gives to the domain of effort estimation in modernizing towards SOA. The framework that is presented, creates the organization necessary to tackle to problem of effort estimation in a systematic way that did not exist before.

In the first place, the thesis identifies the multiple dimensions relevant to forming a complete view of the problem domain from existing research. These dimensions encompass approaches to system and modernization process modeling. Building on this base, an innovative contribution is made consisting of two parts. The first innovative addition is the selection and combination of different parts of the existing approaches. A decision is made on how to fit the complementing parts together. The second innovative addition of this research is the method for effort estimation. It delivers a rating based on the systematization of the issues relevant to effort estimation and the causes of modernization effort. The central concept in this Rating Model is the Point of Modernization. This abstraction decouples the effort estimation from the underlying system models created by existing approaches.

The presented Effort Estimation Framework has two main applications. The first is in trade-off analysis as part of the planning phase of modernization projects. The second is its

use as a frame of reference for future work on impact analysis of the modernization towards SOA.

The framework supports trade-off analysis between effort and gain, because of its construction. It accepts an input in the form of parameters describing the desired gain from the modernization towards SOA such as flexibility and lower coupling. On the other hand the framework produces the effort estimation as output. The framework relates effort estimation to gain and enables in this way the search for an optimal modernization plan with a set of requirements for both entities as a starting point.

The framework also offers a frame of reference for future work on the impact of modernization towards SOA. It contains an organization of the basics of the problem domain in a form allowing extension. This reference can be used in defining a clear scope of future research through its overview of the related areas. In this way, concrete areas can be studied in isolation and afterwards the findings can be related to the whole. This can also be used in the definition of empirical experiments and case studies. The framework eases the design (choice of independent and dependent variables) and the comparison of results.

Finally, such a frame of reference is also useful for work in the industry. It delivers guidance for modernization discussions by relating the major issues of modernizing to each other. It can function as a checklist in assessing the impact of decisions. The framework is at a high enough level to be instantiated for many different legacy systems. Most importantly, the rational behind the construction of the framework can be traced back to the scientific sources and its applicability can be verified for a particular case at hand.

The thesis also discusses the possible influence of issues/factors that fall outside of the framework scope such as business gain, modernization impact on the organization and business and modernization process phases and cost drivers.

5.2 Limitations of the framework

The Effort Estimation Framework is based on a set of assumptions and configurations for two reasons, both of which are aimed at improving the framework. The first is to narrow down the domain of considered systems and modernization strategies so that the effort estimation is more accurate. But at the same time also not making it too specific in order to keep the framework reusable(the framework dilemma, section 2.1). The second reason is to enable the choice, through configuration, for the most suitable effort estimation techniques, by considering the target Service Oriented Architecture and the strategies to modernize towards such an environment.

Next to leading to the above-mentioned improvements, these choices of assumptions and configurations also lead to **limitations**. We consider a limitation to be an ability or feature that the framework does not have, but that is desirable in order to deliver the most complete effort estimation results. Furthermore, we consider the limitations to manifest themselves in one of three possible areas:

- the application area of the framework and the supported input types
- the achieved estimation accuracy of the framework
- the relevancy of the output

Following is the discussion of these limitations. We do this per root cause, starting with the scope and ending with the framework assumptions and configuration.

Scope limitations

Risk – the framework considers the trade-off analysis between effort and gain. This is the result of the impact assumption in section 3.1. There is also another important modernization factor in the literature – risk [87, 94]. It is not included as a factor in the evaluations in the Rating Model and affects the accuracy of the framework's optimal point estimation.
Business domain – the effort estimation is done in the technical domain. This makes it more easily quantifiable, but it also distances it from the business domain which is generally more relevant for the enterprise strategy management. This separation between the Business gain and Technical gain was also made explicit in section 3.1 and limits the both input and output of the framework to technical entities. This is something that the framework certainly needs to be extended in and that is why this issue is also treated in the chapter on future work 6.
Technical implementation – The framework is specified in isolation from the out-of-scope domains. The fact that such a relationship exists has been noted in section 3.1. However, because of the scope of this research, the concrete interaction between the domains has not been defined, such as for example impact propagation between them. Specifically, the separation from the Technical domain creates limitations. The framework does not specify the exact impact propagation to the implementation level an this limits the relevancy of the output it supplies. It still is on a relatively high-level of abstraction and not expressed in concrete realization terms. This limitation, however, is justifiable for two reasons. The first is that this link falls under the category of formal transformations [59] and is considered trivial. The second is that it is implementation technology dependent and this step should, thus, be left for the instantiation and calibration of the framework.

Framework assumptions and configuration limitations

Procedural legacy systems – another limitation on the input is due the assumption L-2 in section 3.3. The effort estimation in the Identification group of the Rating Model could be more accurate through a more effective legacy system modeling if extended with other paradigms (e.g. Object-Oriented). However, this is not a serious limitation as a largest part of the legacy systems considered for modernization do make use of the procedural paradigm.

Business service identification approach – the approach to Business service identification configured in the Control Model in section 3.3.3, limits the accuracy of the effort estimation. Because the models are extracted bottom-up from the existing code they tend to preserve its flaws such as inefficient business process flows or low level of reuse. The bottom-up approach taken in the framework introduces, thus, an error in the estimation as it preserves the system organization mainly in the Business domain. It would eventually be needed to remove these flaws at the cost of extra effort.

Furthermore, the consequence of the current configuration as presented in section 3.3 are the following limitations:

- Input: the approach taken, using SOA properties as input, gives a rather limited range of options and is possibly on a too high-level of abstraction. This could be overcome by extending the target system specification possibilities. For this purpose, we suggest using the approaches for the evaluation of a Service-Oriented system, which were identified in the literature study such as [46, 95, 8].
- Output: the precision of the output is currently limited to the ordinal scale. This is the consequence of the absence of quantitative information about the relationship between the effort for different modernization strategies. This issue was described in the "Strategy effort indicators and rating aggregation" subsection of section 3.3.1.
- Accuracy: the accuracy of the effort estimation is limited due to the following:
  - the level of abstraction used in the architectural models is rather high. It goes down only to components and connectors and not to function level (section 3.3.1, configuration C-2)
  - the choice not to consider the propagation of impact between domains yet. This configuration issue is described in the "Points of Modernization as basis for effort estimation" subsection of section 3.3.1.
  - the Rating Model does not take into account such an impact propagation and assumes a single effort aggregation pass in the order identification, componentization, integration, SOA capabilities.
  - no quantitative use of pattern information. As was argued in section 3.3.1 configuration C-4, the use of design patterns in this estimation framework remains limited. The extend of this limitation is described at the end of section 3.4.3.
  - the absence of concrete quantitative data about the relation of the strategies and indicators to effort. The field of effort estimation for modernization towards SOA is rather unexplored so currently the presented indicators and strategies are only qualitatively related to effort. Each one of them has to be researched to make this relation concrete and also how combining them together influences the estimation.

5.3 Resulting quality of the framework

From the description of the limitations above, we can conclude that the framework is coarse grained in its approach and the resulting estimations. This is due to the absence of empirical results to support a more accurate relationship specification between indicators and effort. It does not, however, lack any essential aspects in rating the modernization effort. It offers a structure that can be further refined by systematically investigating its subparts.

The quantification method's accuracy depends on the correctness of the relationship between effort indicators, modernization strategies and identification of PoM. And also on the weights of these relationships. These should generally be calibrated through industry experience. But even if the method is calibrated the analysis process is generally as good as the information it has to work with. This makes the concrete approaches used in the Reverse

Engineering phase the weakest link. And because we configure the framework to rely on automated evaluation, this introduces the limitations of the absence of expert knowledge.

Chapter 6

Summary, Conclusion and Future Work

6.1 Summary

This thesis is aimed at the problem of impact analysis of the modernization of legacy systems towards Service Oriented Architecture. Chapter 1 specified the need for a method for decision making specifically relying on the estimation of the effort expected for modernization. The goal of the research is to establish a method for the estimation of this modernization effort as part of overall impact analysis.

The solution proposed in this thesis is the Effort Estimation Framework. It is based on existing work from several areas of research. Chapter 2 describes this background in the areas of software architecture (§2.1), service-orientation (§2.2) and modernization approaches (§2.3, §2.4). These existing approaches are used to capture the information on which the effort estimation is based. There are four such important sources of information. In the first place, these are the models of the legacy architecture and Service Oriented Architecture. In addition to this, the process of modernization, the strategies used and their impact is also modeled. In third place, the overall context of the modernization is described such as stakeholders, domains and costs-drivers to put the effort estimation into perspective. Finally, the specific issues in the modernization of software system towards SOA are identified.

The framework itself is presented in chapter 3. It is built in several stages - assumptions (§3.1, §3.2), configuration (§3.3) and specification (§3.4). This allows the adaptation of the framework to specific domain or as a result of further research. Its contribution is the organization of the existing approaches so that they can be used for effort estimation specifically for the modernization towards Service Oriented Architectures (§3.4.1). For the purpose of establishing such an effort estimation, the framework specifies on top of the approaches a Rating Model (§3.4.3) and a process for gathering the required information (§3.4.2). The Rating Model relates measurements of architectural characteristics to effort and aggregates these estimations. Essential for this is the concept of Point of Modernization (PoM). The Rating Model offers an effort estimation output on an ordinal scale. It is based on a distinction between impact types, classification of modernization concerns, ordering of the modernization strategies according to effort and a classification of measurable indicators.

The use of the framework is presented in chapter 5. The chapter describes an empirical experiment that demonstrates the instantiation of part of the framework (§4.2.5, §4.3). The experiment evaluates the effectiveness of the framework based on the measurement of the effort indicator of Scattering. The outcome of the experiment shows the relationship that exists between the framework input of Gain and its output of Effort (§4.4.2). This demonstrates that the Effort Estimation Framework is effective in producing an effort estimation given input of required gain. This relationship is then used to demonstrate the application of the effort estimation for trade-off analysis (§4.5).

Finally, the Effort Estimation Framework and the research results are evaluated in chapter 5. The evaluation treats three aspects. The first is concerned with the application of the proposed framework. The suitability of the framework is discussed for impact and trade-off analysis (§5.1). Secondly, the limitations of the framework are discussed anticipated as a result of the assumptions and choices made (§5.2). Finally, the overall quality of the approach taken in the framework is confirmed (§5.3).

6.2 Conclusion

The main conclusion from this thesis is that automated estimation of modernization effort towards SOA is feasible. The theoretical context (chapter 2) shows that there are existing methods describing the main aspects of the problem of modernization effort estimation. What is missing is a method to bring them together. This thesis shows that they can be combined to form a structure for tackling the problem (chapter 3).

The suggested Effort Estimation Framework shows how the different aspects/methods from software architecture and modernization can be integrated. They complement each other in different areas which makes it possible to build an all-round view of the causes of effort in legacy modernization. The resulting framework divides the problem of effort estimation in separate fragments (phases, domains, responsibilities, modernization strategies). This lowers the complexity of the problem as estimations can be made for the different parts separately and then aggregated. The Rating Model makes this possible through the Points of Modernization and the categorizations it uses.

Furthermore, the empirical part of the research using the Effort Estimation Framework (chapter 4) leads to establishing the following facts:

- the Gain and Effort in the constructed framework are related to each other through on the scattering and granularity variables
- this relationship can be used for trade-off analysis and to find an optimal solution with given required levels of Effort and Gain
- the discovery of an unexpected parameter which could be useful - the percentage of complex business processes
- the parameters used are not enough to model all the expected characteristics of the Effort estimation development

We conclude with the remark that this research is only an initial attempt and requires much more work, both theoretical and empirical. A vision on the direction of this future work is described in the following section.

6.3 Future work

This section presents the big picture around the findings of this research, emphasizing on the future possibilities. We describe the path towards an accurate effort estimation of modernization projects (Appendix C.56), the eventual goal in legacy system modernization, and position our current work on it.

The path towards a good estimation of modernization effort is divided in stages in terms of cost. Each stage introduces a change in scale and units of measurement, increasing the accuracy of the *Rating Model*. This gradation is shown on the left side in figure C.56. In the middle is shown the central for this research process of modernizing. Based on the modernization strategies that can be applied to perform this process, we presented a *Rating Model*. This model is concentrated in the stage in the estimation path - the *Difference Identification*. The highest accuracy level achievable here based on the available information is an estimation on the ordinal scale. The availability of the source information is determined in the preceding stage of *Model Extraction* and *SOA specification* for the legacy and target systems respectively. Their goals are determined during the *Portfolio Analysis* stage. Following the *Difference Identification* stage is the *Trade-off Analysis* weighting the possible modernization strategies against each other. Finally comes the stage that translates the effort into cost through a *Cost Model*. Each of these stages and their possibilities are clarified in the following subsections.

Portfolio Analysis

The existing approach to portfolio analysis is rather limited (section 2.3). In addition to this, the results of this essential process are not yet related to the framework. One of the assumptions of the research is that this analysis has been successfully performed and its results can readily used as input for the effort estimation. The future challenge is to extend the framework by shifting its input side further in the direction of the Business Goals. This will make the link possible between the Business Gain and the SOA gain (section 3.1).

A second area for improvement is the models used as input for the framework. They are currently fixed, but they depend on the business goals. For example, a custom selection of models could be made in case of a target of greater Return on investment through increased reuse. The same applies for the specification of the target system configuration. The decision for or against SOA as enabling technology and the selection of desired SOA capabilities depends on the set Business Goals. The aim is to link the business need to the technical need. So linking the framework to the output of the Portfolio Analysis process will lead to improved: a) legacy model selection, b) SOA capabilities specification. Altogether, this will make it possible to make a more precise estimation guided directly by the business goals of the company.

Decision Model

The currently used Rating Model does not support any constraints. These do exist in practice and may mean that a sub-optimal solution must be chosen. This means that the estimation

given actually forms and lower limit estimation of the modernization effort. Taking into account constraints such as resources, technology, know-how to make a selection from the available modernization strategies, could lead to an increase in the estimation. These constraints can also be seen as requirements(soft and hard) on the to-be system as for example fixed operating system, programming language, SOA infrastructure supplier, performance. Together these form the *Decision Model*. The final output of the analysis based on this model is a modernization plan containing a chosen strategy covering all Points of Modernization. The criteria for making these choices can differ depending on the goals of the company. For this reason we theorize that an additional *Relation Model* is necessary. It is an interchangeable view on the system on how to prioritize/valuate the constraints incorporated in the *Decision Model*. Possible approaches are to prioritize the application of the redesign strategy to components with a high business value or to consider only black-box strategies for components with a high reusability degree. The *Decision Model* is used to quantify the differences between the solutions the different strategies suggest. This makes it possible to switch over to an interval scale. It also has to include a more elaborate gain valuation of the impact of modernization as explained in section 3.1.

Cost Model

The effort output of the framework is intended as input to a Cost Model. This model will relate the effort to a concrete implementation and its costs. This will make it possible to switch to a ratio scale and use money as output. Here we come back to the types of costs involved in a modernization project mentioned in section 2.3. The cost items mentioned there each belong to one of the three categories:

1. Reengineering Investment Costs (RIC)
2. Operations and Support Costs for Reengineered Components (OSRC)
3. Operations and Support Costs for Legacy Components (OSLC)

Each of these cost items has its origins in one of the four phases of system evolution: A).Plan Evolution, B).Implement, C).Deliver and D).Deploy.

The framework gives an effort estimation that serves as input for the combination A1 - the RIC costs in the Planning stage. The challenge for the future is to extend this framework with the estimation of the effort needed for the calculation of the other possible combinations. In this phase it is also of importance to know how the estimates can be improved. As the output of this phase can be verified with real projects, there is a possibility for calibration through iteration.

Extension current approach

Furthermore, the accuracy of the current approach could be improved by extending it with:

- extend service model identification approximation with business-goal driven service identification (top-down approach)
- add dynamic analysis(Wu [100]): run the system and keep track of the invocation of each business rule to obtain an indication of its business value. higher usage ->higher business value

Bibliography

[1] W. Al Belushi and Y. Baghdadi. An approach to wrap legacy applications into web services. June 2007.

[2] Robert Allen and David Garlan. A formal basis for architectural connection. *ACM Trans. Softw. Eng. Methodol.*, 6(3):213–249, 1997.

[3] Francesca Arcelli, Christian Tosi, and Marco Zanoni. Can design pattern detection be useful for legacy systemmigration towards soa? In *SDSOA '08: Proceedings of the 2nd international workshop on Systems development in SOA environments*, pages 63–68, New York, NY, USA, 2008. ACM.

[4] A. Arsanjani, S. Ghosh, A. Allam, T. Abdollah, S. Gariapathy, and K. Holley. Soma: a method for developing service-oriented solutions. *IBM Syst. J.*, 47(3):377–396, 2008.

[5] Ali Arsanjani and Kerrie Holley. The service integration maturity model: Achieving flexibility in the transformation to soa. In *SCC '06: Proceedings of the IEEE International Conference on Services Computing*, page 515, Washington, DC, USA, 2006. IEEE Computer Society.

[6] Joachim Bayer, Jean-François Girard, Martin Würthner, Jean-Marc DeBaud, and Martin Apel. Transitioning legacy assets to a product line architecture. *SIGSOFT Softw. Eng. Notes*, 24(6):446–463, 1999.

[7] Jesal Bhuta and Barry Boehm. A framework for identification and resolution of interoperability mismatches in cots-based systems. In *IWICSS '07: Proceedings of the Second International Workshop on Incorporating COTS Software into Software Systems: Tools and Techniques*, page 2, Washington, DC, USA, 2007. IEEE Computer Society.

[8] Phil Bianco, Rick Kotermanski, Summa Technologies, and Paulo Merson. Evaluating a service-oriented architecture. Technical Report CMU/SEI-2007-TR-015, Carnegie Mellon University, Pittsburgh, PA, USA, 2007.

[9] Norbert Bieberstein, Robert G. Laird, Dr. Keith Jones, and Tilak Mitra. *Executing SOA: A Practical Guide for the Service-Oriented Architect*. IBM Press, May 2008.

[10] Kevin Bierhoff, Mark Grechanik, and Edy S. Liongosari. Architectural mismatch in service-oriented architectures. In *SDSOA '07: Proceedings of the International Workshop on Systems Development in SOA Environments*, page 4, Washington, DC, USA, 2007. IEEE Computer Society.

[11] Steven J. Bleistein, Karl Cox, and June Verner. Strategic alignment in requirements analysis for organizational it: an integrated approach. In *SAC '05: Proceedings of the 2005 ACM symposium on Applied computing*, pages 1300–1307, New York, NY, USA, 2005. ACM.

[12] T. Bodhuin, E. Guardabascio, and M. Tortorella. Migrating cobol systems to the web by using the mvc design pattern. In *WCRE '02: Proceedings of the Ninth Working Conference on Reverse Engineering (WCRE'02)*, page 329, Washington, DC, USA, 2002. IEEE Computer Society.

[13] B. Boehm, B. Clark, E. Horowitz, C. Westland, R. Madachy, and R. Selby. Cost models for future software life cycle processes: Cocomo 2.0. *Annals of Software Engineering*, 1:57–94, December 1995.

[14] Shawn A. Bohner. Extending software change impact analysis into cots components. In *SEW '02: Proceedings of the 27th Annual NASA Goddard Software Engineering Workshop (SEW-27'02)*, page 175, Washington, DC, USA, 2002. IEEE Computer Society.

[15] Diego Bovenzi, Gerardo Canfora, and Anna Rita Fasolino. Enabling legacy system accessibility by web heterogeneous clients. In *CSMR '03: Proceedings of the Seventh European Conference on Software Maintenance and Reengineering*, page 73, Washington, DC, USA, 2003. IEEE Computer Society.

[16] Paul Brown. *Succeeding with SOA: Realizing Business Value Through Total Architecture*. Addison-Wesley Professional, 2007.

[17] Paul Brown. *Implementing SOA: Total Architecture in Practice*. Addison-Wesley Professional, 2008.

[18] Frank Buschmann, Regine Meunier, Hans Rohnert, Peter Sommerlad, and Michael Stal. *Pattern-oriented software architecture: a system of patterns*. John Wiley & Sons, Inc., New York, NY, USA, 1996. TU-Bib: IN69D22Busc.

[19] G. Canfora, A.R. Fasolino, G. Frattolillo, and P. Tramontana. Migrating interactive legacy systems to web services. *Software Maintenance and Reengineering, 2006. CSMR 2006. Proceedings of the 10th European Conference on*, pages 10 pp.–36, March 2006.

[20] Gerardo Canfora, Aniello Cimitile, Andrea De Lucia, and Giuseppe A. Di Lucca. Decomposing legacy programs: a first step towards migrating to client-server platforms. *J. Syst. Softw.*, 54(2):99–110, 2000.

[21] Gerardo Canfora, Anna Rita Fasolino, Gianni Frattolillo, and Porfirio Tramontana. A wrapping approach for migrating legacy system interactive functionalities to service oriented architectures. *J. Syst. Softw.*, 81(4):463–480, 2008.

[22] Satish Chandra, Jackie de Vries, John Field, Howard Hess, Manivannan Kalidasan, Komondoor V. Raghavan, Frans Nieuwerth, Ganesan Ramalingam, and Justin Xue. Using logical data models for understanding and transforming legacy business applications. *IBM Syst. J.*, 45(3):647–655, 2006.

[23] Chia-Chu Chiang. Automated software wrapping. In *ACM-SE 45: Proceedings of the 45th annual southeast regional conference*, pages 59–64, New York, NY, USA, 2007. ACM.

[24] E. J. Chikofsky and J. H. Cross. Reverse engineering and design recovery: A taxonomy. *IEEE Software*, 7(1):13–17, 1990.

[25] Katja Cremer, André Marburger, and Bernhard Westfechtel. Graph-based tools for re-engineering. *Journal of Software Maintenance*, 14(4):257–292, 2002.

[26] Fred A. Cummins. *Building the Agile Enterprise: With SOA, BPM and MBM*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, September 2008. TU-Bib: IN69D33Cumm.

[27] Andrea De Lucia, Rita Francese, Giuseppe Scanniello, and Genoveffa Tortora. Developing legacy system migration methods and tools for technology transfer. *Softw. Pract. Exper.*, 38(13):1333–1364, 2008.

[28] Concettina Del Grosso, Massimiliano Di Penta, and Ignacio Garcia-Rodriguez de Guzman. An approach for mining services in database oriented applications. In *CSMR '07: Proceedings of the 11th European Conference on Software Maintenance and Reengineering*, pages 287–296, Washington, DC, USA, 2007. IEEE Computer Society.

[29] Merriam-webster's online dictionary, 11th edition. `http://www.merriam-webster.com/dictionary/dendrogram`.

[30] Mark Endrei, Jenny Ang, Ali Arsanjani, Sook Chua, Philippe Comte, Pl Krogdahl, Min Luo, and Tony Newling. *Patterns: Service-oriented Architecture and Web Services*. IBM Redbook, April 2004.

[31] Thomas Erl. *SOA: Principles of Service Design*. Prentice Hall, July 2007.

[32] Thomas Erl. *SOA Design Patterns*. Prentice Hall, December 2008.

[33] M. Ernest and J. M. Nisavic. Adding value to the it organization with the component business model. *IBM Syst. J.*, 46(3):387–403, 2007.

[34] Meng Fan-Chao, Zhan Den-Chen, and Xu Xiao-Fei. Business component identification of enterprise information system: A hierarchical clustering method. In *ICEBE '05: Proceedings of the IEEE International Conference on e-Business Engineering*, pages 473–480, Washington, DC, USA, 2005. IEEE Computer Society.

[35] Erich Gamma, Richard Helm, Ralph Johnson, and John Vlissides. *Design patterns: elements of reusable object-oriented software*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 1995.

[36] David Garlan. Software architecture: a roadmap. In *ICSE '00: Proceedings of the Conference on The Future of Software Engineering*, pages 91–101, New York, NY, USA, 2000. ACM.

[37] David Garlan, Robert Allen, and John Ockerbloom. Architectural mismatch: Why reuse is so hard. *IEEE Softw.*, 12(6):17–26, 1995.

[38] David Garlan and Mary Shaw. An introduction to software architecture. Technical report, Carnegie Mellon University, Pittsburgh, PA, USA, 1994.

[39] Rainer Gimnich. Component models in soa realization. `ftp://ftp.software.ibm.com/software/emea/de/soa/sqm2007_gimnich_presentation.pdf`, 2007.

[40] Eduardo Machado Gonçalves, Marcilio Silva Oliveira, and Kleber Rogerio Bacili. Digitalassets discoverer: automatic identification of reusable software components. In *OOPSLA '07: Companion to the 22nd ACM SIGPLAN conference on Object-oriented programming systems and applications companion*, pages 872–873, New York, NY, USA, 2007. ACM.

[41] I. Garcia-Rodriguez de Guzman, M. Polo, and M. Piattini. An adm approach to reengineer relational databases towards web services. In *WCRE '07: Proceedings of the 14th Working Conference on Reverse Engineering*, pages 90–99, Washington, DC, USA, 2007. IEEE Computer Society.

[42] Neil B. Harrison, Paris Avgeriou, and Uwe Zdun. Using patterns to capture architectural decisions. *IEEE Softw.*, 24(4):38–45, 2007.

[43] Carsten Hentrich and Uwe Zdun. Patterns for business object model integration in process-driven and service-oriented architectures. In *PLoP '06: Proceedings of the 2006 conference on Pattern languages of programs*, pages 1–14, New York, NY, USA, 2006. ACM.

[44] Peter Herzum and Oliver Sims. *Business Components Factory: A Comprehensive Overview of Component-Based Development for the Enterprise*. John Wiley & Sons, Inc., New York, NY, USA, 2000.

[45] H. M. Hess. Aligning technology and business: applying patterns for legacy transformation. *IBM Syst. J.*, 44(1):25–45, 2005.

[46] Helge Hofmeister and Guido Wirtz. Supporting service-oriented design with metrics. In *EDOC '08: Proceedings of the 2008 12th International IEEE Enterprise Distributed Object Computing Conference*, pages 191–200, Washington, DC, USA, 2008. IEEE Computer Society.

[47] Z.-W. Hong and H. Jiau. A design pattern for reengineering windows software applications into reusable corba objects. In *FTDCS '01: Proceedings of the 8th IEEE Workshop on Future Trends of Distributed Computing Systems*, page 215, Washington, DC, USA, 2001. IEEE Computer Society.

[48] John Hutchinson, Gerald Kotonya, James Walkerdine, Peter Sawyer, Glen Dobson, and Victor Onditi. Migrating to soas by way of hybrid systems. *IT Professional*, 10(1):34–42, 2008.

[49] IEEE. Recommended practice for architectural description of software intensive systems, September 2000.

[50] Per Jnsson and Mikael Lindvall. Impact analysis. In *Engineering and Managing Software Requirements*. Springer Berlin Heidelberg, 2005.

[51] Capers Jones. *Assessment and control of software risks*. Yourdon Press, Upper Saddle River, NJ, USA, 1994.

[52] Nicolai Josuttis. *SOA in practice*. O'Reilly, 2007.

[53] Stephen H. Kaisler. *Software Paradigms*. John Wiley & Sons, 2005.

[54] Mira Kajko-Mattsson, Grace A. Lewis, and Dennis B. Smith. A framework for roles for development, evolution and maintenance of soa-based systems. In *SDSOA '07: Proceedings of the International Workshop on Systems Development in SOA Environments*, page 7, Washington, DC, USA, 2007. IEEE Computer Society.

[55] Mark Kasunic and William Anderson. Measuring systems interoperability: Challenges and opportunities. Technical Report CMU/SEI-2004-TN-003, Carnegie Mellon University, Pittsburgh, PA, USA, 2004.

[56] Mansour Kavianpour. Soa and large scale and complex enterprise transformation. In *ICSOC '07: Proceedings of the 5th international conference on Service-Oriented Computing*, pages 530–545, Berlin, Heidelberg, 2007. Springer-Verlag.

[57] Rick Kazman, Mario Barbacci, Mark Klein, S. Jeromy Carrière, and Steven G. Woods. Experience with performing architecture tradeoff analysis. In *ICSE '99: Proceedings of the 21st international conference on Software engineering*, pages 54–63, New York, NY, USA, 1999. ACM. ATAM.

[58] Rick Kazman, Paul C. Clements, Len Bass, and Gregory D. Abowd. Classifying architectural elements as a foundation for mechanism matching. In *COMPSAC '97: Proceedings of the 21st International Computer Software and Applications Conference*, pages 14–17, Washington, DC, USA, 1997. IEEE Computer Society.

[59] Vitaly Khusidman. Adm transformation. architecture-driven modernization and transformation. `http://www.omg.org/docs/admtf/08-06-10.pdf`, 2008.

[60] Vitaly Khusidman and William Ulrich. Architecture-driven modernization: Transforming the enterprise. `http://www.omg.org/docs/admtf/07-12-01.pdf`, 2007.

[61] Phillippe Kruchten. Architecture blueprints—the "4+1" view model of software architecture. In *TRI-Ada '95: Tutorial proceedings on TRI-Ada '91*, pages 540–555, New York, NY, USA, 1995. ACM.

[62] Tobias Kuipers and Leon Moonen. Types and concept analysis for legacy systems. In *IWPC '00: Proceedings of the 8th International Workshop on Program Comprehension*, page 221, Washington, DC, USA, 2000. IEEE Computer Society.

[63] Grace Lewis, Edwin Morris, and Dennis Smith. Analyzing the reuse potential of migrating legacy components to a service-oriented architecture. In *CSMR '06: Proceedings of the Conference on Software Maintenance and Reengineering*, pages 15–23, Washington, DC, USA, 2006. IEEE Computer Society.

[64] Grace A. Lewis, Edwin J. Morris, Dennis B. Smith, and Soumya Simanta. Smart: Analyzing the reuse potential of legacy components in a service-oriented architecture environment. Technical Report CMU/SEI-2008-TN-008, Carnegie Mellon University, Pittsburgh, PA, USA, 2008.

[65] Lars Lundberg, Jan Bosch, Daniel Hggander, and Per olof Bengtsson. Quality attributes in software architecture design. In *Proceedings of the IASTED 3rd International Conference on Software Engineering and Applications*, pages 353–362, 1999.

[66] C. Matthew MacKenzie, Ken Laskey, Francis G. McCabe, Peter F Brown, and Rebekah Metz. Reference model for service oriented architecture 1.0. Technical report, OASIS, October 2006.

[67] Francis G. McCabe, Jeff A. Estefan, Ken Laskey, and Danny Thornton. Reference architecture for service oriented architecture version 1.0. Technical report, OASIS, April 2008.

[68] Alok Mehta and George T. Heineman. Evolving legacy system features into fine-grained components. In *ICSE '02: Proceedings of the 24th International Conference on Software Engineering*, pages 417–427, New York, NY, USA, 2002. ACM.

[69] Nikunj R. Mehta, Nenad Medvidovic, and Sandeep Phadke. Towards a taxonomy of software connectors. In *ICSE '00: Proceedings of the 22nd international conference on Software engineering*, pages 178–187, New York, NY, USA, 2000. ACM.

[70] Danny Miller and Peter H. Friesen. Archetypes of Strategy Formulation. *MANAGEMENT SCIENCE*, 24(9):921–933, 1978.

[71] Abhay Nath Mishra. Business strategy and it-enabled business capabilities: Fits, misfits and firm performance. `http://auapps.american.edu/~alberto/IT/Mishra.ppt`, 2008.

[72] Glenford J Myers. *Reliable software through composite design*. Petrocelli/Charter, 1975.

[73] Ph. Newcomb and L. Markosian. Automating the modularization of large COBOL programs: application of an enabling technology for reengineering. In *Proceedings of the 1st Working Conference on Reverse Engineering*, pages 222–230, 1993. Experience report using the Software Refinery to build a modularization tool for COBOL.

[74] Liam O'Brien, Len Bass, and Paulo Merson. Quality attributes and service-oriented architectures. Technical Report CMU/SEI-2005-TN-014, Carnegie Mellon University, Pittsburgh, PA, USA, 2005.

[75] Liam O'Brien, Len Bass, and Paulo Merson. Quality attributes and service-oriented architectures. Technical Report CMU/SEI-2005-TN-014, Software Engineering Institute of Carnegie Mellon University, 2005.

[76] Liam O'Brien, Paul Brebner, and Jon Gray. Business transformation to soa: aspects of the migration and performance and qos issues. In *SDSOA '08: Proceedings of the 2nd international workshop on Systems development in SOA environments*, pages 35–40, New York, NY, USA, 2008. ACM.

[77] Liam O'Brien, Dennis Smith, and Grace Lewis. Supporting migration to services using software architecture reconstruction. In *STEP '05: Proceedings of the 13th IEEE International Workshop on Software Technology and Engineering Practice*, pages 81–91, Washington, DC, USA, 2005. IEEE Computer Society.

[78] Ipek Ozkaya, Rick Kazman, and Mark Klein. Quality-attribute based economic valuation of architectural patterns. In *ESC '07: Proceedings of the First International Workshop on The Economics of Software and Computation*, page 5, Washington, DC, USA, 2007. IEEE Computer Society.

[79] Mike P. Papazoglou and Willem-Jan Heuvel. Service oriented architectures: approaches, technologies and research issues. *The VLDB Journal*, 16(3):389–415, July 2007.

[80] Daniel Plakosh, Scott Hissam, and Kurt Wallnau. Into the black box: A case study in obtaining visibility into commercial software. Technical Report CMU/SEI-99-TN-010, Carnegie Mellon University, Pittsburgh, PA, USA, 1999.

[81] Harvard Business School Press. *The essentials of strategy*. Harvard Business School Press, 2006.

[82] Robert C. Seacord, Daniel Plakosh, and Grace A. Lewis. *Modernizing Legacy Systems: Software Technologies, Engineering Process and Business Practices*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 2003.

[83] Mary Shaw and Paul C. Clements. A field guide to boxology: Preliminary classification of architectural styles for software systems. In *COMPSAC '97: Proceedings of the 21st International Computer Software and Applications Conference*, pages 6–13, Washington, DC, USA, 1996. IEEE Computer Society.

[84] Dennis B. Smith, Liam O' Brien, and John Bergey. Using the options analysis for reengineering (oar) method for mining components for a product line. In *SPLC 2: Proceedings of the Second International Conference on Software Product Lines*, pages 316–327, London, UK, 2002. Springer-Verlag.

[85] H. M. Sneed. Encapsulating legacy software for use in client/server systems. In *WCRE '96: Proceedings of the 3rd Working Conference on Reverse Engineering (WCRE '96)*, page 104, Washington, DC, USA, 1996. IEEE Computer Society.

[86] Harry M. Sneed. Program interface reengineering for wrapping. In *WCRE '97: Proceedings of the Fourth Working Conference on Reverse Engineering (WCRE '97)*, page 206, Washington, DC, USA, 1997. IEEE Computer Society.

[87] Harry M. Sneed. Risks involved in reengineering projects. In *WCRE '99: Proceedings of the Sixth Working Conference on Reverse Engineering*, page 204, Washington, DC, USA, 1999. IEEE Computer Society.

[88] Harry M. Sneed. Wrapping legacy cobol programs behind an xml-interface. In *WCRE '01: Proceedings of the Eighth Working Conference on Reverse Engineering (WCRE'01)*, page 189, Washington, DC, USA, 2001. IEEE Computer Society.

[89] Harry M. Sneed. Integrating legacy software into a service oriented architecture. In *CSMR '06: Proceedings of the Conference on Software Maintenance and Reengineering*, pages 3–14, Washington, DC, USA, 2006. IEEE Computer Society.

[90] Harry M. Sneed and Stephan H. Sneed. Creating web services from legacy host programs. *Web Site Evolution, IEEE International Workshop on*, 0:59, 2003.

[91] H.M. Sneed. Risks involved in reengineering projects. In *Reverse Engineering, 1999. Proceedings. Sixth Working Conference on*, pages 204–211, Oct 1999.

[92] Michael Stal. Using architectural patterns and blueprints for service-oriented architecture. *IEEE Softw.*, 23(2):54–61, 2006.

[93] S.R. Tilley and D.B. Smith. Perspectives on legacy system reengineering, 1995.

[94] Amjad Umar and Adalberto Zordan. Reengineering for service oriented architectures: A strategic decision model for integration versus migration. *Journal of Systems and Software*, 82(3):448 – 462, 2009.

[95] André van der Hoek, Ebru Dincel, and Nenad Medvidovic. Using service utilization metrics to assess the structure of product line architectures. In *METRICS '03: Proceedings of the 9th International Symposium on Software Metrics*, page 298, Washington, DC, USA, 2003. IEEE Computer Society.

[96] Arie van Deursen and Tobias Kuipers. Identifying objects using cluster and concept analysis. In *ICSE '99: Proceedings of the 21st international conference on Software engineering*, pages 246–255, New York, NY, USA, 1999. ACM.

[97] R. Van Solingen and E. Berghout. *The Goal/Question/Metric Method – A Practical Guide for Quality Improvement of Software Development*. McGraw-Hill Publishing Company, 1999.

[98] Ian Warren. *The Renaissance of Legacy Systems: Method Support for Software-System Evolution*. Springer-Verlag New York, Inc., Secaucus, NJ, USA, 1999.

[99] Claes Wohlin, Per Runeson, Martin Höst, Magnus C. Ohlsson, Björn Regnell, and Anders Wesslén. *Experimentation in software engineering: an introduction*. Kluwer Academic Publishers, Norwell, MA, USA, 2000.

[100] Lei Wu, Houari Sahraoui, and Petko Valtchev. Program comprehension with dynamic recovery of code collaboration patterns and roles. In *CASCON '04: Proceedings of the 2004 conference of the Centre for Advanced Studies on Collaborative research*, pages 56–67. IBM Press, 2004.

[101] Hua Xiao, Jin Guo, and Ying Zou. Supporting change impact analysis for service oriented business applications. In *SDSOA '07: Proceedings of the International Workshop on Systems Development in SOA Environments*, page 6, Washington, DC, USA, 2007. IEEE Computer Society.

[102] Charles Zhang and Hans-Arno. Jacobsen. Quantifying aspects in middleware platforms. In *AOSD '03: Proceedings of the 2nd international conference on Aspect-oriented software development*, pages 130–139, New York, NY, USA, 2003. ACM.

[103] Zhuopeng Zhang and Hongji Yang. Incubating services in legacy systems for architectural migration. In *APSEC '04: Proceedings of the 11th Asia-Pacific Software Engineering Conference*, pages 196–203, Washington, DC, USA, 2004. IEEE Computer Society.

[104] Ying Zou, Terence C. Lau, Kostas Kontogiannis, Tack Tong, and Ross McKegney. Model-driven business process recovery. In *WCRE '04: Proceedings of the 11th Working Conference on Reverse Engineering*, pages 224–233, Washington, DC, USA, 2004. IEEE Computer Society.

Appendix A

Glossary

In this appendix we give an overview of frequently used terms and abbreviations.

Architecture: The fundamental organization of a system embodied in its components, their relationships to each other, and to the environment, and the principles guiding its design and evolution [49]

ADM (Architecture Driven Modernization): an approach to modeling the modernization process of legacy systems. It is shown in figure C.13

Architectural description: A collection of products to document an architecture [49]

Architectural view: A representation of a whole system from the perspective of a related set of concerns [49]

Component: an autonomous, encapsulated computational entity which accomplishes predefined tasks through internal computation and external communication. This definition (2.2) is explained in detail in section 2.1 on page 6.

Dendrogram: a branching diagram representing a hierarchy of categories based on degree of similarity or number of shared characteristics especially in biological taxonomy[1] [29]

Design pattern: a design pattern describes a commonly-recurring structure of communicating components that solves a general design problem within a particular context. The complete definition (2.6) and its context are explained on page 8

Design principle: an accepted industry practice with a specific design goal. The service-orientation design paradigm is comprised of a set of design principles that are applied

[1] http://en.wikipedia.org/wiki/Dendrogram

together to achieve the goals of service-oriented computing [32]

Development View: one of four architectural views according to Kruchten [61] that focuses on the software module organization in subsystems, layers and libraries defining grouping and visibility. This view is shown for the Service-Oriented set of systems in figure C.6

Effort: the cost of the changes needed to modernize (see definition 3.2)

Framework: a generic architecture complemented by an extensible set of components (see definition 2.7)

Gain: the potential value changes deliver (see definition 3.1)

Impact: what needs to be modified in a system in order to make a change or the consequences on the system if the change is implemented(see definition 2.14)

Indicator: "Strategy effort indicators and rating aggregation" in section 3.3.1

Modernization strategy: an approach for handling the difference in a Point of Modernization and transforming its set of legacy entities into the desired set of target system entities.

PoM (Point of Modernization): an entity identifying a difference between an as-is and a to-be model, which is the base for the effort estimation analysis (see section 3.3.1 on page 34)

Process View: one of four architectural views according to Kruchten [61] that focuses on representing the runtime aspects of a system such as communication and synchronization and non-functional requirements such as performance, availability, systems integrity and fault-tolerance. This view is shown for the Service-Oriented set of systems in figure C.5

Rating scale: the measurement scale used to express the effort in the Rating Model of the framework in. It can range from a nominal to a ratio scale and is explained in detail in section 3.3.1 on page 33

SCA (Service Component Architecture): a modeling standard for the architecture of Service-Oriented systems (see figure C.2)

SDO (Service Data Object): a data modeling standard for Service-Oriented systems (see figure C.3)

Viewpoint: A specification of the conventions for constructing and using a view. A pattern

or template from which to develop individual views by establishing the purposes and audience for a view and the techniques for its creation and analysis [49]

Appendix B

MATLAB code used for performing the experiment

This appendix contains the code listings of the prototype implementation in MATLAB of the Abstraction phase of the research experiment.

Listing B.1: Business Model abstraction

```
% IN:   M - the adjecency matrix of the graph
% OUT:  entry-nodes == all nodes for which the row contains >0 ONES and the column==0 ONES
%
% entry_nodes    = [entry_node_id1, ...., ...                ]
% subtrees       = [subtree_nodes_of_entry_node1, ..., ... ]
entry_nodes_mask = arrayfun(@(i)(sum(M(i,:)) > 0) && (sum(M(:,i)) == 0), 1:size(M));
entry_nodes = find(entry_nodes_mask);
% find the subtree for each entry-node
Ms = sparse(M);
subtrees = arrayfun(@(i)graphtraverse(Ms, i), entry_nodes, 'UniformOutput', false);
```

Listing B.2: Visualization of Business Process abstraction for entry-node #5174

```
M2 = zeros(size(M,1));
entry_node = 5174;
subtree_arr = subtrees(find(entry_nodes==entry_node));
subtree_arr = subtree_arr{1};
for node = subtree_arr
    row = M(node, :);
    col = M(:, node);

    row_new = zeros(1, 5386);
    col_new = zeros(5386, 1);
    for to_node = subtree_arr
        row_new(1, to_node) = row(1, to_node);
    end
    for from_node = subtree_arr
        col_new(from_node, 1) = col(from_node, 1);
    end
    M2(node, :) = row_new;
    M2(:, node) = col_new;
end
% prune empty nodes
isDel = @(id) ((sum(M2(id,:)) == 0) && (sum(M2(:,id)) == 0));
delMask = arrayfun(isDel, 1:size(M2));
delMask = arrayfun(@(i)not(i), delMask);
M3 = M2(:,delMask);
M3 = M3(delMask,:);
% get ids
ids = find(sum(M2));
ids = [ids, entry_node];
ids = sort(ids);
% convert to string
ids_str = cell(1, numel(ids));
for i = 1:numel(ids)
    ids_str{i} = int2str(ids(1,i));
end
% display graph
h = view(biograph(M3, ids_str));
```

Listing B.3: Architectural Model abstraction

```
% calculate distances between all nodes based on connectivity matrix M
M_sparse = sparse(M);
DIST_MATRIX = zeros(size(M));
for from_node = 1:size(M)
    disp(['from_node',num2str(from_node)]);
    for to_node = 1:size(M)
        if from_node ~= to_node
            [dist, path, pred] = graphshortestpath(M_sparse, from_node, to_node);
            DIST_MATRIX(to_node, from_node) = dist;
        end
    end
end

% format matrix to the input format of the linkage function
U = triu(DIST_MATRIX);
U = U';
L = tril(DIST_MATRIX);
% superimpose matrices
DIST_MATRIX2 = L;
for row = 1:size(U);
    for col = 1:size(U);
        if (DIST_MATRIX2(row,col) == Inf)
            if U(row, col) ~= Inf
                DIST_MATRIX2(row, col) = U(row, col);
            else
                % replace Inf with a relatively large number
                DIST_MATRIX2(row,col) = 15;
            end
        end
    end
end

% do the hierarchical clustering for the dendrogram
DIST_MATRIX3 = squareform(DIST_MATRIX2);
TREE = linkage(DIST_MATRIX3, 'weighted');
[H, T] = dendrogram(TREE, 0);
```

Listing B.4: Business to Architectural model mapping for all granularities

```
% calculate the scattering for all possible clusterings
SURF_MATRIX = zeros(5386:1026);
for g = 1:5386
    disp(['g=',num2str(g)]);
    [H, T] = dendrogram(TREE, g);
    process_to_component_mapping = arrayfun(@(process)arrayfun(@(node)T(node,1),
                                subtrees{process}), 1:1026, 'UniformOutput', false);
    scattering_occurrences = arrayfun(@(process)numel(unique(
                                process_to_component_mapping{process})), 1:1026);
    SURF_MATRIX(g,:) = scattering_occurrences;
end

% calculate the distribution of the scattering
SURF_MATRIX2 = zeros(5386:410);
for g = 1:5386
    map = SURF_MATRIX(g,:);
    distribution = zeros(1,410);
    s = sort(unique(map));
    for i = s
        distribution(1,i) = numel(find(map == i));
    end
    SURF_MATRIX2(g,:) = distribution;
end
```

Appendix C

Figures

Figures-I. Software Architecture (section 2.1)

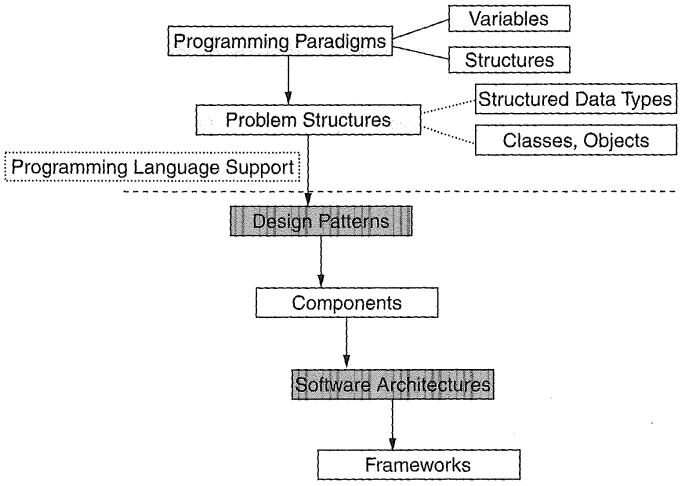


Figure C.1: Hierarchy of structural paradigms in software architecture

Figures-II. Service Oriented Architecture (section 2.2)

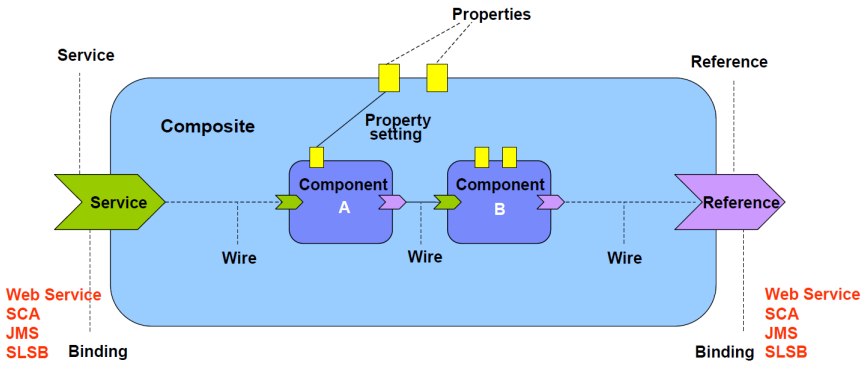


Figure C.2: Service Component Architecture (SCA)

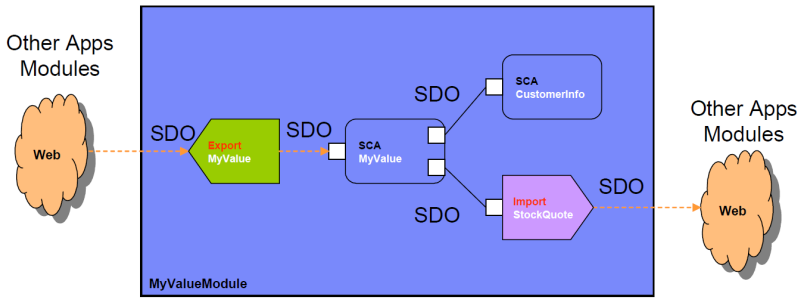


Figure C.3: Service Component Architecture using Service Data Object (SDO)

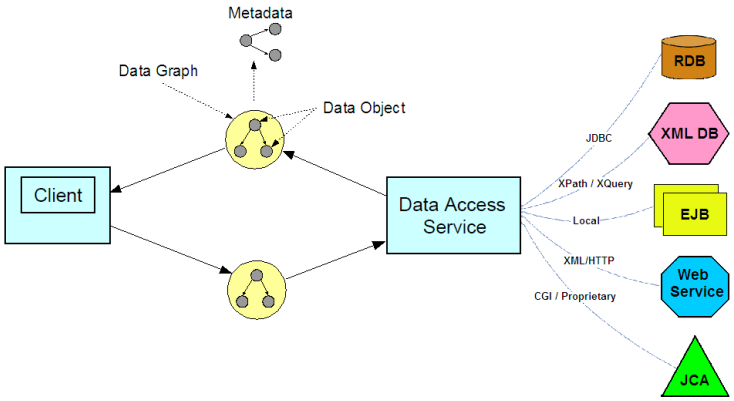


Figure C.4: Data Access Service (DAS)

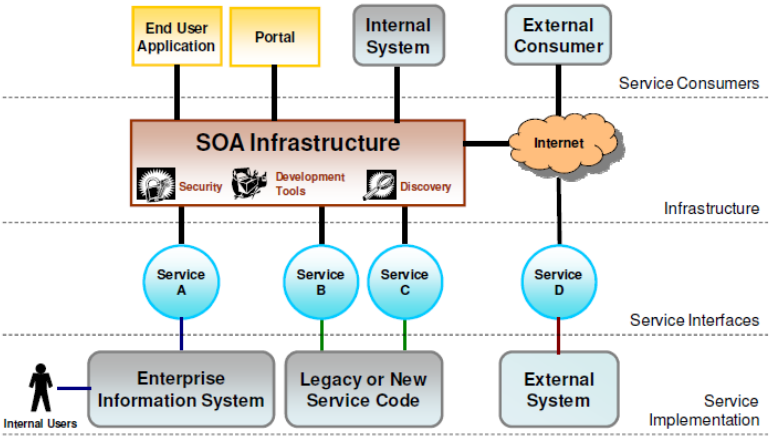


Figure C.5: SOA Process View

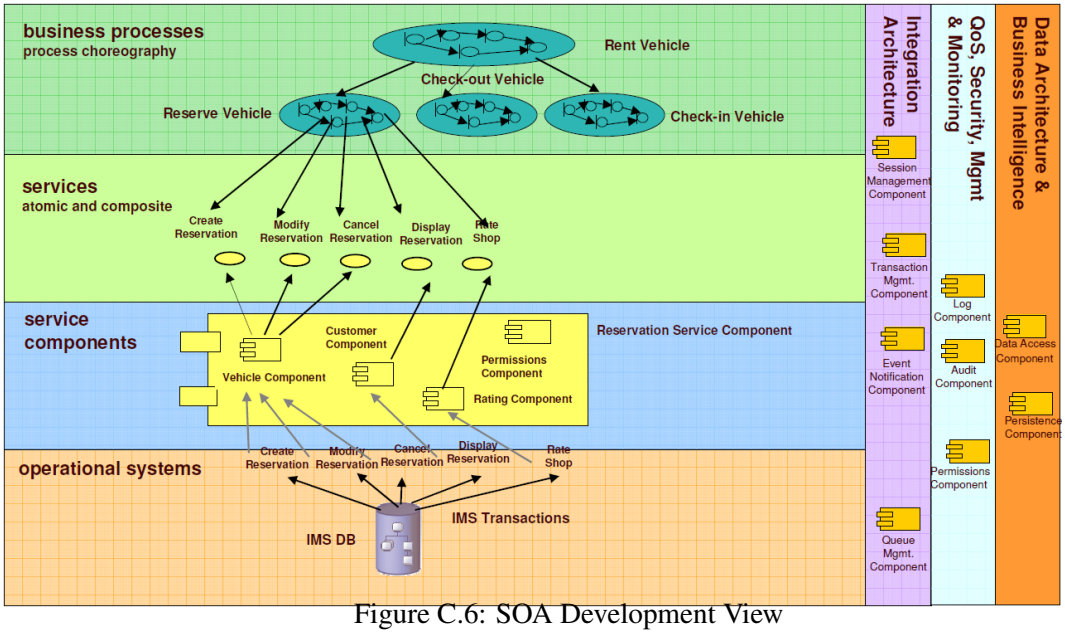


Figure C.6: SOA Development View

Figures-III. Renaissance (section 2.3.1)

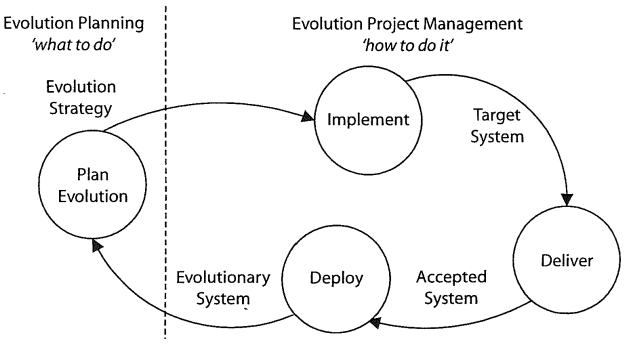


Figure C.7: Incremental Process model

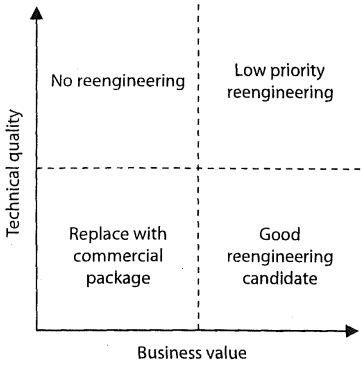


Figure C.8: Portfolio analysis

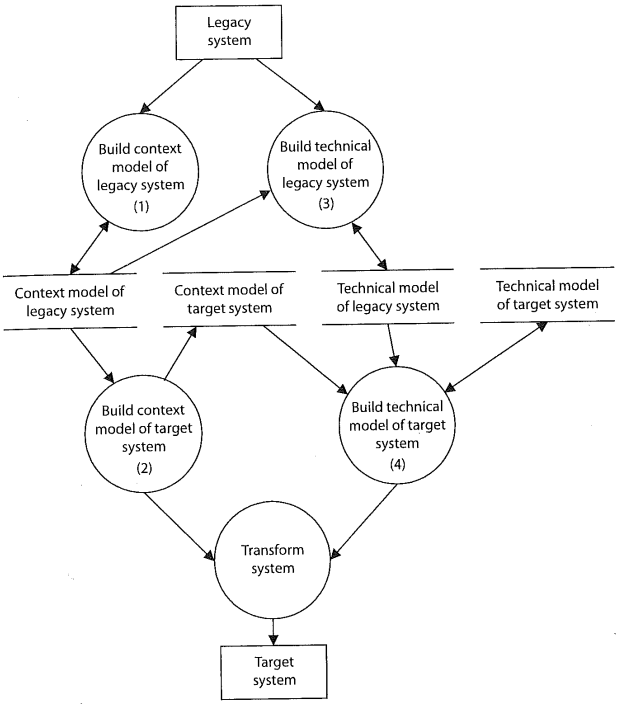


Figure C.9: Data flow diagram for evolution modeling

Evolution strategy		Description
Continued maintenance		The accommodation of change in a system, without radical change to its structure, after it has been delivered and deployed.
Reengineering	Revamp	The transformation of a system by modifying or replacing its user interfaces. The internal workings of the system remain intact, but appear to have changed to the user.
	Restructure	The transformation of a system's internal structure without changing any external interfaces.
	Rearchitecture	The transformation of a system by migrating it to a different technological architecture.
	Redesign with reuse	The transformation of a system by redeveloping it utilizing some legacy components.
Replace		Total replacement of a system from scratch.

Figure C.10: Renaissance - Evolution strategies

Repository folder	Process model phase			
	Plan Evolution	Implement	Deliver	Deploy
Business	produced used	used	used	used
Legacy system	produced used	used		
Target system	produced	produced used	produced used	used
Evolution Strategy	produced used	used		
Project Management		produced used	produced used	used
Test		produced used	used	

Figure C.11: Renaissance - information sources for performing modernization

Figures-IV. ADM (section 2.3.2)

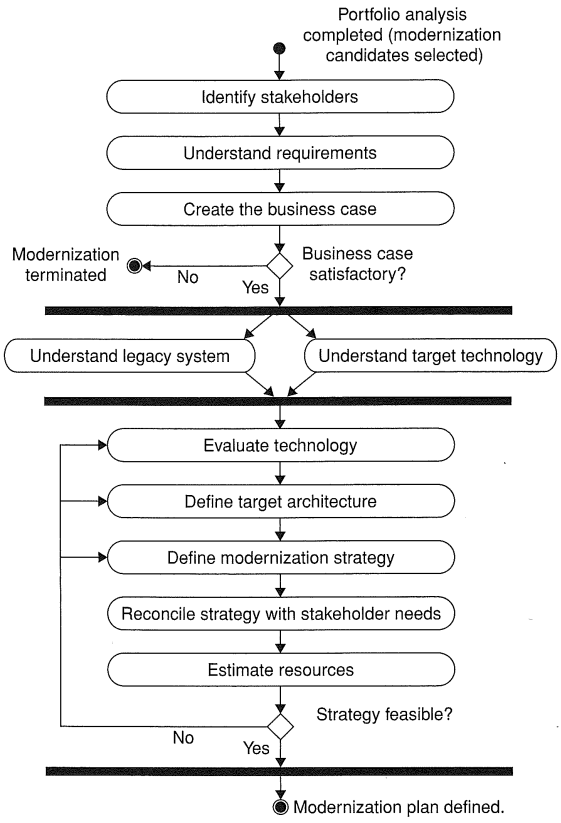


Figure C.12: Risk-management modernization approach (RMM)

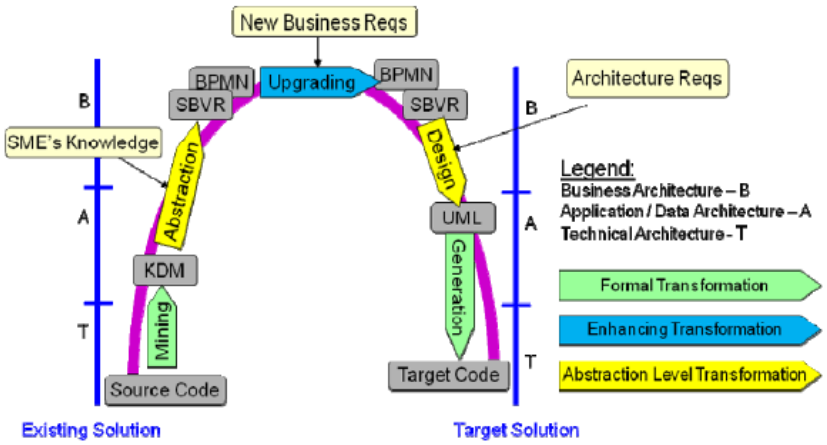


Figure C.13: ADM Transformation types in the horseshoe model

Figures-V. SMART (section 2.3.3)

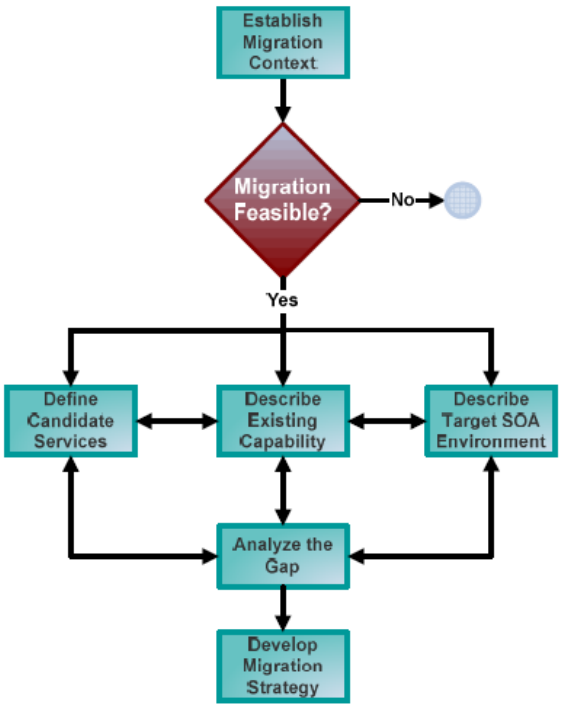


Figure C.14: SMART process for the identification of a modernization strategy towards SOA

Figures-VI. Modernization (section 2.4)

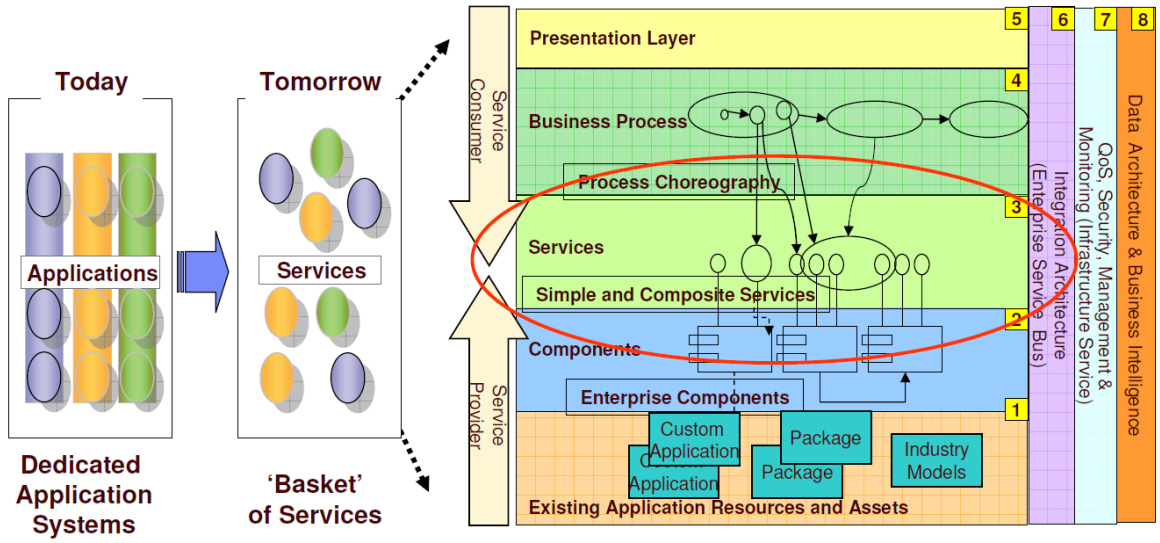
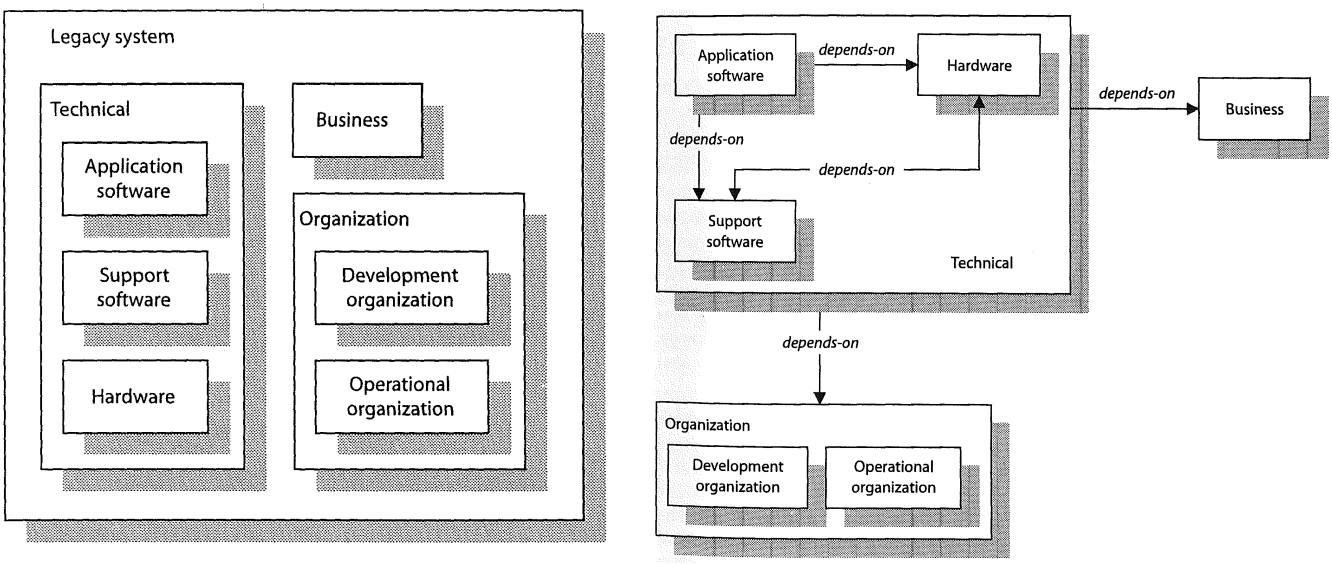


Figure C.15: Architectural goal of the modernization towards SOA



(a) Legacy system composition

(b) Legacy system components relations

Figure C.16: Legacy systems components impacted by modernization

Figures-VII. Effort Estimation Framework (section 3)

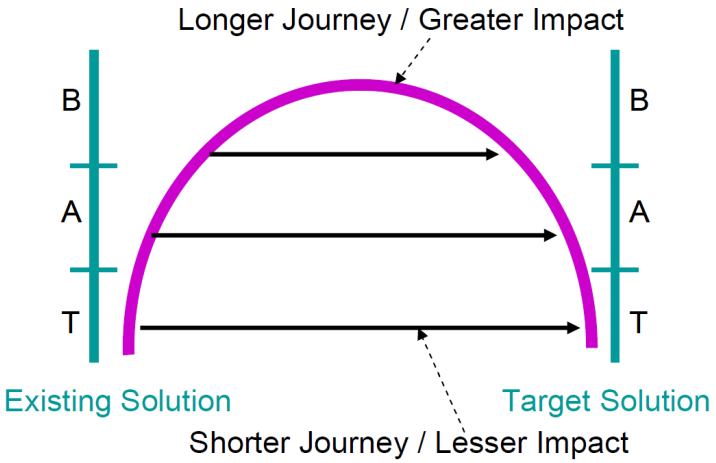


Figure C.17: Impact domains involved in system modernization (ADM)

	Business	Application/Data	Technology
Concepts	• Business Rule • Business Process • Business Activity • Business Object	• Component – exposed interfaces – type classification (layer, concern, relative level of granularity) – dependencies/constraints = on data types (used variables) = on finctionality (called components) – implementation technology binding • Connector – connector class – intrface A – interface B – embedded functionality (synchronoous / asynchronous) – technology binding • Interface – data input/output types • – control input/output types • Patterns – layers; pipe-and-filter; blackboard – Facade, Mediator, Singleton, Abstract Factory, Bridge, Decorator, Adapter, Proxy, Command, Chain of Responsibility, State, Observer	Technology stack – OS – DBMS / App Server – Middleware – Frameworks / Libraries / Runtime environments – Build- / Test- / Deploy facilities
Data Sources	Application/Data Architecture	• Module dependency graph • Data flow graph • Call graph	Deployment diagram
Architectural Views	Logical View	• Development View • Process View	Physical View

Table C.18: Framework entities for modeling the legacy system

	Business	Application/Data	Technology
Concepts	• Capability • Business Process • Business Service • Process Choreography • Service Provider • Service Consumer • Service Mediator	SCA concepts + • Service contract/interface • Simple/Composite service • Agnostic/Specific Service • Logic-, Entity- and Utility Services • Service Task • Resource • Service referenced object • Service owned object	• SOA recomendet technology stack
Data Source	• 8x SOA Design Principles • Requirements • Industry model	• 8x SOA Design Principles • Requirements • Industry model	• 8x SOA Design Principles • Requirements
Architectural Views		• Development View • Process View	

Table C.19: Framework entities for modeling the target SOA system

	Separation of Concerns			Satisfaction of Concerns
	Identification	**Componentization**	**Integration**	
Architectural View	Based on: META_I – Legacy META_I – SOA	• Development View • Service Component Architecture • Service Data Object	• Process View • Service Component Architecture • Service Data Object	Quality Attributes View
Concepts		• SCA_Composite • SCA_Component • SCA_Property • SCA_Service • SCA_Binding • SCA_Reference • SDO_DataGraph • SDO_DataObject • *Entitty Service* • Logic Service	• SCA_Wire • SCA_Service • SCA_Reference • SDO_DataGraph • SDO_DataObject • Utility Services	• Communication capability • Dynamic connectivity • Routing capabilities • Endpoint discovery • Integration capabilities • Transformation capabilities • Reliable messaging capabilities • Security capabilities • Transaction capabilities • Management & monitoring • Scalability capabilities
Strategies		*WHITE-BOX*: • RESTRUCTURE: keep Composites, Services, Properties; change Components, Wires • REDESIGN_WITH_REUSE: keep Components; change Composite and Wires • REPLACE_NEW	*BLACK-BOX*: • WRAPPER: change interface and/or Binding • ADAPTER: data; logic • REPLACE_COTS	• ADD • REMOVE
Impact indicators		**SOA Design principles –** *Reusability, Statelessness, Autonomy, Composability, Coupling*	***SOA Design principles*** *– Contracts, Abstraction,Composability, Coupling*	• SOA capability availability • Architectural style choices • SOA metrics

Table C.20: Framework entities for modeling the modernization process

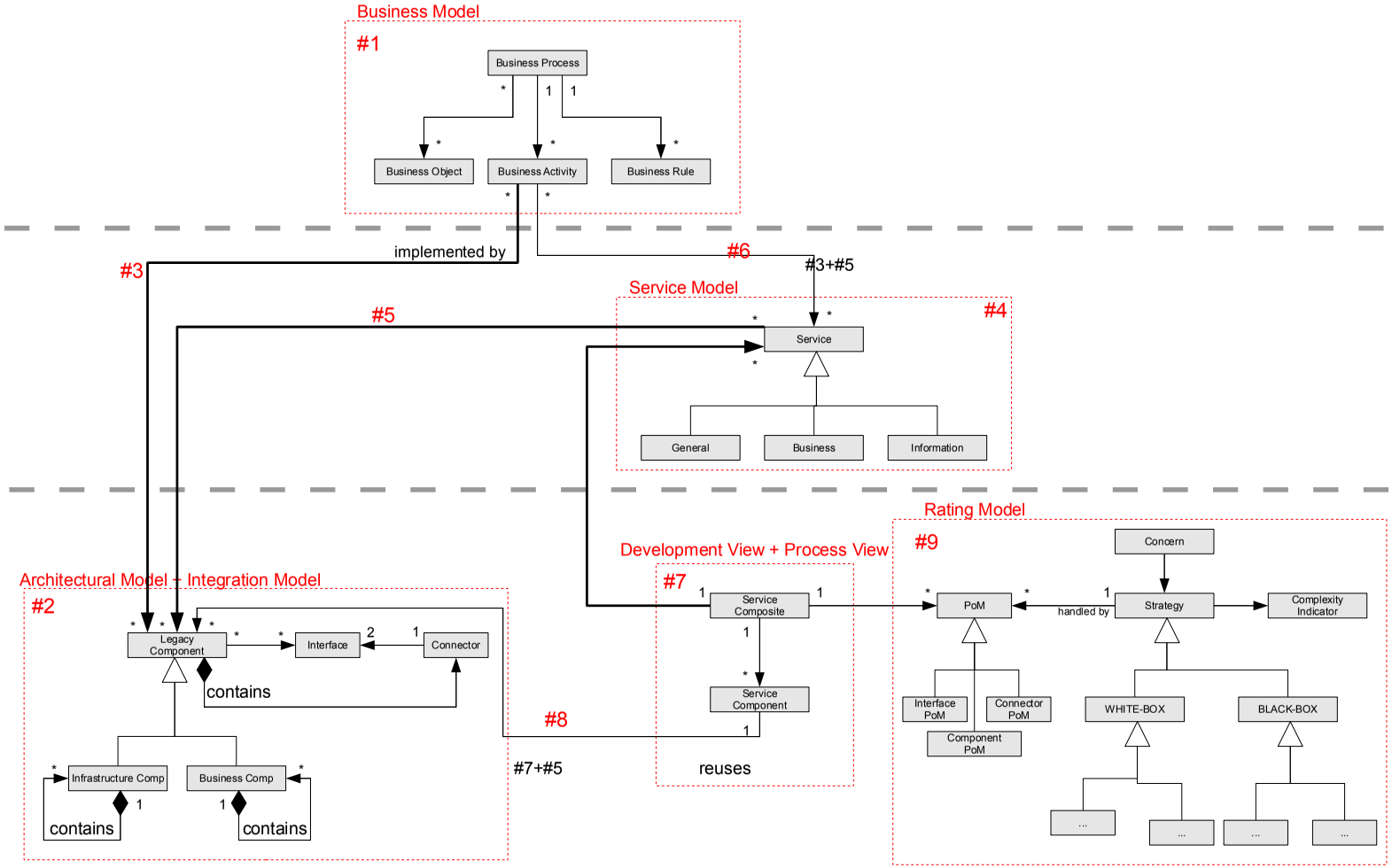


Figure C.21: Framework relationships between models and the contained entities

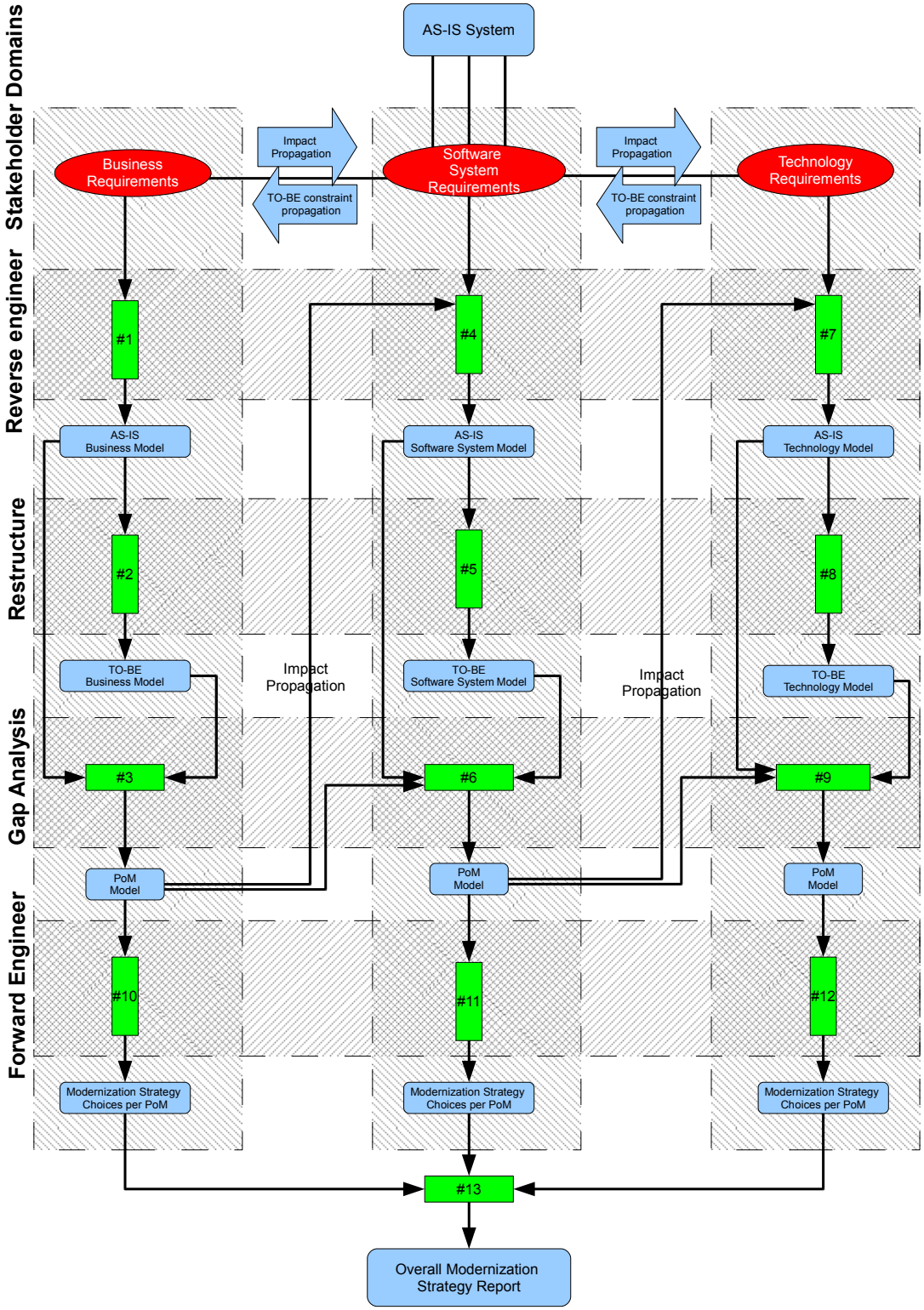


Figure C.22: Framework - impact analysis process

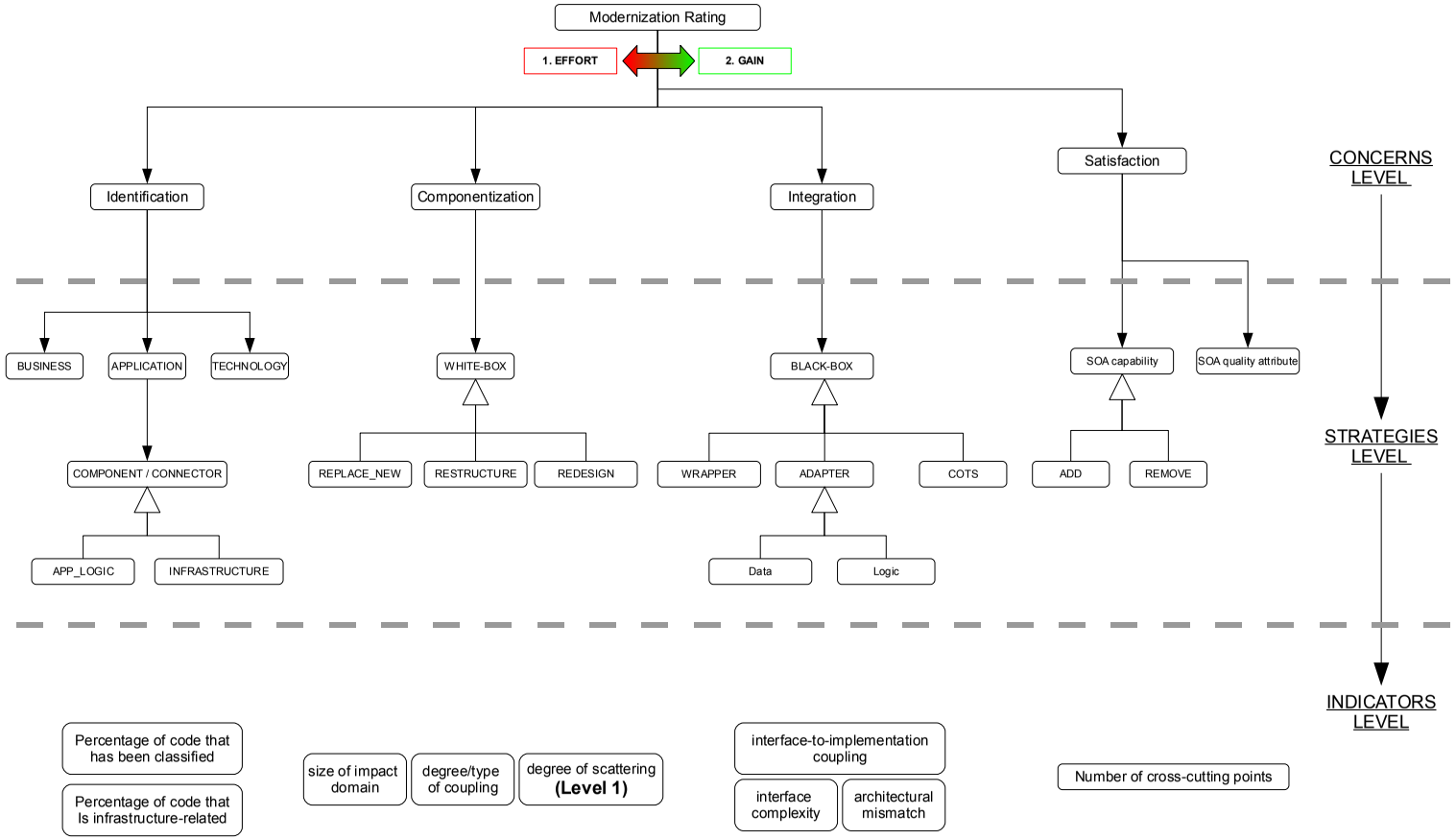


Figure C.23: Framework - Rating Model

Figures-VIII. Experiment (section 4)

Experiment design (section 4.2.5)

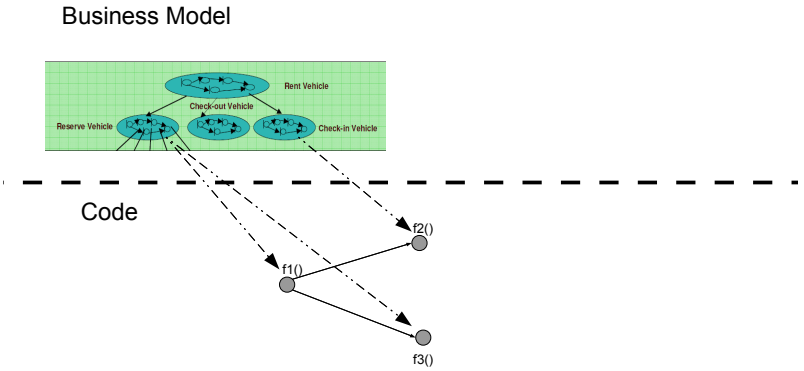


Figure C.24: Business Model extraction and traceability to the code level

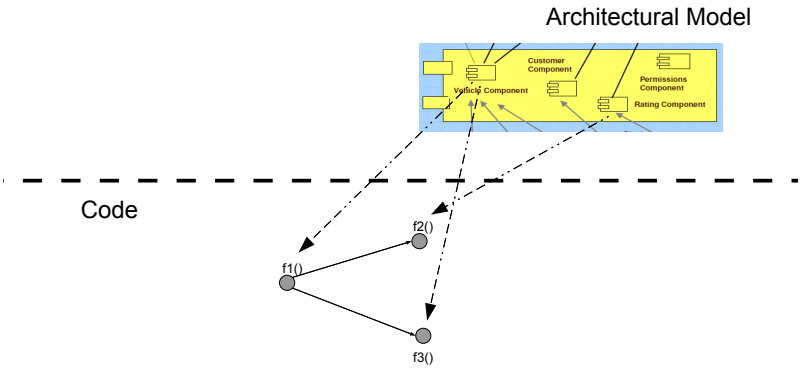


Figure C.25: Architectural Model extraction and traceability to the code

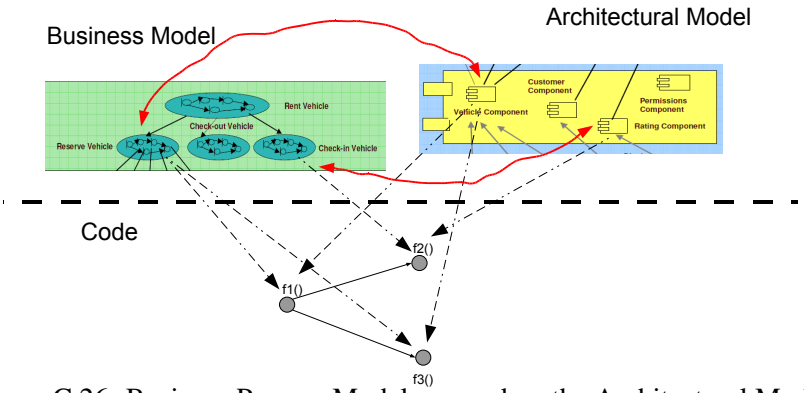


Figure C.26: Business Process Model mapped on the Architectural Model

Experiment instrumentation (section 4.2.6)

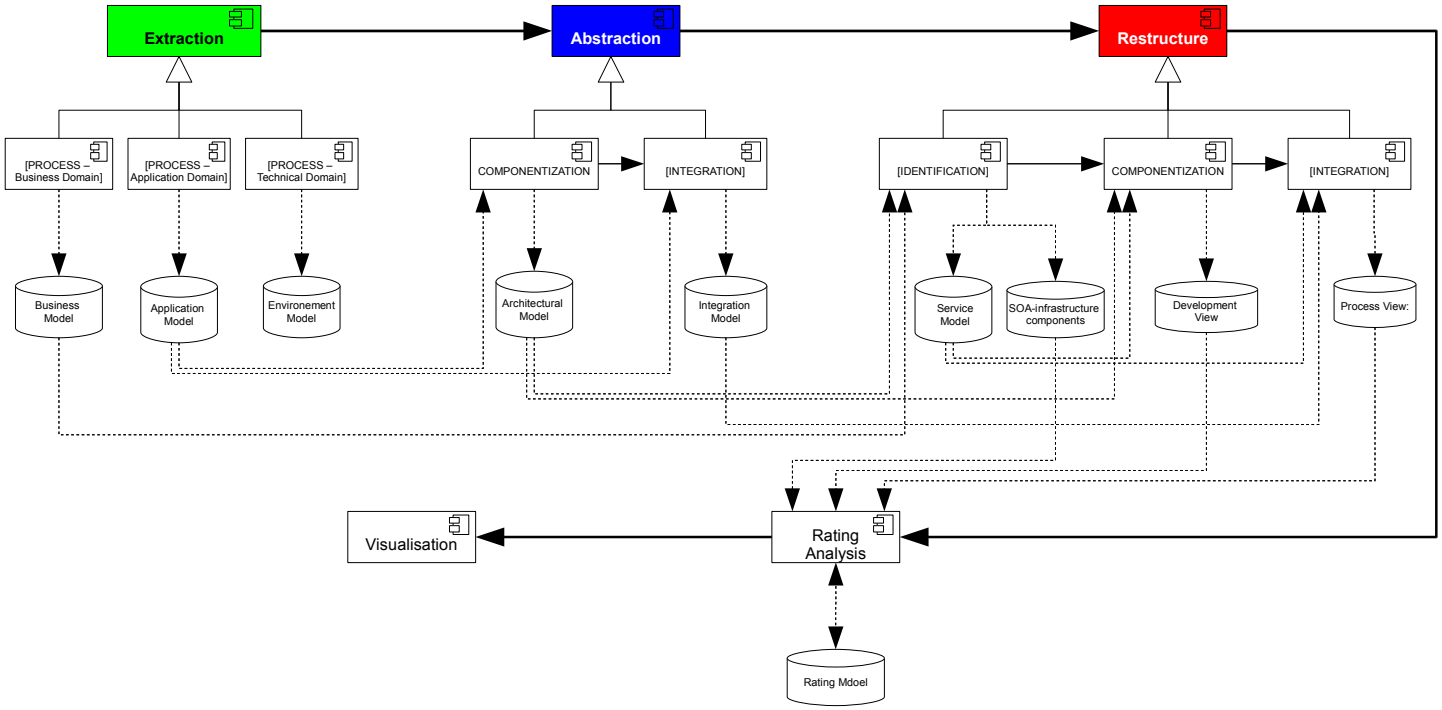


Figure C.27: Overview of the experiment software prototype design

Experiment validity (section 4.2.7)

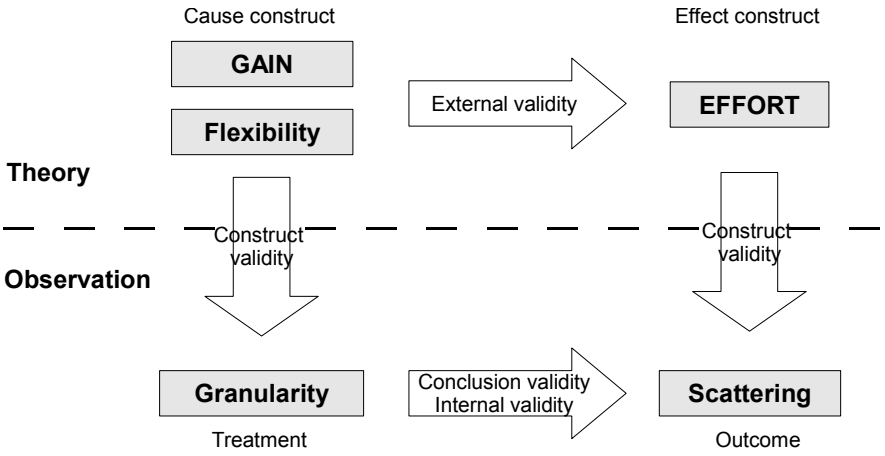


Figure C.28: Experiment validity (adapted from [99])

Experiment operation (section 4.3)

Type1	ID1	Name1	Label1	RelNameUser	Type2	ID2	Name2	Label2	RelName2
PROGRAM	2	A1DBX3	A1DBX3	Calls Program thru Decision	DECISION	6	A1DBX3@1.156.14	A1DBX3.Calls.OU-CE-FIN-OU-ID-SQL-SEL-F	ResolvesToProgramEntry
PROGRAM	2	A1DBX3	A1DBX3	Calls Program thru Decision	DECISION	7	A1DBX3@1.182.18	A1DBX3.Calls.EDALVZ2-PK-SQL-SEL-F	ResolvesToProgramEntry
PROGRAM	2	A1DBX3	A1DBX3	Calls Program thru Decision	DECISION	8	A1DBX3@1.235.14	A1DBX3.Calls.LOCA-EXCHG-RATE-SEL-DRV-F	ResolvesToProgramEntry
PROGRAM	2	A1DBX3	A1DBX3	Calls Program thru Decision	DECISION	9	A1DBX3@1.247.14	A1DBX3.Calls.DEC152-TO-CHAR-CVT-F	ResolvesToProgramEntry
PROGRAM	2	A1DBX3	A1DBX3	Calls Program thru Decision	DECISION	10	A1DBX3@1.252.14	A1DBX3.Calls.CHAR-RIGHT-ALIGN-DRV-F	ResolvesToProgramEntry
PROGRAM	4	AA01A	AA01A	Calls Program thru Decision	DECISION	11	AA01A@1.128.14	AA01A.Calls.EDAVVH3-PN-NAME-SQL-SEL-F	ResolvesToProgramEntry
PROGRAM	7	AA0BW1	AA0BW1	Calls Program thru Decision	DECISION	12	AA0BW1@1.1002.14	AA0BW1.Calls.OU-BUS-FCT-OU-ID-SEL-DRV-F	ResolvesToProgramEntry
PROGRAM	7	AA0BW1	AA0BW1	Calls Program thru Decision	DECISION	13	AA0BW1@1.1052.14	AA0BW1.Calls.OU-CE-WL-STOCK-SQL-SEL-F	ResolvesToProgramEntry
PROGRAM	7	AA0BW1	AA0BW1	Calls Program thru Decision	DECISION	14	AA0BW1@1.1108.14	AA0BW1.Calls.R-ACWSXY-F	ResolvesToProgramEntry
PROGRAM	7	AA0BW1	AA0BW1	Calls Program thru Decision	DECISION	15	AA0BW1@1.1248.18	AA0BW1.Calls.FU-ADDR-SQL-EXIST-VAL-F	ResolvesToProgramEntry
PROGRAM	7	AA0BW1	AA0BW1	Calls Program thru Decision	DECISION	16	AA0BW1@1.1334.18	AA0BW1.Calls.FU-ADDR-SQL-EXIST-VAL-F	ResolvesToProgramEntry
PROGRAM	7	AA0BW1	AA0BW1	Calls Program thru Decision	DECISION	17	AA0BW1@1.350.14	AA0BW1.Calls.OU-NONCE-SEL-DRV-F	ResolvesToProgramEntry
PROGRAM	7	AA0BW1	AA0BW1	Calls Program thru Decision	DECISION	18	AA0BW1@1.400.14	AA0BW1.Calls.OU-BUS-FCT-OU-ID-SEL-DRV-F	ResolvesToProgramEntry
PROGRAM	7	AA0BW1	AA0BW1	Calls Program thru Decision	DECISION	19	AA0BW1@1.452.14	AA0BW1.Calls.OU-REL-VW-EXIST-SQL-SEL-F	ResolvesToProgramEntry
PROGRAM	7	AA0BW1	AA0BW1	Calls Program thru Decision	DECISION	20	AA0BW1@1.505.14	AA0BW1.Calls.OU-CE-WL-RV-SQL-SEL-F	ResolvesToProgramEntry
PROGRAM	7	AA0BW1	AA0BW1	Calls Program thru Decision	DECISION	21	AA0BW1@1.593.14	AA0BW1.Calls.OU-NONCE-SEL-DRV-F	ResolvesToProgramEntry
PROGRAM	7	AA0BW1	AA0BW1	Calls Program thru Decision	DECISION	22	AA0BW1@1.633.14	AA0BW1.Calls.OU-CE-WL-STOCK-SQL-SEL-F	ResolvesToProgramEntry
PROGRAM	7	AA0BW1	AA0BW1	Calls Program thru Decision	DECISION	23	AA0BW1@1.717.14	AA0BW1.Calls.OU-NONCE-SEL-DRV-F	ResolvesToProgramEntry
PROGRAM	7	AA0BW1	AA0BW1	Calls Program thru Decision	DECISION	24	AA0BW1@1.767.14	AA0BW1.Calls.OU-BUS-FCT-OU-ID-SEL-DRV-F	ResolvesToProgramEntry
PROGRAM	7	AA0BW1	AA0BW1	Calls Program thru Decision	DECISION	25	AA0BW1@1.819.14	AA0BW1.Calls.EDAFVB1-ENTRY-SEL-F	ResolvesToProgramEntry
PROGRAM	7	AA0BW1	AA0BW1	Calls Program thru Decision	DECISION	26	AA0BW1@1.894.20	AA0BW1.Calls.R-AA7NX7-F	ResolvesToProgramEntry

Figure C.29: Excerpt from the call map export of the subject legacy system generated with Relativity

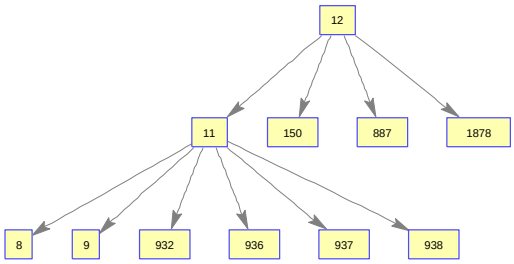


Figure C.30: Business Process for entry-point node #12

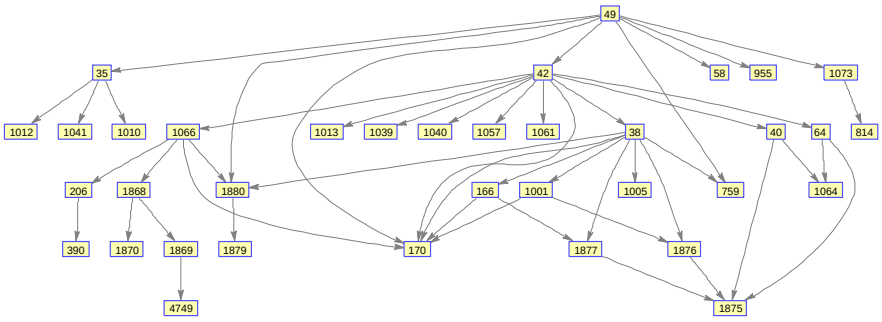


Figure C.31: Business Process for entry-point node #49

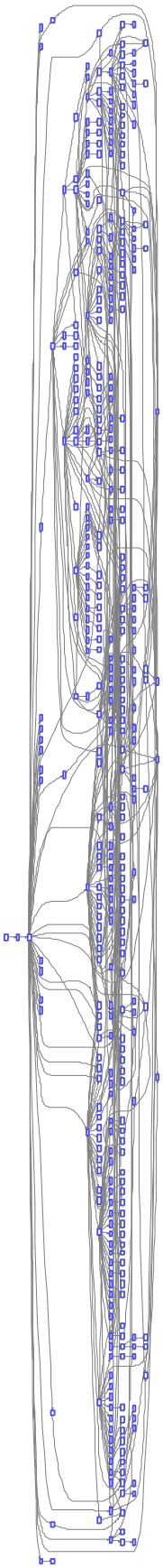


Figure C.32: Business Process for entry-point node #5174

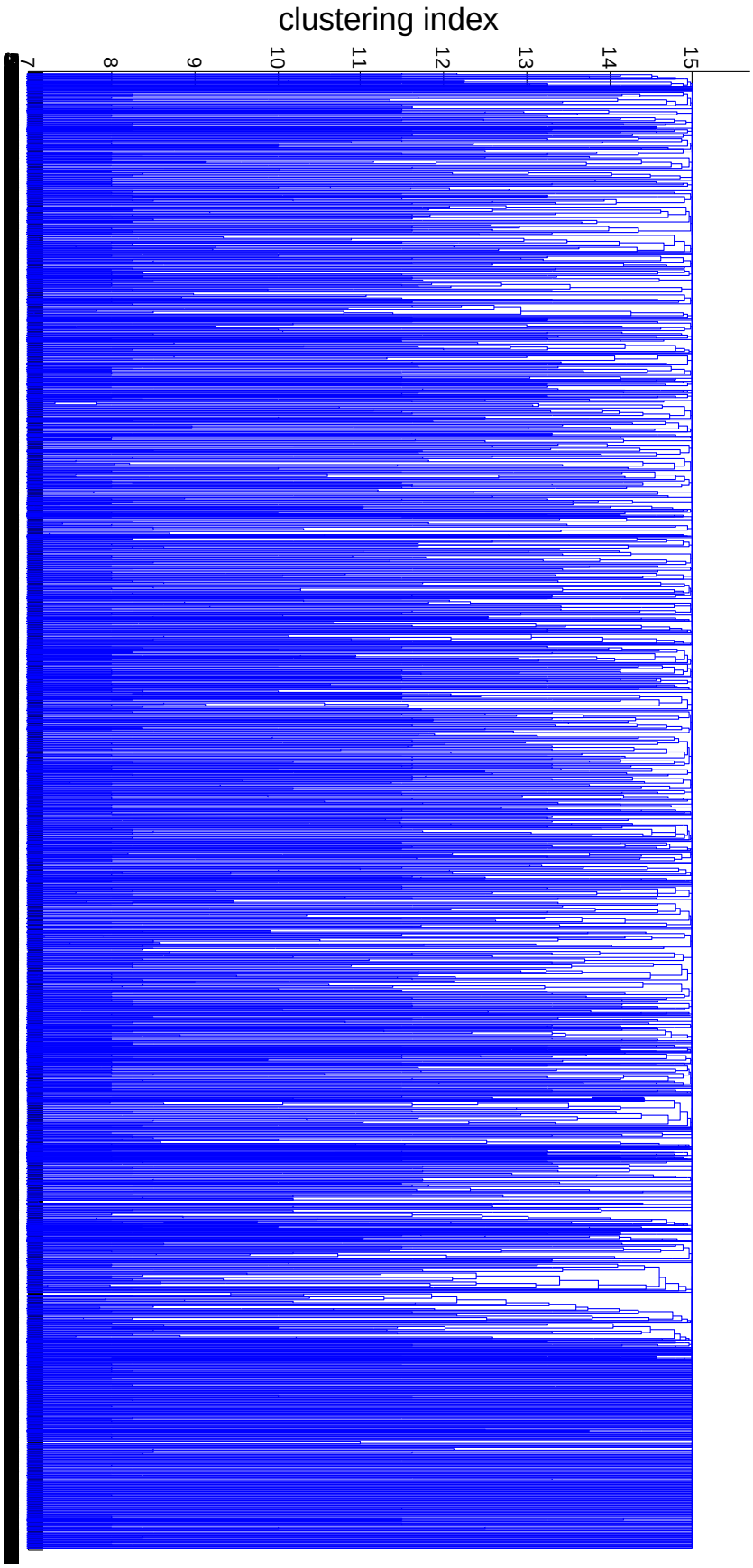


Figure C.33: Architectural Model - complete hierarchical clustering dendrogram of the LLS Request Handling subsystem

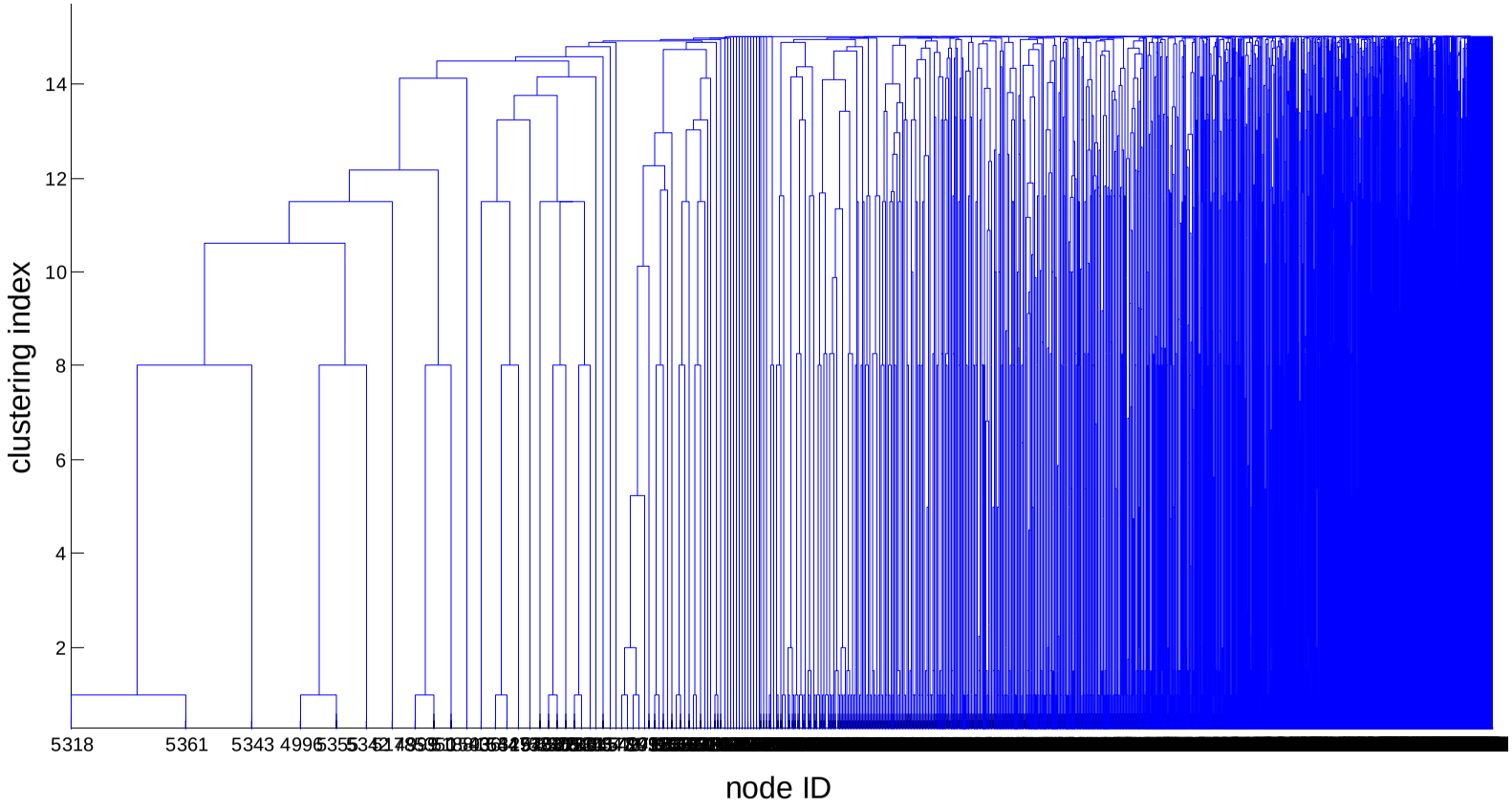


Figure C.34: Architectural Model - complete hierarchical clustering dendrogram of the LLS Request Handling subsystem. The leftmost nodes are magnified.

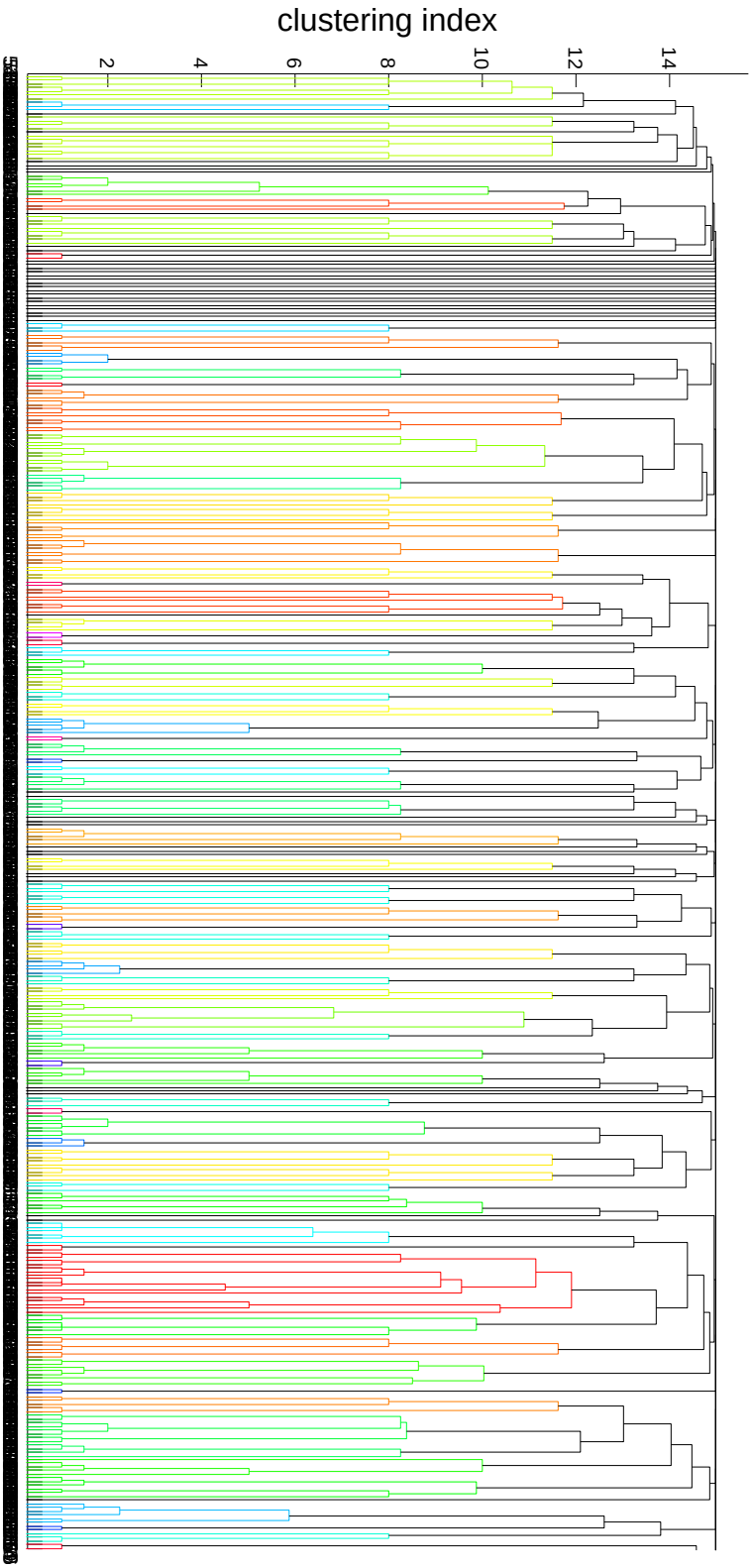


Figure C.35: Component definition based on threshold in hierarchical clustering (threshold=12)

Experiment analysis (section 4.4)

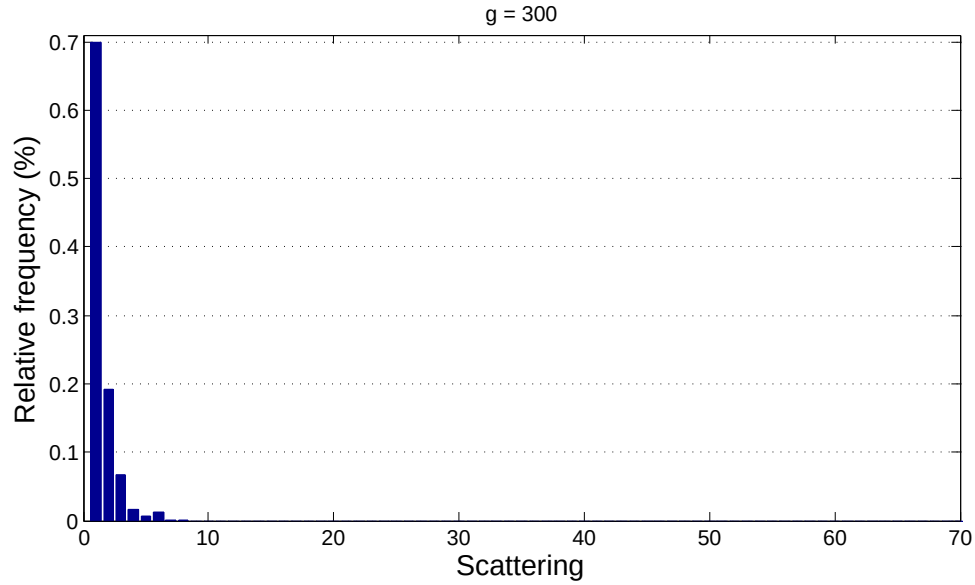


Figure C.36: D_{300} – scattering distribution for granularity level of 300

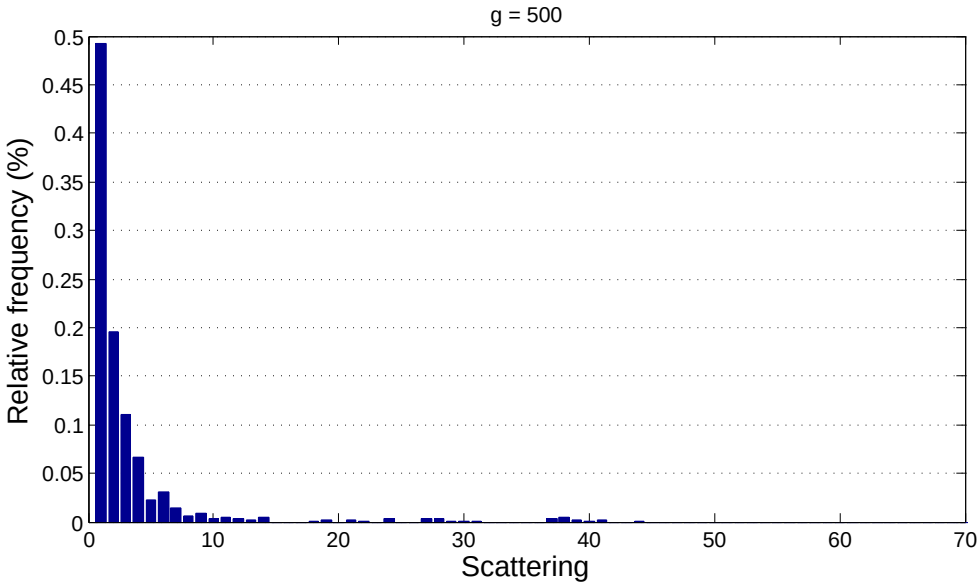


Figure C.37: D_{500} – scattering distribution for granularity level of 500

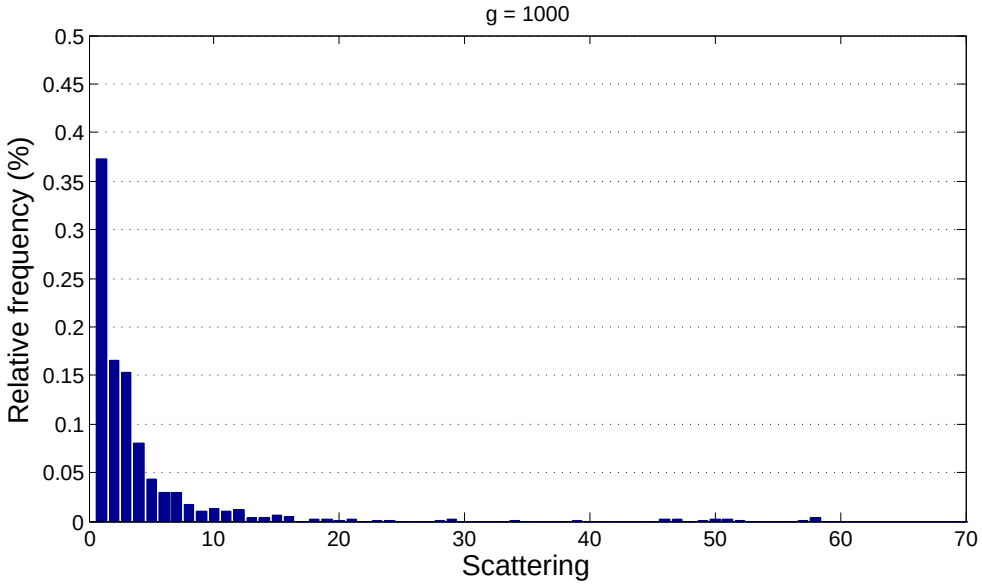


Figure C.38: D_{1000} – scattering distribution for granularity level of 1000

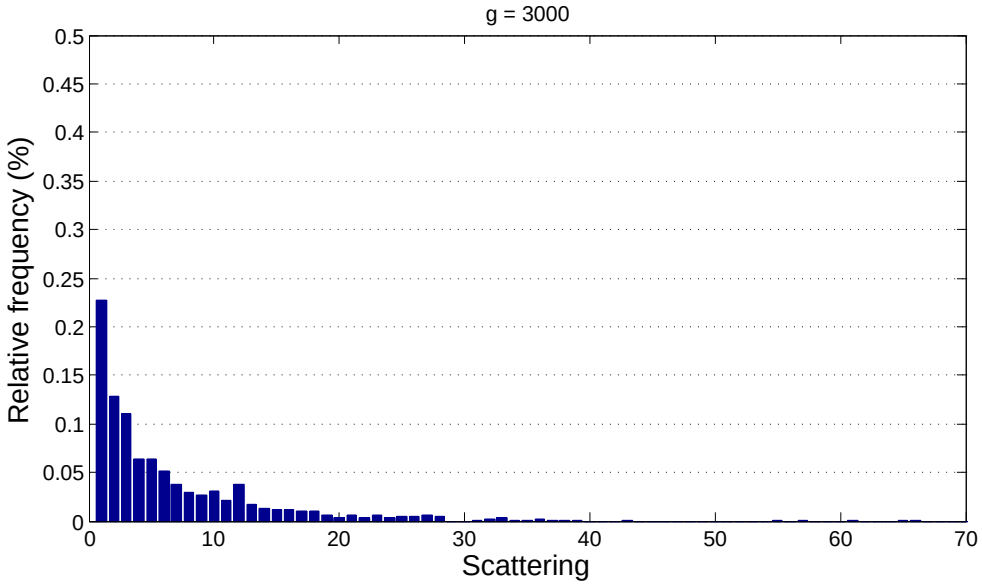


Figure C.39: D_{3000} – scattering distribution for granularity level of 3000

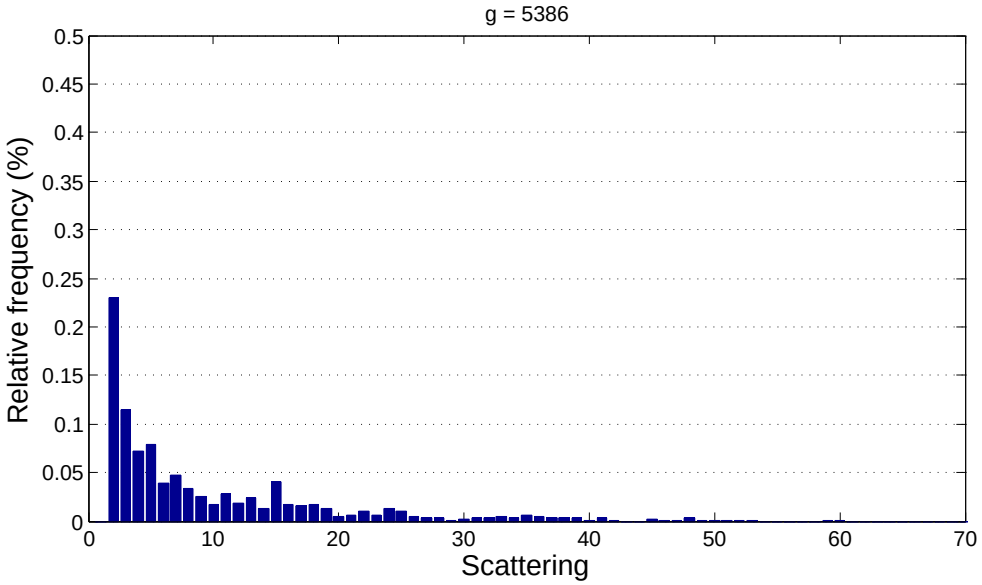


Figure C.40: D_{5386} – scattering distribution for granularity level of 5386

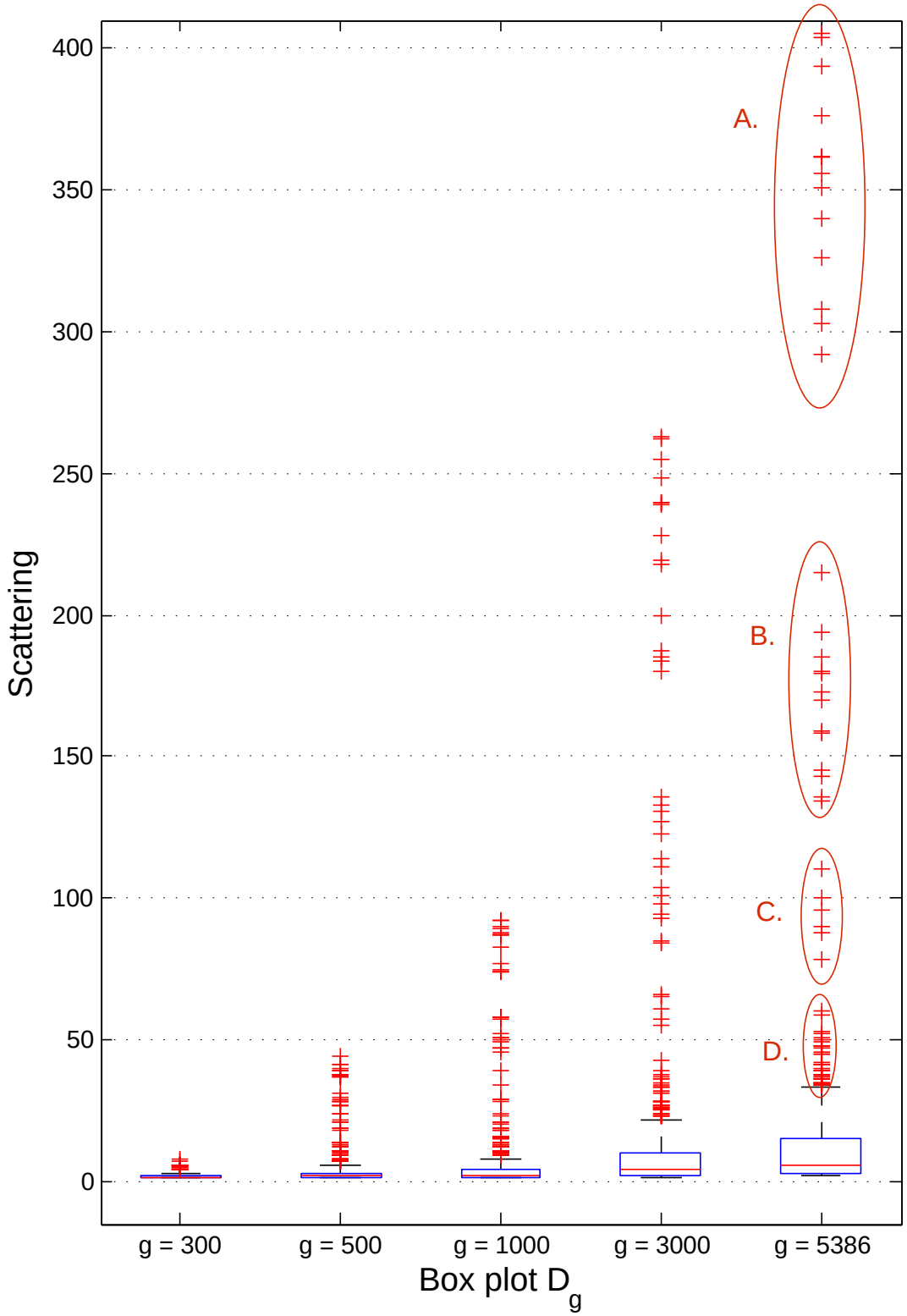


Figure C.41: Box plot of five scattering distributios D_g with g=300, 500, 1000, 3000, 5386

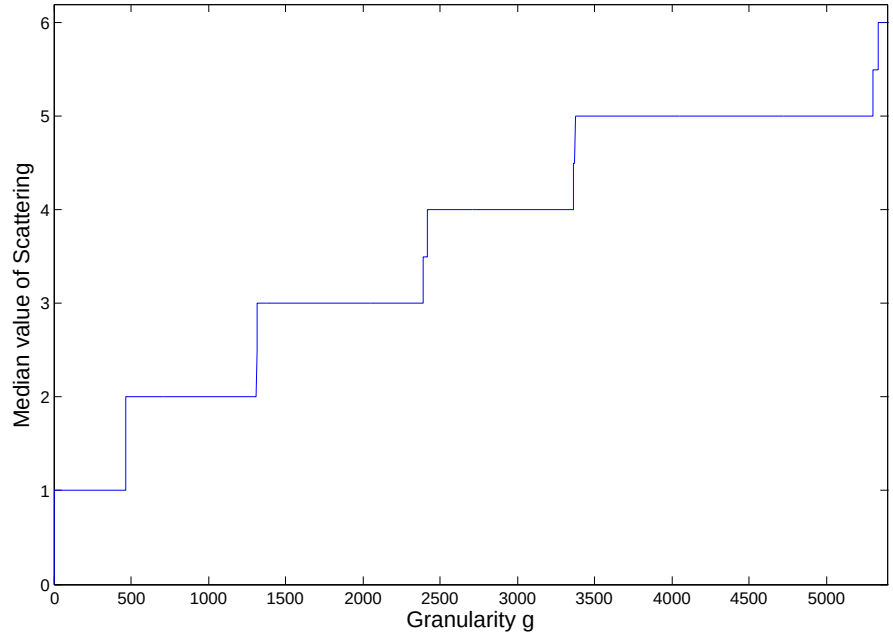


Figure C.42: Median value of the Scattering distributions for all granularities

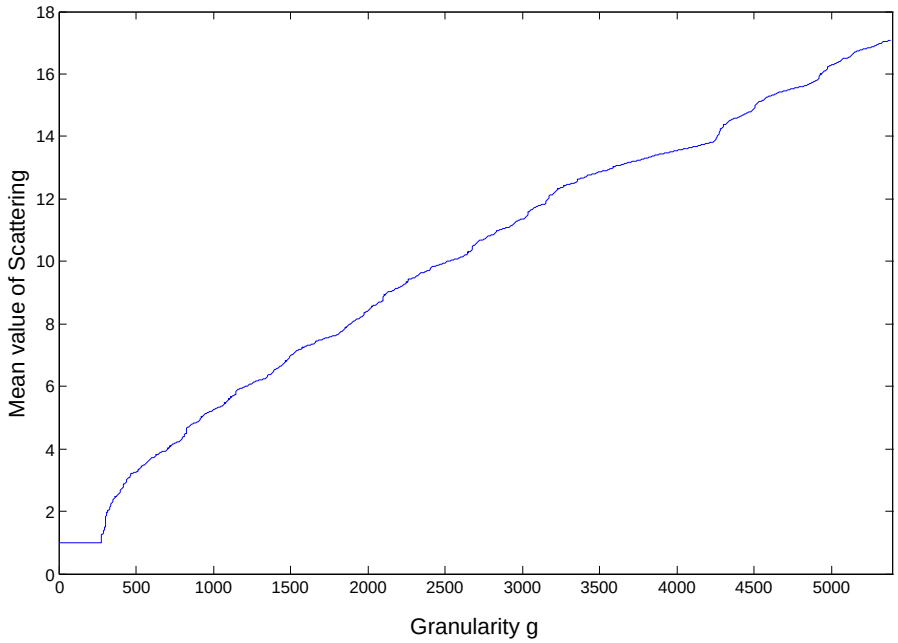


Figure C.43: Mean value of the Scattering distributions for all granularities

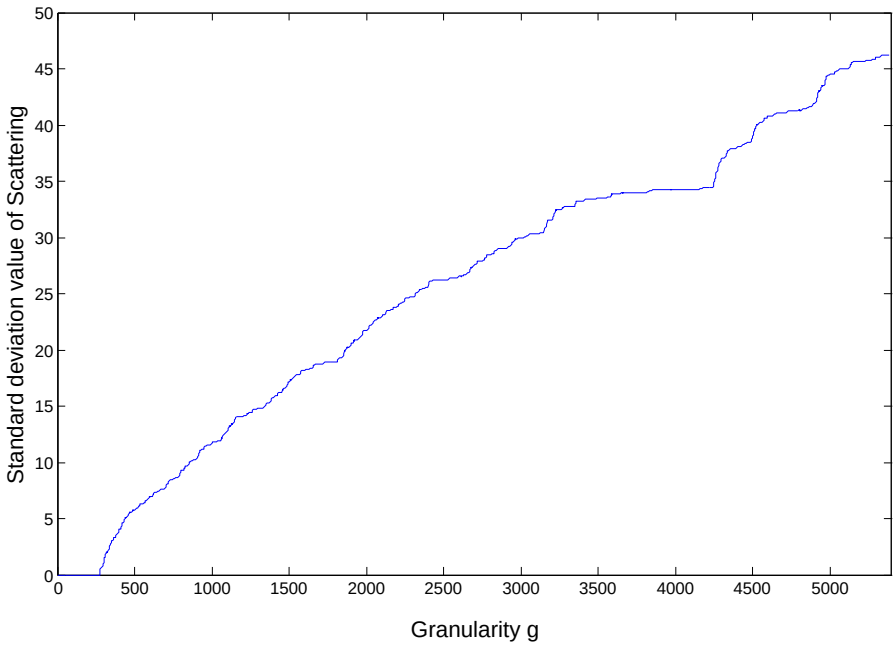


Figure C.44: Standard deviation value of the Scattering distributions for all granularities

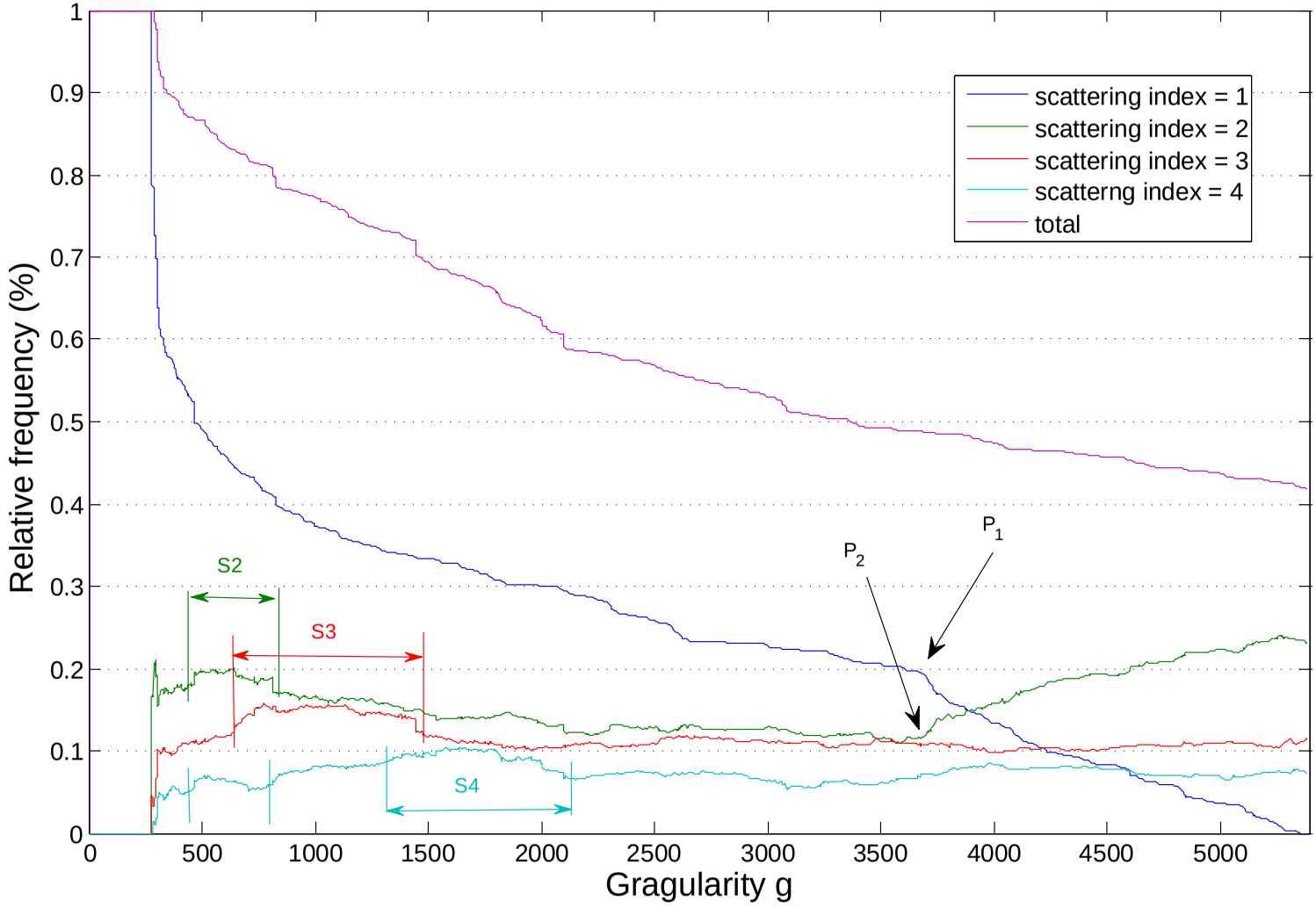


Figure C.45: Relative distribution of scattering indexes 1,2,3 and 4. The figure shows the ripple effect from lower to higher granularities (S2 $\rightarrow$ S3 $\rightarrow$ S4). This is caused by business processes getting more scattered over the implementing them components due to an increased granularity. Also shown is a turning point in the scattering development (P_1, P_2). At granularity level 3650, the granularity is so high that the simple two component business process start getting split

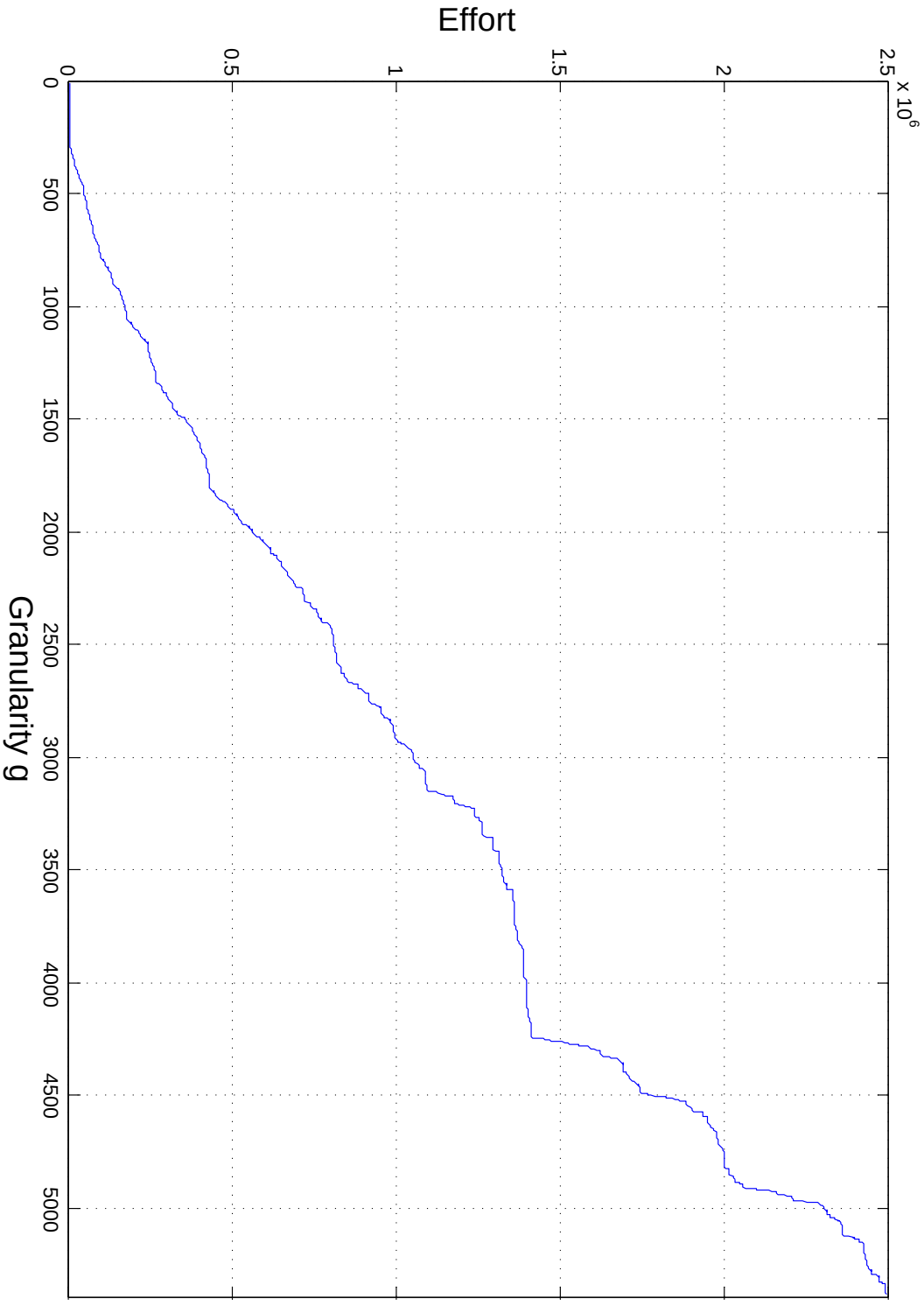


Figure C.46: Total Effort based on the Scattering over each granularity level

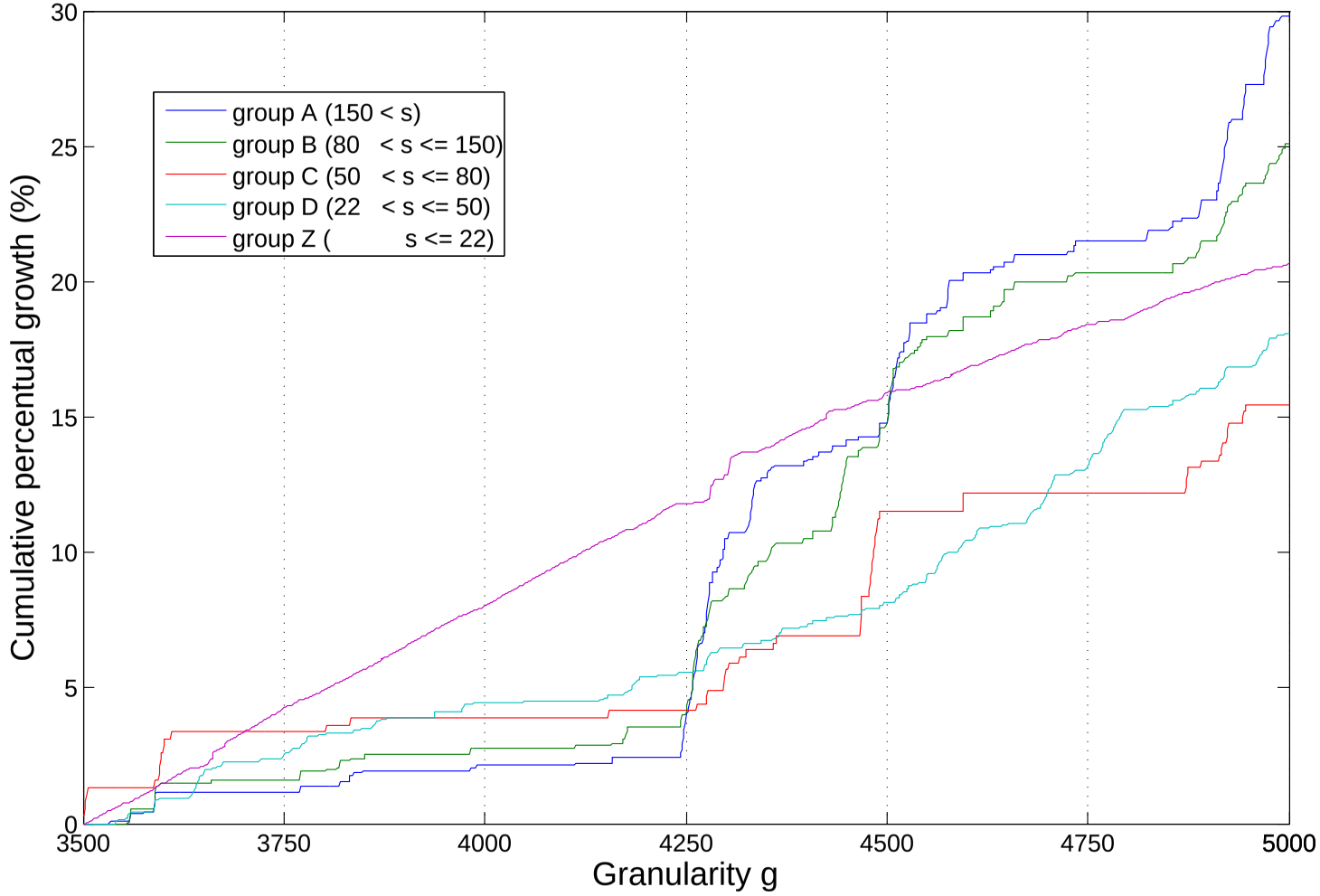


Figure C.47: Difference in growth of the scattering between groups of business processes with different complexity (A, B, C, D, Z). The higher the business process complexity the sharper the growth. This growth spurt of the scattering explains the sudden growth in the overall Effort above granularity 4250.

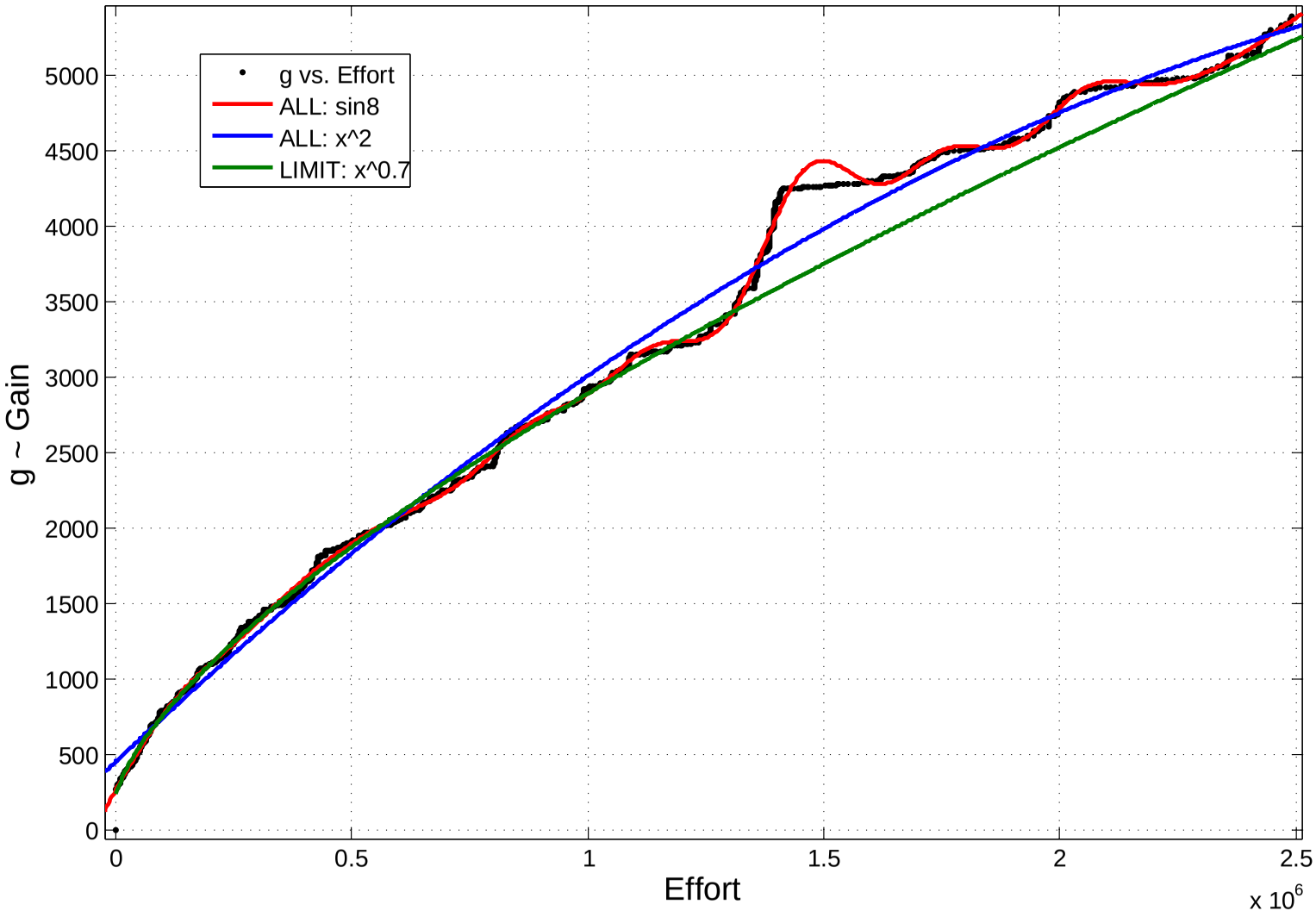


Figure C.48: Best approximations through regression analysis of the Gain/Effort relationship. The limited set of points (LIMIT) for Effort $<= 1.25x10^6$ has an infinitely increasing approximation ($x^{0.7}$). However the trend in the complete set of data points (ALL) is best approximated by a function with a maximum (x^2). The best fitting approximation of the whole data set is the periodic sum of sin functions.

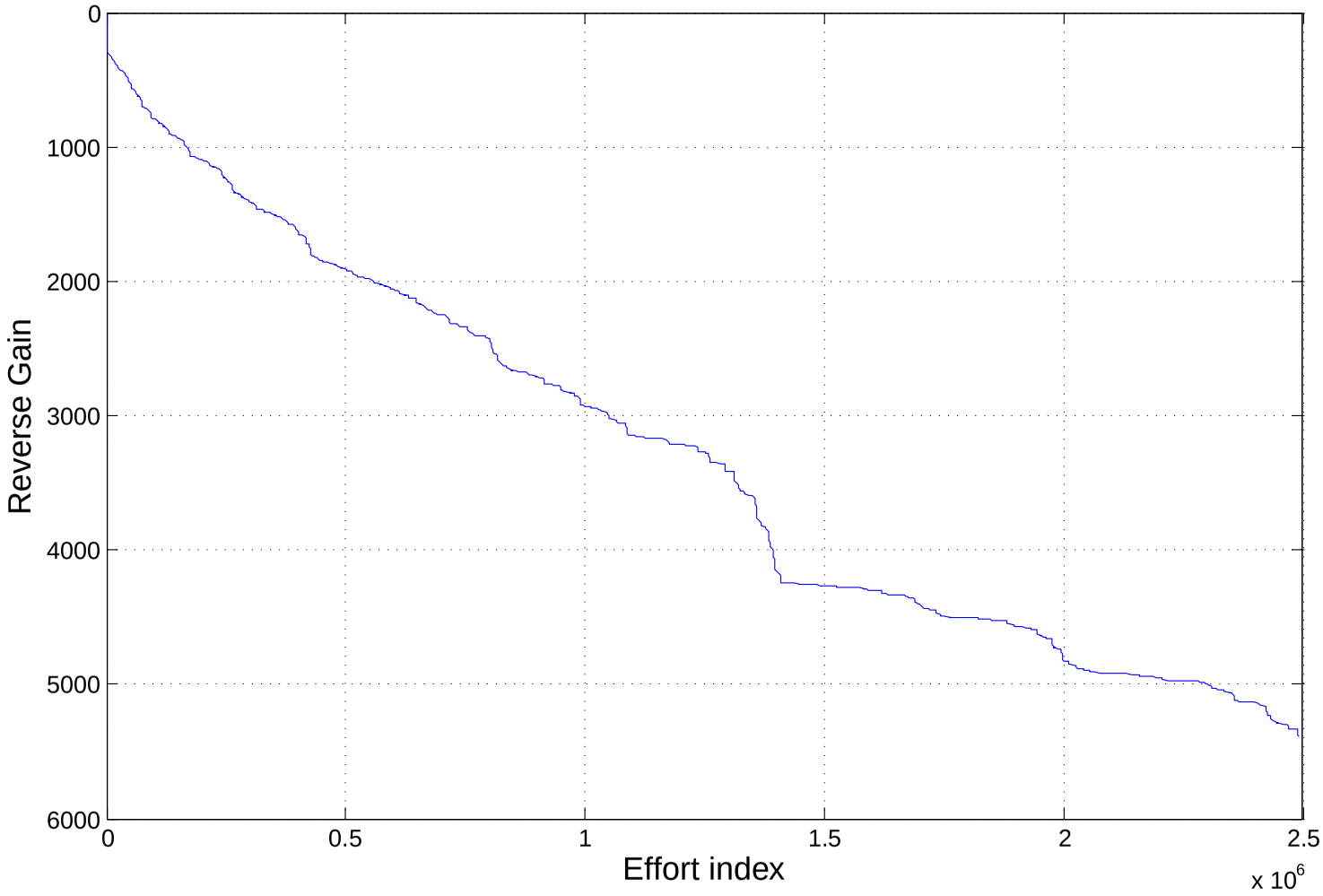


Figure C.49: Trade-off analysis - Pareto frontier. The optimization function is Min $\sum Effort + (1 - Gain)$. The optimal solutions are the ones closer to the origin of the graph.

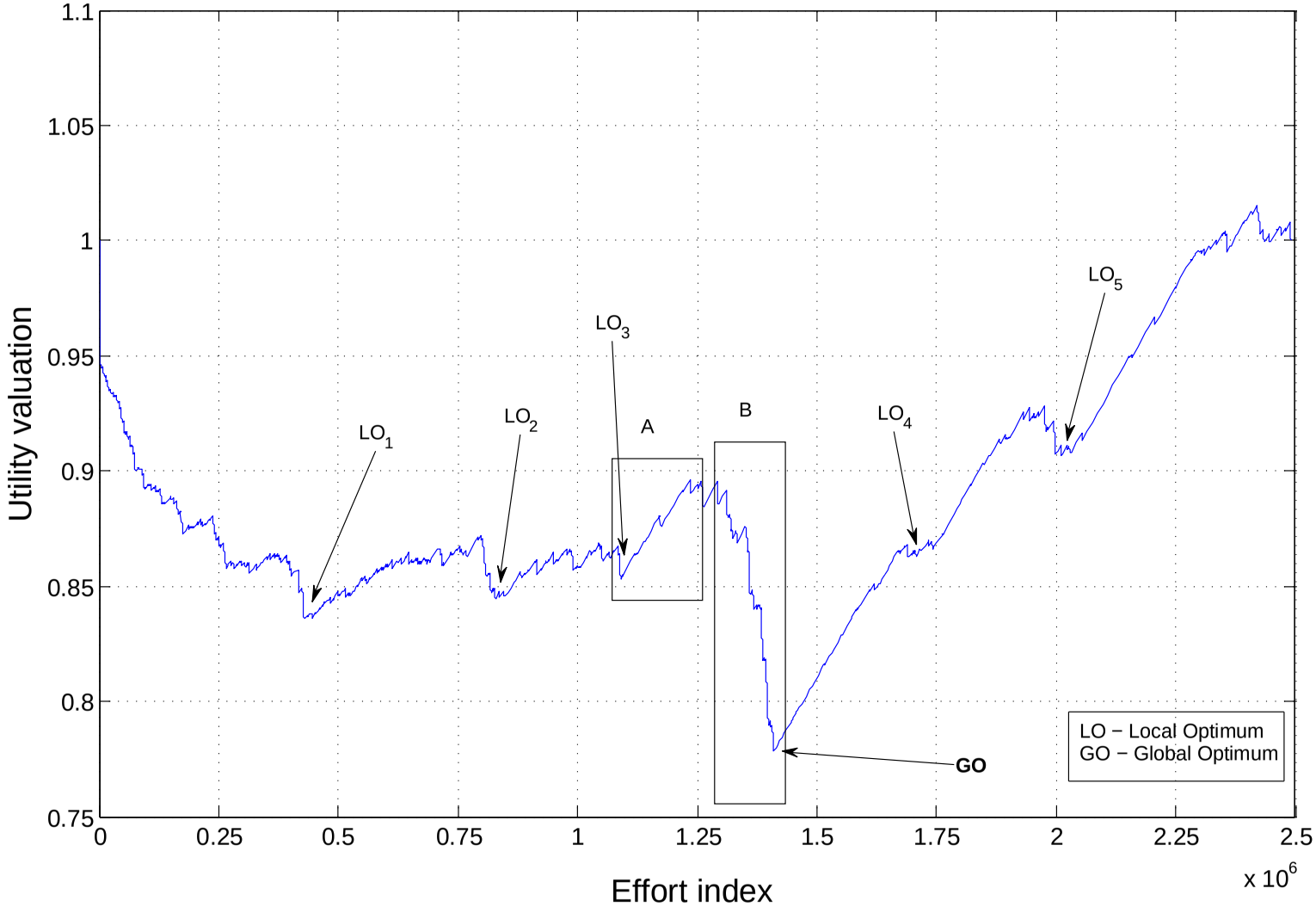


Figure C.50: Optimal solutions of the trade-off analysis based on the Gain and Effort data. The figure shows the global optimum (GO) and several local optima in order of valuation (LO_1, LO_2, LO_3, LO_4, LO_5). The optimization function is Min $\sum Effort + (1 - Gain)$, as shown in C.49.

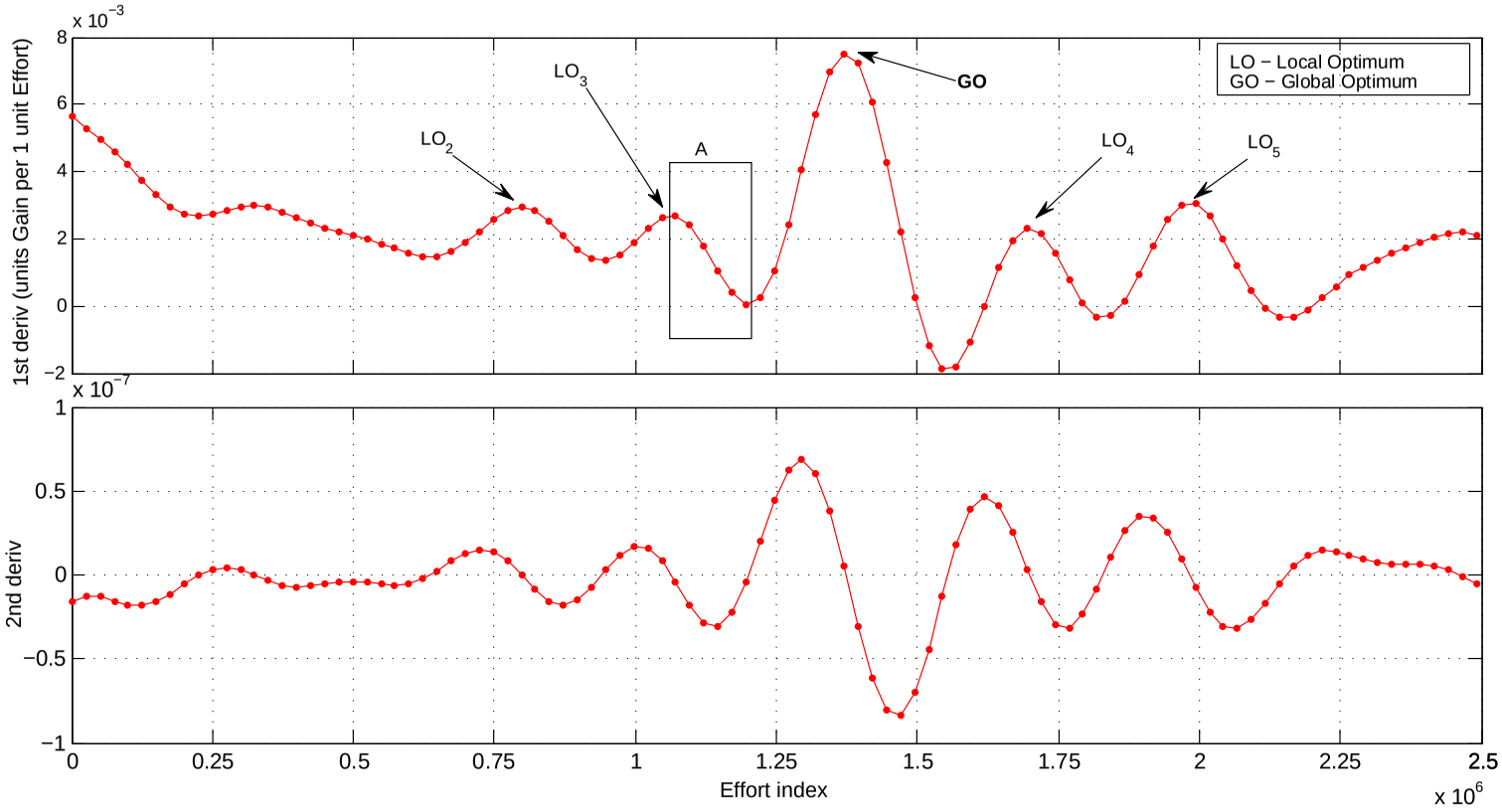


Figure C.51: Trade-off analysis - rate of change around optimal solutions

Experiment future work (section 4.6.1)

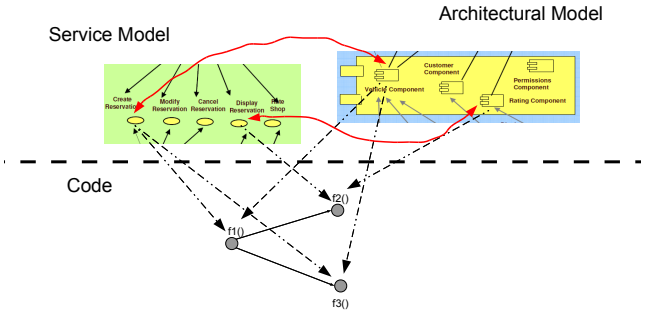


Figure C.52: Experiment future work - Service Model mapped to the Application Model legacy components

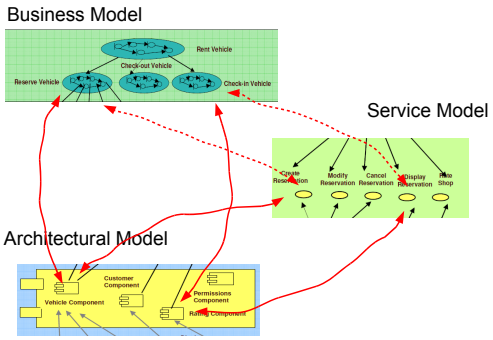


Figure C.53: Experiment future work - Business Model mapped to the Service Model using the Application Model

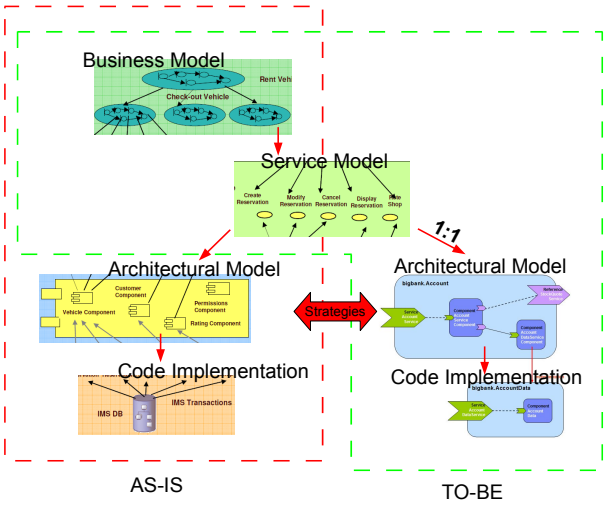


Figure C.54: Experiment future work - building the to-be architectural model using SCA concepts based on an 1-to-1 mapping of the services and the implementing components

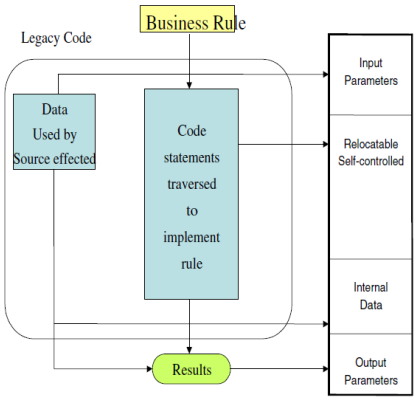


Figure C.55: Experiment future work - service encapsulation model [89]

Figures-IX. Future work (section 6.3)

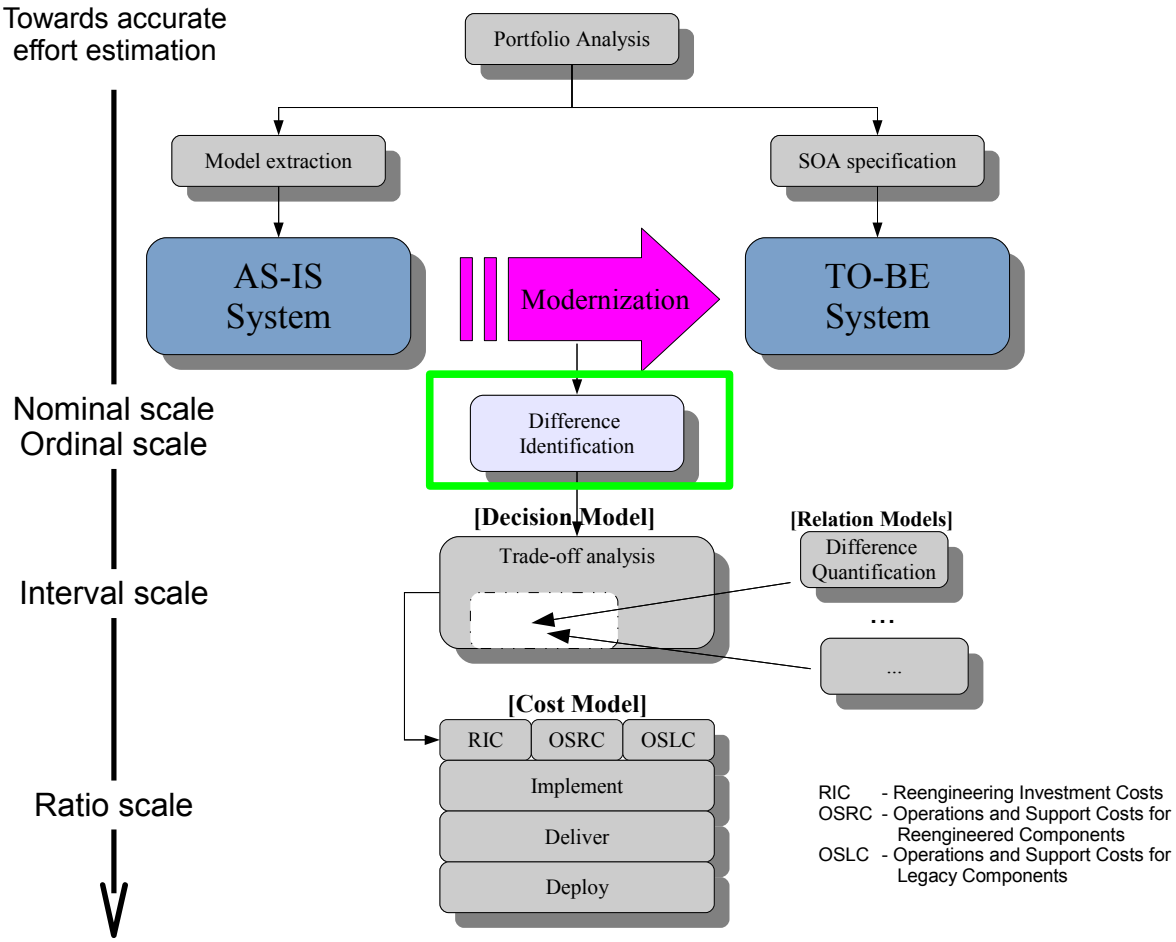


Figure C.56: Future work overview. There are several phases in the process of reaching an effective effort estimation. This research has focused on the Difference identification phase. There are however other phases that need further work such as portfolio analysis, decision models and the cost estimation base on the effort