Simulations of Very Early Life Cycle Quality Evaluation

The Softgoal Simulation Tool

by

Eliza Chiang

B.ASc, The University of British Columbia, 2001

A THESIS SUBMITTED IN PARTIAL FULFILMENT OF
THE REQUIREMENTS FOR THE DEGREE OF

MASTER OF APPLIED SCIENCE

in

THE FACULTY OF GRADUATE STUDIES

(Department of Electrical and Computer Engineering)

We accept this thesis as conforming
to the required standard

. . . .

. . . .

THE UNIVERSITY OF BRITISH COLUMBIA

October 7, 2003

Abstract

The purpose of this research was to develop a tool for quality estimation and decision making during very early life cycle of software system, when data is scarce and information is unavailable. This thesis begins with the introduction and related work chapters, which identifies the problems with achieving quality software and the current state-of-the-art of system modeling and simulation. Then, the requirement analysis technique proposed in this paper is presented: first, a model is needed to capture the tradeoffs/synergy between software quality attributes and applicable design alternatives. We have adopted the softgoal framework by *Chung, Nixon, Yu and Mylopoulous* [28] in modeling such knowledge for analysis. Second, an inference engine is built to automatically execute the text-encoded softgoal framework. Thirdly, we incorporate *Monte Carlo* simulation to explore the wide range of behaviors in the model, and summarize these behaviors with a treatment-learning tool named TAR2. Five case studies have been presented through chapter 3 to chapter 8 to demonstrate the usage of the *Softgoal Simulation Tool* to model and simulate various systems. This thesis concludes with a summary of research results, contributions to the related software engineering field, and further research directions.

Contents

List of Tables

List of Figures

Preface

Prior related publications:

- Simulations for Very Early Life Cycle Quality Evaluation; E. Chiang and T. Menzies; *Software Process: Improvement and Practive (to be appear)*; 2003
- Simulations for Very Early Life Cycle Quality Evaluation; E. Chiang and T. Menzies; *In Proceedings: Prosim'03; Portland State University*, May 3-4, 2003, Portland, USA

Acknowledgements

This research was conducted at University of British Columbia and West Virginia University, partially under NASA contract NCC2-0979. In part, this work was sponsored by the NASA Office of Safety and Mission Assurance under the Software Assurance Research Program led by the *NASA IV&V* Facility. Reference herein to any specific commercial product, process, or service by trade name, trademark, manufacturer, or otherwise, does not constitute or imply its endorsement by the United States Government.

Chapter 1

Introduction

There was a time when we rely solely on physical items such as hammers, shovels, wires, chemicals to do the job, or typewriters and telephones to keep us in touch with our clients and each other. Over time, our reliance on software has grown to such an extend that life without software products seem fictitious. As noted by the U.S. President's Information Technology Advisory Committee[31]: "Software is the new physical infrastructure of the information age. It is fundamental to economic success, scientific and technical research, and national security."

Is it good or bad to rely so much on software? Clearly, software technology has enabled us to do more, do it quickly, and do it with greater precision. But in the rush of improving our products and processes with software, we may have brought in the new speed and functionality together with risks and vulnerability. The President's Committee stated its concern: "The Nation needs robust systems, but the software our systems depend on is fragile"[31]. Fragile software may work some of the time, but is prone to functioning improperly under variable environment, and its failure can lead to severe problems not only to users but also to developers and maintainers. For example, a severe MCI WorldCom outage in August 1999 left thousands of business customers without access to the Internet, emails and other services [119]. One customer, the Chicago Board of Trade, lost four days of services worth hundreds of millions. The failure was due to problem with software provided by Lucent Technologies. Although Lucent assumed full responsibility for the failure, a spokesman for the company said that the same problem had never happened to other clients of their software[80]. Lucent, therefore, had to replicate the problem that had never occurred, retesting after the fix, and maintaining the software for the changes occurred to the software, users, and the environment. Managing such process is not an easy task, especially when changes involve in the requirement and design of the software.

1.1 "Fragile" Software

Some common system behaviors where software fragility manifests are:

- Unavailability and unreliability;
- Security;
- Unpredictable performance;

Many examples reveal that problems with software fragility persist. A few of them are listed below:

- In 1998 and 1999, companies such as eBay, Schwab, and E-Trade were affected by Web-site failure - unavailability ranging from 15 minutes to 2 hours - resulting in significant loss of revenue, substantial drop in market confidence, and even investigations by the Securities and Exchange Commission [140].
- In February 1998, Vladimir Levin was sentenced to 36 months in prison for using computer systems to steal millions of dollars from Citibank in 1994. Between June and August 1994, Levin accessed the Citibank computer, user identification codes, and passwords belonging to bank customers, with which he transferred funds from these accounts to he and his co-conspirators accounts [30].
- Numerous examples are cited in [38] of hackers exploiting systems of Internet Service Providers, credit bureaus such as TRQ and Equifax, and corporation such as Tower Records and ESPN, gaining access to information such as credit card numbers and important identifications.
- Government web pages and university sites have continued to be the target of electronic vandalism.

Here, software failures are the result not of improper function, but of inappropriate reaction to unexpected or uncontrollable environment. While many similar stories on software vulnerabilities are found daily in newspapers and magazines, the industries and researchers constantly look for ways to make quality software: to make it more reliable, secure, and performs predictably.

1.2 Defining Quality

So what does it mean by *quality* software? Reliability, security, predictability, and other similar quality attributes can be subjective, and are dependent in part on who is observing and assessing it. Users may measure the software quality by its product characteristics (e.g., reliability and availability), while manufacturers may adopt a process view, and focus on quality during production and after delivery (e.g., reusability, rework reduction). Moreover, different stakeholders - users, customers, managers, domain experts, and developers - come to the project with diverse expectations and interests. Developers may see system failures as quality defects, but security professionals may see it the other way around. If the definitions of quality are not made explicit, confusion and misunderstanding can lead to bad decisions and poor products.

Quality standards and their interpretations may also vary depending on the particular system being considered. For example, faults tolerated in word-processing software would not in general be acceptable in safety or mission-critical system. Moreover, businesses have to consider the balance between quality and its associated risk in managing software products. In deciding what level of quality is acceptable, managers have to account for the quality required as well as the effects of quality (or the lack of it) in the market[51]. Quality demands effort: to manage configurations, improve processes, audit designs, perform V&V (Verification and Validation), adopt new technologies, etc. It may be cheaper to produce a product of suboptimal quality, but customers may not be willing to tolerate the faulty software; or it may be expensive for developers to assure a particular level of quality, but the customers are not willing to pay the cost passed on to them.

Sometimes there is tension among the strategies used to address different aspects of quality. Attempting to fulfill one quality attribute may sacrifice others. For example, additional security measures may adversely affect performance by requiring additional processing steps or by executing compute-intensive algorithms (e.g., encryption and authentication). Conversely, some techniques used to make a system more reliable may actually reduce security by introducing additional "trust relationships" among system components that can be exploited. To evaluate these trade-offs, resolve conflicts, and obtain the best overall software quality may not be a trivial task.

For all these reasons, software quality is difficult to define and hard to deal with. Yet, dealing with quality attributes can be vital for the success of a software system. This thesis research work aims as defining a way to address software quality attributes in a systematic manner, and hence assisting in decision making for achieving high quality software. The proposed approach is developer-directed: it takes the developers' viewpoint of specific quality attributes on the system under consideration, along with expert knowledge and development techniques to produce customized solution. Developers can evaluate the best direction to build the system according to the level of quality desired. Moreover, it utilizes qualitative, natural-language style descriptive terms for easy and fast prototyping. More concrete description on the research approach is summarized at the end of the chapter.

Prior to the discussion on the research approach, the next section presents an investigation on another issue that the proposed tool addresses - the importance of early quality assessment in reducing faults and improving the final product quality.

1.3 The Role of Early Quality Assessment

Figure 1.1 shows the results of a survey by the United States Government Accounting Office into the success, or otherwise, of some software projects. Only $< 5\%$ of the projects among all are being used, and that a large portion of the unsuccessful software projects are "delivered but never used"(47%). This may suggests that, while most of the failed software products meet all the required functional properties for them to be "deliverable", they may not have the demanded quality standards to be "usable"[1]. This has costed developers and users significant amount of effort and money.

This appalling situation does not only occur in government agencies: a survey of 600 firms (by Boehm in [13]) indicated that 35 percent of them had at least one runaway[2] software project. Most postmortems of these software projects disasters have indicated that their problems would have been avoided if there had been an explicit early concern with idenfiying and resolving their high-risk elements. Boehm also suggested (in [12]) that the cost for a fault found in the field may be a 1000 times higher

[1]This should not be totally blamed on software development, as often times purchasers are unable to identify the desired requirements.

[2]The definition of a software runaway in the book [51] is: "a project that goes out of control primarily because of the difficult of building the software needed by the system."

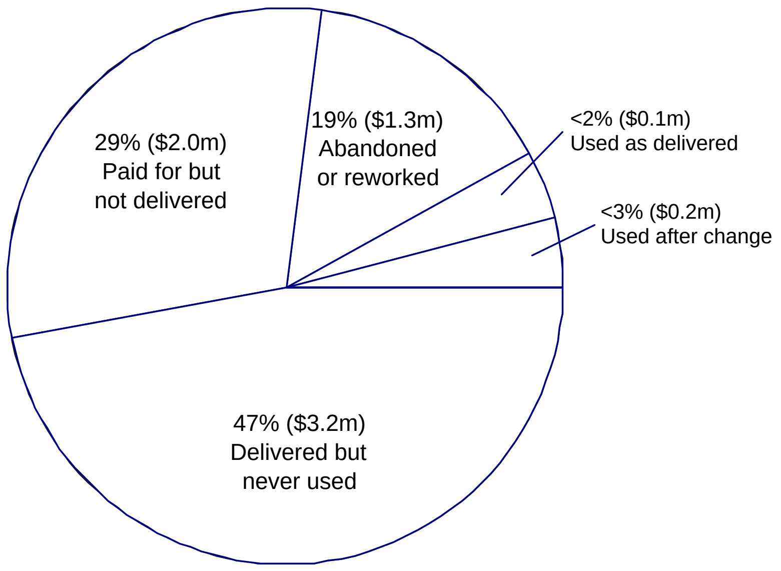


Figure 1.1: Some USA Federal software projects.

than the cost for a fault found in the requirements analysis phase. Similar argument is presented by Moller in [92], which stated that faults detected early in the software project cause only 3% of total fault costs. Moller also presented statistics on fault occurance in various stages of development cycle, two of which are extracted and shown in figures 1.2 and 1.3. Figure 1.2 shows what percentage of fault, according to their own investigations and industrial experiments, arise in different development phases and the percentage which will be detected in the individual test and review phases, as well as in the field. The width of the fault stream is a measure of the number of fault still undetected. As the figure shows, a considerable increase of quality and decrease of cost could be achieved if faults are successfully detected and removed earlier in the development process. To further support this claim, figure 1.3 shows which percentages of costs have to be spent for the activities in the different phases of the software life cycle. While in the requirements analysis and system design phases only 17% and 29% of the costs are used for fault correction, in integration and system test it is much higher - 66% of the total costs.

The above figures suggest that costs increase the later faults are detected in the development life-cycle; consequently, it is advantageous to implement mechanisms for early detection and prevention of faults, such as design flaws. The tool proposed in this thesis is designed to assist decision making in the early life-cycle of software development, so that potential faults can be avoided at an early stage. It also deals with development issues encountered during this stage - uncertain/inconsistent beliefs, and data drought - which are to be discussed in the next section.

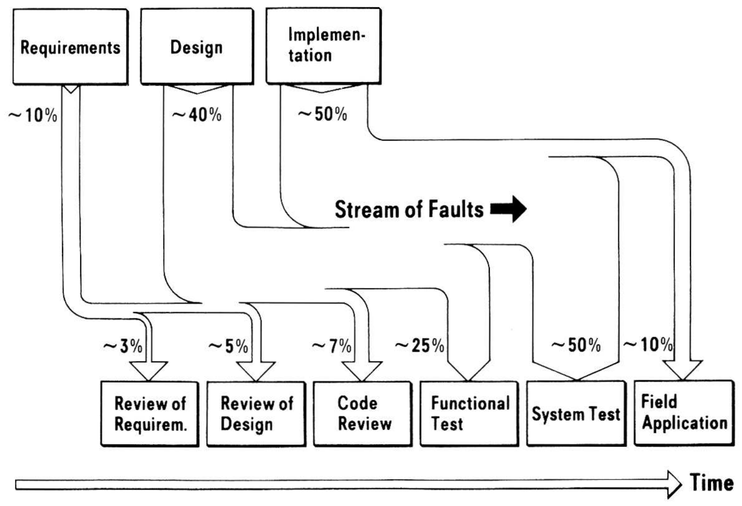


Figure 1.2: Fault statistics showing time of origin and time of detection. [92, fig.10.1]

1.3.1 Issues in Early Estimation and decision making

Uncertainty and Inconsistency

In the early stage of software life-cycle, not much about the exact relationship between design options and the product overall quality is known. Often times developers may have ideas on how each design options affect individual qualitative attributes, in a qualitative manner. For example, a developer may say "implementing shared-data architecture *helps* the space performance of a system". Qualitative influence such as "*helps*" are subjected to individual beliefs, and are thus prone to be inconsistent. However, it is also intuitive to specify this connection in such a simple qualitative format. Therefore, the goal of the tool in this thesis is to find stable conclusion (as of what decisions to make) across uncertainties imposed by subjective knowledge.

Lack of Domain Knowledge

Aside from inconsistent beliefs, another problem with drawing conclusions from a softgoal framework is the lack of supportive data. In the current software engineering practice, there is not much data available to perform statistical analysis on them [85]. This is especially true during the early lifecycle of software development, when decisions are made based on uncertain and subjective knowledge. And in the case of advanced technologies and systems, there is little past experience to learn from. Without supportive data, the relevance of any conclusion drawn from a softgoal framework is questionable. In spite of this, estimations on the potential risks and benefits of design decisions during the earlier requirement phase is essential, because these early decisions have

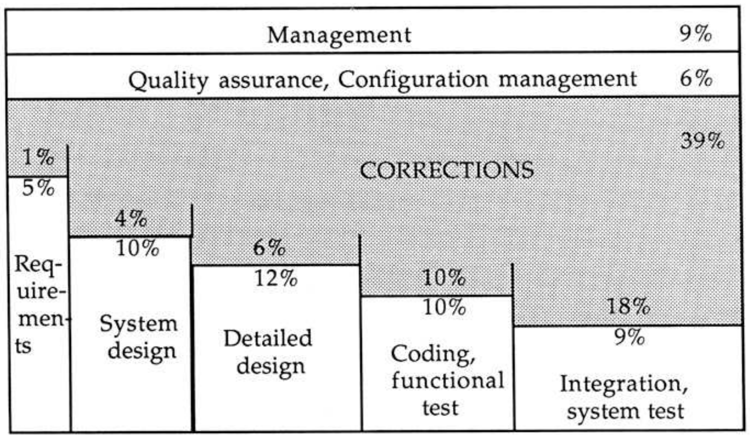


Figure 1.3: Percentages of fault costs compared with the development cost. [92, fig.10.2]

the most leverage to influence the development to follow. The tool presented in this thesis, the *Softgoal Simulation Tool*, is designed to aid decision making in times such as early software development lifecycle, a time when domain knowledge is incomplete and inconsistent.

1.4 Research Approach

The foundation of the *Softgoal Simulation Tool* is based on the idea of System Modeling and Simulation. Models are simplifications of reality and help people to clarify their thinking and improve their understanding of the world. Models can be used for experimentation. A computer model, for instance, can compress times and space and allow many system changes to be tested rather than on an actual system. If a change does not perform well in a model of a system, it is questionable as to whether it will perform well in the actual system itself. One of the main motivations for developing a simulation model is that it is an inexpensive way to gain important insights when the system is still under-development, or when the costs, risks or logistics of manipulating the real system are prohibitive.

The following sections introduce the approaches of system quality modeling and simulation taken by the *Softgoal Simulation Tool*. The premise behind the methodology is also explained.

1.4.1 Softgoal Modeling

The *Softgoal Simulation Tool* adopts the notion of *softgoals*, taken from *Chung, Nixon, Yu and Mylopoulous* [28], to model the quality attributes and other business goals within a system. As there are no clear-cut criteria to determine whether these goals are satisfied, the term "*satisficed*" is used to describes the goals being accomplished at some acceptable level of needs. In [28] *Chung et. al.* define an entire softgoal modeling framework featuring tradeoffs and inter-dependencies between system quality attributes and design alternatives. For example, an analyst can connect two softgoals and say (e.g.,) *softgoal1 helps softgoal2* where *"helps"* is the second strongest of the four qualitative influences defined in a softgoal framework[3]. But their framework is a paper design only: an automatic mechanism is ought to be implemented for the simulation of Softgoal Framework. The basis of the implementation for automatic simulation is discussed in the next section. More about the Softgoal Framework will be presented in chapter 3.

1.4.2 Model Simulation

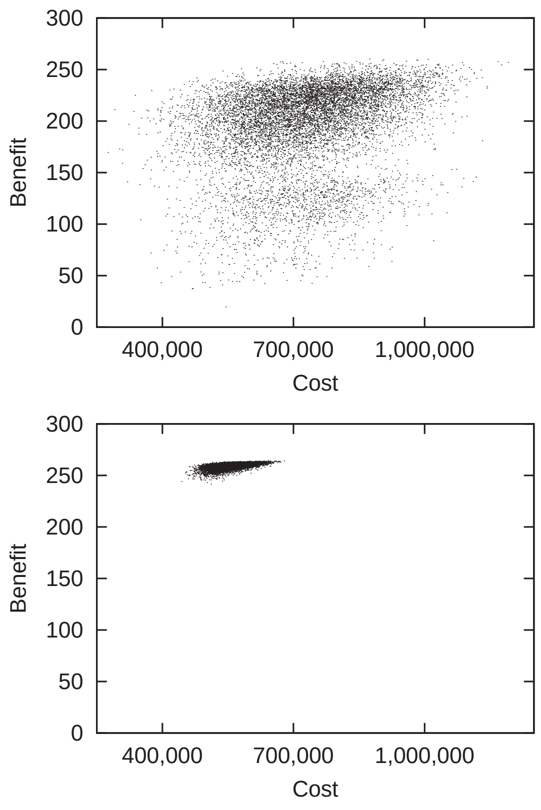


Figure 1.4: An example of treatment learning results

[3]The qualitative influences defined by Chung et al [28] are "MAKE", "HELP", "HURT" and "BREAK".

The premise of the methodology behind the model simulation is that within a large space of uncertainties generated from a model, there often exist emergent stable properties [86]. If isolated, these properties can be used to drive a system towards the more/less preferred direction. In order to find such consistent behaviors, we apply "bounded" randomness (i.e., guesses that fall within some defined range) to handle imprecise knowledge, and utilize *Monte Carlo* simulation [62] to explore a wide range of system behaviors. This generates a large range of behaviors which must be analyzed. TAR2 treatment learner, an analytic tool developed by *Menzies and Hu* [87], is employed to automatically summarize these behaviors, and return recommendations that can drive the system to some preferred mode. For example, *Feather and Menzies* [42] describe one application that used formal requirements model written at *NASA Jet Propulsion Laboratory (JPL)* [138] for deep space satellite design. The formal model could generate a cost and a benefits figure for each possible configuration of the satellite (some 10^{30} options in all). The black dots at the top figure in figure 1.4 shows what happens after 30,000 *Monte Carlo* simulations of that model: note the very wide range of cost and benefits. After treatment learning, a small number of constraints on the satellite configurations were found that, after 30,000 more *Monte Carlo* simulations, yielded the black dots at the bottom figure of figure 1.4. There are two important features of these black dots at the bottom figure. Firstly, compared to the initial black dots (shown in the top figure), the variance in the costs and benefits is greatly reduced. Secondly, the mean value of the costs and benefits are improved; i.e., reduced cost and higher benefits. The success of the *Feather and Menzies* application lead to the speculation that one might understand the space of options within softgoals via *Monte Carlo* simulation and treatment learning.

As for the implementation of the *Softgoal Simulation Tool*, it is designed to be lightweight and highly customizable to different business goals. Our approach is somewhat different from the standard simulation methods in the software engineering or process modeling community. Standard methods include *distributed agent-based simulations* [29], *discrete-event simulation* [55, 65, 75][4], *continuous simulation* (also called *system dynamics*) [4, 128], *state-based simulation* (which includes *Petri Net* and *data flow* approaches) [7, 54, 81], our methods are closer to logic-based [15, chapter 20], or *rule-based* simulations [91]. In our approach:

- A model is defined that is a set of logical constraints between variables.
- A solution is generated from that model that satisfies those constraints.

In the ideal case, all model constraints can be satisfied. However, in the case of models generated from competing stakeholders, this may not be possible. Hence, our approach offers a range of operators which tries to satisfying all, many, or one of a set of constraints. The appropriate selection of operators depends on the business at hand. In the case of software quality assurance, for example, one might combine all softgoals within the framework with logic ANDs to model the strictest quality assurance scheme[5], or with logic ORs for the loosest. Such setting may be useful when users are

[4]See also the http://imaginethatinc.com web site

[5]Some frameworks may not yield any relevant conclusion when inferred under the strictest constraint. It

dealing with different level of system quality requirements. Users may also use a combination of the available operators to create framework that most resemble the actual system. Examples on how these operators can be configured to suit individual business needs are presented in the case studies section (chapters 5 to 8).

1.5 An Overview of Softgoal Modeling and Simulation

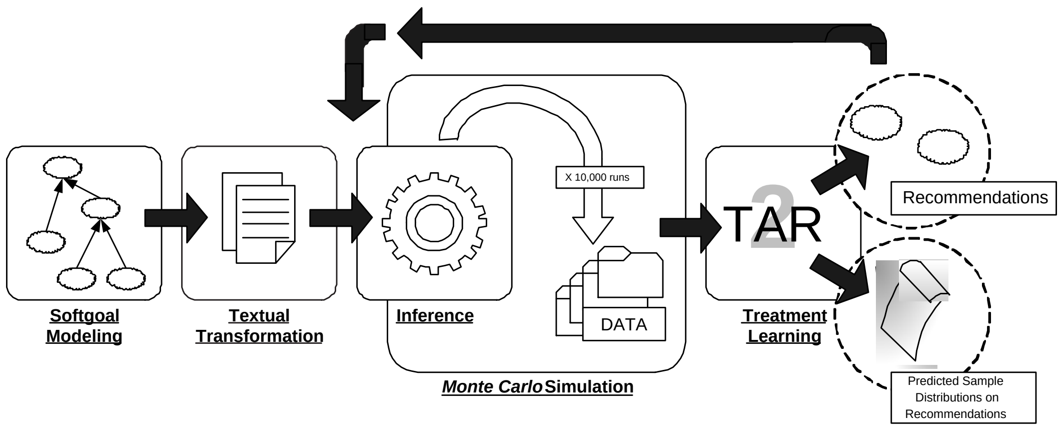


Figure 1.5: Softgoal Simulation Overview

This section presents an overview of the analysis technique used by the *Softgoal Simulation Tool*: as shown in figure 1.5, a Softgoal Framework model is first transformed into textual executable. Then, the model is fed to the inference engine, which is run many times to generate a wide variety of system behaviours (this is labelled "*Monte Carlo Simulation*" in the figure 1.5). Afterwards, TAR2 treatment learner is used to pinpoint consistent properties across the wide range of behaviours within the software framework, properties that can drive the system towards some preferred state. The treatment learning result is then fed to the inference engine as constraints, and the process continues until no further relevant result can be obtained by the treatment learner.

1.6 Chapter Plan

The rest of this thesis is organized as follows: Chapter 2 presents some related works cited in literature on system quality and other modeling techniques. Topics include: software process modeling, system dynamics modeling, influence diagram, *Petri-net*,

is because frameworks may compose of softgoals that benefit some while harm other softgoals, and this may prevent the inference engine to draw any conclusion if a majority of softgoals need to be satisficed.

and other models for early quality evaluation. Some common problems on general system modeling are also discussed. An investigation on common simulation parameters, an overview on software reliability engineering and qualitative reasoning, are presented at the end of the chapter. A new data mining technique named *treatment learning* is also introduced in this chapter.

Chapter 3 introduces the Softgoal framework modeling technique in thorough detail. First, using a hypothetical example of **Web Browser Selection Framework**, the graphical syntax of the original framework proposed by *Chung et. al.* is explained. Second, the textual syntax of the graphical framework transformation is described. Numerious parameter settings and framework configurations are also investigated. Third, a logical view of the Softgoal framework is created to explain the general concepts of a Softgoal framework inference. Lastly, various definitions on cost and benefit functions are explained.

Chapter 4 presents the implementation details of the *Softgoal Simulation Tool*; it begins with the introduction of the inference concept and some alternative implementation approaches, follows by a step-by-step walk-through to further explain the inference process. The cost and benefit scoring rules are presented, and some example calculations are followed afterwards. Then, the concepts of *Monte Carlo* simulation and *treatment learning* are introduced; their roles in the model simulation are also explained.

Chapter 5 to 8 are the case-study chapters: Chapter 4 applies the Softgoal simulation technique to the *Keyword in Context(KWIC)* framework, extracted from [28]. The framework concerns a particular phase in Software Engineering, namely *software architecture design*, and features different architectural options and their impacts on various system quality attributes. Simulations on two distinct framework settings are performed, and the treatment learning results reflect how TAR2 discovers ranges of consistent behavior among the space of inconsistent information within the *KWIC* framework.

Similar to Chapter 5, Chapter 6 deals with various organizational architecture styles, and their impacts on the overall quality of the target web-based application. Chapter 7 presents an application on a credit card information system, focuses on the performance requirements of the system during credit card authorization. Chapter 8 introduces a real-world business application on an advance satellite design project (SR-1); the objective is to find out a set of V&V (verification and validation) activities on a software project that would optimize the benefits while lower the costs. Variants of frameworks on these case studies are constructed to model different quality requirements and business concerns, and results show that the *Softgoal Simulation Tool* has succeeded in drawing stable conclusions on most of the framework settings.

Chapter 9 provides a summary of the Softgoal simulation results as well as future research ideas. The contributions of this thesis research is discussed, and an overall summary is presented to conclude the entire thesis.

Chapter 2

Related Work

The modeling and simulation technique of the *Softgoal Simulation Tool* can be classified as a type of *software process modeling and simulation*. This chapter presents an investigation on this discipline, as well as other related works associated with modeling and simulation methods. Further, some common problems of current modeling techniques are discussed.

Other related field of interest, such as Software Reliability Engineering, Qualitative Reasoning, and a new data mining method called *Treatment Learning*, are also presented.

2.1 Modeling for Quality

2.1.1 Software Process Modeling and Simulation:

Software process refers to a set of activities undertaken to manage, develop and maintain software systems. It focuses on the development process rather than on the product(s) output. *Software process modeling* is the construction of an abstract representation of the architecture, design or definition of the software process, plus optional automated support for modeling and for simulation.

There are different approaches of process modeling: process can be modeled at different levels of abstractions (generic versus tailored models) with different goals (descriptive verses prescriptive). Types of information extracted from process models are categorized into four perspectives as in [36]:

- *functional* - represents what process elements are being performed, and what flows of informational entities are relevant to these process elements;
- *behavioral* - represents when and under which conditions the process elements are implemented;
- *organizational* - represents where and by whom in the organization the process elements are implemented; and
- *informational* - represents the information entities output or manipulated by a process, including their structure and relationship.

The goal of prescriptive modeling is to define the required or recommended means to carry out the process. McChesney[83] has classified this group of models into *manual prescriptive models* and *automated prescriptive models*. *Manual prescriptive* *models* can be standards, methodologies and methods centered on management, development, evaluation and software life cycle, and organizational life cycle support processes. Examples of models belonging to this category include object-oriented methodologies and knowledge engineering methodologies. *Automated prescriptive models* are computerized specifications of software processes that perform activities related to assistance, support, management, and/or computer-assisted software production technique. The *Softgoal Simulation Tool* and other *process simulation models* belongs to this category, as they all involved a certain degree of computerization for automatic simulation.

Simulation modeling is a effective way to model systems that are too complex for static models or other techniques to usefully represent. Complexity encountered in real system can be (i)system uncertainty and stochasticity; (ii)dynamic behavior; and (iii)feedback mechanisms. The *Softgoal Simulation Tool* captures the uncertainty related to early life cycle system development by employing random variables and stochastic simulation technique. It does not address system dynamicity as the softgoal framework structure may become overly complicated. There is no feedback mechanism being explicitly modeled in a softgoal framework. The implications of a decision is being *feed forwarded* to other parts of the system represented by the framework.

A variety of multi-paradigm approaches have been proposed for modeling software processes, including:

- state-based process models;
- general discrete event simulation;
- system dynamics;
- rule-based languages;
- Petri-net models;
- queuing models;
- scheduling approaches[63];

Many of these approaches are already at a programming language level of abstraction. This may hurdle the modeling activity as it forces the engineer to consider many technical aspects of the modeling formalism. On account of this, the *Softgoal Simulation Tool* adopts both graphical and natural-language representation to ease modeling efforts.

Software process modeling is being used to address a variety of issues: from strategic management of software development[144], control and operational management[104][107], to supporting process improvements[26][113][120] and understanding[40][77][142], to software project management training[26]. Its scope of applications ranges from narrow portions of life cycle to long-term product evolution, involving a single project or multiple concurrent projects. Common elements addressed by the model include key activities and tasks, primary objects (e.g., code units, designs, problem reports), vital resources (e.g., staff, hardware), activity dependencies, object flows among activities, iteration loops, decision points, and other structural interdependencies[63].

Depending on the specific purpose of the model, many different variables can be devised as the results of process simulation. Typical results variables are: effort/cost, cycle-time (i.e., duration, schedule, time to market), defect level, staffing requirements over time, staff utilization rate, cost/benefit or other economic measures, throughput/productivity, and queue lengths (backlogs).

The input parameters to the model largely depend upon the result variables desired and the process elements it identifies. Examples of several input parameters include: amount of incoming work (in terms of *lines-of-code(LOC)* or function points), effort for design (as a function of size) and for code reworks, number of defects identified for correction, hiring rate, and resource constraints. Detailed discussion on these measures is presented in section 2.3 of this chapter.

2.1.2 System Dynamics

System dynamics is a methodology and computer simulation modeling technique for studying and managing complex feedback systems. Originally developed in the 1950s to help corporate managers improve their understand of industrial processes[45], system dynamics is now being applied extensively to problems such as public management and policy[46], biological & medical modeling, energy[98], and social sciences.

System dynamics models are formulated using continuous quantities interconnected in loops of information feedback and circular causality. The quantities are generally expressed as levels (stocks of accumulations), rates (flows), and information links representing the feedback loops. This modeling approach involves defining problems dynamically in terms of graphs over time. It strives for an endogenous, behavioral view of the dynamics of the system, so that the model can reproduce the dynamic problem by itself. Ultimately, understandings and applicable policy insights are derived from the resulting model and corresponding changes are implemented [117].

The simplest possible dynamic system is one which satisfies a first order, linear, differential equation. The generic form of the differential equation is as follows:

$$\mathbf{x}'(t) = \mathbf{f}(\mathbf{x}, \mathbf{p}) \tag{2.1}$$

where $\mathbf{x}$ is a vector of levels, $\mathbf{p}$ is a set of parameters and $\mathbf{f}$ is a nonlinear vector-valued function. State variables are represented by the levels.

Mathematical models have been used to represent the dynamicity of engineering systems. Examples include: *translational motion* (spring), *rotational motion* (torsional spring), electrical elements (resistor, capacitor, inductor), and various electrical and mechanical systems. In recent years, this technique is increasing being applied to model software project dynamics. When used for software engineering, system dynamics provides an integrative model that supports analysis of the interaction between activities such as code development, project management, testing, hiring, and training. For example: Abdel-Hamid has developed a generic system dynamics model of software development[4]. Others have modified it to support project and process management in particular organizations[78][124]. Madachy[79] used system dynamics to model the effect of performing formal inspections, particularly the effects on cost, schedule, and quality throughout the life cycle. It uses system dynamics to model the interrelated flows of tasks, errors and personnel throughout different development phases and is calibrated to industrial data. The *Softgoal Simulation Tool* discussed here addresses the software quality problem in a different aspect: it focuses on the design decision making process during the early development stage, when the engineer is still trying to understand the problem and has vague ideas about how to constrain its solution. System dynamics, on the other hand, analyses the behavior of some solidly-defined processes, and is often applied when ideas have crystallized into solutions specified in terms of functionalities and measurable factors.

Software products such as *Stella*[56], *Powersim*[108], *Vensim*[133], and *Dynasys*[58], are available at the market for quantitative system dynamics modeling. All these software products are graphically oriented: the models are developed directly as stock-flow-diagrams on the screen; some commercial products (IVensim PLE) are even free for educational purposes. Alternatively, standard spreadsheet software (e.g., *Excel*) can be used for very basic system dynamics evaluation.

2.1.3 Cost Model - COCOMO II

COCOMO II is a model that allows one to estimate the cost, effort, and schedule when planning a new software development activity. It consists of three submodels, each one offering increased fidelity the further along one is in the project planning and designing process. Listed in increasing fidelity, these submodels are called the Application Composition, Early Design, and Post-architecture models. Until recently, only the Post-architecture model had been implemented in a calibrated software tool. The tool provides a range on its cost, effort, and schedule estimates, from best case to most likely to worst case outcomes. It also allows a planner to easily perform “what if” scenario exploration, by quickly demonstrating the effect of adjusting requirements, resources, and staffing might have on predicted costs and schedules.

The original COCOMO model was first published by Dr Barry Boehm in 1981. The COCOMO project aims at developing an open-source, public-domain software effort estimation model. The project has collected information on 161 projects from commercial, aerospace, government, and non-profit organizations[27]. As of 1998, the projects represented in the database were of size 20 to 2000 KSLOC (thousands of lines of code) and took between 100 to 10000 person months to build.

COCOMO measures effort in calendar months where one month is 152 hours (and includes development and management hours). The core intuition behind COCOMO-based estimation is that as systems grow in size, the effort required to create them grows exponentially, i.e., $effort \propto KSLOC^x$. More precisely:

$$months = a * \left(KSLOC^{\left(1.01+\sum_{i=1}^{5} SF_i\right)}\right) * \left(\prod_{j=1}^{17} EM_j\right)$$

where a is a domain-specific parameter, and KSLOC is estimated directly or computed from a function point analysis. SF_i are the scale factors (e.g., factors such as “have we built this kind of system before?”) and EM_j are the cost drivers (e.g., required level of reliability).

Software effort-estimation models like COCOMO-II should be tuned to their local domain. Off-the-shelf “untuned” models have been up to 600% inaccurate in their estimates, e.g., [93, p165] and [66]. However, tuned models can be far more accurate. For example, [27] reports a study with a bayesian tuning algorithm using the COCOMO project database. After bayesian tuning, a cross-validation study showed that COCOMO-II model produced estimates that are within 30% of the actuals, 69% of the time. However, the historical data needed for tuning the model is not always available. Thus, the *Softgoal Simulation Tool* aims to explore the local model as much as we can, with the use the combination of *Monte Carlo* simulation and a data classification method named *treatment learning*.

2.1.4 Influence Diagram

Influence diagram [121] (a form of Bayes nets) has been used to sketch out subjective knowledge, then assess and tune each knowledge variable based on available data. The basic structure of an influence diagram is introduced in section 23. Since the early 1980’s, influence diagrams have been used in a wide variety of different applications, such as medical diagnosis, clinical decisions, and optimization of oil spill response. In the field of software development, this modeling scheme is used by *Burgess et al* [21] in evaluating requirements that are candidates to be included in the next release of some software. Influence diagram offers a global view of the probabilistic structure of the decision problem, and the interdependence between decisions and uncertain events. Thus, it helps decision makers to visualize how management decisions and uncertain events relate to each other and their combined effect on the final outcome of decision.

Some advantages and disadvantages of Influence diagram modeling is listed below:

+ Influence diagram offers a framework in which decision makers and analysts can discuss the problem without invoking any formal mathematical, probabilistic, or statistical notation;

+ Complex problem can be represented in a concise format, instead of pages of narrative explanation;

+ An Influence diagram can provide a degree of sensitivity analysis to show how much effect particular decisions or uncertain events have upon the final outcomes;

- Although it is a very simple model to understand, an influence diagram is not easy to construct; it takes time to identify the uncertainty events and their relationships with decisions, and their effects on the utility value. Setting up the probability tables for simulation also requires time.

- The size of the influence diagram is restricted by the computation power of the software that drives the simulation. The larger the framework and the more complicated the interdependency network, the more the processing power required.

- To improve the precision of the model, the organization has to fine-tune the probabilities with information such as expert advice, past experience and statistical data. This information is hard to obtain, costly, and is not always available.

Experiment on implementing an Influence diagram for softgoal modeling has been performed. Despite the advantages listed above, the Influence diagram approach was found to be inappropriate for the *Softgoal Simulation Tool* due to performance issues and other concerns, which are discussed in Chapter 23.

Many software tools exist that implement the influence diagrams directly and perform the evaluations. Examples of these tools include *DATA*[132], *Decide*[127], *DPL*[8], *ExpressionTreeExpressionTree*. Non-commercial packages, such as *GeNIe*[22] and *SMILE*[41], are also available for educational uses.

2.1.5 Petri Nets and Fuzzy Petri Nets

Petri Nets are a graphical and mathematical modeling tool for describing and studying information processing systems. As a graphical tool, Petri nets can be used as a visual-communication aid similar to flow charts, block diagrams, and networks. *Tokens* are used in these nets to simulate the dynamic and concurrent activities of systems. As a mathematical tool, it is possible to set up state equations, algebraic equations, and other mathematical models governing the behavior of systems. Thus, they can serve as a communication medium between practitioners and theoreticians.

Petri nets have been proposed for a wide variety of applications. Two successful application areas are performance evaluation and communication protocols. Others include modeling and analysis of distributed systems, concurrent and parallel programs, and flexible manufacturing/industrial control systems. In the context of software engineering, Bandinelli[11] proposed a software process modeling environment named *SPADE*, in which its modeling language is based on high-level Petri nets. Kusumoto[74] proposed a software projects development modeling and estimation (on factors such as cost, quality, and delivery date) method, based on a variant of Petri nets called *GSPN(Generalized Stochastic Petri-net)*[1]. This method has been applied to real software projects and the estimated value are compared with the actual data.

Fuzzy Petri Nets, a formalism that combines fuzzy set theory and Petri Net theory, is another technique for representing uncertain knowledge about a system state. It has been applied to robotic systems [24], programmable logic controller [9], damage assessments[76] and other expert systems. This modeling method may be incorporated with the inference engine of the *Softgoal Simulation Tool*. However, within a Petri Net, the way membership functions attached to each tokens and certainty factors associated with transitions can be hard to understand. For business users that are involved in constructing the requirement model, a simple modeling convention is much preferred to a sophisticated one. Moreover, Petri-net-based models tend to become too large for analysis even for a modest size system[94]. Thus, using Petri Net for the implementation may not be very practical, and hence not being implemented.

[1]A *stochastic Petri net(SPN)* is a Petri net where each transition is associated with an exponentially distributed random variable that expresses the delay from the enabling to the firing of the transition. In a case where several transitions are simultaneously enabled, the transition that has the shortest delay will fire first. *Generalized stochastic Petri nets(GSPN)* is an extension of SPN that has two types of transition (timed and immediate). A timed transition has an exponentially distributed firing rate, and an immediate transition has no firing delay. [94]

2.1.6 Other Models for Early Quality Prediction

Acuna & Juristo [5] has developed a *Capabilities-Oriented* Software Process Model that captures the *competencies* of the people who perform the development activities. Based on this model, they proposed methods that (i) determine the capabilities of the members of a development team; and (ii) assign people to perform roles depending on their capabilities and the capabilities demanded by the roles. This human competencies model has been empirically validated, and experiments performed to confirm that such technique improves software development.

Modeling tools have been developed for the estimation of software reliability[2]. *Software Reliability Growth Model(SRGM)* is a modeling technique which capture the process of fault detection and removal during the testing phase in order to forecast failure occurrences (and hence software reliability) during the operational phase. There exist tools such as ROBUST[39] that use SRGMs to provide early estimation of the testing effort required based on static metrics obtained during the development phase. Ramani et. al. proposed a high-level design of a *Software Reliability Estimation and Prediction Tool*(SREPT), which supports reliability prediction throughout the software life cycle, right from the architectural phase to the operational phase. However, the development progress of the actual tool remains unknown. More on software reliability is discussed in later section (section 2.4).

In [118], Runeson & Wohlin proposed a *state hierarchy usage model* to capture the complex and dynamic usage behavior for large software systems. This model can be transformed into a discrete-time *Markov chain*[3], and hence Markov Decision Theory can be applied on the model for analysis purpose. Wesslen & Wohlin [143][146] applied this model as the basis of usage case generation, which serves as the input to the dynamic analysis. Their method supports quality and reliability estimation early at the phase of specification and design document in the software life cycle (i.e., before the implementation phase).

Folmer and Bosch [43] proposed a *Usability Framework* which models the relationship between usability and software architecture. For example: the *wizard* architecture pattern is linked to the “guidance” usability property, which in turn is linked to the “learnability” usability attribute. This framework look very similar to the *Softgoal Framework*, with usabilities correspond to NFR Softgoals and architecture elements as Operationalizing Softgoals. However, such framework is only used as a guideline to the development process. There is no simulator or computerized inference method to find out the best combinations of architecture elements.

2.2 Common Problems

Software modeling technique has not been used to their full advantage. Typical reasons sited can be generalized as follow:

[2]Software reliability is defined as the probability of failure-free software operation for a specified period of time in a specified environment[10].

[3]*Markov chain* is another class of decision problems that incorporate the idea of *influence diagram* but with some constraints lifted. This is based on the assumption that a decision may be conditionally independent of certain pieces of available information.

- Models tend to be difficult to build and maintain;
- The metric data for the models can be difficult to acquire (or may not exist);

As a result, simulation modeling may be perceived as costly to build and difficult to obtain timely and accurate results from them. This section presents an investigation on common problems and some possible resolutions on simulation modeling.

2.2.1 Cost

In general, the higher the level of detail and fidelity to the original (development projects, process etc), the greater amount of effort and expertise required to build the simulation model. Highly-skilled model developers with knowledge associated with specific simulation tool or language are essential for designing models of complex systems. Novice personnel requires more time to build the model, collect data and analyze complex simulations. It is also necessary to provide training to managers who will utilize model results to make decision. Together, these cost factors prevent the simulation modeling technique to be widely adopted by the industries.

Obtaining quantitative and qualitative data for simulation models is another significant cost associated with model development. Companies may not have collected all the data that are needed for the model or in forms that are readily usable. Sometimes it is necessary to interview staff or extract data from written documents. Even if the data exists, organization issues concerning company politics, security and confidentiality can confound data collection efforts for the model.

One possible approach to lower the cost of model development is by designing a general purpose simulation model. Raffo et. al. [114] defined the concept of a *generalized software process simulation model (GPSM)* that can be tailored quickly to a particular project scenario and deployed rapidly to be used. GPSM incorporates the idea of modularization for software reuse and develop software product families, and aims at lowering modeling cost by (i) reducing model complexity; (ii) making the mechanics of building models easier; (iii) making it easier to acquire model data, and (iv) reducing the amount of effort and expertise needed.

Nonetheless, as stated by the authors, that great care is required when adapting the GPSM to a specific context. Many of the assumptions and empirical results used in one context may not be easily transferable to another context. For example, Kemerer[66] used a single set of projects from an anonymous organization to check the accuracy and consistency of four popular cost models (*SLIM*, *COCOMO*, *Function Points*, and *ESTIMAGS*). The results indicate that there is a wide variation in the results from the uncalibrated models. Calibrated models produce better results, but an organization must invest time to build a database sufficient for model calibration. Hence, the problem associated with data has still remained unresolved.

Therefore, the goal of the *Softgoal Simulation Tool* is to minimize the time and expertise required for modeling and simulation. By adopting a simple graphical framework syntax and natural-language textual input, users are expected to build a model customized to their target system quickly and easily. To further reduces the cost, the tool requires only subjective knowledge as the input of the model, and eliminates the use of databases and solid statistics. Chapters 3 details on the input parameters to the softgoal model and how they are interpreted by the tool.

2.2.2 Model Validation

Simulation model often undergoes calibration and validation procedures to ensure its accuracy. Validation can be performed by inspections and walkthroughs of the model. Actual data should be used to validate the model empirically and to calibrate it against real-world results. Ahmed et. al [6] proposed a *Software Process Model Evaluation Framework*, which consists of five quality aspects for evaluation: *syntactic quality*, *semantic quality*, *pragmatic quality*, *test quality*, and *maintainability*, each has a checklist of fulfilling criteria. To evaluate a model according to this framework, however, required domain and modeling expertise, manual checking, and/or user surveys, which can be a time-consuming and costly process. But the main obstacle to model validation is the lack of relevant, desired data in practical setting. Even if data is available, it is too often that these variables have not been measured or not carefully measured [63].

To cope with this situation, Kellner and Raffo [64] suggest the following strategies:

- Adjust available values to approximate desired variables;
- Look for alternative source of records (e.g., not only summary reports but also the original inspection forms) and construct values from them;
- Utilize typical values taken from the literature;

Similar to other modeling methods, the softgoal modeling and simulation technique requires validation by domain experts to ensure its accuracy. Nonetheless, much less statistical data is required as the tool deals only with subjective, non-quantified knowledge.

2.2.3 Problems with Quantification

When modeling "hard" variables such as production, cash flow and so forth, there are few difficulties in quantification. However, it is often necessary to introduce "soft" variables, such as Customer Satisfaction and New Order Inflow Rate, to the model for simulation. For quantitative deterministic models, these uncertainties can significantly mislead the output results. One may argue that it is the general patterns of behavior, rather than numerical precision, that is being concerned about for system model simulation. However, when a number of uncertainties combine and multiply in the model equations, the outputs can be so inaccurate that the policy inference drawn from them may be illusive.

One remedy of quantification problems is to adopt a stochastic approach: rather than using (deterministic) point estimates, stochastic simulation employs random numbers drawn from a specified probability distribution. Many simulation runs are required because the random numbers, and hence the result variables, differ from run to run. The result variables are analyzed statistically across a batch of simulation runs; this is termed *Monte Carlo* simulation, and is also the approach taken by the *Softgoal Simulation Tool*.

Another suggestion is to consider adopting a qualitative modeling approach instead of common quantitative analysis. Coyle advised in his paper on system dynamics, that “qualitative modeling can be useful in its own right, and that quantification may be unwise if it is pushed beyond reasonable limits”[34]. Coyle further elaborated his argument with reviews of some qualitative models and showed that they are effective in leading to policy insights. In [73], Kuipers formalized the qualitative structure and behavior descriptions as abstractions of differential equations and continuously differentiable functions. The proposed QSIM algorithm has been applied to various works, including Modeling and Simulation[47][35][4]. In recent papers, Ramil and Smith[115][116] applied such techniques to modeling long-term software evolution. Qualitative simulation has also been applied on software development projects[130]. See 2.5 for more details on qualitative modeling.

Lastly, one may calibrate these uncertain variables from data obtained from past experience and/or expertise. Again, this may be costly and may not be applicable to all situations.

The *Softgoal Simulation Tool* does not encounter with these problems as it only demands for subjective knowledge in a qualitative manner.

2.3 Input Parameters for Quality Evaluation

A list of parameters popular for quality evaluation is presented as follows:

1. amount of incoming work (measured in terms of software size (LOC) or function points);
2. effort for design as a function size;
3. defect detection efficiency during testing or inspections;
4. effort for code rework as a function of size and number of defects identified for correction;
5. defect removal and injection rates during code rework;
6. decision point outcomes; number of rework cycles;
7. hiring rate; staff turnover rate;
8. personnel capability and motivation, over time;
9. amount and effect of training provided;
10. resource constraints; and
11. frequency of product version releases;

[4]Both concerned with physical systems.

These quantitative values are either treated as constants or inter-dependent parameters of some functions. Apart from justifying the functions and evaluation methods, many of the metrics listed above (such as the source code size and the number of bugs reported) can only be obtained at the late phase of the software life cycle. Thus, quality evaluation relying on these metrics is often performed after product has been finalized. Evaluation performed during early development life cycle, on the other hand, rely mostly on past experiences, expert opinions, and statistical data analysis. Experiences and advices are subjective knowledge, while statistical data is not always available. The *Softgoal Simulation Tool* is developed targeting this situation, and does not rely on any of the above quantitative parameters.

2.4 Software Reliability Engineering

Software reliability engineering is the quantitative study of the operational behavior of software-based systems with respects to user requirements. Its goal is to predict, measure, and manage the reliability of software-based systems to maximize customer satisfaction[96]. The study of reliability engineering includes (1) Software reliability measurement (assessment) and estimation (prediction); (2) Effects of product and development process metrics and factors (activities) on operational software behavior; (3) Application of this knowledge in specifying and guiding software development, testing, acquisition, use, and maintenance[134].

Reliability is the probability that a system, or a system component, will deliver its intended functionality and quality for a specified period of "time" and under specified conditions. As software reliability will depend on how software is used, software usage information is an important part of reliability evaluation. This includes information on the environment in which software is used, as well as the information on the actual frequency of usage of different functions (or operations, or features) that the system offers. The usage information is quantified through *operational profiles*, which is a set of relative frequencies (or probabilities) of occurrence of disjoint software operations during its operational use. Operational profiles are used to select test cases and direct development, testing and maintenance efforts towards the most frequently used or most risky components. A detail discussion of operational profile issues can be found in [95].

A metric that is commonly used to describe software reliability is *failure intensity*[5], defined as the number of failures experienced per unit "time" period. Two typical models are the "basic execution time" (BET) model [52][97] and the Logarithmic-Poisson execution time (LPET) model [97]. Both models assume that the testing uses operational profiles, and that every detected failure is immediately and perfectly repaired.

Another important measure is *availability*, defined as the probability that a system, or a system component, will be available to start a mission at a specified "time"[48]. Reliability and availability models are used to predict future *unavailability*[6] of a sys-

[5]When a system in operation does not deliver its intended functionality and quality, it is said to *fail*. A *failure* is an observed departure of the external result of software operation from software requirements or user expectations [2, 3, 1]. A *fault*, on the other hand, is a defective, missing, or extra instruction, or a set of relation instructions, that is the cause of one or more actual or potential failures.

[6]Unavailability is the opposite of availability.

tem. The prediction would differ from the true value depending on how well the model describes the system.

To practice software reliability engineering, a sound and solid software *verification and validation* plan, process and activities throughout software life cycle are essential. Examples of activities include software requirements traceability analysis and evaluation, system and acceptance test plan generations, design & code traceability analysis, test cases design, generation and execution. The use of inspections and reviews is encouraged, and evaluation of "inherited"[7] code (using operational profiles appropriate to the application) is also recommended.

A crucial activity of reliability engineering is defining *operational profiles* and associated test cases. Construction of an operational profile is preceded by definition of a *customer* profile, a *user* profile, a system *mode* profile, and a *functional* profile. Information for creating a *customer* profile may be obtained from marketing data of related systems, modified by marketing estimates that take into account the new system's appeal. *User* profile can be derived from the customer profile by refinement: looking at each customer group and determining what user groups exists. Note that a system's users are not necessarily identical to its customers. A customer is the person, group, or institution that is acquiring the system. A user is a person, group, or institution that employs, not acquires, the system.

A system mode is a set of functions or operations that are grouped for convenience in analyzing execution behavior. A system can have several mode and can switch between them. A system *mode* profile is the set of system modes and their associated occurrence probabilities. Modes can be defined on the bases of user groups, environment (e.g., overload vs. normal, high-noise vs. low-noise), criticality, user experience, platform, how user employs system operations to accomplish system functions.

To obtain *functional* profile, it is necessary to break each system mode into user-oriented functions needed for its operations and associated probabilities (e.g., features plus the environment). A function may represent one or more tasks, operations, parameters, or environmental variables (e.g., platform, operating system). It should be noted that the functions will tend to evolve during system development and the profiling process is an iterative one.

The final steps are the definition of an operational profile through explicit listing of operations and generation of test cases. The operations are the ones that are tested. Their profile will determine verification and validation resources, test cases and the order of their execution. The operations are to be associated with actual software commands and *input states*, which are then sampled in accordance with the associated probabilities to generate test cases.

The need to recognize software problems early, so that appropriate corrections can be taken is obvious. However, the construction of an operational profile, which is crucial for all reliability modeling methods, often encounters difficulties during the early phase of software development. Data and knowledge needed for system *mode* and *functional* profiles, such as the environment, user experience, and criticality, may not be available or confirmed. Several authors have published models that attempt to relate

[7]Inherited code includes legacy code, or "acquired", "reused" code.

some early software metrics, such as the size of the code and cyclomatic numbers[8], to the reliability model of a system [67, 16]. But even these metrics are not present during very early development phases such as the requirement specification and design phases. The software quality modeling technique proposed in this thesis eliminates the use of such metrics, thus makes it possible to perform estimation during the very early life cycle of software development.

2.5 Qualitative Reasoning

The construction of a quantitative simulation model has required the specification of mathematical relationships between the attributes or variables involved in the model. If such relationships are not known precisely, the model builder has to make assumptions. The issue is critical when empirical data is scarce and knowledge about the relationship is imprecise. Qualitative reasoning techniques, on the other hand, allow modeling at a higher level where such assumptions are not necessary. The Softgoal framework (by Chung et. al[28]) utilizes qualitative approach to model the inter-relationships between various quality factors. This section presents a brief review on qualitative reasoning.

There are generally three approaches to build qualitative engineering models. These approaches have in common the fact that they deal with abstractions of real numbers into positive negative, and zero valued quantities rather than dealing with numbers themselves. The first approach is that of Kuipers [73] who takes a set of differential equations, abstracts them to consider only their qualitative influence, and then uses them as a set of constraints on the possible values of the state variables. This approach has been implemented as the QSIM software system [73]. The second approach, by de Kleer et. al. [37], is to build a collection of components, each of which has a well-defined qualitative behavior described by qualitative differential equations, and connect these components together to build a qualitative model. The *ENVISION* software system is implemented based on this work. The third approach is Forbus' *Qualitative Process Theory (QPT)* [44], which allows sets of objects to have group behaviors over and above their individual ones, thus providing a far richer modeling language[101]. Chung et. al.'s qualitative modeling approach can be classified as a variant of Kuipers'. For the implementation of the *Softgoal Simulation Tool*, the qualitative symbols are quantified with "bounded" random numbers in order to make the reasoning less abstract while modeling the impreciseness of qualitative value.

Qualitative model simulation is a powerful technique for creating and using models in the presence of incomplete information. The abstraction of the model to consider only its qualitatively distinct behaviors is appropriate considering the sparseness of the empirical data. Nevertheless, the majority of software process simulation models adopt a quantitative approach because they often address finer-grained planning and managerial issues[116].

[8]The McCabe cyclomatic complexity has been proposed in the literature as an important determinant of defects [131], [135] It is a measure of a subprogram's internal complexity from the perspective of logic control flows. The cyclomatic complexity measure generally becomes available later in the development process when a subprogram body is implemented in terms of its control flow logic.

```
lstat <= 11.66
|   rm <= 6.54
|   |   lstat <= 7.56 THEN medhigh
|   |   lstat > 7.56
|   |   |   dis <= 3.9454
|   |   |   |   ptratio <= 17.6 THEN medhigh
|   |   |   |   ptratio > 17.6
|   |   |   |   |   age <= 67.6 THEN medhigh
|   |   |   |   |   age > 67.6 THEN medlow
|   |   |   dis > 3.9454 THEN medlow
|   rm > 6.54
|   |   rm <= 7.061
|   |   |   lstat <= 5.39 THEN high
|   |   |   lstat > 5.39
|   |   |   |   nox <= 0.435 THEN medhigh
|   |   |   |   nox > 0.435
|   |   |   |   |   ptratio <= 18.4 THEN high
|   |   |   |   |   ptratio > 18.4 THEN medhigh
|   |   rm > 7.061 THEN high
lstat > 11.66
|   lstat <= 16.21
|   |   b <= 378.95
|   |   |   lstat <= 14.27 THEN medlow
|   |   |   lstat > 14.27 THEN low
|   |   b > 378.95 THEN medlow
|   lstat > 16.21
|   |   nox <= 0.585
|   |   |   ptratio <= 20.9
|   |   |   |   b <= 392.92 THEN low
|   |   |   |   b > 392.92 THEN medlow
|   |   |   ptratio > 20.9 THEN low
|   |   nox > 0.585 THEN low
```

Attributes used in the decision tree:

age =	proportion of houses built prior to 1940
b =	information on racial mixture in the suburb
dis =	weighted distances to five employment centers
$lstat$ =	living standard
nox =	nitric oxides concentration
$ptratio$ =	parent-teacher ratio at local schools
rm =	number of rooms

Figure 2.1: A decision tree learned from the HOUSING database at the UC Irvine Machine learning repository `http://www.ics.uci.edu/~cmerz/mldb.tar.Z`. Generated using WEKA's J4.8 algorithm [145] with the command line `J4.8 -C 0.25 -M 10`.

2.6 Data Mining and Treatment Learning

Data mining is a summarization technique that reduces large sets of examples to a small understandable pattern using a range of techniques taken from statistics and artificial intelligence [145, 53, 14]. As an example, figure 2.1 shows a decision tree learned by a typical decision tree data miner. This tree is generated from hundreds of examples of houses in the Boston area. Each branch of the tree tell us how we might recognize $high$, $mediumHigh$, $mediumLow$ and low quality houses.

Traditional data mining tools try to build the most complex or the most expressive models. A standard data miner hunts for detailed summaries of a set of examples. Such data miners can generate extensive and lengthy *descriptions* of things, and extracting that information for modeling can be a waste of time. The *Treatment Learning* technique takes a different approach: unlike traditional data mining methods (such as decision tree induction[110][111] and simulated annealing[68]) which give extensive descriptions on datasets, treatment learning seeks minimal differences between sets[88]. These differences can be seen as the control variables that can lead to better system behaviors. Not only does treatment learning reduce the option space of the target domain, but it also aids decision making by finding the key issues that have the greatest impact towards system behavior. For example, standard data mining method may generate decision trees with thousand of nodes, while treatment learner will generate useful models that are less than a dozen lines long. To further elaborate the merits of treatment learning over other data mining tools, a comparison between the Boston housing decision tree (figure 2.1) and the treatment learner TAR2 result is presented in the following.

To compare 2.1 to TAR2, we first convert the tree to the rule format generated by TAR2. There are many sophisticated methods for translating trees to rules [145]. However, in this case, the simple method of reading each branch as a separate rule is as good as any other method. For example, the first three lines of 2.1 can be collapsed to a rule with two tests:

$$ruleJ48a \begin{cases} IF: & lstat \leq 7.56\ AND\ rm \leq 6.54 \\ THEN: & medhigh \end{cases}$$

A policy for house hunting might be found by combining some of the *decision values* at all the branch points in figure 2.1; i.e., the values that select certain branches and reject others. In all, there are seven attributes in figure 2.1 with fifteen decision values including $lstat = 11.66$ and $rm = 6.54$.

The same data, summarized by TAR2, generates a much smaller model with but only two decision values ($rm = 6.6$, $ptratio = 15.9$):

$$ruleTAR2a \begin{cases} IF: & rm \geq 6.6\ AND\ ptratio \leq 15.9 \\ THEN: & \text{97\% of the found houses} \\ & \text{will be high quality} \end{cases}$$

As shown in the above example, TAR2 produces much simpler output than that of the decision tree in figure 2.1. For humans making decisions, overly elaborated models can complicate the situation. Researches on cognitive behavior of human decision making noted that human often use simple *satisficing* models rather than intricate ones[50]. For these characteristics, treatment learner is believed to be a better option for analyzing data generated from the softgoal simulation process.

For the *Softgoal Simulation Tool*, TAR2 treatment learner[57][9] has been used to perform analysis on the datasets generated by the *Monte Carlo* process. Vast amount of publications regarding to TAR2 and its applications are available at [84], [90], [85], [87], [89], and [86].

[9]The tool can be downloaded at `www.ece.ubc.ca/twiki/view/Softeng/TreatmentLearner`.

- **Validation**
  e.g., check that data models generated from the model matches known domain values
- **Screening**
  e.g., dimensionality reduction via, say, ignoring input variables that are not highly correlated to outputs
- **Sensitivity Analysis**
  e.g., execute using the minimum and maximum of all input values
- **True Uncertainty Analysis**
  e.g., treat each input as a random variable with a mean and standard deviation, then perform *Monte Carlo* simulations
- **Optimization**
  e.g., find some combination of inputs that improve the output values

Figure 2.2: Sensitivity analysis methods

2.7 Sensitivity Analysis

A standard method for understanding numeric models is some sort of *sensitivity analysis*. Sensitivity analysis is a huge field that includes many techniques (some of which have overlapping definitions). One survey[69] argues that what is called “sensitivity analysis” divides up into tasks shown in figure 2.7; i.e., validation, screening, true sensitivity analysis, uncertainty analysis, and, finally, finding generating some optimization policy. Validation and screening usually proceeds the other tasks but it is seems to be a matter of personal style whether or not a sensitivity analysis follows uncertainty analysis.

Traditional sensitivity analysis is a labor-intensive, time-consuming task. Researchers in this area can spend their whole career working in just one of the areas shown in figure 2.7. Often, extensive knowledge is required about the internal features of the model being simulated. Unless project managers have access to specialists in sensitivity analysis, then they may not be able to understand all the implications of the output of the software process models that they execute.

A unique feature of the modeling technique in this thesis is the use of data mining to reduce the effort and skill-level required for sensitivity analysis. Software project managers would merely need to tag some of the model output variables with “utilities”, i.e., their evaluation of how important certain variables are to them. Data miners would then perform many random simulations, cache the results, then automatically learn what input parameters lead to preferred outputs.

This approach assists with many of the tasks shown in figure 2.7. Validation is still an issue, but treatment learning tool can automate the dimensionality reduction of screening. *Monte Carlo* simulation is applied to sample input at random. And the recommendation generation by TAR2 is similar to optimizing. This approach has been investigated by Feather and Menzies [42]: their research found that TAR2's heuristically generated alert points yielded solutions very close to the solutions generated by more complete data mining methods such as simulated annealing and genetic algorithms.

2.8 Chapter summary

This chapter reviews current modeling methods that attempt to improve software quality, as well as some related technologies on this thesis research. Each methodology has its unique merits and specific field of application, and it is up to the managers to decide which one is right for the situation. However, using these models to develop early software quality estimations has been difficult: some are complex and require modeling experts, and most require project attributes that are not well known or understood during the early development life cycle. Some models try to prorate the effort estimate to the earlier phase base on experience or a theoretical basis, yet both of these inputs depend on the use of expertise and statistical data. These resources are often costly and may not be always available for modeling use. Moreover, these modeling techniques assist in decision making by describing the effect on the system under the current implementation, and do not explicitly gives suggestions that will lead to a better system. This thesis proposes the *Softgoal Simulation Tool*, a novel modeling and simulation tool that targets the problem of over-dependence on data and expertise, provides a simple environment for software managers to estimate software quality during the early development stage of the system, and thus ponder on different decisions base on the estimate. This tool features:

- a simple syntax that allows users to sketch out the inter-dependencies between software quality attributes and design decisions;
- the use of subjective knowledge, thus no statistical data is needed;
- automatic simulation without the need of human intervention;
- provides suggestions that will drive the system to a certain (better, or worse) direction;

By eliminating obstacles such as data-dependence and complicated modeling syntax, project managers are able to perform quality evaluation during the early life cycle of software development, and focus on making the most appropriate decisions that lead to quality software.

Chapter 3

Softgoal Modeling and Simulation

The next two chapters present detailed explanations on a novel approach of system modeling and simulation, in which the *Softgoal Modeling* technique is combined with *Monte Carlo* simulation and treatment learning. This chapter provides an overview and discusses the preliminary steps for preparing Softgoal Framework for simulation analysis. First, the Softgoal Modeling approach is introduced. Second, the textual syntax of transforming a graphical Framework is described. Numerous parameter settings and framework configurations are also explained. Third, a logic view of a Softgoal Framework is presented to explain the general concepts of framework inference. Lastly, definitions of various cost and benefit functions are presented.

3.1 Overview of Softgoal Framework

Softgoal framework consists of three types of softgoals: the *Non-Functional-Requirement* (*NFR*) *softgoals*, the *operationalizing softgoals*, and the *claim softgoals*. *NFR softgoals* represent quality requirements such as “time-performance”. *Operationalizing softgoals* comprise of possible solutions or design alternatives to achieve the *NFR softgoals* (e.g., “incorporate javascript in online storefront”). *Claim softgoals* argue the rationale and explain the context for a softgoal or interdependency link (e.g., a claim may argue that “client-side scripting loads faster”). As there are no clear cut criteria for success, *NFR softgoals* may not be absolutely achieved, yet they can be sufficiently satisficed[1][123].

NFR softgoals can have an associated *priority*. *Priority softgoals* are shown in a softgoal framework with exclamation marks (`!`, `!!`), or `critical/veryCritical` textually. *Priority* specifies how important a softgoal is to be fulfilled for the success of the system.

Contribution represents the interdependency between softgoals, as well as the influence a (*claim*) softgoal has on an interdependency link[2]. Listed in an increasingly positive magnitude, these contributions include BREAK (`--`), HURT (`-`), UNKNOWN (`?`), HELP (`+`) and MAKE (`++`). *Contributions* can also be combined among multiple softgoals and/or interdependency links through logic operations such as AND and OR.

[1] Coined by *H.A. Simon* (United State social scientist and economist), “satisficed” is defined as to be satisfied with a minimum or merely satisfactory level of performance, profitability etc., rather than a maximum or optimum level. In the context of the softgoal framework, a softgoal is said to be satisficed when it is achieved not absolutely but within acceptable limits.

[2] In graphical terms, they are the labels of the arrows between softgoals.

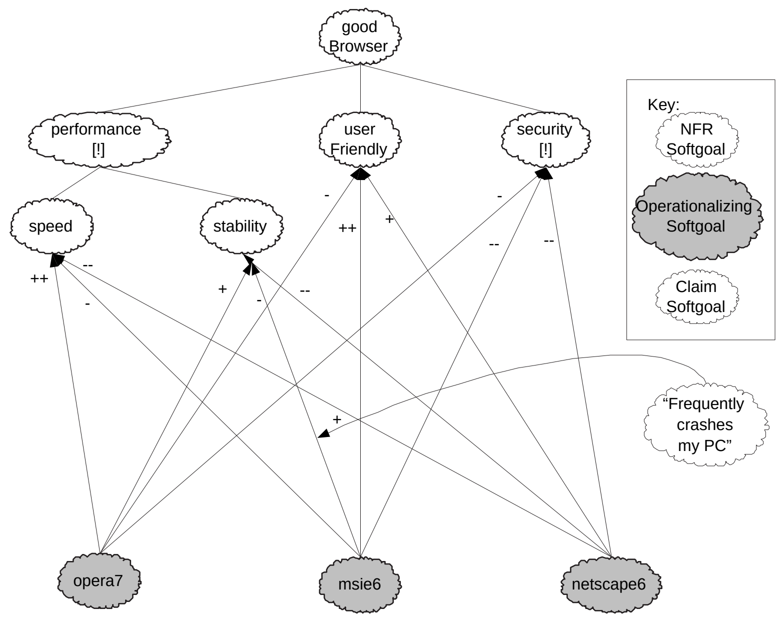


Figure 3.1: Web Browser Selection Softgoal Framework

A hypothetical framework on **Web Browser Selection** is set up to illustrate how the softgoal framework models design options and quality attributes. This framework will be used to explain the inference process and the implementation details of the *Softgoal Simulation Tool* throughout this chapter.

3.1.1 Web Browser Selection - A Sample Application of Softgoal Framework

The web is a part of the internet that offers textual, visual, and multimedia contents together in a presentation-type format. A web browsers is an application that renders web pages viewable to users. Web browsers such as Netscape[19], Internet Explorer[136], Opera[20][3], Neoplanet[18], and Mozilla[17], are freely available for PC users. For most people, one browser is sufficient for simple web-surfing purpose. As each browser has its unique features and downfalls, the concept of *Softgoal Framework* can be used for choosing the default browser among all the available options. Thus, the *Web Browser Selection* Softgoal Framework is created as an illustrative example to demonstrate how the softgoal notation is applied to define a tradeoff problem.

The framework shown in figure 3.1 describes the tradeoffs among web browsers and the desired browser quality attributes. To simplify this example, a small fraction of all quality attributes are taken into account, and only three web browsers (Opera7.0,

[3]The advertisement-free version of Opera 7.11 costed USD39, as of June 2003.

Microsoft Internet Explorer 6.0, and Netscape 6.0) are considered in the implementation of softgoal framework. As shown in the figure, a *goodBrowser* has to be of good *"performance"* and *"security"*, and be *"userFriendly"*. The *NFR softgoals "Performance"* and *"security"* are important criteria, and are thus labeled with ! sign indicating high priority. For precise quality assessment of different browsers, *"performance"* is decomposed into two *NFR Softgoals*, *"speed"* and *"stability"*. The web browsers *"opera7"*, *"msie6"*, and *"netscape6"* are the *operationalizing softgoals* in the framework. The strengths and weaknesses of these browsers are stated in the framework as labeled edges towards various *NFR softgoals*. For example, Internet Explorer is infamous with its security loopholes, but offers consistent web page rendering and intuitive user interface. These ideas are captured in the softgoal framework by a BREAK (`--`) arrow drawn from the *operationalizing softgoal "msie6"* to the *NFR softgoal "security"*, and a MAKE (`++`) arrow towards the *"userFriendly"* softgoal. Other tradeoff relationships between *NFR* and *operationalizing softgoals* are expressed in the same manner.

The Browser Selection framework discussed above sets an example on how Softgoal Modeling technique can be applied to other systems to capture the tradeoffs/synergy between quality attributes and design alternatives. Once the Softgoal framework is defined, it can be encoded into text format for automatic inference.

The framework syntax is organized in a simple, natural-language structure so that users can code with minimal guidance. Before we proceed onto the details of the framework syntax, we present a fragment of the textual definition for the Browser Selection framework:

```
rule1 says goodBrowser if
  performance of browser
  any_of userFriendly of browser
  any_of security of browser.
rule2 says performance of browser if
  speed of browser
  any_of stability of browser.
  :
  :
rule6 says security of browser if
  unhurt by opera7
  any_of unbroken by msie6
  any_of unbroken by netscape6
```

For the complete textual definition of the Browser Selection framework as well as other framework definitions, please refer to A.2 in the appendix section.

3.2 Softgoal Framework Syntax and Keywords

The defined Softgoal framework is encoded into text format, which is transformed into Prolog [139] specific terms for automatic inference. We extended the input syntax of

JANE [84], a prepositional rule-based language, to represent the graphical structure of the framework in text. The JANE syntax is of the form:

```
conclusion if preconditions
```

To integrate with softgoal framework additional features and to allow more flexibility, we extended this syntax as described in the BNF[4] as shown:

```
<rule> ::== <stakeholder> says <conclusion>
            if <precondition> |
            <combine_logic> <precondition>
            .
<conclusion> ::== <nfr_softgoal>
<precondition> ::==
            <softgoal> |
            <contribution> by <softgoal> |
            <contribution> by <softgoal>
            with <contribution> by <claim_softgoal>
            .
<softgoal> ::=
            <nfr_softgoal> |
            <claim_softgoal> |
            <operationalizing_softgoal>
```

`<rule>` captures a quality attribute (`<conclusion>`) and its satisficing criteria proposed by a `<stakeholder>`. Each criterion is declared in `<precondition>` and chained with `<combine_logic>`. The `<rule>` is valid only if the `<stakeholder>` is to be trusted; it is not considered otherwise.

`<precondition>` can be a `<softgoal>` to be fulfilled, or a `<softgoal>` combined with a `<contribution>`. `<contribution>` determines the effect a `<softgoal>` has when satisficed. For instance, if the `<contribution>` of a `<softgoal>` has a negative effect, then satisficing the `<softgoal>` will invalidate the `<conclusion>`, and vice versa.

Finally, `<softgoal>` can be a quality attribute `<nfr_softgoal>`, a design option (`<operationalizing_softgoal>`), or a claim (`<claim_softgoal>`).

The above syntax is common across all Softgoal frameworks. Depending on the problem domain, certain fields (such as `and`) can be modified to accept additional attributes and varied inference scheme.

3.3 Logical Operators

3.3.1 Concept

When several qualitative contributions are asserted onto a softgoal, combination logic is utilized to sum their influences. Since such combination methods are themselves

[4]BNF is an acronym for "Backus Naur Form". John Backus and Peter Naur introduced for the first time a formal notation to describe the syntax of a given programming language.[99]

heuristic and prone to change, we model them explicitly in a syntax that makes their definition plain and easily customizable. As previously discussed, attributes in `<combine_logic>` are mapped to terms which specify the logical operation and parsing order during inference. The logic terms currently implemented to the *Softgoal Simulation Tool* are AND, OR, and ANY. The rule structure below illustrates the inferring scheme of logical operations:

```
conclusion if
   precondition1 op precondition2 op precondition3.
```

For op `and`, all the preconditions have to be satisficed in order to satisfice the conclusion. The inference engine examines all preconditions, and drops the rule if any one of the preconditions is not satisficed. For op `or`, one satisficed precondition among all is sufficient to satisfice the conclusion. The inference engine tries out each of the preconditions until none of them is satisficed, in which the rule is dropped. `Any` can be viewed as a greedy logic or: the inference engine tries to prove one of more of `precondition1` and `precondition2` and `precondition3`. `any` is useful when searching for subsets that contribute to certain conclusion. For example:

```
minimize_damage_in_rainydays if
   have_umbrella any
   wear_rain_coat any
   indoor_activities.
```

Any of the preconditions (`have_umbrella`, `wear_rain_coat`, or `indoor_activities`) is sufficient to satisfice the conclusion (`minimize_damage_in_rainydays`), but it is more desirable to prove more than one.

As seen from the above logic definitions, chaining preconditions with op `and` imposes the strictest constraint towards fulfilling the rule. In other words, when some softgoals are chained by `and`, if the concluding softgoal they contribute to is satisficed in a “world”, then all of them must have been satisficed in that particular world. On the other hand, if these softgoals are chained with op `or` instead, then the inference would result in the least of them being satisficed for the same “world”. Op `any` is similar to op `or` in its satisficing criteria, but the inference engine would try to prove more than one of the chained softgoals. These different logic operations may be used in building the framework to address different concerns. For example, when the goal is to find solution which optimizes the concluding softgoal, chaining its preconditions with `and` ensures that the resulting “world” most satisfice the conclusion. `or` may be used when the objective is to find the least preconditions that best satisfice the concluding softgoal. `any` is useful when it is logically impossible to have all preconditions satisficed (i.e., chaining preconditions with `and` results in no solution to the framework).

Descriptions of the complete inference process can be found in chapter 4.1.

3.3.2 Actual Implementation

As previously mentioned, the `<preconditions>` of a rule are chained by `<combine_logic>`. Several predefined keywords included `all_of`, `any_of`, `any_one_of`, `one_of`,

Rule Structure:

```
conclusion if
   precondition1 op precondition2 op precondition3.
```

Logic		To Prove `<conclusion>`			To Disprove `<conclusion>`		
L to R	Random	# Positives	# Negatives	Tries	# Positives	# Negatives	Tries
and	rand	all	0	all	dc	1	1
or	ror	1	dc	1	0	all	all
any	rany	1	dc	all	0	all	all
xor	rxor	1	all - 1	all	0;2	dc	2;all
nand	rnand	0	all	all	1	dc	1
nor	rnor	dc	1	1	all	0	all
nany	rnany	dc	1	all	all	0	all

Key:
L to R: `<preconditions>` are solved from left to right;
Random: `<preconditions>` are solved in random order;
# Positives: number of satisficed `<preconditions>` required in order to satisfice the `<conclusion>`;
# Negatives: number of denied `<preconditions>` permitted to satisficed the `<conclusion>`;
Tries: number of `<preconditions>` the inference engine attempt to prove/disprove;
all: all `<preconditions>`;
dc: don't care;

Table 3.1: Implementation of Internal Logic Types

and `none_of`. They are mapped to different textitinternal logic types as shown in the following code segment:

```
infer(all_of, rand).
infer(any_of, rany).
infer(any_one_of, ror).
infer(one_of, rxor).
infer(none_of, rnand).
```

Additional keywords can be defined, and be mapped to various logic types according to user preference.

Internal logic types such as `rand`, `rany`, `ror` are the randomized versions of types `and`, `any`, and `or`. For `op (and, any, or)`, the inference engine will attempt to solve the preconditions in the order from left to right. On the other hand, the preconditions are attempted in random order when chained with `(rand, rany, ror)`.

Table 3.1 describes the implementation of the internal logics of the *Softgoal Simulation Tool*. Using logic `rand` as an example: all `<preconditions>` chained with this logic are to be proved true in order to satisficed the `<conclusion>` of a `<rule>`. In other words, none of the `<preconditions>` are allowed to be not satisficed[5] to satisfice the `<conclusion>`. The inference engine will attempt to prove all `{preconditions>` to enforce such restrictions. To refute the `<conclusion>`

[5]either denied or not known

and disprove the `<rule>`, only one denied `<precondition>` is needed; the number of satisficed `<preconditions>` will not be taken account for. Therefore, the inference engine will only attempt to disprove one `<preconditon>` to disprove the `<rule>`.

Consider the implementations of logics `or` and `any`; both have the same requirements for number of satisficed/denied `<preconditions>` to fulfill the `<conclusion>` (i.e., to prove the `<rule>`), however, all the `<preconditions>` will be attempted on if chained with logic `any`. Whereas in the case of `or`, the inference engine will not solve the rest of the `<preconditions>` once one has been proven.

Please see appendix A.1.1 for the actual code segment.

3.3.3 Inferring keyword `with`

For generic Softgoal frameworks, verbs such as `made`, `helped`, `unhurt` and `unbroken` [6] are termed as `<contribution>`. In the study of the *KWIC* framework (chapter 5) and many other case studies, we have interpreted `made` and `helped` as having positive effects, whereas `unhurt` and `unbroken` as having negative effect towards the concluding quality attribute stated at the start of each rule (e.g., `modifiability of dataRep` shown in `rule 1`). Attributes such as `all_of`, `any_of`, `any_one_of` are called `<combine_logic>`, which are mapped onto various logical operations (AND, OR, etc) and parsing sequences during inference. `<combine_logic>` determines the inference scheme on a set of preconditions (e.g., `modifiability of process, dataRep` and `function` as stated in `rule9` at the beginning of this section). Again, new attributes can be added and their mappings changed according to the problem domain.

The `with` attribute is used specifically in chaining a `<claim_softgoal>` and its effect on the `<contribution>` of another softgoal towards the concluding softgoal. Consider the following rule as example:

```
he says palatability of pizza if
  made by pepperoni of crust
  all_of made by stuffedCheese of crust
    with helped by claim of pizzaParlor.
```

In this example, whether or not `stuffedCheese of crust` contributes to `palatability of pizza` is dependent on the validity of the `claim of pizzaParlor`. If `pizzaParlor` is not to be trusted, his claim on the contribution of `stuffCheese of crust` to `palatability of pizza` is no longer supported. Hence, the `palatability of pizza` solely depends on `pepperoni of crust`.

3.3.4 Configuring `<contribution>`

All `<contributions>` are classified into two types: they either assert positive (`good_effect(...)`) or negative (`bad_effect(...)`) impacts towards their corresponding `<softgoals>`. Some sample declarations of `<contribution>` types are shown below:

[6] `made`, `helped`, `unhurt` and `unbroken` denote MAKE, HELP, HURT and BREAK respectively

Case 1: `<precondition> ::== <contribution> by <softgoal>`		
`<contribution>`	`<softgoal>`	`<precondition>`
$+$	satisficed	satisficed
$+$	denied	denied
$-$	satisficed	denied
$-$	denied	satisficed

Case 2: `<precondition> ::== <contribution0> by <softgoal>` `with <contribution> by <claim_softgoal>`		
`<contribution>`	`<claim_softgoal>`	`<precondition>`
$+$	satisficed	`<contribution0> by <softgoal>`
$+$	denied	$disappear$
$-$	satisficed	$disappear$
$-$	denied	`<contribution0> by <softgoal>`

Key:
$+$: `<contribution>` which has a positive impact towards the `<precondition>`;
$-$: `<contribution>` which has a negative impact towards the `<precondition>`;
$disappear$: `<precondition>` vanishes within the `<rule>` and no longer affect the `<conclusion>`;

Table 3.2: Implementation of Internal Logic Types

```
good_effect(helped).
good_effect(made).
bad_effect(unhurt).
bad_effect(unbroken).
```

The inference schemes of `<contribution>`-`<softgoal>` pair and the extra `<claim_softgoal>` support within a `<precondition>` are described in table 3.2. A `<precondition>` is satisficed only if its satisficed `<softgoal>` is supported by a positive `<contribution>`, or its denied `<softgoal>` is refuted by a negative `<contribution>`. For *case 2* where `<claim_softgoal>` is involved, the `<precondition>` is considered only if a proven `<claim_softgoal>` is supported by its associated positive `<contribution>`, or a false `<claim_softgoal>` is refuted by a negative `<contribution>`. Otherwise, the entire `<precondition>` is taken out from the `<rule>` and no longer has any impact on the `<conclusion>`.

As previously mentioned, the *Softgoal Simulation Tool* is developed to handle uncertain and conflicting knowledge commonly occurred in real world problems. Often times, business users face situation where the effect of some operations towards softgoals remained refutable. To resolve this modeling problem, users can specify this ill-defined `<contribution>` keyword to be a set of well-defined `<contributions>`. The *Softgoal Simulation Tool* will randomly select one from the set to represent the `<contribution>` keyword for each framework inference. The following presents some framework code segments as an illustrative example:

```
rule14 says standbyMode of avgs if
  positively_influenced by rav01 of ral
  positively_influenced by tav01 of tal...
```

For the above rule, `positively_influenced` is declared to be either `helped` or `made`, as shown in the code below:

```
roulette(positively_influenced,helped).
% good effect with a lower score
roulette(positively_influenced,made).
% good effect with a higher score
```

During one framework inference, the *Softgoal Simulation Tool* randomly selects `helped` or `made` to represent the `<contribution>` keyword `positively_influenced` associated with the `<softgoal>` `rav01 of ral1`, and selects again for the keyword associated with `tav01 of tal1`. Therefore, the actual `<contribution>` representing the keywords `positively_influenced` associated with `rav01 of ral1` and `tav01 of tal1` can be different. After the actual representation has been selected for each keyword instance, it will be cached and will remain the same for the entire inference.

Here is another sample rule and `<contribution>` keywords declaration:

```
ruleD says performanceAndOperation of faultCriticality if
  any_of moderately_rated by cam of dart
  any_of critically_rated by diagnosticMode of avgs...

good_effect(critically_rated).
bad_effect(lowly_rated).

roulette(moderately_rated,critically_rated). % good effect everytime
roulette(moderately_rated,lowly_rated). % sometimes good sometimes bad
```

The above code is used in the case study of the ***NASA IV&V* Activity Prioritization**. For further details please refer to chapter 8 of the case study section.

3.3.5 Configuring `<stakeholder>`

The `<stakeholder>` is declared at the beginning of each `<rule>`. As mentioned in the framework syntax section (3.2), the `<conclusion>` softgoal cannot be satisficed unless the `<stakeholder>` is to be trusted. `<stakeholders>` can be declared in the following fashion:

```
stakeholder rule0.
stakeholder rule1.

rule0 says goodness of system if
modifiability of system...
rule1 says modifiability of system if
helped by c1
any_of modifiability of process...
```

To validate every rules associated with a class of `<stakeholders>` requires only one additional clause:

```
rightness(stakeholder).
rightness(god).
```

3.4 Graphical Representation of Framework during Inference

This section explains how the various concepts described in the above sections are applied to the reasoning of a textual framework. A sample textual softgoal framework is created as in 3.2, and a tree-structured graphical representation is presented to explain the inference process.

The circles in figure 3.2 represent the `<softgoal>` and `<stakeholder>` nodes. As shown in the figure, **rule c** becomes a tree-like structure, with softgoal nodes `mushroom of pizza` and `pepper of pizza` combined with the AND logic as described by **logic table 3**. The logical outcome is combined with `<stakeholder> mary` with an "and" gate, and traverses up towards the parent node ¬`vegetable of pizza`. **Rule b** is decomposed in a similar fashion as **rule a**. For **rule a**, the `<with>` keyword results in an intermediate node connecting softgoals `stuffedCheese of crust` and `claim of pizzaParlor` with the combining logic shown in **logic table 4**. The intermediate node is then combined with the precondition softgoals `saltiness of pizza`, ¬`sogginess of crust`, and ¬`vegetable of pizza` with the ANY logic shown in **logic table 1**. Its result is combined with the stakeholder `tom` with an "and" gate, and the outcome goes to the top-level parent node `palatability of pizza`.

Thorough descriptions on how the inference is performed can be found in chapter 4.1.

3.5 Cost and Benefit Settings for Evaluation

As mentioned in section 3.2, each inference on the framework results in a cost and benefit score. To compute these values, qualitative factors such as softgoal priorities (`!`, `!!`) and contributions (`-`, `++`) are involved. Numerical values are required to represent these factors during automatic inference, yet there is no definition available for quantification. This section outlines the approach taken by our simulation tool to handle the calculations of benefit and cost under such limitation.

3.5.1 Handling Source of Uncertainty within Softgoals

As there is no conventional basis for quantifying subjective knowledge (e.g., HELP, MAKE), a quantification rule is thus created to state the rankings of various qualitative strengths. Under this rule, the mean of all quantified parameters must satisfy some numerical constraints. A typical quantification rule is stated below:

```
0 <= score("--") <= score("-") <= 1 <= score("+") <= score("++") <= 2
```

A score less than 1 reflects a weakening effect. Having fixed a (0..1) range for weakening, a range between 1 and 2 is used to reflect strengthening of the influence.

The *Softgoal Simulation Tool* allows a wide variety of scoring functions. In addition to fixed values, the score can be specified as a bounded random number, a Gaussian

Rule a:

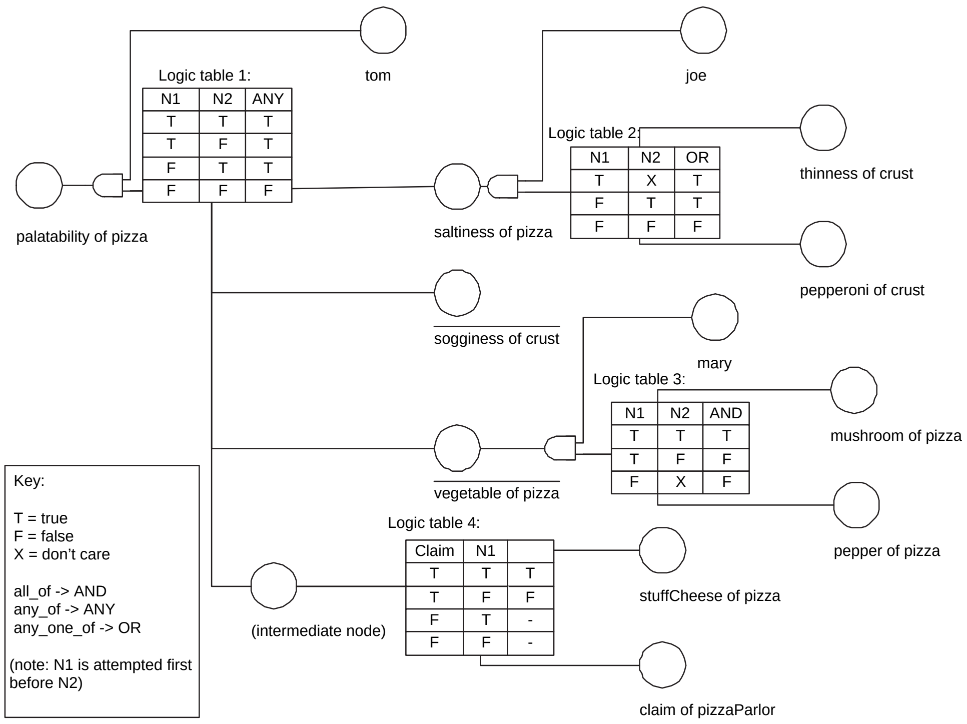
```
tom says palatability of pizza if
  saltiness of pizza
  any_of unhurt by sogginess of crust
  any_of unbroken by vegetable of pizza
  any_of made by stuffedCheese of crust
    with helped by claim of pizzaParlor.
```

Rule b:

```
joe says saltiness of pizza if
  helped by thinness of crust
  any_one_of made by pepperoni of crust.
```

Rule c:

```
mary says vegetable of pizza if
  mushroom of pizza
  all_of pepper of pizza.
```



Logic table 1:

N1	N2	ANY
T	T	T
T	F	T
F	T	T
F	F	F

Logic table 2:

N1	N2	OR
T	X	T
F	T	T
F	F	F

Logic table 3:

N1	N2	AND
T	T	T
T	F	F
F	X	F

Logic table 4:

Claim	N1	
T	T	T
T	F	F
F	T	-
F	F	-

Key:

T = true
F = false
X = don't care

all_of -> AND
any_of -> ANY
any_one_of -> OR

(note: N1 is attempted first before N2)

Figure 3.2: Sample Softgoal Textual Framework and its Graphical Representation

with pre-defined mean and variance values, or a bounded number following *Beta Distribution*. Users can adjust the scoring functions according to the business model and the degree of inconsistency of domain knowledge .

Examples of scoring functions are shown below:

```
score(helped, multiply, 1.4).
% fixed value
score(made, multiply, range(1.5,1.8))).
% 1.5 < number < 1.8
score(unhurt, multiply, randomize(normal(0.6,0.2), [[=<,2],[>=, 0]])).
% Gaussian with mean=0.6, variance=0.2, restriction: 0 =< number =< 2
score(unbroken, multiply, randomize(skewed_range(0.6, 0.1, 0.8)).
% beta distribution with mean=0.6, range=(0.1, 0.8)
```

The scoring functions of the **Web Browser Selection** framework is shown below as another example:

```
score(helped,  multiply, randomize(normal(1.4,0.2), [[>=, 0], [=<,2]])).
score(made,  multiply, randomize(normal(1.8,0.2), [[>=, 0], [=<,2]])).
score(unhurt,  multiply, randomize(normal(0.6,0.2), [[=<,2],[>=, 0]])).
score(unbroken,  multiply, randomize(normal(0.2,0.2), [[>=, 0], [=<,2]])).

% weight(Severity, SoftgoalName, Weight)
weight(veryCritical,_,randomize(normal(2,0.3), [[>=, 0], [=<,3]])).
weight(critical,_,randomize(normal(1.5,0.3), [[>=, 0], [=<,3]])).
weight(normal,_,randomize(normal(1,0.3), [[>=, 0], [=<,3]])).

score(stakeholder, 0).
score(default, 1).  % default benefit score
```

Moreover, users can customize the methods that the inference engine uses for combining the influence of child nodes. It takes the format:

```
score(Logic_keyword, Arithmetic_combine_rule, Multiplier)
```

`Logic_keyword` refers to the combination logic keywords such as `all_of` and `any_of`. `Arithmetic_combine_rule` defines the arithmetic functions for combining the benefit of all satisficed child nodes. `Multiplier` is the number by which the benefit score is multiplied, and passes over to the parent node. The *Softgoal Simulation Tool* currently supports arithmetic functions such as `add`(sum up the benefit scores of all nodes), `min`(choosing the minimum benefit scores from all child nodes), `max`(finding the maximum among all), `multiply`(multiply the scores for all child nodes), and `average`.

The following combination rules come from the **Web Browser Selection** framework (from section 3.1.1):

```
score(all_of, min, 1). % minimum benefit score
score(any_of, add, 1).  % summation
score(one_of, max, 1).  % maximum benefit score
score(any_one_of, max, 1).
score(none_of, min, -1). % minimum benefit score multiply by -1
score(claim2parent, multiply, 1).
% multiply benefit(claim) with benefit(claim target)
```

The cost of each softgoal can be configured similar to that of Benefit definition. The following shows the cost definition of the SR-1 project in (chapter 8) as an example:

```
basic_cost(high, randomize(range(2,10))).
cost_variance(_AllPriceType,0.04).
% for x(mean)=1
scaling_factor(high, notHigh, 0.7).
scaling_factor(high, veryHigh, randomize(skewed_range(1.2, 1.1,1.7))).
% F=(1.1, 1.7)L, mean(F)=1.2L
scaling_factor(high, extremelyHigh, randomize(skewed_range(1.6,1.2,1.7))).
% F=(1.2, 1.7)L, mean(F)=1.6L

softgoal rav01 of ral costed notHigh.
softgoal dav09 of dal costed high.
softgoal dav10 of dal costed veryHigh.
```

As mentioned before, the cost and benefit computed in each inference produce a “rating” on the “desirability” of a particular “world”. These ratings, once obtained by performing *Monte Carlo* process, are used by a treatment learner for classification. Details on treatment learning are given in section 4.5.

Chapter 4

Implementation

This chapter details the implementation of the Softgoal Framework Simulation processes. First, the concepts of framework inference approach and other possible alternatives are discussed, follows by a step-by-step walk through of the inference process to further explain the concept. The rules of scoring the inferred framework is then presented. Lastly, the ideas of *Monte Carlo* Simulation and *treatment learning*[87], and how they are incorporated into the *Softgoal Simulation Tool*, are explained.

4.1 Inference Scheme

After a softgoal framework is constructed for the target system, it is fed to the *Softgoal Simulation Tool* for automatic inference. The methodology for inferring the softgoal framework structure is described in this section. It begins with the discussion on the design concerns, follows by an investigation on the abductive reasoning approach taken in implementing the inference engine. Experiments done with alternative implementation strategies are also discussed.

4.1.1 Design Concerns

Here are a list of concerns to be addressed in implementing the inference engine of the *Softgoal Simulation Tool*:

- Simplicity - the inference engine should adopt a reasoning approach that is simple to understand and yet well-studied and sound.
- Speed - as the framework inference needs to be executed multiple times (10,000 or more), it should not take minutes to run one inference.
- Scalablily - it should be able to handle large framework.
- Flexibility - it should accept variations of scoring rules and graph-walking schemes.

Since the *Softgoal Simulation Tool* deals with uncertainties and imprecise numbers, numerical accuracy is not an issue. Moreover, the simulation tool relies on the treatment learning tool TAR2 (discussed in section 4.5) which classifies framework behaviors generated by *Monte Carlo Simulation* (section 4.4). It is the difference in numerical scorings between each inference run that impacts the simulation result. Therefore, the *Softgoal Simulation Tool* have not considered integrating math processing software packages (such as *Matlab* or *Maple*).

4.1.2 Abductive Inference Approach

Charles S. Peirce defines the concept of “abduction” as “...the process of forming an explanatory hypothesis.”[103, V]. In [106], abductive and default reasoning are paired with *explanation* and *prediction*: *Explanation* is where the proposition to be explained (the explanandum[1]) is an observation in the world, and we want to explain why this occurred. *Prediction* is where the explanandum is unknown and the problem is to determine whether to predict the explanatum or not (forming a formalization of default reasoning). Kakas[61] offered the following formalization:

Given: $T = Theory$, $G = Observation$, and $\Delta \subseteq T$ *such that:*

$$T \cup \Delta \vdash G \tag{4.1}$$

$$T \cup \Delta \textit{is consistent.} \tag{4.2}$$

Definitions offered by other authors such as Bylander[23], Konolige[72], and Paul[102], while differed in their presentation formats, captured the essence of abductive reasoning: that a set of hypotheses can give good explanation to the observation, provided that the set does not give rise to contradiction.

Abduction is often thought of as a kind of backward reasoning. Where *modus ponens* infers from $p \rightarrow q, p$ to q, abduction infers from $p \rightarrow q, q$ to p. From a logical point of view, reasoning backward may be seen as conjectural, or presumptive thinking; nonetheless, abduction is a human cognitive process of adopting explanations, and is the only logical operation which introduces new idea. As in [103, V], Peirce claims that abduction can be represented as “a perfect definite logical form”: “The surprising fact, C, is observed; but if A were true, C would be a matter of course. Hence, there is reason to suspect that A is true”.

Many important kinds of intellectual tasks, such as medical diagnosis, fault diagnosis, scientific discovery, and natural language understanding have been characterized as abduction. The model of abductive inference has been used in various diagnostic expert systems (RED[60], TIPS[109], PATHEX/LIVER[126], MDX2[129], and QUAWDS[141]), imitating mankind’s “mode” of dealing with uncertain information in a very successful way.

4.1.3 Concept

The acyclic graphical representations (example shown in figure 3.2) define the qualitative causal relationships of the Softgoal Framework. The following rule:

```
x says goal if
  c.
y says c if
  unbroken by e.
z says c if
  made by e
  all_of made by f.
```

[1]Explanandum is the thing that needs to be explained.

is interpreted in its logical form as:

$$goal \rightarrow x \wedge c. \tag{4.3}$$

$$c \rightarrow y \wedge \bar{e}. \tag{4.4}$$

$$c \rightarrow z \wedge e \wedge f. \tag{4.5}$$

Using the theory of abductive reasoning, $goal$ is assumed true and other nodes are *abduced* as long as they do not violate the truth (i.e. the truth of $goal$). In this example, two "worlds" can lead to the fulfillment of $goal$ with no contradiction:

$$goal \rightarrow x \wedge c \wedge y \wedge \bar{e}. \tag{4.6}$$

$$goal \rightarrow x \wedge c \wedge e \wedge f. \tag{4.7}$$

The inference engine utilizes a top-down approach in finding possible "worlds", hereby reasoning the Softgoal Framework. It explores the framework based on the truth of the top-level goal and traverse downwards. The top-level softgoal node of a Framework represents an abstraction of overall quality of the target domain. Each search performed on the framework generates a consistent "world" - a scenario where the top-level softgoal is satisficed when a set of softgoals are satisficed / denied. This "world" can be different for each search, depending on the topology and randomness embedded in the framework definition. After a "world" is generated, its "goodness" is rated by computing the "benefit" and "cost" of this particular "world" based on various user-configured parameters. The complete inference approach of the *Softgoal Simulation Tool* is explained with step-by-step walk-throughs of the graph traversal, presented in section 4.2. The "benefit" and "cost" calculations are described in section 4.3 in this chapter.

4.1.4 Other Inference Strategies

- Heuristic Methods
  Many other inference schemes can be developed to reason the Softgoal Framework. An experiment previously done on Softgoal Framework used the following heuristic implementation: for each inference, the leaf nodes (i.e. *operationalizing* and *claim softgoals*) are randomly selected to be satisficed/denied, and its costs assigned (satisficed=1, denied=0). Then, their influence towards the parent non-leaf nodes (i.e. benefit scores) are computed based on the `<contributions>` (e.g., `++`, `+`)[2] and `<combine_logics>` (e.g., `rand`, `rany`)[3]. These scores propagate bottom-up to the top-level parent node, in which the overall benefit score is determined. Unlike the current abductive approach, there is no concept of denied non-leaf softgoal nodes under this approach, only a non-leaf softgoal

[2] $1 \geq X_{++} \geq X_{+} \geq X_{-} \geq X_{--} \geq -1$.

[3] The benefit of a `rand` combo is the minimum of all input child-node influences. The benefit of a `ror` is the maximum of all input influences. Likewise, `rany` is the average of input influences.

node with negative benefit scoring. Moreover, each framework inference is a complete graph search, where each node is considered for benefit calculations.

Heuristic inference can be very flexible and customizable to individual user's needs; it can be developed free of complicated graph reasoning theories, provided that all users understand and agree upon certain heuristics. Nevertheless, without the support of some well established theories (such as *Bayesian*), the trustworthiness of the inference results has to be further justified. Many experiments have to be performed on the heuristic algorithm alone, in order to verify the result produced is not just some ad hoc basis.

- Bayesian Reasoning
  Bayesian reasoning theory is a quantitative, probabilistic approach of solving decision problems. Its algorithm is widely studied and well understood, and inference tools widely available. To apply Bayesian theory in defining the softgoal framework, one needs to translate the relationship between operational softgoals (design alternatives) and NFR softgoals (software quality attributes) into probabilities (e.g., 0.6, 0.8). These numerical values may not be as intuitive as linguistic descriptions ("HELP", "MAKE"). Moreover, when data required to determine the likelihoods for the calculation of posterior probabilities is not available, certain assumption has to be made. These assumptions negate the authenticity of the answer obtained by Bayesian inference. Therefore, Inference Diagram may not be the best for early lifecycle requirement modeling.

  In spite of these shortcomings, an experiment on applying Bayesian theory to solve Softgoal Framework problem has been performed on the *Keyword in Context* case study(chapter 5). To perform Bayesian Inference, the Softgoal Framework is transformed into an *Influence Diagram* - a representation of decision problem that can be analyzed by Bayesian decision theory[82]. *NFR Softgoals* and *Claim Softgoals* become the *Random Variables*, with their *probability* of being satisficed on *Decision* nodes and other *random variables* in the *Influence Diagram*. *Operationalizing Softgoals* become the *Decision* node as they are interpreted as having deterministic values (either true or false) in terms of *Bayesian* reasoning theory. `<contributions>` are translated into the *conditional probabilities* associated with each softgoal, and `<priorities>` become the factors the overall *utility* of the transformed *Influence Diagram*. Sample transformations can be found in the appendix section (A.4).

  Similar to the *Softgoal Simulation Tool* reported in this thesis, the code for *Bayesian* Softgoal Framework inference is implemented entirely in *Prolog*. The *Bayes Net* meta-interpreter portion of the inference engine is borrowed from [105][4]. A major drawback of this implementation is that it can only handle a small Softgoal Framework. When tested on the *Keyword in Context* framework (chapter 5), it is found that the inference engine is unable to process the entire transformed *Influence Diagram*[5]. The speed of such inference engine is also disappointing: one inference of a reduced-size *Keyword in Context* framework takes

[4]Available at: `http://www.cs.ubc.ca/spider/poole/ci/ci_code.html`
[5]Stack-size errors resulted despite manually increasing the working memory size.

approximately 5.75 seconds. Comparing with the current abductive approach which takes the maximum of 5 minutes for 10,000 inference[6], the performance of the *Bayesian* implementation is unsatisfactory.

4.2 Inference Walk-Through

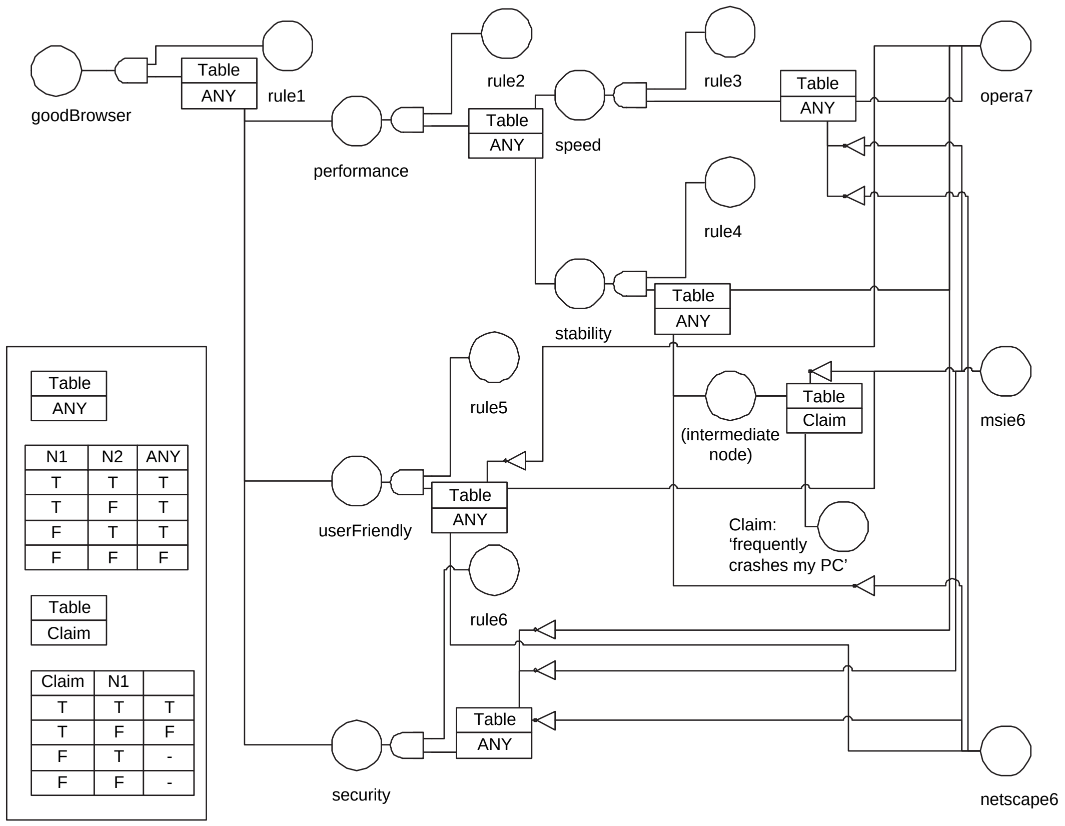


Table
ANY

N1	N2	ANY
T	T	T
T	F	T
F	T	T
F	F	F

Table
Claim

Claim	N1	
T	T	T
T	F	F
F	T	-
F	F	-

Figure 4.1: Web Browser Framework: Logical View

Using the **Web Browser Selection** framework discussed in chapter 3.1.1 as an example, this section describes the flow of framework traversal approach taken by the *Softgoal Simulation Tool*. Three variations of "worlds", i.e. graph-walk routes and associated scorings, are presented (figures 4.2, 4.3 and 4.4) to explain how the *Softgoal Simulation Tool* reasons a given framework problem. The Softgoal Framework is converted into logical view (as shown in figure 4.1) for clarity purpose. The steps taken by the inference engine (for one graph walk) are labeled (in numbers) for each inference

[6]Time taken for inferring the complete *KWIC* framework with all logics set to `any_of`.

variation, and the satisficed/denied softgoal nodes are marked with checks/crosses. Benefits and costs are calculated after the completion of each graph walk. Details on scoring are provided in section 4.3.

4.2.1 Graph Walk

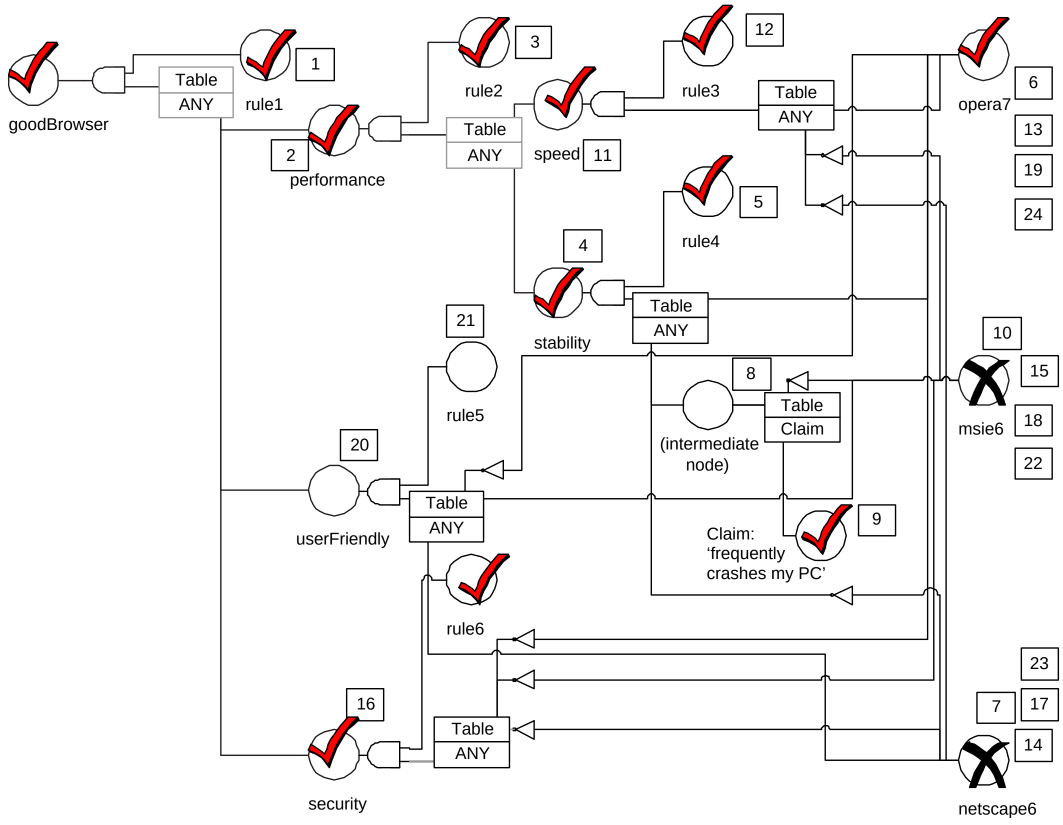


Figure 4.2: Inferring Web Browser Framework: logic `rany`

Figure 4.2 shows a possible graph walk route for inferring the **Web Browser Selection** framework. Descending from the top-level softgoal node *"goodBrowser"*, the tool assumes *"rule1"* to be true and randomly picks one of the second-level nodes - *"performance*, *userFriendly* and *security* - to prove. Softgoal *"performance"* is selected, assumed to be satisficed, and the tool infers downwards to *rule2* and the third-level nodes. *Stability* is picked first under the logic `rany`, and hence *rule4* is assumed true. Since none of the leaf-node *opera7*, *msie6*, its supporting claim *"frequently crashes my PC"* and *netscape6* are "assumed" to be satisficed or denied, they took the values according to the rule (i.e. *opera7*=satisficed, *msie6*=denied , *"frequently..."*=satisficed

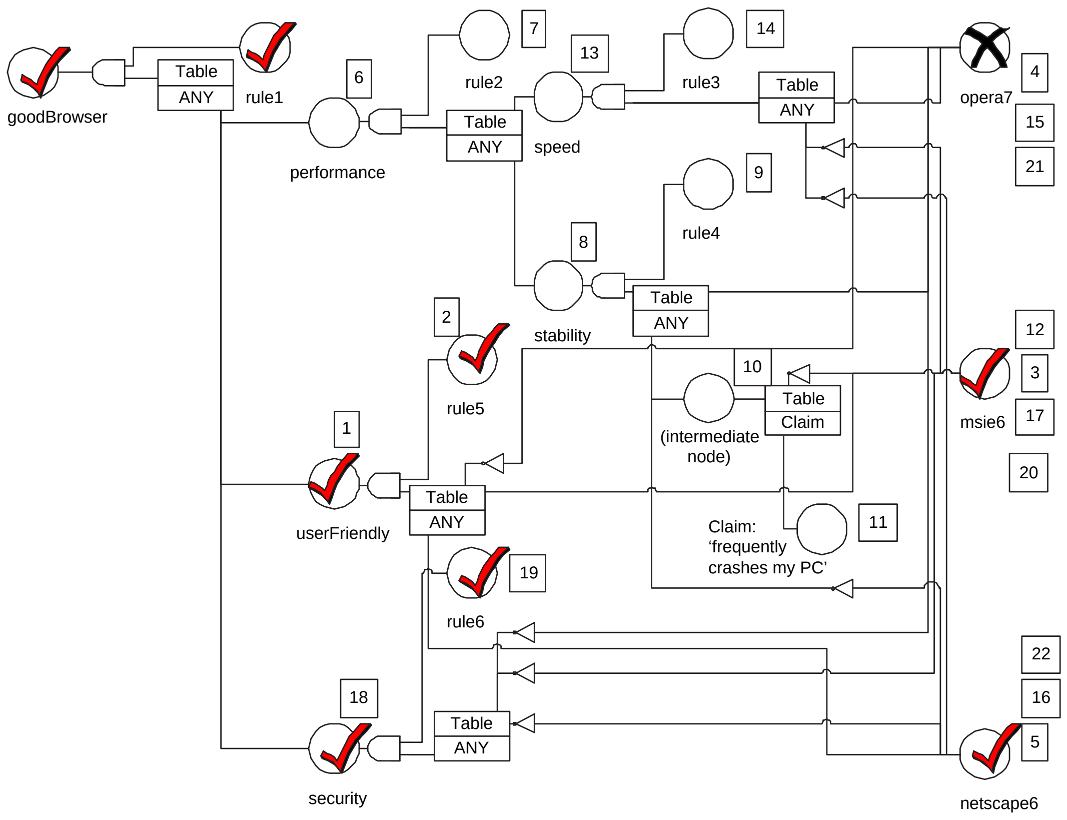
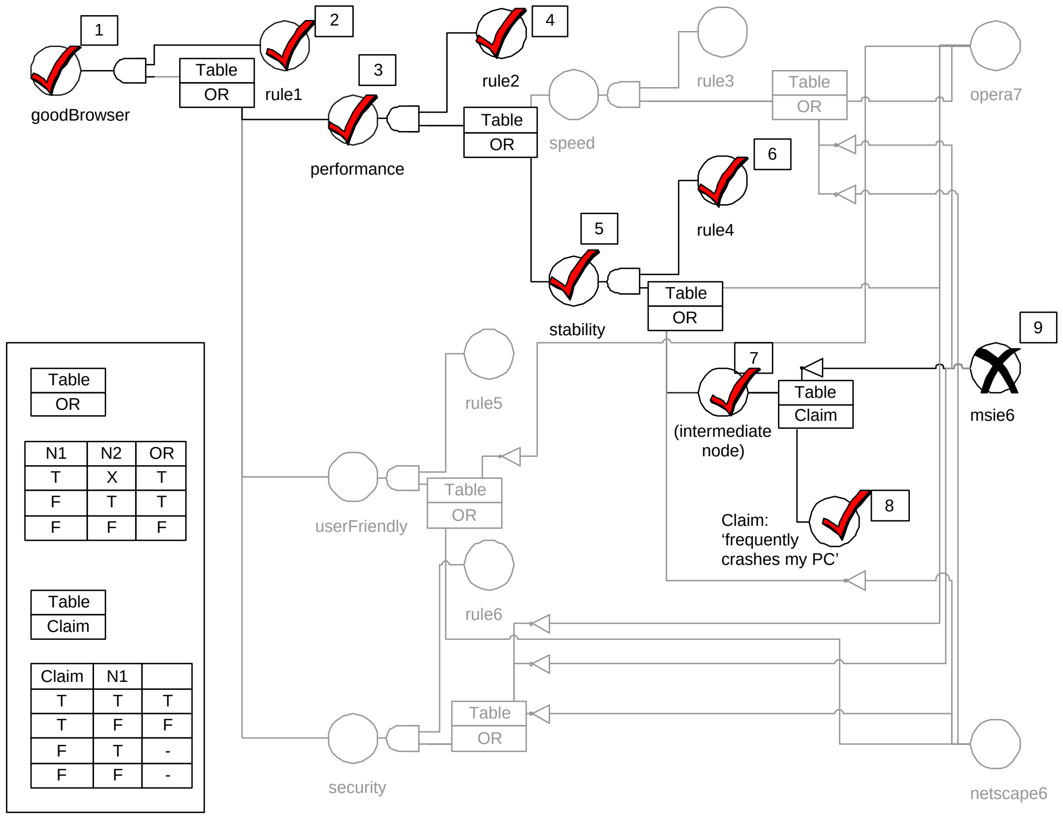


Figure 4.3: Inferring Web Browser Framework: logic `rany` (variation)

and *netscape6*=denied to satisfice their parent softgoal node (*stability*). Afterwards, it tries to prove the softgoal node *speed* to be satisficable; as the leaf-nodes requirements of being satisficed are the same as the softgoal node *stability*, *speed* is also proved satisficed. Coming back to the second-level softgoal nodes, *security* is picked under the `rany` logic. *Rule6* is assumed true, and the inference engine goes on to prove *opera7*, *msie6* and *netscape6*. Both *msie6* and *netscape6* complies with the satisficing criteria, and hence *security* is satisficed under logic `rany`. Lastly, the left over node *userFriendly* is picked, and since none of the leaf-node fulfill its satisficing criteria, the softgoal *userFriendly* is thus not being satisficed.

Figure 4.3 is another possible route taken by the inference engine to reason the Softgoal Framework. By taking different graph walk route, the end results of satisficed/denied softgoals, and hence the overall cost and benefit scores, are different. One run of *Monte Carlo* simulation generates a wide range of scores and corresponding framework behaviors, which are recorded and classified by the treatment learning tool TAR2. (Details on *Monte Carlo* simulation and TAR2 can be found in the coming sections.)



N1	N2	OR
T	X	T
F	T	T
F	F	F

Claim	N1	
T	T	T
T	F	F
F	T	-
F	F	-

Figure 4.4: Web Browser Framework: logic `ror`

Figure 4.4 illustrates the graph walk route of the *Web Browser Selection* framework under a different configuration of combination logics. In this example, the combine logic `rany` is replaced with `ror`, and the other settings have remained the same. Solid lines represents the route taken by the inference engine, and dotted lines otherwise. Note that some routes are skipped by the inference engine, and many softgoals remain unknown under the logic `ror`. By comparing figure 4.4 with figures 4.2 and 4.3, it is apparent that the framework behaviors and scores generated under one logic combination can be very distinct from another.

4.3 Scoring

4.3.1 BNF for Scoring Rules

Benefit computation of one softgoal involves in factors such as: *priority*, default weight, and its precondition softgoals that leads to its being satisficed (if any). The BNF described below specifies the calculation rules:

```
<benefit> ::==
  function(<score[weight]>,
           <benefit_combine>) |
  function(<score[weight]>)

<score[weight]> ::==
  function(<arithmetic[priority]>,
           <value[priority]>, <weight>)
```

For each non-leaf-softgoal (i.e. softgoals that have preconditions), a recursive descent is executed and the returned benefit values (`<score[precondition]>`) are combined according to some combination rule (`<arithmetic[combine_logic]>`).

```
<benefit_combine> ::==
  function(<arithmetic[combine_logic]>,
           <score[precondition1]>,
           ... <score[preconditionN]>)
```

In cases where a precondition consists of a softgoal and a contribution factor (e.g., `helped`, `made`), the benefit values is computed by combining its `<benefit>` weight with the (numeric) contribution weight by some arithmetic operation (`<arithmetic[contribution]>`). Otherwise, its default `<benefit>` weight is taken.

```
<score[precondition]> ::==
   function(<benefit>)|
   function(<benefit>, <arithmetic[contribution]>,
            <value[contribution]>)
```

Arithmetic operations such as summation(`add`), average, min, max, and multiplication are supported by the inference.

```
<arithmetics[X]> ::==
     add | average | min | max | multiply
```

The overall benefit of a “world” is equivalent to the benefit of the top-level softgoal.

4.3.2 Calculation Walk-Through

Using figures 4.2, 4.3 and 4.4 as examples, this section demonstrates how the benefits and costs are calculated. As defined in section 3.5.1, the benefits of child nodes are summed up under the logic `rany`. The cost of all web browser choices (i.e. *operationalizing softgoals* is 1. Randomized values of various keywords are given as follows:

randomized(`helped`)=1.4
randomized(`made`)=1.8
randomized(`unhurt`)=0.6
randomized(`unbroken`)=0.2
randomized(`critical`)=1.5
randomized(`normal`)=1.0

The calculation corresponds to figure 4.2 is shown below:

`rule3`:
`made by opera7` $\Rightarrow 1 * 1.8 = 1.8$
`unhurt by msie6` $\Rightarrow 0 * 0.6 = 0$
`unbroken by netscape6` $\Rightarrow 0 * 0.2 = 0$
`any_of` $\Rightarrow 1.8 + 0 + 0 = 1.8$
$\Longrightarrow$ `speed of browser` $= 1.8$

`rule4`:
`helped by opera7` $\Rightarrow 1 * 1.4 = 1.4$
`with helped by 'frequently crashes my PC'` $\Rightarrow 1 * 1.4 = 1.4$
`unhurt by msie6` $\Rightarrow 1.4 * 0 = 0$
`unbroken by netscape6` $\Rightarrow 0 * 0.2 = 0$
`any_of` $\Rightarrow 1.4 + 0 + 0 = 1.4$
$\Longrightarrow$ `stability of browser` $= 1.4$

`rule2`:
`speed of browser` $\Rightarrow 1.8$
`stability of browser` $\Rightarrow 1.4$
`any_of` $\Rightarrow 1.8 + 1.4 = 3.2$
$\Longrightarrow$ `performance of browser (critical)` $= 3.2 * 1.5 = 4.8$

`rule6`:
`unhurt by opera7` $\Rightarrow 1 * 0.6 = 0.6$
`unbroken by msie6` $\Rightarrow 0 * 0.2 = 0$
`unbroken by netscape6` $\Rightarrow 0 * 0.2 = 0$
`any_of` $\Rightarrow 0.6 + 0 + 0 = 0.6$
$\Longrightarrow$ `security of browser (critical)` $= 0.6 * 1.5 = 0.9$

`rule1`: `performance of browser` $\Rightarrow 4.8$
`userFriendly of browser` $\Rightarrow 0$

`security of browser` $\Rightarrow 0.9$
`any_of` $\Rightarrow 4.8 + 0 + 0.9 = 5.7$
$\Longrightarrow$ `goodBrowser` $= 5.7$

Therefore, the scores obtained for the "world" of figure 4.2 are: cost=1 and benefit=5.7

With the same arithmetic, the "world" shown in figure 4.3 scores 2 for cost and 4.1 for benefit. Figure 4.4 scores 0 for both cost and benefit.

4.4 *Monte Carlo* Simulation

Monte Carlo Simulation[62] is a stochastic technique[7] used to solve mathematical problems by random sampling. Originally developed for the Manhattan Project during World War II, it is now applied to a wide disciplines such as nuclear reactor design, econometrics, stellar evolution, stock market forecasting etc.

A simulation can be deterministic, stochastic, or mixed. In the deterministic case, input parameters are specified as single values, and only one simulation run is needed for a given set of parameters. On the other hand, stochastic modeling recognizes the inherent uncertainty in many parameters and relationships. Rather than using (deterministic) point estimates, stochastic variables are random numbers drawn from a specified probability distribution. The result variables are analyzed statistically across a batch of simulation runs: this is termed *Monte Carlo* simulation.

Monte Carlo methods randomly select value (within a fixed range) to create scenarios of a problem. Each time a value is randomly selected, it forms one possible scenario and solution to the problem. The random selection process is repeated many times to create multiple scenarios, some of which are more probable and some less probable. When repeated for many scenarios (10,000 or more), the average solution will give an approximate answer to the problem.

In the design of the *Softgoal Simulation Tool*, the concept of *Monte Carlo* method is applied for exploring the wide range of behaviors of the Softgoal Framework. 10,000 inferences are performed for each framework, resulted in a large variety of Softgoal behaviors and overall ratings (i.e. costs and benefits). Data obtained from *Monte Carlo* simulation is then summarized by the treatment learning tool TAR2, which is introduced in the next section.

4.5 Treatment Learning

Treatment learning is a data classifying technique that seeks a small number of control variables among large datasets. Much of its features are detailed in the related work chapter 2.6.

To allow TAR2 treatment learner to classify each "world" according to its cost and benefit score, a classification and ranking scheme is required to map ranges of costs/benefits to appropriate categories. Using the *Web Browser Selection* framework as example, each "world" is rated based on the following classification scheme: the

[7]The word "stochastic" means that it uses random numbers and probability statistics to obtain an answer.

	Benefit			
Cost	vlow	low	high	vhigh
zero	10	5	2	1
one	12	7	4	3
two	14	9	8	6
three	16	15	13	11

Table 4.1: class rankings for Web Browser Selection framework

ranges of benefit were sub-divided into four bands - vlow, low, high and vhigh (in increasing magnitude), whereas cost is sub-divided by its discrete values (from 0 to 3). Each band has roughly the same number of samples. Combining each band of the cost and benefit yields 16 classes (see table 4.1 for the ranking function described). This scheme takes account on both benefit and cost with slight preference towards lower cost. For example, Cost=zero,Benefit=high has a higher ranking than Cost=one,Benefit=vhigh. Base on the class ranking (table 4.1), TAR2 searches the datasets for the candidate attribute ranges, i.e., ranges that are more common in the highly ranked classes than the other classes[8]. In the domain of *Web Browser Selection*, such a candidate is the browser(s) that can lead to the best result for web-surfing. Knowing this range of attributes can greatly assist in making design decisions, as the space of considerations is narrowed down to only the attributes that would assert positive/negative impacts towards the system.

	Benefit					
Cost	≤3.2	≤9.6	≤16	≤22.4	≤28.8	
0	34					34
1	23	10				33
2	29	4				33
3						
total	86	14				100

0: no treatment

	Benefit					
Cost	≤3.2	≤9.6	≤16	≤22.4	≤28.8	
0						
1	45	21				66
2						
3	6	21	7	1		34
total	50	42	7	1		100

I: opera7=y

	Benefit					
Cost	≤3.2	≤9.6	≤16	≤22.4	≤28.8	
0						
1	32					33
2	37	26	5			67
3						
total	69	26	5			100

II: msie6=y

	Benefit					
Cost	≤3.2	≤9.6	≤16	≤22.4	≤28.8	
0						
1	34					34
2	38	25	4			66
3						
total	71	25	4			100

III: netscape6=y

Table 4.2: Sample Distributions on Web Selection Framework

With the class ranking shown in table 4.1, TAR2 analyzed the datasets of the *Web Browser Selection* framework and recommended *opera7* as the preferred web browser. The sample distributions are then organized into percentile matrix, as shown in figure

[8]By reversing the class ranking, an entirely different concern is addressed, in which TAR2 finds the attribute ranges more common for an undesirable system.

4.2, to verify TAR2's recommendation on improving the system behavior. Percentile matrices of other browser options are also presented as a comparison. From these distribution matrices, it is observed that using *opera7* results in the best overall cost/benefit distribution amongst other browsers, and hence the best web browsing experience.

A sample of treatment learning output can be found in appendix A.3.

4.5.1 Incremental Treatment Learning

Prior to the discussion on the case studies in the next section, we introduce the idea of *incremental treatment learning* strategy. To apply incremental treatment learning, a *Monte Carlo* simulator executes and generates datasets on the softgoal framework model. TAR2 condenses this dataset to a set of proposed treatments. After some discussions, users add the approved treatments as constraints for another round of *Monte Carlo* simulation. This cycle repeats until users see no further improvement. This simulation technique has been applied on the case studies in the following chapters.

Chapter 5

The *Keyword In Context* (*KWIC*) Framework

5.1 *KWIC* Domain Overview

Software architecture has emerged over time as a natural evolution of design abstractions, as engineers have searched for better ways to understand their software and new ways to build larger, more complex systems. In [122], Shaw and Garlan used the example problem proposed by Parnas[100] to show how different architectural solutions provide different benefits.

Parnas proposed the following problem in his 1972 paper: *"The KWIC [Keyword in Context] index system accepts an ordered set of lines; each line is an ordered set of words, and each word is an ordered set of characters. Any line may be circularly shifted by repeatedly removing the first word and appending it at the end of the line. The KWIC index system outputs a listing of all circular shifts of all lines in alphabetical order. "*

Shaw and Garlan outlined four architectural designs for the *KWIC* system in their book([122]). Two of the solutions are originated from Parnas' paper: one based on functional decomposition with shared access to data representations (*Shared Data*), and a second based on a decomposition that hides design decisions(*Abstract Data Type*). The third solution uses an implicit invocation style by Garlan, Kaiser and Notkin[49]. The fourth is a pipeline soultion inspired by the Unix index utility.

The *KWIC* framework presented in this case study is a graphical expression of the above architectural assessment constructed by Chung et.al[28].

5.2 Softgoal Framework Description

The framework shown in figure 5.1 defines the tradeoffs among *NFRs* and the architectural design alternatives within the *KWIC* domain. The top-level *NFR softgoals* - *Comprehensibility, Modifiability, Performance, Reusability* - are the quality requirements to be satisficed. The design alternatives - *Shared Data, Abstract Data Type, Implicit Invocation, Pipes and Filters*, populate the bottom-level as operationalizing softgoals. The sub-softgoals at the middle-level of the framework are obtained by decomposing the top-level *NFR softgoals*. In figure 5.1, for example, Modifiability considerations for a system are decomposed into concerns for data representation, processes and functions. The links from the operationalizing softgoals to *NFR softgoals* indicate the positive/negative impacts for each design alternatives had among quality factors. For instance, the implicit invocation regime makes an architectural design more extensible but requires more space, thus contributing to the corresponding softgoals (*Extensibility[function]* and *SpacePerformance[system]*) respectively as illustrated in figure 5.1. Arguments such as *expected size of data is huge* is used to justify the statement: *Pipe & Filter[Target System]* BREAKS(`--`) *Space Performance[System]*, and is represented by Claim softgoal (*Claim [c4]*).

In addition, the `!!` symbol associated with the *NFR softgoal Modifiability[Data Rep]* indicates that it is a high priority quality attribute to be satisficed. Several other attributes, such as *TimePerformance[System]*, *Deletability[Function]*, *Updatability[Function]*, also serve as critical factors for overall system quality.

5.3 Experiments and Result on the *KWIC* system

As detailed in section 5.1, the *KWIC* framework models the architectural design alternatives and quality attributes that are of business concern. The objective of this experiment is to look for design alternatives that would significantly impact the *KWIC* system among inconsistent knowledge such as trade-offs and beliefs.

In this section, we present two different logical interpretations applied onto the framework topology with respect to different business concerns. The rationale behind these interpretations and our observations are also discussed.

5.3.1 Experiment 1: Rigorous Quality Assurance

This experiment is intended for system designs where strict quality assurance is mandatory. The goal is to find out what design alternatives would optimize system quality attributes. As shown in figure 5.2.I, the *NFR softgoals* are combined with logic AND, meaning that all the softgoals have to be satisficed in order to satisfice their parent softgoal. To satisfice the *NFR softgoals* immediately above, the inference would try to satisfy the operationalizing softgoals as many as possible, and hence they are combined with logic ANY. The top-level *NFR softgoals* below the overall goal “goodness of system” are combined with logic ANY[1].

This above schematic is applied to the *KWIC* framework, and its implications is explained as follows: the *operationalizing softgoals*, namely the *sharedData[targetSystem]*, *abstractDatatype[targetSystem]*, *implicitInvocation[targetSystem]*, and *pipe&Filter[targetSystem]*, are attached to *modifiability[DataRep]* with ANY, meaning that the inference engine will try to prove as many of the operationalizing softgoals as it can to satisfice the *modifiability[DataRep]*. Satisficing *modifiability[System]* means all its precondition softgoals - *modifiability[Process]*, *modifiability[DataRep]*, and *modifiability[Function]* - are satisficed, for they are combined with an AND. As it is impossible to find a “world” where the *comprehensibility[System]*, *modifiability[System]*, *performance[System]* and *reusability[System]* are satisficed at the same time, they are chained with ANY (instead of AND) so that the top-level goal *goodness[System]* can be satisficed.

[1]ANY is used instead of AND to allow proper inference on this particular framework, as it is impossible to satisfice all the desired system quality attributes represented by the top-level nfr softgoals.

	Benefit					
Cost	vvlow	vlow	low	high	vhigh	vvhigh
zero	26	17	10	5	2	1
one	28	19	12	7	4	3
two	30	21	14	9	8	6
three	32	23	16	15	13	11
four	34	25	24	22	20	18
five	36	35	33	31	29	27

Table 5.1: class rankings for KWIC framework

Calculations on benefits and costs, as well as other parameters, are summarized in figure 5.3. Notice that deletability[System] in figure 5.2 does not associate with any *operationalizing softgoal*. Thus, it is assumed that certain operation is performed for its fulfillment, and the cost of this unknown operation is equal to 1. The class ranking function is described in figure 5.1.

Results from incremental treatment learning of the *KWIC* framework using the rigorous quality assurance settings are summarized in table 5.2.0, I, II and III. In order to clearly show how TAR2 condenses data ranges to improve the mean of the more preferred class, the results for each of the incremental process are presented as percentile matrix. Each cell is colored on a scale ranging from white (0%) to black (100%).

As this experiment represents the case where business users put more focus on software quality (i.e. benefit scores) verses costs, the benefit improvement is emphasized in the following discussion on treatment results.

Table 5.2.0 shows resulting data ranges when no constraint was imposed on the architectural design options and claims. Table 5.2.I, II, III shows the result of applying incremental treatments to figure 5.2.I. Note that as the key decisions accumulate, the variance in behavior decreased and the mean benefit scores improved. The mean benefit drifted from <5.5 before treatment (table 5.2.I) to <11 at treatment **Round 4** (table 5.2.III). Moreover, the number of samples fell into the high benefit ranges (<27.5 and <32) increased after treatment. Base on this result, developers may focus on key issues that would greatly impact overall software quality, such as whether or not to implement shared Data for the system. Alternatively, if in some dispute situation, an analyst could use *c2; c4; c5* as bargaining chips. Since these claims have little overall impact, our analyst could offer them in any configuration as part of some compromise deal in exchange for the other key decisions being endorsed.

As a last note on the result shown in table 5.2.I, II, III, we observed that cost increases as treatments accumulate. In other words, rigorous quality assurance costs the most but doubles the average benefit. With this new information, users are now informed enough to intelligently debate the merits of rigorous quality assurance over its cost.

	Benefit						
Cost	<5.5	<11	<16.5	<22	<27.5	<32	Total
0							
1	3.86						3.86
2	32.9	0.27					33.17
3	10.01	3.49	0.02				13.52
4	6.66	22.68	5.88	0.45	0.02		35.69
5	1.06	6.74	4.94	0.95	0.07		13.76
total	54.49	33.18	10.84	1.4	0.09		100

0. **Round 1** no treatment

	Benefit						
Cost	<5.5	<11	<16.5	<22	<27.5	<32	Tital
0							
1							
2	27.84						27.84
3	10.71	7.05	0.28				18.04
4	4.26	8.89	1.7	0.11	0.01		14.97
5	2.83	19.2	14.68	2.24	0.19	0.01	39.15
total	45.64	35.14	16.66	2.35	0.2	0.01	100

I. **Round 2** Constraint: sharedData of targetSystem=yes

	Benefit						
Cost	<5.5	<11	<16.5	<22	<27.5	<32	Total
0							
1							
2							
3	25.21	2.7					27.91
4	8.86	18.51	2.13	0.04			29.54
5	3.15	21.13	15.5	2.45	0.32		42.55
total	37.22	42.34	17.63	2.49	0.32		100

II. **Round 3** Constraint: implicitInvocation of targetSystem=yes,sharedData of targetSystem=yes

	Benefit						
Cost	<5.5	<11	<16.5	<22	<27.5	<32	Total
0							
1							
2							
3							
4	10.34	24.86	3.52	0.08		‘	38.8
5	4.68	31.01	21.51	3.64	0.34	0.02	61.2
total	15.02	55.87	25.03	3.72	0.34	0.02	100

III. **Round 4** Constraints: abstractDataType of targetSystem=yes, c3=yes, implicitInvocation of targetSystem=yes, sharedData of targetSystem=yes

Table 5.2: Percentage distributions of benefits and costs seen in 10,000 runs of *KWIC* Framework Variant I (figure 5.2.I) - Rigorous Quality Assurance

	Benefit						
Cost	<14.67	<29.33	<44	<58.67	<73.33	<87	Total
0							
1	3.06						3.06
2	6.99	10.36	0.62	0.02			17.99
3	4.18	25.26	11.92	1.41	0.1		42.87
4	1.72	12.88	13.26	3.61	0.31	0.04	31.82
5	0.27	1.57	1.77	0.59	0.06		4.26
total	16.22	50.07	27.57	5.63	0.47	0.04	100

0. **Round 1** no treatment

	Benefit						
Cost	<14.67	<29.33	<44	<58.67	<73.33	<87	Total
0							
1	12.34	0.02					12.36
2	9.18	18.66	1.77	0.05			29.66
3	4.71	26.32	16.1	3.24	0.24	0.02	50.63
4	0.5	3.29	2.8	0.7	0.06		7.35
5							
total	26.73	48.29	20.67	3.99	0.3	0.02	100

I. **Round 2** Constraints: c4=yes, pipeAndFilter of targetSystem=no

	Benefit						
Cost	<14.67	<29.33	<44	<58.67	<73.33	<87	Total
0							
1	11.89	0.01					11.9
2	9.03	18.6	1.95	0.04			29.62
3	4.77	26.17	17.05	3.47	0.29	0.01	51.76
4	0.38	2.77	2.91	0.56	0.1		6.72
5							
total	26.07	47.55	21.91	4.07	0.39	0.01	100

II. **Round 3** Constraints: c3=yes, c4=yes, pipeAndFilter of targetSystem=no

	Benefit						
Cost	<14.67	<29.33	<44	<58.67	<73.33	<87	Total
0							
1	20.34	0.05					20.39
2	8.38	28.29	4.62	0.18			41.47
3	1.84	15.66	15.84	4.27	0.48	0.05	38.14
4							
5							
total	30.56	44	20.46	4.45	0.48	0.05	100

III. **Round 4** Constraints: c2=yes, c3=yes, c4=yes, pipeAndFilter of targetSystem=no

Table 5.3: Percentage distributions of benefits and costs seen in 10,000 runs of *KWIC* Framework Variant II (figure 5.2.II) - Weak Quality Assurance

5.3.2 Experiment 2: Weak Quality Assurance

Often, when outside consultants are called in to offer a rapid assessment on how to improve a problematic project, they seek the fewest actions that offer the most benefit. To handle this situation, we defined a variation of the KWIC framework to simulate a weaker form of quality assurance. This assurance scheme is a simple modification in terms of its logical operations (i.e. swapping logical operators between softgoals). As shown in figure 5.2.II, the *NFR softgoals* are combined with logic OR, meaning that the parent softgoal is satisficed when one of its contributing softgoals is satisficed. Similarly, only one of the *operationalizing softgoal* is needed to fulfill the satisficing criteria of the *NFR softgoal* immediately above. The parameter configuration for inference is the same as of rigorous quality assurance (figure 5.3). In order to find the least preferred behavior, the class rankings are reversed as opposed to that of the rigorous quality assurance scheme (figure 5.1). Incremental treatment learning is performed on figure 5.2.II, and the results are shown in table 5.3.0, I, II and III.

The goal of this experiment is to determine what would negatively impact software quality in the most liberal quality assurance scheme. Comparing table 5.3.III (after treatments) with table 5.3.0 (before treatments), the number of samples fell into the lowest benefit range (<14.7) increased, which showed that benefit suffered as treatments accumulated. The results also suggest that, using the weaker form of quality assurance scheme, the overall software quality of the *KWIC* system suffers if *Pipe & Filter* is not implemented. Hence, users may center their discussions on the possibilities of implementing the Pipe & Filter option. Most importantly, high cost solutions can be avoided (note those results 35% over cost=3 in table 5.3.III) without degrading overall benefits.

We have presented the treatment learning results of the *KWIC* framework experiments on two distinct settings, and how these settings address different business concerns. Even though their implications are different, these experimental results demonstrated how TAR2 discovers a range of consistent behavior among the space of inconsistent information. Also, as incremental treatment is applied, variance is reduced and mean values of preferred classes improved.

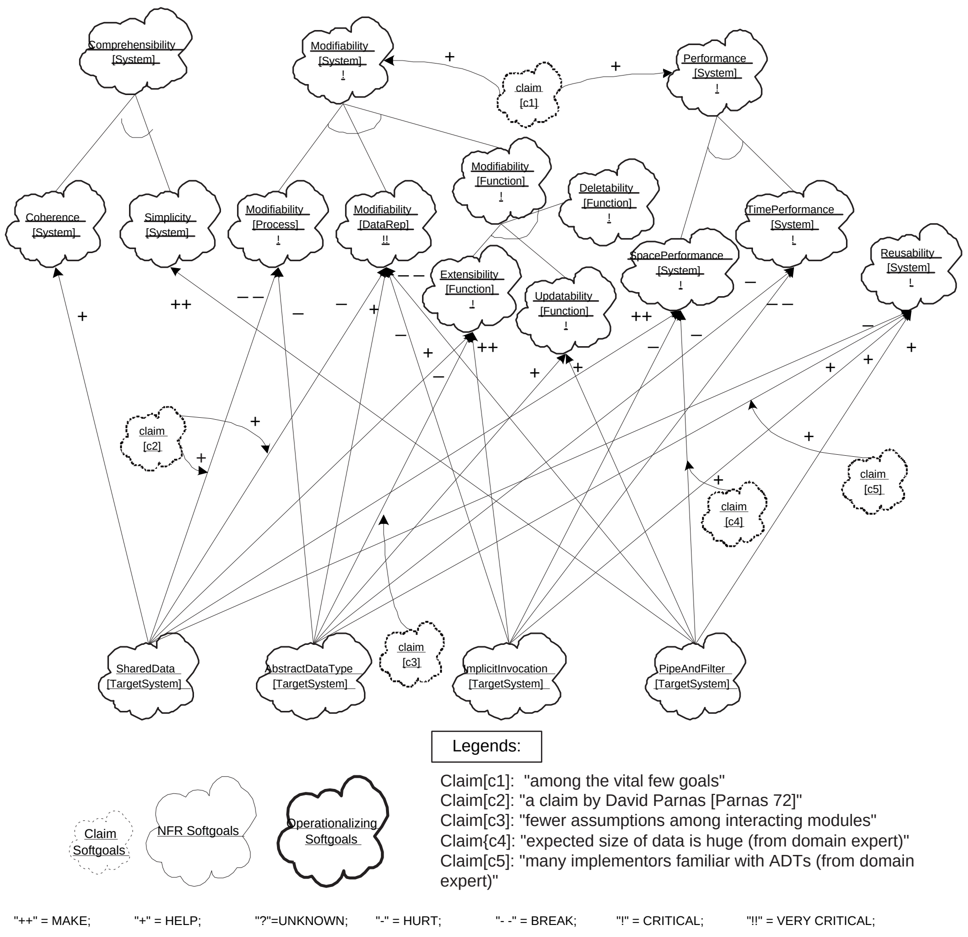

Comprehensibility [System]
Modifiability [System] !
claim [c1]
Performance [System] !
Coherence [System]
Simplicity [System]
Modifiability [Process] !
Modifiability [DataRep] !!
Modifiability [Function] !
Deletability [Function] !
Extensibility [Function] !
Updatability [Function] !
SpacePerformance [System] !
TimePerformance [System] !
Reusability [System] !
claim [c2]
claim [c3]
claim [c4]
claim [c5]
SharedData [TargetSystem]
AbstractDataType [TargetSystem]
ImplicitInvocation [TargetSystem]
PipeAndFilter [TargetSystem]
Legends:
Claim Softgoals
NFR Softgoals
Operationalizing Softgoals
Claim[c1]: "among the vital few goals"
Claim[c2]: "a claim by David Parnas [Parnas 72]"
Claim[c3]: "fewer assumptions among interacting modules"
Claim{c4]: "expected size of data is huge (from domain expert)"
Claim[c5]: "many implementors familiar with ADTs (from domain expert)"
"++" = MAKE; "+" = HELP; "?"=UNKNOWN; "-" = HURT; "- -" = BREAK; "!" = CRITICAL; "!!" = VERY CRITICAL;

Figure 5.1: the *KWIC* framework

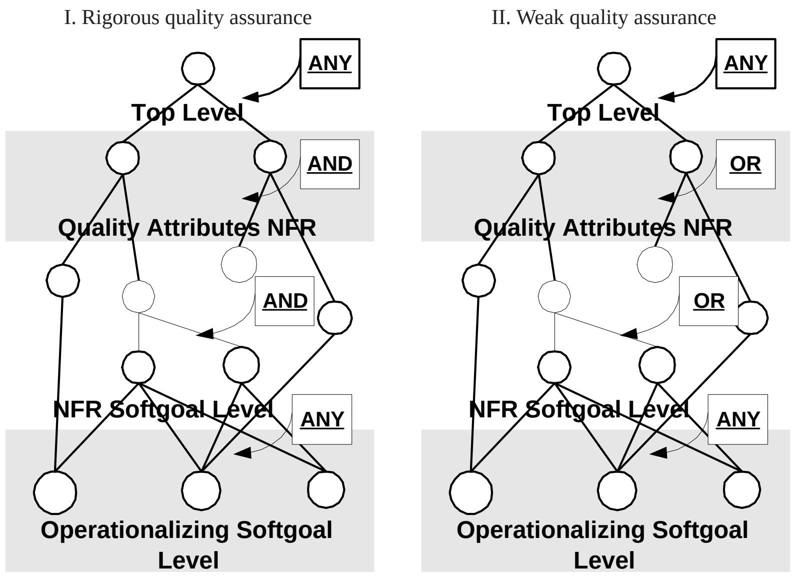

I. Rigorous quality assurance
II. Weak quality assurance
ANY
Top Level
AND
OR
Quality Attributes NFR
NFR Softgoal Level
Operationalizing Softgoal Level

Figure 5.2: KWIC Framework Variants

`<combine_logic>`	`<op>`	`<arithmetic[op]>`

all_of
any_of
any_one_of
	`rand`
	`rany`
	`ror`
		minimum
		summation
		maximum

`<contribution>`	`<value>[contribution]`	`<arithmetic>[contribution]`

helped
made
unhurt
unbroken
	mean=1.4
	mean=1.8
	mean=0.6
	mean=0.2
		multiply
		multiply
		multiply
		multiply

`<priority>`	`<value>[priority]`	`<arithmetic>[priority]`

veryCritical
critical
normal
	mean=2.0
	mean=1.5
	mean=1.0
		multiply
		multiply
		multiply

`<softgoal>`	`<softgoalType>[softgoal]`	`<cost>[softgoalType]`
operationalizing softgoal	any type	1

Figure 5.3: Parameter Settings for KWIC framework

Chapter 6

Choice of Software Organization Styles

6.1 Domain Overview

Organizational theory is a study of alternative (business) organization that are used to model the coordination of business stakeholders (individuals, organizations, or systems) to achieve common goal. According to Kolp, modeling the organizational context within which a software system will eventually operate is an important part of the requirement engineering process [70]. In addition to the classical architectural styles defined by *Shaw and Garlan*[122], *Castro et al* [25] developed *Organization Architecture Styles* based on concepts and design options from research in organization management. Some of these organization architecture styles include *flat structure, pyramid, joint venture, structure-in-5, takeover, co-optation, bidding*, etc. These architecture styles focus on business process and non-functional requirements of the application.

This chapter presents an investigation on *Software Organizational Styles* [70], particularly *joint venture, pyramid*, and *co-optation*, and their applicability on the architecture of an e-business system[1].

For web-based applications such as e-business, there are a variety of architectural styles available to assist in modeling. Some examples are *Thin Web Client*, *Thick Web Client*, *Dynamic* and *Web Delivery* [32]. These techniques focus on the technological aspects of software development, but do not emphasize on the many quality attributes that clients demand. By investigating the tradeoffs between software organization (architectural) styles and the demanded quality attributes, with the *Softgoal Simulation Tool*, we may reveal the key of optimizing the quality of this e-business application.

Before proceeding to next section on framework description, the organizational styles discussed in this chapter and their characteristics are summarized as follows:

Pyramid is a hierarchical organizational style where the actors[2] at the lower levels are under supervision from the actors at the higher levels. The crucial mechanism is direct supervision from the apex [71]. Immediate actors (such as manager and supervisor) route strategic decisions and authority from the apex to the operating level. This architectural style aids dynamicity, and lowers the cost for accomplishing evolvability and modifiability. It is suitable for deploying simple distributed systems, but not huge

[1]The softgoal framework used in this study is directly taken from [25]

[2]An actor is "a role that someone or something in the environment can play in relation to the business"[59]. An "individual actor" (also referred to as a "user") is defined to be an instance of class actor. Further, the same person (or other item) can assume more than one role.

ones like multi-agent systems involving in many kinds of agents.

Joint Venture, by definition, refers to a contractual agreement joining together two or more parties for the execution of a particular business undertaking, in which all parties agree to share the profits and losses. When applying this organizational style to software architecture, it means two or more actors sharing resources and responsibilities via *delegation of authority* to a specific *Joint Management* actor. Inside the *Joint Venture*, each actor manages itself and interacts directly with other actors exchange services and data. Actors outside the *Joint Venture* supply services or support tasks for the organization core.

Co-optation is the act of incorporating representatives of external system into the decision making and behavior of the organization. The purpose is to obtain resources and support from external source, while sacrificing confidentiality and authority. The organization, and its co-optated actors, have to reconcile with the policies of the external system.

6.2 Softgoal Framework Description

The softgoal framework for this study is shown in figure 6.1. At the top level, the overall "goodness" of the system consists of its *Availability*, *Security*, and *Adaptability*. *Availability* is further refined into *Integrity*, *Usability*, and *Response Time* of the system, in which *Integrity* is decomposed into *Accuracy* and *Completeness* of the system. *Security* is decomposed into system *Authorization*, *Confidentiality*, *External Consistency*, and that *Authorization* is further decomposed into *Identification*, *Authentication* and *Validation*. *Adaptability* of the system is refined into *Dynamicity* and *Updatability*. *Dynamicity* is refined into *Real-time Maintainability* and *Evolvability*, which is further refined into *Run-time Modifiability* and *Extensibility* of the system. These system quality attributes serve as the *non-functional requirement (NFR) softgoals* of the framework. The organization architectural styles, i.e. *Pyramid*, *Joint Venture*, and *Co-optation*, are the *operationalizing softgoals*; each of them asserts different *contributions*e.g. "++", "-") towards various *NFR softgoals* of the framework[3]

Unlike the *KWIC* framework presented in chapter 5, some *NFR softgoals* assert positive or negative effects to other *NFR softgoals*. For example, system *Authorization* improves *Accuracy* but harms system *Dynamicity*; system *Availability* benefits *Security*. These effects are treated as *contributions* in the framework and during automatic inference.

The claim *"external agents can acquire trusted information"* supports the interdependency-links "*Pyramid* helps system *Identification*", "*Pyramid* made system *Confidentiality*", and refutes the link "*Co-optation* hurts *External Consistency*". Each of the claims "Possible Conflicts" individually imposes positive contributions towards *Response Time* & *Security*, and *Adaptability* & *Security*.

For this particular e-business system, *Security* and data *Integrity* are two major concerns among all other quality attributes. These two *NFR softgoals* are assigned

[3]Other applicable organizational styles can be integrated with the adopted framework shown in figure 6.1. However, the framework is kept unmodified because the contributions of other styles towards many of the *NFR softgoals* are not known.

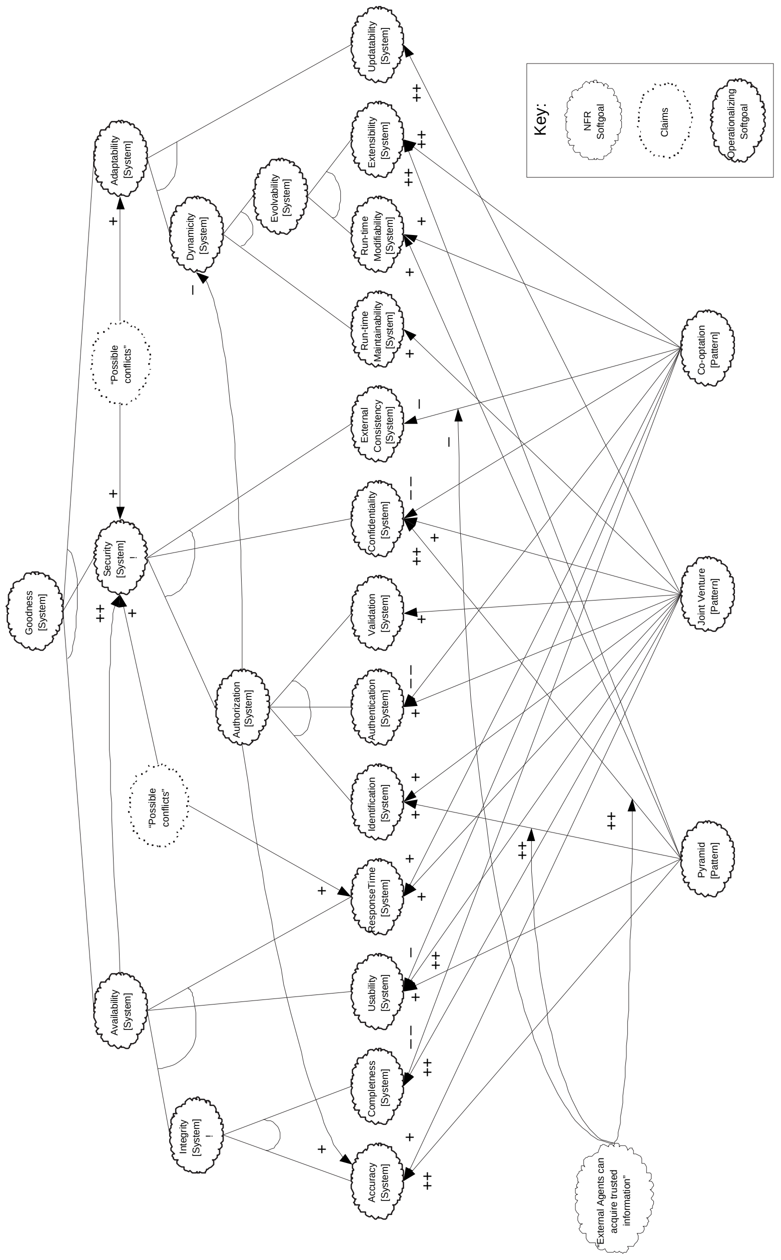


Figure 6.1: Architecture Patterns Framework

higher priorities than other softgoals [25].

6.3 Softgoal Framework Variants

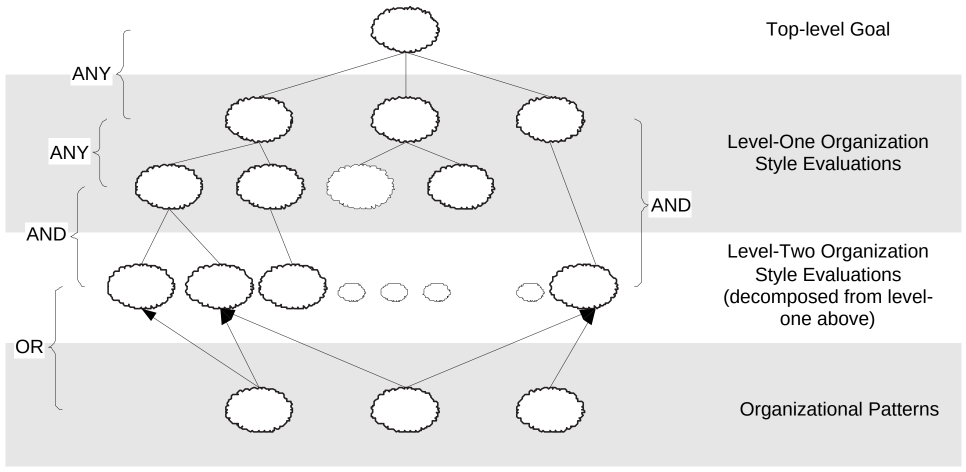

Top-level Goal
ANY
ANY
AND
OR
AND
Level-One Organization Style Evaluations
Level-Two Organization Style Evaluations (decomposed from level-one above)
Organizational Patterns

Figure 6.2: Software Organization Style Framework Variants 1: Rigorous Quality Assurance

Similar to the study on the *KWIC* framework (chapter 5), two variants were implemented for rigorous (figure 6.2) and weak (figure 6.3) quality assurance:

6.3.1 Rigorous Quality Assurance

Figure 6.2 shows the framework model for system demanded high quality. Developers of web-applications nowadays face tremendous competitions in the market, and they often have to excel in product quality in order to attract customers. This rigorous quality assurance configuration of the framework can be applied in such situation. As shown in the figure, the top-level goal is linked to the *Level-One Organization Style Evaluations*, i.e. *NFR softgoals*,with logic ANY. The softgoal hierarchies within this level are also linked with ANY, meaning that the inference mechanism will try to prove as many softgoal as possible to satisfice the parent softgoal. Each of the *Level-One Organization Style Evaluations* is AND-decomposed into its *level-two* softgoals for strictest satisficing requirements.

The *Level-One Organization Style Evaluations*, such as *Availability*, *Security* and *Adaptability*, are the more general quality attributes that the system must satisfice. These general *NFR softgoals* are refined into more specific system qualities, which are classified as *Level-Two Organization Style Evaluation* softgoals in figure 6.2. The original attempt of modeling the rigorous quality assurance scheme is to combine all *Organization Style Evaluations* with logic AND for the strictest constraints, but the

Softgoal Simulation Tool found no solution for such model. In order to achieve proper inference, the top two levels of logic compositions are replaced with ANYs.

The *operationalizing softgoals*, i.e. the organization architectural styles, are linked to the *Level-Two Organization Style Evaluation (NFR softgoals)* with OR, because only one of the three *Organization Styles* will be selected to implement the e-business application. To further comply with this setting, the *Softgoal Simulation Tool* will automatically deny the other *Organization Styles* once one is achieved during an inference. In other words, only one *Organization Style* can be assumed true in a "world".

The textual definition segment for this setting is shown below:

```
pick_one_in([
  pyramid of archPattern,
  jointVenture of archPattern,
  co_optation of archPattern
  ]).
```

For the complete textual definition of the *Software Organization Style*, please refer to appendix A.2.3.

6.3.2 Weak Quality Assurance

Contrary to section 6.3.1, the variant shown in figure 6.3 is configured to model the loosest quality satisficing requirement. Often times when time is a limiting factor, developers have to sacrifice robust quality control for meeting deadlines. In this case, managers may want to know which organization patterns will have the greatest impact on the loosest definition of product quality. The weak quality assurance scheme can be applied in such situation. As shown in the figure, all softgoals (*NFR softgoals*, *Operationalizing softgoals*, *Claim softgoals* are OR-decomposed to model the weak quality assurance scheme. Moreover, only one of the three *Organization Styles* (*Pyramid*, *Joint Venture*, *Co-optation*) is allowed for a particular "world"[4].

6.4 Experiments and Results

Both variants are analyzed with the *Softgoal Simulation Tool* with the parameter configurations and class ranking shown at tables 6.4 and 6.1 respectively. 10-way-cross-validation is performed to verify TAR2's treatment conclusions, and the unstable cases are discarded. Results are compiled into sample distribution tables and presented in tables 6.2 and 6.3.

6.4.1 Rigorous Quality Assurance

With the class ranking specified in figure 6.1, simulation is performed on figure 6.2 and treatment learning is applied to find treatments for the **most** and the **least preferred outcome**. After cross-validating the treatment recommendations, it is found that the

[4]For complete textual definition for this framework variant, please see appendix A.2.3

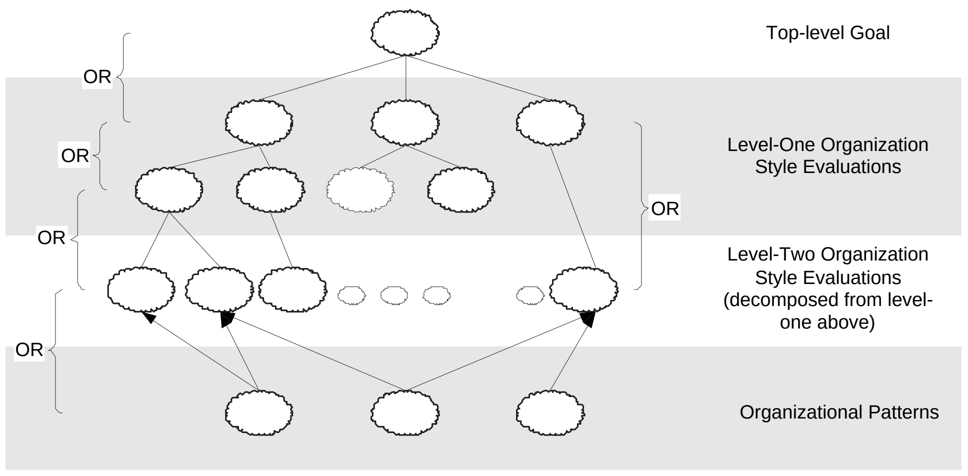

Top-level Goal
Level-One Organization Style Evaluations
Level-Two Organization Style Evaluations (decomposed from level-one above)
Organizational Patterns
OR
OR
OR
OR
OR

Figure 6.3: Software Organization Style Framework Variants 2: Weak Quality Assurance

	Benefit					
Cost	vvlow	vlow	low	high	vhigh	vvhigh
vvlow	26	17	10	5	2	1
vlow	28	19	12	7	4	3
low	30	21	14	9	8	6
high	32	23	16	15	13	11
vhigh	34	25	24	22	20	18
vvhigh	36	35	33	31	29	27

Table 6.1: class rankings for *Software Organization Style* framework

conclusions drawn by TAR2 on the **most preferred outcome** are too unstable, and thus excluded from result discussion. The sample distributions of the **least preferred outcome** before and after treatments are presented in table 6.2. Sample distribution tables (in percentile matrices) on choosing other *Organization Styles* are also presented for comparison.

According to TAR2's conclusion on the rigorous quality assurance framework variant, choosing *Co-optation* style will lead to a undesirable system, i.e. low benefit and high cost. Comparing table 6.2.I with other tables validates TAR2's result:

- Table 6.2.0 represents the system outcome before treatment, in which the cost and benefit score distributions are more favorable than the scores after treatment (table 6.2.I). As shown in the tables, samples at the lowest benefit category ($\leq$15.4) raised from 50% to 99%, and those at benefit scores $\leq$46.2 were nearly eliminated after treatment. Cost remains unchanged because all *Organization Styles* were configured to have the same cost, and only one style is picked in one

`<combine_logic>`	`<op>`	`<arithmetic[op]>`
all_of	`rand`	minimum
any_of	`rany`	summation
any_one_of	`ror`	maximum
`<contribution>`	`<value>` `[contribution]`	`<arithmetic>` `[contribution]`
helped	mean=1.4	multiply
made	mean=1.8	multiply
unhurt	mean=0.6	multiply
unbroken	mean=0.2	multiply
`<priority>`	`<value>` `[priority]`	`<arithmetic>` `[priority]`
critical	mean=2.0	multiply
normal	mean=1.0	multiply
`<softgoal>`	`<softgoalType>` `[softgoal]`	`<cost>` `[softgoalType]`
operationalizing softgoal	any type	random(1,2)

Figure 6.4: Parameter Settings for *Software Organization Style* framework

"world" during every softgoal simulations.

- Tables 6.2.II.a and 6.2.II.b show sample distributions when *Joint Venture* and *Pyramid* are chosen to implement the target e-business system. Both tables maintained more desirable benefit distributions than the one resulted from applying TAR2 recommanded treatment (table 6.2.I), which has the most samples at the lowest benefit category ($\leq$15.4).

6.4.2 Weak Quality Assurance

Using the same class ranking as the Rigorous Quality Assurance experiment, the weak quality assurance framework variant (figure 6.2 is analyzed with the *Softgoal Simulation Tool*, and the treatments found by TAR2 are 10-way-cross-validated to ensure their reliability. It is revealed that TAR2's recommended treatments for the **most preferred outcome** are too unstable to be accountable, and are therefore discarded in this study.

Shown in tables 6.3.0, 6.3.I and 6.3.II are the sample distributions for the **least preferred outcome** of the e-business application before and after incremental treatments. The sample distribution before treatments appeared to be very undesirable as far as benefit score is concerned[5]. The incremental treatments (as recommended by TAR2) result in similar undesirable sample distribution. Comparing TAR2's recommended treatment results with other alternatives (table 6.3.III.a. and 6.3.III.b.) shows that neither one of the *Organization Styles* can significantly worsen the target system under the weak quality assurance scheme.

[5]Notice that the cost factor for all the percentile matrices presented in this study are not of specific concern. It is because only one *Organization Style* can be assumed true at one time, and thus each inference on the framework will result in zero cost (no treatment) or the random cost for one *operationalizing softgoal* (after treatments).

	Benefit				
Cost	≤15.4	≤46.2	≤77	>77	total
<0.6					
<1.2	10	9	1		20
<1.8	30	27	3		60
<2.4	10	9	1		20
total	50	45	5		100

0. no treatment

	Benefit				
Cost	≤15.4	≤46.2	≤77	>77	total
<0.6					
<1.2	20				20
<1.8	59	1			60
<2.4	20				20
total	99	1			100

I. treatment learning conclusion: *Co_optation*

	Benefit				
Cost	≤15.4	≤46.2	≤77	>77	total
<0.6					
<1.2	6	12	2		20
<1.8	17	38	5		60
<2.4	6	12	2		20
total	29	62	8		100

II.a. *Joint Venture*

	Benefit				
Cost	≤15.4	≤46.2	≤77	>77	total
<0.6					
<1.2	17	2			20
<1.8	52	8			60
<2.4	18	2			21
total	88	12			100

II.b. *Pyramid*

Table 6.2: Sample Distributions on Organization Style Framework Rigorous Quality Assurance Variant (figure 6.2): **Least Preferred Outcome**

6.4.3 Discussion

This study has explored frameworks and settings which the *Softgoal Simulation Tool* failed to draw meaningful conclusions. As demonstrated in the experimental results on both framework variants, the model behaviors and scores can sometimes be too uncorrelated for TAR2 to draw stable conclusion. Moreover, in some situation, variations of model behaviors may yield the same score, which forbids TAR2 to produce meaningful treatment learning results that are relevant to the target system. A major difference between the *Organization Style* framework in this study and other frameworks is that there are only four variations of design decisions allowed[6]. More research is needed to be done to determine whether this is related to the inability of getting useful results with some variants of the *Organization Style* framework.

Despite its shortcoming, the *Softgoal Simulation Tool* can find useful conclusion with the rigorous quality assurance framework variant (figure 6.2) with its treatment learner TAR2. Comparisons with the sample distributions of alternatives further validated its correctness at pin-pointing key attributes that can drive the system towards some perferred mode.

[6]The four variations are (1)do only *Pyramid*; (2)only *Joint Venture*; (3)only *Co-optation*; or (4)nothing at all.

	Benefit			
Cost	≤5.6	≤16.8	>16.8	Total
<0.6	43			43
<1.2	12			12
<1.8	34	1		34
<2.4	11			11
total	99	1		100

0. no treatment

	Benefit			
Cost	≤5.6	≤16.8	>16.8	Total
<0.6	48			48
<1.2	10			10
<1.8	31			31
<2.4	10			10
total	99	1		100

I. **Round1**: Constraints: not "*external agents can aquire trusted information*", not *Co_optation*

	Benefit			
Cost	≤5.6	≤16.8	>16.8	Total
<0.6				
<1.2	20			20
<1.8	60	1		60
<2.4	20			20
total	99	1		100

II. **Round 2**: Constraints: "*external agents can aquire trusted information*", not *Co_optation*, *Joint Venture*, not *Pyramid*

	Benefit			
Cost	≤5.6	≤16.8	>16.8	Total
<0.6				
<1.2	20			20
<1.8	59			59
<2.4	20			20
<2				
total	100			100

III.**a**. Constraints: "*external agents can aquire trusted information*", *Co_optation*

	Benefit			
Cost	≤5.6	≤16.8	>16.8	Total
<0.6				
<1.2	20			20
<1.8	59			60
<2.4	21			21
<2				
total	99	1		100

III.**b**. Constraints: "*external agents can aquire trusted information*", *Pyramid*

Table 6.3: Sample Distributions on Organization Style Framework Weak Quality Assurance Variant (figure 6.3): **Least Preferred Outcome**

Chapter 7

Credit Card Authorization

This chapter presents an application on the softgoal simulation technique on a credit card information system. This study focuses on the performance requirements of the system during credit card authorization, particularly the response time and storage space requirement. The softgoal framework is first discussed, follow by the introductions of different framework variants. The experimental results of these variants are also presented.

7.1 Softgoal Framework Description

The functionalities of a bank's credit card system include transaction authorizations, account update, and card cancellation. Information on card holders and merchants is also maintained. To be competent in the market, a credit card system should be capable of providing fast and accurate services to its customers. For example, customers expect fast response time for purchase authorizations, and an effective space usage for storing customer information is always desired. Sales authorization is the dominant workload of a bank's credit card system, therefore it is vital to maintain its optimal performance. This study considered the performance requirement during credit card authorization, in terms of the response time and space required for storing sales information.

The softgoal framework presented here is a modification of the credit card system analysis found in [28]. It is a merge between the time requirement [28, fig 11.15] and space requirement [28, fig 11.18] frameworks. Some of its softgoals have been expanded according to the catalogue descriptions found in [28, Chapter 8] and [28, Chapter 9]. Moreover, some of the words used to describe the softgoals in the original graphs have been changed to reflect the correct logic.

Figure 7.1 shows the softgoal framework studied in this experiment. The system adopts a layered structure, and the softgoals are organized to reveal the implementation layers they belong to. At the top "catalog" level, the NFR softgoal *performance[authorize(card,money)]* is refined into time performance (*timePerformance[authorize(card,money)]*) and space performance (*spacePerformance[attribute(card)]*). Also shown at this level are response time (*responseTime[authorize(card,money)]*) which contributes to time performance, as well as main memory (*mainMemory[attribute(card)]*) and secondary storage (*secondaryStorage[attribute(card)]*), both contribute to space performance of the system.

The response time softgoal is further refined into code common for all card types (*responseTime[commonCode(authorize)]*) and code specific to certain card types (*responseTime[nonCommonCode(authorize)]*). Further down to *layer 3*, response time for common code execution (*responseTime[commonCode(authorize)]*) is decomposed into three subsoftgoals: *responseTime[retrieve(card*status)]*, represents the execution time for retrieving card status, *responseTime[access(card*remaining)]* for accessing remaining credits, and *responseTime[otherOperation(authorize)]*, for other operations. *ResponseTime[access(card*remaining)]* is broken down into *responseTime[retrieve(card*remaining)]* and *responseTime[update(card*remaining)]*, each represent the execution time for database retrieval and update. System developers have to decide which of these database transactions are to be performed first. Such design options are defined by the softgoals *performFirst[retrieve(card*remaining)]*, *performFirst[update(card*remaining)]* and *performFirst[retrieve(card*status)]*. [1]

On the other side of the softgoal framework are the storage components of the credit card system. The softgoal *secondaryStorage[attribute(card)]* is decomposed according to the type of card attributes, i.e. the card status (*secondaryStorage[status]*), credit remaining (*secondaryStorage[creditRemaining]*) and other card attributes (*secondaryStorage[otherAttrs]*). As there is no information regarding to the main memory space, the corresponding softgoal is not expanded.

Four other operationalizing softgoals are lied at the bottom of *layer 2*. They are *Few attribute per tuple(card*status)*, *earlyFixing[status]*, *Replicate derived attribute(card*remaining)*, and *lateFixing[otherAttrs]*. These implementation alternatives assert different impacts to the response time and storage space of the credit card system.

Fixing is the mapping of action or function requested to the instructions required to accomplish the action, and the mapping of information required to the corresponding data[125]. Early fixing (e.g. at compilation time) leads to a more responsive system, but it has a negative impact on storage requirement for the system considered in this experiment. Late Fixing, on the other hand, is the opposite of early fixing, and has a negative impact upon time performance[2]. Nonetheless, it affects positively to the space requirement for storing other attributes of the credit card system studied in this experiment.

Softgoals *Few attribute per tuple(card*status)* and *Replicate derived attribute(card*remaining)* deal with the representation of attributes in relational database. If a tuple contains few attributes (e.g. vertical splitting), the response time of data retrieval may be improved, depending on the way data is accessed. In the case of credit card authorization system, using "few attribute per tuple" to store card status improves time performance of the system. The effect of using "replicate derived attribute" on time performance, however, varies between database operations. It impacts positively on response time for retrieval, but negatively for update on credit card remaining.

The objective of this experiment is to access the performance of the system during

[1]Notice that the operational softgoals *performFirst of retrieve(card*status)*, *performFirst of update(card*status)* and *performFirst of retrieve(card*creditremaining)* cannot be performed simultaneously. In other words, only one of these softgoal can be true at a time, and the rest are set to be false. This is an attempt of simulating different job prioritization approaches, and their impact to the overall system. The softgoal analysis tools (lurch) have been modified to handle such setting.

[2]It affects responsive time because the code for linking the action/information to the function/data must be executed every time the action/information is used.[125]

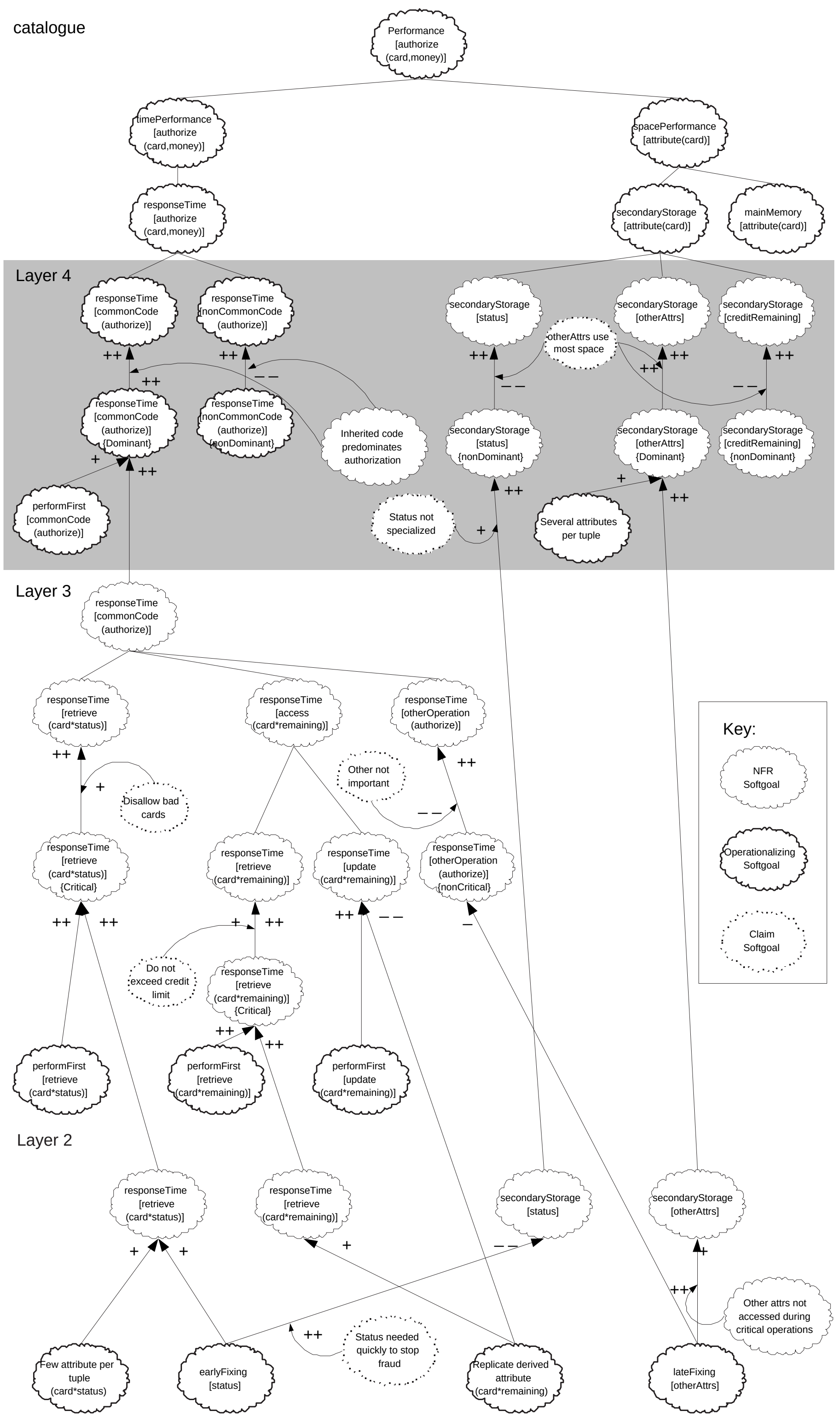

catalogue
Performance [authorize (card,money)]
timePerformance [authorize (card,money)]
spacePerformance [attribute(card)]
responseTime [authorize (card,money)]
secondaryStorage [attribute(card)]
mainMemory [attribute(card)]
Layer 4
responseTime [commonCode (authorize)]
responseTime [nonCommonCode (authorize)]
secondaryStorage [status]
secondaryStorage [otherAttrs]
secondaryStorage [creditRemaining]
otherAttrs use most space
responseTime [commonCode (authorize)] {Dominant}
responseTime [nonCommonCode (authorize)] {nonDominant}
Inherited code predominates authorization
secondaryStorage [status] {nonDominant}
secondaryStorage [otherAttrs] {Dominant}
secondaryStorage [creditRemaining] {nonDominant}
performFirst [commonCode (authorize)]
Status not specialized
Several attributes per tuple
Layer 3
responseTime [commonCode (authorize)]
responseTime [retrieve (card*status)]
responseTime [access (card*remaining)]
responseTime [otherOperation (authorize)]
Key:
NFR Softgoal
Operationalizing Softgoal
Claim Softgoal
Other not important
Disallow bad cards
responseTime [retrieve (card*status)] {Critical}
responseTime [retrieve (card*remaining)]
responseTime [update (card*remaining)]
responseTime [otherOperation (authorize)] {nonCritical}
Do not exceed credit limit
responseTime [retrieve (card*remaining)] {Critical}
performFirst [retrieve (card*status)]
performFirst [retrieve (card*remaining)]
performFirst [update (card*remaining)]
Layer 2
responseTime [retrieve (card*status)]
responseTime [retrieve (card*remaining)]
secondaryStorage [status]
secondaryStorage [otherAttrs]
Other attrs not accessed during critical operations
Status needed quickly to stop fraud
Few attribute per tuple (card*status)
earlyFixing [status]
Replicate derived attribute (card*remaining)
lateFixing [otherAttrs]

Figure 7.1: Credit Card Authorization Framework

	Benefit			
Cost	vlow	low	high	vhigh
vlow	10	5	2	1
low	12	7	4	3
high	14	9	8	6
vhigh	16	15	13	11

Figure 7.2: Class Rankings for Credit Card Authorization Framework

`<combine_logic>`	`<op>`	`<arithmetic[op]>`

all_of
any_of
any_one_of
	`rand`
	`rany`
	`ror`
		minimum
		summation
		maximum

`<contribution>`	`<value>`
	`[contribution]`
		`<arithmetic>`
		`[contribution]`

helped
made
unhurt
unbroken
	mean=1.4
	mean=1.8
	mean=0.6
	mean=0.2
		multiply
		multiply
		multiply
		multiply

`<priority>`	`<value>`
	`[priority]`
		`<arithmetic>`
		`[priority]`

veryCritical
critical
nonCriticall
dominant
normal
nonDominant
	mean=2.0
	mean=1.5
	mean=0.5
	mean=2.0
	mean=1.0
	mean=0.5
		multiply
		multiply
		multiply
		multiply
		multiply
		multiply

`<softgoal>`	`<softgoalType>`
	`[softgoal]`
		`<cost>`
		`[softgoalType]`

operationalizing *softgoal*	any type	1

Figure 7.3: Settings for Credit Card Authorization Framework

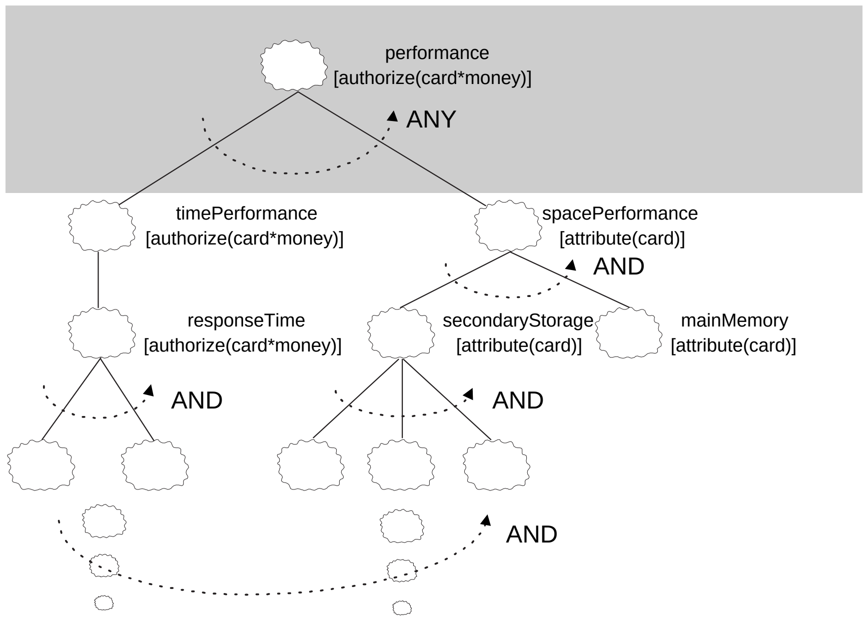


Figure 7.4: Credit Card Authorization Framework Variant 1

card authorization with respect to different system design approaches. Four framework variants, differed by their logic compositions, were created to model individual business concern. The structures of these framework variants are shown in figures 7.4 to 7.7. These frameworks are executed with the *Softgoal Simulation Tool*, using the parameter settings in tables 7.2 and 7.3. The next session presents the modeling intents, the experimental results and their implications for each of the constructed frameworks.

7.2 Framework Variant Descriptions and Experiment Results

7.2.1 Variant 1:

This framework variant, as shown in figure 7.4, is an attempt to model the credit card authorization system with the strictest performance requirement. The top level softgoal and the level below are connected with logic ANY, while the rest of the softgoals are all connected with logic AND. The original attempt was to specify all softgoal relations with logic AND, in order to find out the impacts of different system design approach under the strictest settings. In this configuration, however, every random walk in the graph would result in a cost of 0, meaning that no operational softgoal would satisfice the top level softgoal. Thus, framework inference constraints are relaxed by replacing the upper-most level ANDs with ANYs, while the rest of the framework are left unchanged. The *Softgoal Simulation Tool* executed this framework to find out the operations that would assert the most/least favorable effects on the system, under the strictest softgoal satisficing scheme. The compiled results are shown in tables 7.1 and 7.2.

	Benefit						
Cost	≤0.77	≤1.5	≤2.3	≤3.1	≤3.8	≤4.6	Total
0,1	16	28	6				50
2,3	3	5	1				8
4,5	12	21	4				37
6,7	1	2					4
Total	31	57	11	1			100

round0: no treatment

	Benefit						
Cost	≤0.77	≤1.5	≤2.3	≤3.1	≤3.8	≤4.6	Total
0,1	16	28	6				50
2,3	5	10	2				17
4,5	10	19	4				33
6,7							
Total	32	56	11	1			100

round1: not earlyFixing of status at layer(2)

	Benefit						
Cost	≤0.77	≤1.5	≤2.3	≤3.1	≤3.8	≤4.6	Total
0,1	15	28	5				49
2,3	5	10	2				17
4,5	11	19	4				34
6,7							
Total	32	57	11	1			100

round2: **round1** + status needed quickly to stop fraud at layer(2)

	Benefit						
Cost	≤0.77	≤1.5	≤2.3	≤3.1	≤3.8	≤4.6	Total
0,1	16	29	6				50
2,3	5	9	2				16
4,5	11	18	4				34
6,7							
Total	32	56	11	1			100

round3: **round2** + status not specialized at layer([4, prioritization])

	Benefit						
Cost	≤0.77	≤1.5	≤2.3	≤3.1	≤3.8	≤4.6	Total
0,1	33	56	10	1			100
2,3							
4,5							
6,7							
Total	33	56	10	1			100

round4: **round3** + not inherited code predominates authorization at layer([4, commonCode])

	Benefit						
Cost	≤0.77	≤1.5	≤2.3	≤3.1	≤3.8	≤4.6	Total
0,1	32	56	11	1			100
2,3							
4,5							
6,7							
Total	32	56	11	1			100

round5: **round4** + not otherAttrs use most space at layer([4, individualAttributes])

Table 7.1: Credit Card Authorization System Framework Variant 1: Incremental Treatments to make system better

Table 7.1 showed the incremental treatments obtained for making the system better. As shown in **round0** and **round1**, the number of samples at the highest cost categories (cost=6,7) were dropped from 4% to 0%. Benefit scores suffered correspondingly due to cost improvement. After applying all the recommended treatments (as shown in **round5**), all the samples saturated at the lowest cost category (cost=0,1) while the benefit scores have relatively small differences compared to the initial state (no treatment).

Table 7.2 are the results obtained from treatments that would make the system worse. By the end of performing all treatments (as shown in **round7**), all samples were driven to the highest cost category (Cost=6,7) with relatively small differences in benefit scores when comparing with the initial state (**round0**).

Note that, however, the graph used in the above analysis takes into account of the unknown factors, such as *mainMemory[attributes(card)]*. Under the current inference mechanism, this node is always assumed true and a random benefit weight is assigned for each simulation. Although this topology may be a better approximation to real world system, the TAR2 classifications and resulting recommendations may be skewed.

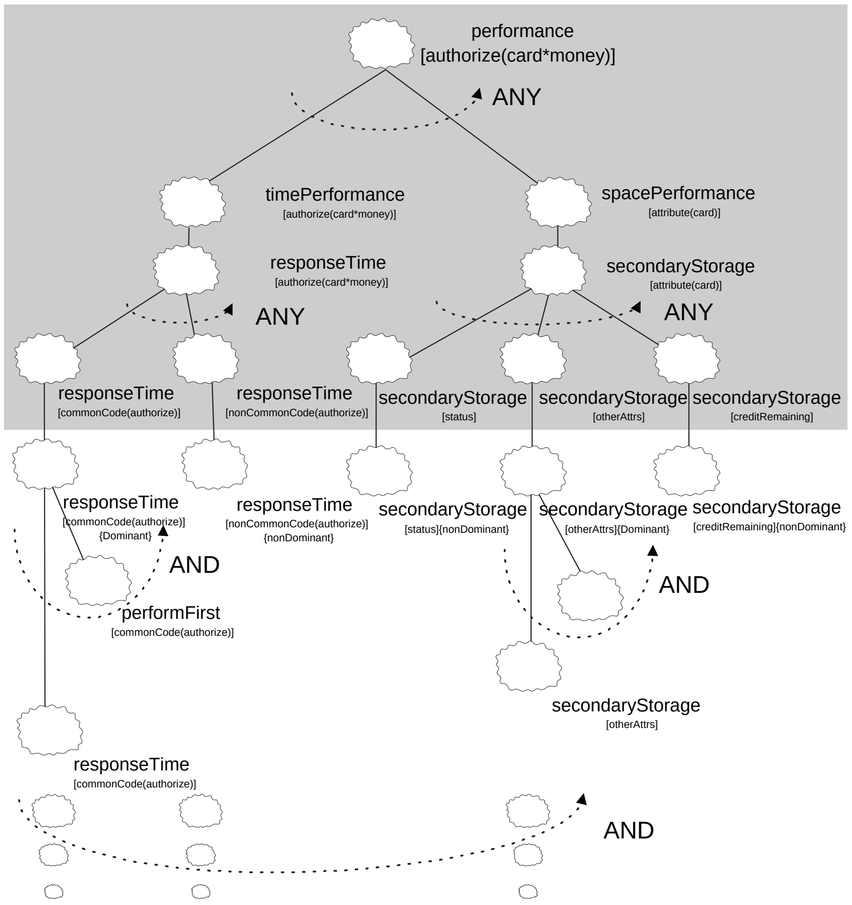


Figure 7.5: Credit Card Authorization Framework Variant 2

Therefore, the following experiments are performed with unknown nodes eliminated from the graph.

7.2.2 Variant 2:

The topology of the softgoal graph (shown in figure 7.5) is the same as that of **variant 1**, with the node *mainMemory of attributes(card)* eliminated. Also similar to **variant 1**, this graph is constructed to determine the most/least favorable operations and their effects on the system, using the strictest scheme to satisfice the performance requirement. The results from the experiments are shown in tables 7.3 and 7.4.

Table 7.3 showed the treatment recommended for a better system. It is found that applying *lateFixing of otherAttrs at layer(2)* (**round1**) immediately resulted in a decrease of samples in the lower-cost category (from 58% to 49%, a 13.3% drop) and an increase in the higher-cost category (from 42% to 51%, +18%). However, the resulting benefit scores are drifted towards higher values. Comparing the samples in round 0 and round 1 showed that the number decreased by 0.34% in the lower-benefit category [0,9.33] (from 95% to 94%), and increased by 50% in the high-benefit category (9.33,18.7] (from 4% to 6%). After four treatments (**round4**), the percentage of samples within the low-benefit category [0,9.33] dropped to 81% (-17%), while the percentage increased to 17%(+325%) for the high-benefit (9.33, 18.7]. Consequently, the number of lower-cost samples decreased by 12% to 51%. The samples in the higher-cost categories increased by 17%, and are concentrated at the Cost=6,7 categories. The result suggested that “lateFixing of otherAttrs at layer(2)” is the key to achieving the performance requirements of the credit card authorization system.

For treatments that could make the system worse, it is found that after one treatments (*disallow bad cards at layer([3, operationComponents])*), samples in the low-cost categories (Cost=0,1,2,3) dropped by 16% (from 58% to 49%), while those in higher-cost categories (Cost=4,5,6,7) raised by 21% (from 42% to 51%). The scores for each of the benefit category remained unchanged. From **round3** onwards, all samples were saturated at the lowest benefit score category (0, 9.33] while costs got higher. At **round6**, no sample could be found at the lower cost categories (Cost=0,1,2,3).

7.2.3 Variant 3:

As oppose to experiment **variant 1**, this framework variant (figure 7.6) imposes the weakest performance satisficing requirement; all the nodes below the top-level softgoals are connected with the logic OR. As business users often concern about cost-effectiveness, TAR2 analysis on such graph composition can be used to find the cheapest way to achieve the performance requirement. Moreover, it is useful in finding the operations that would harm the system even under the loosest performance satisficing requirements.

Table 7.5 shows the treatments that would harm the performance of the credit card authorization system. By applying *fewAttributePerTuple of (card*status) at layer([2, operationComponents])* in **round1**, the samples migrated to higher-cost categories while responded minimally to the benefit scores. At the end of performing all treatments (**round 9**), benefit scores improved significantly when comparing with the initial no-treatment table(**round0**): lower-benefit categories (0,207] reduced from 99% to 98% (-1%), while higher-cost categories (207,413] increased from 1% to 2% (+100%). Costs increased consequently due to increased benefit: data samples were saturated in the higher-cost regions (Cost=4,5) as opposed to that in round 0, where all samples were in the low cost categories (Cost=0,1,2,3). Note that many of TAR2’s recommendations were claims, thus users can focus on discussing the creditability of claims such as *disallow bad cards at layer([3, operationComponents]), do not exceed credit limit at layer([3, operationComponents]), status not specialized at layer([4, prioritization])...* etc, as the validity of these claims can significantly impact the performance requirement of the system.

For treatment recommendation to make system worse, table 7.6 shows the sample distributions with each treatment applied. By not implementing *lateFixing of otherAttrs* (shown in **round1**), the samples migrated to the lower cost regions (87% to 94% at

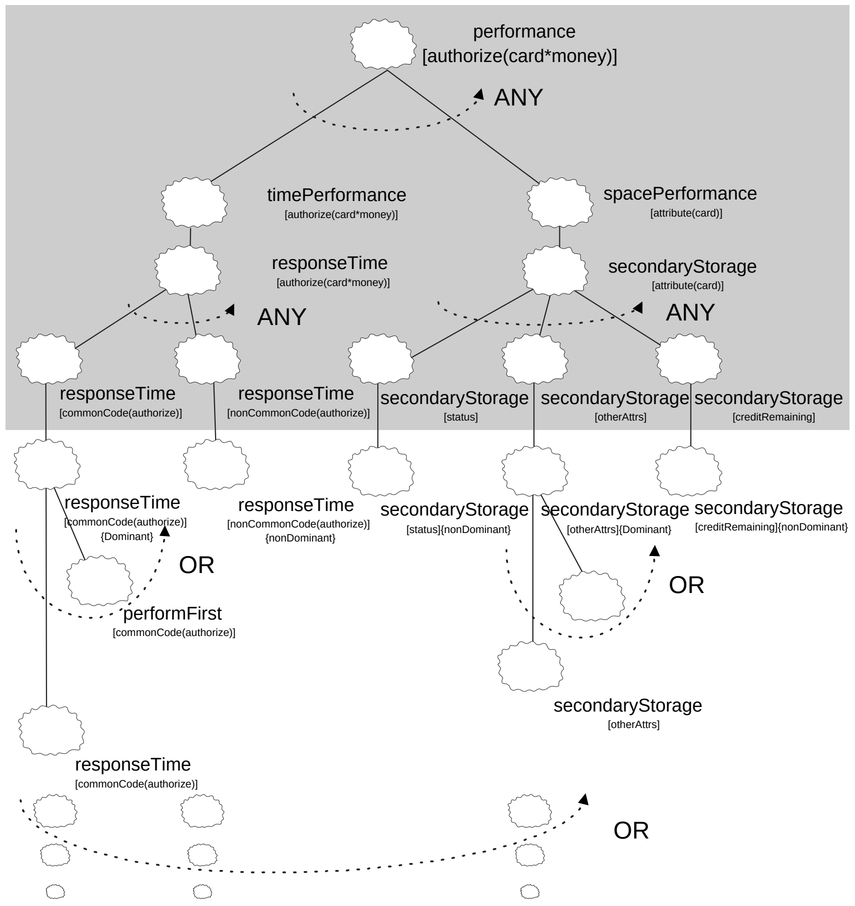


Figure 7.6: Credit Card Authorization Framework Variant 3

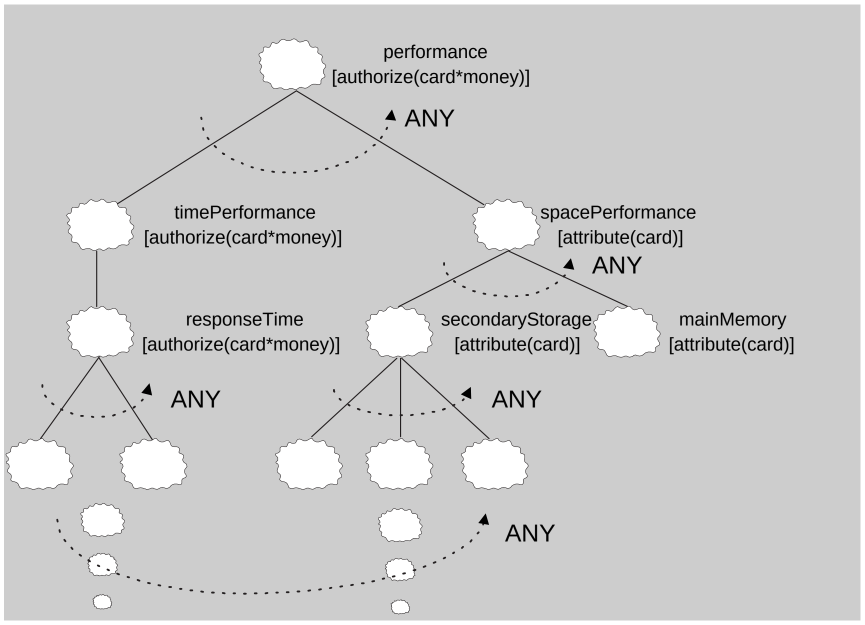


Figure 7.7: Credit Card Authorization Framework Variant 4

Cost=0,1; 13% to 6% at Cost=2,3), while has insignificant impact to the benefit scores. As treatment progressed, samples became more saturated at the low cost region. But by **round3**, all samples were saturated at the lowest benefit region. From this result, business users can now focus on validating claims such as *inherited code predominates authorization at layer([4, commonCode])*, as refuting such claim can negatively impact the system performance significantly.

7.2.4 Variant 4:

Figure 7.7 showed the structure of this framework variant. In this framework, all the logic relations between softgoals are ANYs, meaning that every subtrees were explored during each graph traversal. This logic configuration allows the most comprehensive investigation on the impacts of every softgoals towards the system performance. It is also useful in evaluating options when a balance emphasis on requirement and cost factor is preferred.

In achieving a better system according to this graph settings, table 7.7 presented the TAR2 recommendations and sample distributions. After applying two treatments (**round2**), samples fell within the low benefit category (0,574] increased (from 95% to 97%, +2%), while those within the higher-cost category dropped (from 6% to 3%, -37.6%; (574,11725]). However, samples in the lowest cost categories (Cost=0,1) increased (from 33% to 51%, +54%), and the samples in the highest cost categories (Cost=6,7) were reduced from 12% to 0% (-100%). While benefit worsen, cost improved in a more significant manner. This result implies that the performance improved in general, if *replicateDerivedAttribute of (card*creditRemaining)at layer([2, operationComponents])*' and *lateFixing of otherAttrs at layer(2)* are not implemented.

For softgoals that worsen system performance, table 7.8 showed that costs would be lowered if the claim *inherited code predominates authorization at layer([4, commonCode])* is refuted, as shown in **round1**. Such claim refutation eliminated all samples in the higher cost category. Nonetheless, all samples migrated towards the lowest benefit category. Note that almost all of the TAR2 recommendations were claims (*inherited code predominates authorization at layer([4, commonCode])*, *status needed quickly to stop fraud at layer(2)*, *status not specialized at layer([4, prioritization])*, *not otherAttrs use most space at layer([4, individualAttributes])*). Thus users may ponder the truthfulness of these claims as they can have significant impacts towards the performance of the target system.

			Benefit				
Cost	≤0.77	≤1.5	≤2.3	≤3.1	≤3.8	≤4.6	Total
0,1	11	19	3				33
2,3	92	14	3				25
4,5	9	17	3				30
6,7	4	7	1				12
Total	32	57	11	1			100

round0: no treatment

			Benefit				
Cost	≤0.77	≤1.5	≤2.3	≤3.1	≤3.8	≤4.6	Total
0,1	11	19	4				34
2,3	8	14	3				25
4,5	9	16	3				30
6,7	4	6	1				12
Total	32	56	11	1			100

round1:
performFirst of retrieve(card*creditRemaining) at layer([3, operationComponents])

			Benefit				
Cost	≤0.77	≤1.5	≤2.3	≤3.1	≤3.8	≤4.6	Total
0,1							
2,3	11	19	3				34
4,5	18	30	5				54
6,7	4	7	1				13
Total	32	56	11	1			100

round2: **round1** +
earlyFixing of status at layer(2)

			Benefit				
Cost	≤0.77	≤1.5	≤2.3	≤3.1	≤3.8	≤4.6	Total
0,1							
2,3	10	19	3				33
4,5	16	28	6				51
6,7	6	9	2				17
Total	32	57	11	1			100

round3: **round2** +
replicateDerivedAttribute of (card*creditRemaining)at layer([2, operationComponents])

			Benefit				
Cost	≤0.77	≤1.5	≤2.3	≤3.1	≤3.8	≤4.6	Total
0,1							
2,3							
4,5	16	29	6				51
6,7	16	28	5				50
Total	32	57	10	1			100

round4: **round3** +
severalAttributesPerTuple at layer([4, prioritization])

			Benefit				
Cost	≤0.77	≤1.5	≤2.3	≤3.1	≤3.8	≤4.6	Total
0,1							
2,3							
4,5	10	19	4				33
6,7	22	38	8				67
Total	32	56	11	1			100

round5: **round4** +
performFirst of commonCode(authorize)at layer([4, commonCode])

			Benefit				
Cost	≤0.77	≤1.5	≤2.3	≤3.1	≤3.8	≤4.6	Total
0,1							
2,3							
4,5							
6,7	32	56	12				100
Total	32	56	12	1			100

round6: **round5** +
lateFixing of otherAttrs at layer(2)

			Benefit				
Cost	≤0.77	≤1.5	≤2.3	≤3.1	≤3.8	≤4.6	Total
0,1							
2,3							
4,5							
6,7	33	56	11	1			100
Total	33	56	11	1			100

round7: **round6** +
fewAttributePerTuple of (card*status)at layer([2, operationComponents])

Table 7.2: Credit Card Authorization System Framework Variant 1: Incremental Treatments to make system worse

	Benefit						
Cost	≤9.33	≤18.7	≤28.0	≤37.3	≤46.7	≤55.0	Total
0,1	32						32
2,3	23	3					26
4,5	29						29
6,7	11	2					13
Total	95	4					100

round0: no treatment

	Benefit						
Cost	≤9.33	≤18.7	≤28.0	≤37.3	≤46.7	≤55.0	Total
0,1	33						33
2,3	13	3					16
4,5	26						26
6,7	22	3					25
Total	94	6					100

round1: lateFixing of otherAttrs at layer(2)

	Benefit						
Cost	≤9.33	≤18.7	≤28.0	≤37.3	≤46.7	≤55.0	Total
0,1	33						33
2,3	13	3					17
4,5	25						25
6,7	22	3					25
Total	94	6	1				100

round2: **round1** + other Attrs not accessed during critical operations at layer(2)

	Benefit						
Cost	≤9.33	≤18.7	≤28.0	≤37.3	≤46.7	≤55.0	Total
0,1							
2,3	46	3					49
4,5	9						9
6,7	39	2					42
Total	94	5	1				100

round3: **round2** + severalAttributesPerTuple at layer([4, prioritization])

	Benefit						
Cost	≤9.33	≤18.7	≤28.0	≤37.3	≤46.7	≤55.0	Total
0,1							
2,3	41	8	1				51
4,5							
6,7	41	8					49
Total	81	17	2				100

round4: **round3** + otherAttrs use most space at layer([4, individualAttributes])

Table 7.3: Credit Card Authorization System Framework Variant 2: Incremental Treatments to make system better

	Benefit						
Cost	≤9.33	≤18.7	≤28.0	≤37.3	≤46.7	≤55.0	Total
0,1	32						32
2,3	20	5	1				26
4,5	30						29
6,7	10	2					13
Total	91	7	1				100

round0: no treatment

	Benefit						
Cost	≤9.33	≤18.7	≤28.0	≤37.3	≤46.7	≤55.0	Total
0,1	33						33
2,3	11	5	1				16
4,5	34						34
6,7	14	2					17
Total	91	7	1				100

round1: disallow bad cards at layer([3, operationComponents])

	Benefit						
Cost	≤9.33	≤18.7	≤28.0	≤37.3	≤46.7	≤55.0	Total
0,1	33						33
2,3	11	5	1				17
4,5	33						33
6,7	14	2					17
Total	91	7	1				100

round2: **round1** +
replicateDerivedAttribute of (card*creditRemaining) at layer([2, operationComponents])

	Benefit						
Cost	≤9.33	≤18.7	≤28.0	≤37.3	≤46.7	≤55.0	Total
0,1	34						34
2,3	17						17
4,5	33						33
6,7	16						16
Total	100						100

round3: **round2** +
not lateFixing of otherAttrs at layer(2)

	Benefit						
Cost	≤9.33	≤18.7	≤28.0	≤37.3	≤46.7	≤55.0	Total
0,1							
2,3	51						51
4,5	33						33
6,7	16						16
Total	100						100

round4: **round3** +
earlyFixing of status at layer(2)

	Benefit						
Cost	≤9.33	≤18.7	≤28.0	≤37.3	≤46.7	≤55.0	Total
0,1							
2,3	34						34
4,5	50						50
6,7	16						16
Total	100						100

round5: **round4** +
performFirst of update(card*creditRemaining) at layer([3, operationComponents])

	Benefit						
Cost	≤9.33	≤18.7	≤28.0	≤37.3	≤46.7	≤55.0	Total
0,1							
2,3							
4,5	50						50
6,7	50						50
Total	100						100

round6: **round5** +
severalAttributesPerTuple at layer([4, prioritization])

Table 7.4: Credit Card Authorization System Framework Variant 2: Incremental Treatments to make system worse

	Benefit						
Cost	≤207	≤413	≤620	≤826	≤1033	≤1238	Total
cost=0	86						87
cost=2	13						13
cost=4							
cost=6							
Total	99	1					100

round0: no treatment

	Benefit						
Cost	≤207	≤413	≤620	≤826	≤1033	≤1238	Total
cost=0	43						43
cost=2	56						57
cost=4							
cost=6							
Total	99	1					100

round1: fewAttributePerTuple of (card*status)at layer([2, operationComponents])

	Benefit						
Cost	≤207	≤413	≤620	≤826	≤1033	≤1238	Total
cost=0	43						43
cost=2	56						57
cost=4							
cost=6							
Total	99	1					100

round2: **round1** + disallow bad cards at layer([3, operationComponents])

	Benefit						
Cost	≤207	≤413	≤620	≤826	≤1033	≤1238	Total
cost=0							
cost=2	89	1					89
cost=4	10						10
cost=6							
Total	99	1					100

round3: **round2** + performFirst of retrieve(card*status)at layer([3, operationComponents])

	Benefit						
Cost	≤207	≤413	≤620	≤826	≤1033	≤1238	Total
cost=0							
cost=2	48	1					49
cost=4	50	1					51
cost=6							
Total	98	2					100

round4: **round3** + replicateDerivedAttribute of (card*creditRemaining)at layer([2, operationComponents])

	Benefit						
Cost	≤207	≤413	≤620	≤826	≤1033	≤1238	Total
cost=0							
cost=2	49	1					50
cost=4	49	1					50
cost=6							
Total	98	2					100

round5: **round4** + do not exceed credit limit at layer([3, operationComponents])

	Benefit						
Cost	≤207	≤413	≤620	≤826	≤1033	≤1238	Total
cost=0							
cost=2	32	2					34
cost=4	65	1					66
cost=6							
Total	97	3					100

round6: **round5** + inherited code predominates authorization at layer([4, commonCode])

	Benefit						
Cost	≤207	≤413	≤620	≤826	≤1033	≤1238	Total
cost=0							
cost=2	33	1					34
cost=4	64	2					66
cost=6							
Total	97	3					100

round7: **round6** + status not specialized at layer([4, prioritization])

	Benefit						
Cost	≤207	≤413	≤620	≤826	≤1033	≤1238	Total
cost=0							
cost=2	45	3					48
cost=4	52						52
cost=6							
Total	97	3					100

round8: **round7** + not otherAttrs use most space at layer([4, individualAttributes])

	Benefit						
Cost	≤207	≤413	≤620	≤826	≤1033	≤1238	Total
cost=0							
cost=2							
cost=4	98	2					100
cost=6							
Total	98	2					100

round9: **round8** + earlyFixing of status at layer(2)

Table 7.5: Credit Card Authorization System Framework Variant 3: Incremental Treatments to make system better

	Benefit						
Cost	≤177	≤354	≤531	≤708	≤885	≤1061	Total
0,1	86	1					87
2,3	13						13
4,5							
6,7							
Total	99	1					100

round0: no treatment

	Benefit						
Cost	≤177	≤354	≤531	≤708	≤885	≤1061	Total
0,1	93	1					94
2,3	6						6
4,5							
6,7							
Total	99	1					100

round1: not lateFixing of otherAttrs at layer(2)

	Benefit						
Cost	≤177	≤354	≤531	≤708	≤885	≤1061	Total
0,1	92	1					93
2,3	6						7
4,5							
6,7							
Total	99	1					100

round2: **round1** + other not important at layer([3, operationComponents])

	Benefit						
Cost	≤177	≤354	≤531	≤708	≤885	≤1061	Total
0,1	100						100
2,3							
4,5							
6,7							
Total	100						100

round3: **round2** + not inherited code predominates authorization at layer([4, commonCode])

	Benefit						
Cost	≤177	≤354	≤531	≤708	≤885	≤1061	Total
0,1	100						100
2,3							
4,5							
6,7							
Total	100						100

round4: **round3** + not earlyFixing of status at layer(2)

Table 7.6: Credit Card Authorization System Framework Variant 3: Incremental Treatments to make system worse

	Benefit						
Cost	≤574	≤1150	≤1725	≤2299	≤2874	≤3448	Total
0,1	33						33
2,3	25						25
4,5	26	3					30
6,7	10	2					12
Total	95	5	1				100

round0: no treatment

	Benefit						
Cost	≤574	≤1150	≤1725	≤2299	≤2874	≤3448	Total
0,1	33						33
2,3	34						35
4,5	22	2					24
6,7	8	1					8
Total	97	3					100

round1: not replicateDerivedAttribute of (card*creditRemaining)at layer([2, operationComponents])

	Benefit						
Cost	≤574	≤1150	≤1725	≤2299	≤2874	≤3448	Total
0,1	51						51
2,3	16						16
4,5	30	2					33
6,7							
Total	97	3					100

round2: **round1** + not lateFixing of otherAttrs at layer(2)

Table 7.7: Credit Card Authorization System Framework Variant 4: Incremental Treatments to make system better

	Benefit						
Cost	$\leq$573	$\leq$1146	$\leq$1719	$\leq$2291	$\leq$2864	$\leq$3436	Total
0,1	33						33
2,3	25						25
4,5	26	3	1				30
6,7	10	2					12
Total	94	5	1				100

round0: no treatment

	Benefit						
Cost	$\leq$573	$\leq$1146	$\leq$1719	$\leq$2291	$\leq$2864	$\leq$3436	Total
0,1	66						66
2,3	34						34
4,5							
6,7							
Total	100						100

round1: not inherited code predominates authorization at layer([4, commonCode])

	Benefit						
Cost	$\leq$573	$\leq$1146	$\leq$1719	$\leq$2291	$\leq$2864	$\leq$3436	Total
0,1	65						65
2,3	35						35
4,5							
6,7							
Total	100						100

round2: **round1** + not earlyFixing of status at layer(2)

	Benefit						
Cost	$\leq$573	$\leq$1146	$\leq$1719	$\leq$2291	$\leq$2864	$\leq$3436	Total
0,1	67						67
2,3	33						33
4,5							
6,7							
Total	100						100

round3: **round2** + status needed quickly to stop fraud at layer(2)

	Benefit						
Cost	$\leq$573	$\leq$1146	$\leq$1719	$\leq$2291	$\leq$2864	$\leq$3436	Total
0,1	66						66
2,3	34						34
4,5							
6,7							
Total	100						100

round4: **round3** + status not specialized at layer([4, prioritization])

	Benefit						
Cost	$\leq$573	$\leq$1146	$\leq$1719	$\leq$2291	$\leq$2864	$\leq$3436	Total
0,1	100						100
2,3							
4,5							
6,7							
Total	100						100

round5: **round4** + not otherAttrs use most space at layer([4, individualAttributes])

Table 7.8: Credit Card Authorization System Framework Variant 4: Incremental Treatments to make system worse

Chapter 8

NASA IV&V Activity Prioritization - A Study on the SR-1 Project

This study demonstrates the capability of the *Softgoal Simulation Tool* to “learn” treatments from incomplete data. The overview and the goal of our study are outlined first, followed by details on the investigations and analysis process performed in this study. The treatment learning strategy is then defined, and result of our experimentations presented.

8.1 The SR-1 Project

This section begins with some basic facts and terminologies being used throughout this case study. Then the objective of our study is defined, and the issues on data unavailability addressed.

8.1.1 Introduction

The *NASA* SR-1 [33] program refers to the technologies involved in advance satellite design. It is under a contract from *NASA*’s *Space Launch Initiative (SLI)*. In this study, we focus on the software components of this technology, specifically on a list of *Catastrophic/Critical/High Risk (CCHR)* functions and the standards used for evaluating their risk and criticality.

The *NASA IV&V* Facility is one of the organizations performing *V&V* on software projects such as SR-1. Verification & Validation (*V&V*) is a system engineering process employing a variety of software engineering methods, techniques, and tools for evaluating the correctness and quality of a software product throughout its life cycle [137]. Independent *V&V* (*IV&V*) is performed by organization that are technically, managerially, and financially independent of the development organization.

The *Criticality Analysis and Risk Assessment (CARA)*[33] process is a quantitative analysis used by the *NASA IV&V* personnel to determine the appropriate scope of *V&V* on a project. *CARA* is based on the notion that a function that has high criticality and high risk requires more extensive inspections than a function of lower criticality/risk. The *CARA* analysis evaluates and rates the criticality and risk of software functions based on factors such as size and complexity of program code. These ratings are then used to calculate the *CARA* score for each of these functions. Appropriate *IV&V* resources are assigned based on these scores.

8.1.2 Overview and Objective of this study

Like many other companies, project management at *NASA IV&V* Facility has to deal with business issues such as delivery deadlines and resource allocations. It is every manager's goal to optimize resource usage, reduce project costs while meeting deadline dates. On the other hand, each *IV&V* analysis activities consume different degree of resources, and some of these activities perform better in the *V&V* process than the others. Finding out which of these *V&V* activities are more powerful, and less costly at the same time, would be helpful for project resource management and task prioritization. The objective of our study, therefore, is to look for the analysis activities that are more cost-effective than others.

In our study of the SR-1 project, we applied the softgoal framework idea to sketch out the inter-dependencies between *IV&V* Analysis Activities and the Criticality/Risk assessments on SR-1 functions, which are summarized as follows:

- **Criticality** and **Risk Criteria**, such as *Performance and Operation*, are viewed as the quality attributes which each validated SR-1 functions are trying to satisfice. They are the *NFR softgoals* in the SR-1 framework.
- **SR-1 Software Functions** (e.g., *vehicle management*) are also viewed as *NFR softgoals*, as no software validation process can guarantee these functions to be absolutely flawless. Nonetheless, their correctness can be sufficiently satisficed by applying *IV&V* analysis activities.
- ***IV&V* Analysis Activities** serve as the *operationalizing softgoals* in the framework.
- ***CARA* Ratings** (*catastrophic, critical, high, moderate, low*) defines the impacts of each function towards SR-1's overall criticality and risk factors upon failure. Thus, they become the inter-dependencies between **SR-1 Software Functions** and the **Criticality** and **Risk Criteria**.
- **Effectiveness of Analysis Activities** relates the ***IV&V* Analysis Activities** to the applicable **SR-1 Software Functions**.
- **Significance** of **SR-1 Functions** defines its *priority*.

To illustrate the above idea, a sample segment of the SR-1 framework is shown in figure 8.1. Figure 8.2 shows the criticality and risk ratings of the SR-1 functions, as well as their analysis levels resulting from the *CARA* process. The analysis activities these levels provide by *NASA IV&V* for requirements, design, code, and test are listed in figure 8.4. Each of these SR-1 functions maps to an analysis level, which is mapped to a set of activities assigned to analyze the function. For example, the function "'Target State Filter" (*f[tFilter]*) is assessed to be level "Limited"(*L*), thus the analysis activities:

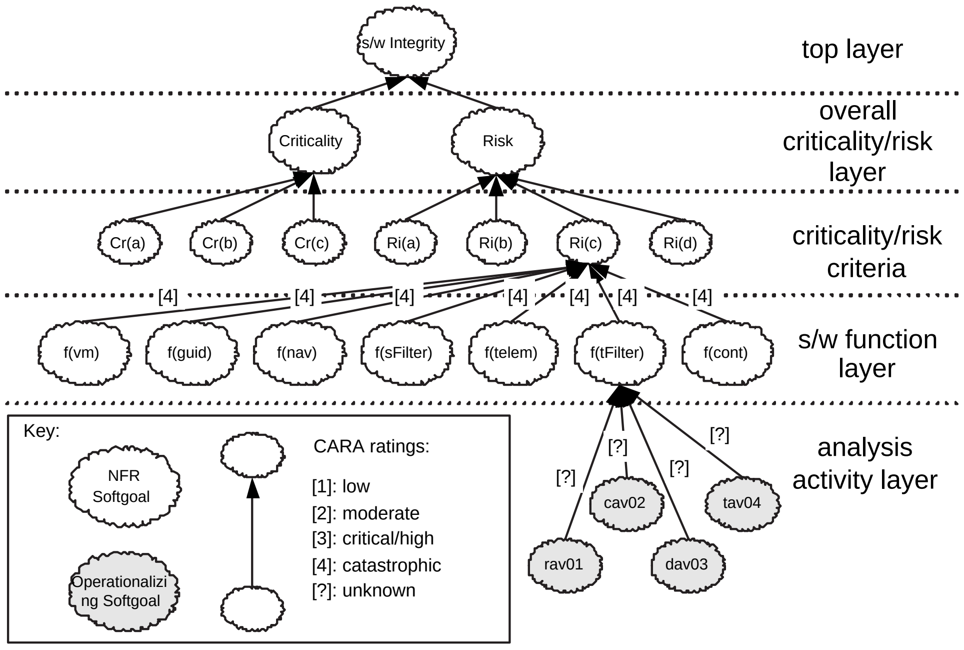


Figure 8.1: Segment of the SR-1 framework

rav01 - rav09, dav01 - dav09, cav01 - cav06, and *tav01 - tav07* are performed to validate the integrity of this function. Figure 8.3 shows the cost of these analysis activities in qualitative terms (*low, high* and *veryHigh*). The term "cost" is used to generalize factors such as time, manpower, and other *IV&V* resources.

As we proceeded on our analysis, we found that the SR-1 softgoal framework displayed typical features of real world systems - lack of domain knowledge and supporting data. First of all, we were unable to obtain any expert opinions regarding to the effectiveness of each analysis activities. Similarly, we have no information on which of the SR-1 function is more important than the others. Moreover, there is discrepancy in the scaling factors for cost calculations. These factors would defeat traditional quantitative approach in requirement analysis. With our proposed technique, however, we were able to perform inference and draw useful conclusion in spite of the lack of domain knowledge. The next session details how we handled the incomplete information in the SR-1 framework.

8.2 Softgoal Framework Construction

The list below describes the uncertainty factors in the SR-1 framework, translated into softgoal framework-specific terminologies:

- The contributions of *operationalizing softgoals* (analysis activities) to *NFR soft-*

Function	Criticality			Risk				
	Cr[a]	Cr[b]	Cr[c]	Ri[a]	Ri[b]	Ri[c]	Ri[d]	Level
f[vm]	4	4	2	3	2	3	3	F
f[guid]	4	4	2	3	2	2	3	F
f[nav]	4	4	2	3	2	2	3	F
f[sFilter]	4	4	2	3	1	2	3	F
f[telem]	4	1	3	3	1	3	2	F
f[tFilter]	4	1	2	3	2	2	3	L
f[cont]	4	4	1	2	1	2	1	L
f[cam]	2	1	1	3	3	2	3	B
f[exInf]	3	2	1	1	1	2	1	N
f[oReqm]	3	3	3	1	2	2	3	F
f[sMode]	3	1	3	3	2	2	3	L
f[aMode]	3	1	3	3	2	2	3	L
f[tMode]	3	1	3	3	2	2	3	L
f[dMode]	3	1	2	3	2	2	3	L
f[sbMode]	1	3	1	1	1	2	2	L
f[pMode]	3	1	1	1	1	2	2	N
f[aexInf]	3	1	1	2	1	2	1	N
f[comm]	3	1	1	1	1	3	1	N

Figure 8.2: CARA ratings on SR-1 software functions and corresponding Analysis Level

goals (SR-1 functions);

- The priorities of all *NFR softgoals*;
- The uncertainties intrinsic to the use of qualitative representations (e.g., "catastrophic *CARA*" ratings, "*veryHigh*" cost);

To allow inference while accounting for the above factors, we have made some assumptions and corresponding adjustments to the inference process. They are listed as follows:

- Analysis activities will always contribute positively to the integrity of SR-1 functions; i.e., all the *operationalizing softgoals* will either HELP or MAKE their parent *NFR softgoals*. To comply with this assumption, each positive contribution is randomly chosen to be either HELP or MAKE by our *Softgoal Simulation Tool* during inference;
- Performing *V&V* on either `catastrophically_rated` or `critically/highly_rated` functions is assumed to be always beneficial towards the overall safety of SR-1 software. Also, we assumed that doing *V&V* to `lowly_rated` SR-1 functions has negative impacts on the framework. The rationale of such assumption is that the additional workload may hinder job performance of *V&V* specialists, and hence out-weighed the gains;
- For the `moderately_rated` functions, the effect is assumed to be either positive or negative. Therefore, such rating is transformed to be either `lowly_rated` or `critically_rated` during inference;

Code	Level	Cost
Requirements Analysis Activities		
rav01	B,L,F,C	notHigh
rav02	B,L,F,C	notHigh
rav03	B,L,F,C	notHigh
rav04	B,L,F,C	notHigh
rav05	B,L,F,C	notHigh
rav06	B,L,F,C	notHigh
rav07	B,L,F,C	notHigh
rav08	B,L,F,C	notHigh
rav09	B,L,F,C	notHigh
rav10	F,C	veryHigh
rav11	F,C	veryHigh
rav12	F,C	veryHigh
rav13	F,C	veryHigh
rav14	C	extremelyHigh
Design Analysis Activities		
dav01	L,F,C	high
dav02	L,F,C	high
dav03	L,F,C	high
dav04	L,F,C	high
dav05	L,F,C	high
dav06	L,F,C	high
dav07	L,F,C	high
dav08	L,F,C	high
dav09	L,F,C	high
dav10	F,C	veryHigh
dav11	F,C	veryHigh
dav12	F,C	veryHigh
dav13	F,C	veryHigh
dav14	C	extremelyHigh

Code	Level	Cost
Code Analysis Activities		
cav01	L,F,C	high
cav02	L,F,C	high
cav03	L,F,C	high
cav04	L,F,C	high
cav05	L,F,C	high
cav06	L,F,C	high
cav07	F,C	veryHigh
cav08	F,C	veryHigh
cav09	F,C	veryHigh
cav10	F,C	veryHigh
cav11	F,C	veryHigh
cav12	C	extremelyHigh
Test Analysis Activities		
tav01	B,L,F,C	high
tav02	B,L,F,C	high
tav03	B,L,F,C	high
tav04	B,L,F,C	high
tav05	B,L,F,C	high
tav06	B,L,F,C	high
tav07	B,L,F,C	high
tav08	L,F,C	veryHigh
tav09	L,F,C	veryHigh
tav10	F,C	veryHigh
tav11	F,C	veryHigh
tav12	C	extremelyHigh

Figure 8.3: Analysis Activities, applicable Analysis Levels for SR-1's functions, and Cost

Code	Requirements Analysis Activity
Rav01	Verify documentation meets intended purpose, has appropriate detail and all necessary elements.
Rav02	Validate ability of requirements to meet system needs
Rav03	Verify Traceability to and from parent requirements
Rav04	Analyze data/adaptation requirement
Rav05	Analyze Testability, Qualification requirements
Rav06	Analyze Data FnotHigh, Control FnotHigh, moding and sequencing
Rav07	Assess development metrics
Rav08	Analyze development risks/mitigation plans
Rav09	Analyze Timing and Sizing requirements
Rav10	Review developer timing/sizing, loading engineering analysis
Rav11	Perform engineering analysis of key algorithms
Rav12	Review/use developer prototypes or dynamic models
Rav13	Develop alternative static representations (diagrams, tables)
Rav14	Develop prototypes or models. Perform timing/sizing/loading analysis. Apply formal methods
Code	Design Analysis Activity
Dal01	Verify documentation meets intended purpose, has appropriate detail and all necessary elements
Dal02	Validate ability of design to meet system needs
Dal03	Verify Traceability to and from requirements
Dal04	Analyze database design
Dal05	Analyze design Testability, Qualification requirements
Dal06	Analyze design Data FnotHigh, Control FnotHigh, moding, sequencing
Dal07	Analyze control logic, error/exception handling design
Dal08	Assess design development metrics
Dal09	Analyze development risks/mitigation plans
Dal10	Review developer timing/sizing, loading engineering analysis
Dal11	Perform design analysis of select critical algorithms
Dal12	Review/use developer prototypes or dynamic models
Dal13	Develop alternative static representations (diagrams, tables)
Dal14	Develop prototypes or models. Perform timing/sizing/loading analysis. Apply formal methods
Code	Code Aalysis Activity
Cal01	Verify documentation meets intended purpose, has appropriate detail and all necessary elements
Cal02	Verify Traceability to and from design
Cal03	Verify Architectural design compliance
	(structure, external I/O, and CSCI executive moding, sequencing and control)
Cal04	Verify supportability and maintainability
Cal05	Access code static metrics
Cal06	Verify CSU and CSC level logical structure and control fnotHigh
Cal07	Verify internal data structures and data fnotHigh/usage
Cal08	Verify error and exception handling
Cal09	Verify code and external I/O data consistency
Cal10	Verify correct adaptation data and ability to reconfigure
Cal11	Verify correct operating system and run time libraries
Cal12	Verify selected algorithms, correctness and stability under full range of potential input condition
	Verify code data compliance with data dictionary. Verify compliance with coding standards

Figure 8.4: Analysis Activities Keys to figure 8.3

Code	Test Analysis Activity
Tal01	Analyze System level verification requirements to verity that test definition, objectives, plans and acceptance criteria are sufficient to validate system requirements and operational needs associated with CCHR functions
Tal02	Verify Software Test Plan qualification testing methods and plans are sufficient to validate software requirements and operational needs
Tal03	Verify test cases traceability and coverage of software requirements, operational needs and capabilities
Tal04	Verify software STD test case definition inputs, expected results, and evaluation criteria comply with STP plans and testing objectives
Tal05	Analyze correct dispositioning of software test anomalies
Tal06	Validate software test results compliance with test acceptance criteria
Tal07	Verify trace and successful completion of all software test case objectives
Tal08	Verify ability of software test environment plans and designs to meet software testing objectives
Tal09	Verify regression tests are sufficient to determine that the software is not adversely affected by changes
Tal10	Analyze STD procedures for test setup, execution, and data collection; confirm procedures completely and correctly test referenced requirements, confirm inspection and analysis completely verifies referenced requirements
Tal11	Monitor execution of software testing as needed
Tal12	Analyze select CSC test plans, procedures, and results to verify adequate logic path coverage, testing of full range of input conditions, error and exception handling, key algorithm stability, and performance in compliance with the design.
	Perform life cycle IV&V on software test environment components

Figure 8.5: Analysis Activities Keys to figure 8.3 (continue)

- All the *NFR softgoals* have the same priorities;
- The numeric values of the qualitative terms fall within pre-defined ranges, as shown in figure 8.6;

Regarding to the cost discrepancy, two versions of cost functions were presented to us. Figure 8.7 describes these functions.

Other settings used in inferring SR-1 framework are shown in figure 8.6. The resulting SR-1 framework consists of 48 *operationalizing softgoal* nodes, 28 *NFR softgoal* nodes, and hundred of edges representing softgoal contributions. As this framework was too large to be legible, we were unable to present it in this paper.

After the SR-1 softgoal framework was constructed, we carried out our investigations using the *Softgoal Simulation Tool*. Details on the experimental settings (class ranking functions, logic configurations, etc) and treatment learning results are given in the next section.

8.3 Experiments and Results

The class ranking function used for the SR-1 framework is similar to that of the KWIC framework. The range of benefits and costs were sub-divided into four bands (from *vlow* to *vhigh*), and each band consists of roughly the same examples. Table 8.1 shows these class rankings. Two studies were conducted based on this ranking function:

`<combine_logic>`	`<op>`	`<arithmetic[op]>`
all_of	`rand`	minimum
any_of	`rany`	summation
any_one_of	`ror`	maximum
`<contribution>`	`<value>` `[contribution]`	`<arithmetic>` `[contribution]`
helped	mean=1.4	multiply
made	mean=1.8	multiply
catastrophically_rated	mean=1.9	multiply
critically_rated	mean=1.4	multiply
highly_rated	mean=1.1	multiply
lowly_rated	mean=0.4	multiply

Figure 8.6: Miscellaneous settings for SR-1 framework

version	1	2
`<notHigh>` (X)	$mean(X) = 1$	$mean(X) = mean(Y) * 0.7$
`<high>` (Y)	$Y = mean(X) * 10$	$mean(Y) = 2,$ $0 < Y < 10$
`<veryHigh>` (Z)	$Z = mean(Y) * 10$	$mean(Z) = Y * F,$ $mean(F) = 1.2; 1.1 \leq F \leq 1.7$

Figure 8.7: Two versions of cost function

- In the “MOST PREFERRED” study, TAR2 looks for behaviors that would contribute to the integrity of the SR-1 functions.
- Conversely, in the “LEAST PREFERRED” study, we reversed the order of class ranks to find treatments, which would assert negative impact to the framework.

We constructed and experimented on two variations of the SR-1 framework, which differed in their logical compositions[1]. Figure 8.8 showed the weakest quality assurance scheme build to represents realistic business situation, where analysts try to perform as many analysis activities to fulfill the integrity of a SR-1 function, and hence the overall software quality. In this framework, the analysis activities at the bottom

[1]The appropriateness of these framework variants in representing the real situations has yet to be determined by *NASA* experts.

	Benefit			
Cost	vlow	low	high	vhigh
vlow	10	5	2	1
low	12	7	4	3
high	14	9	8	6
vhigh	16	15	13	11

Table 8.1: class rankings for SR-1 framework

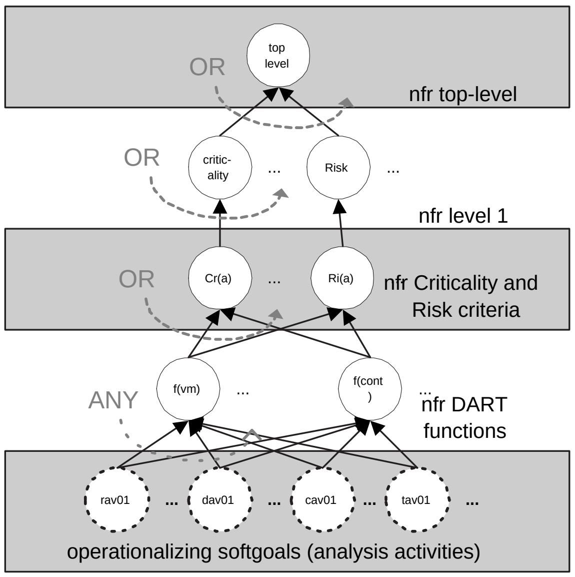


Figure 8.8: SR-1 framework: 1

level were chained with an ANY, and attached to their corresponding SR-1 functions. The SR-1 functions were bind to each of its *critical/risk* criteria with an OR, which were also bind to their upper-level softgoals with OR. For instance, *rav01-rav09* and *tav01-tav07* are chained with ANY and attached under SR-1 function *f[cam]*, meaning that the associated activities would be proven as many as it could during inference to satisfice *f[cam]*. Function *f[cam]*, together with other functions (such as *f[vm]*, *f[guid]*, *f[nav]* etc.) were chained to the *risk* criteria *Ri[d]* with logic OR. Therefore, satisficing one SR-1 function would be sufficient to satisfice the *risk* criteria. Similarly, the *risk* criteria *Ri[a]* to *Ri[d]* are combined with OR under the overall *risk* softgoal, which was connected to the top-level softgoal with an ANY. In other words, satisficing one *risk* criteria would be enough to satisfice the overall *risk* softgoal, which would lead to the top-level softgoal being satisficed.

To compare with figure 8.8, we proposed another SR-1 framework to represent rigorous quality assurance. For this, we derived two prototypes of such a framework, presented in figure 8.9 and 8.10. Figure 8.10 defined the strictest form of quality assurance, in which all the *NFR softgoals* (i.e., SR-1 functions, risk/criticality criteria, risk and criticality softgoals, and the top-level softgoal) are combined with AND, except for the bottom level *operationalizing softgoals* (i.e., analysis activities), which were combined with ANY. We reviewed this configuration and found it corresponded to a "utopia" model of rigorous quality assurance, since it is impractical in real world situation to fulfill the complete set of quality requirements as implied by this model. Because of its lack of practical applications, we abandoned this model and experimented on a more "pragmatic" rigorous quality assurance model, as shown in figure 8.9. In this

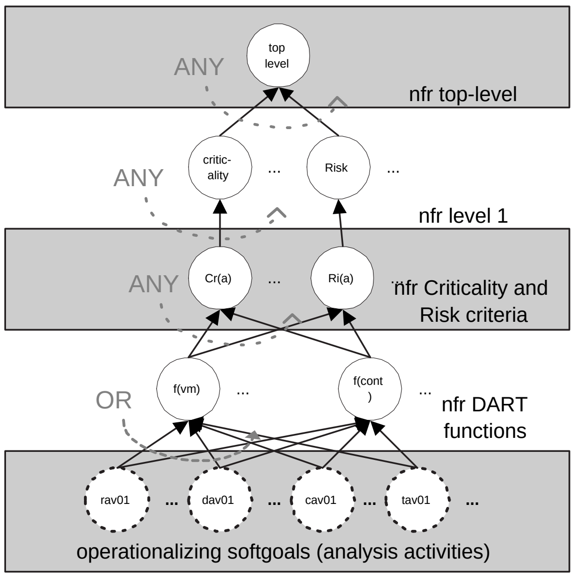


Figure 8.9: SR-1 framework: 2

“pragmatic” model, ANY were used to replace AND in the “utopia” model as a weaker form of conjunctive logical operator. Further, OR was used to replace ANY to further relax the satisficing constraint. Under this scheme, one satisficed analysis activities at the bottom level of the framework would be sufficient to fulfill the SR-1 function softgoal at the upper level. The inference engine would attempt to satisfice these SR-1 function softgoals as many as possible to satisfice the *criticality/risk* criteria softgoals, which would also be attempted as many as possible for the upper level overall *criticality/risk* softgoals. Finally, the top-level softgoal would be satisficed when either one or both of the overall *criticality/risk* softgoals were satisficed. We found this model to be a closer match to the real world business case; hence it was used in performing the experiments in our studies.

After the framework variations were defined, *Monte Carlo* simulations were applied to each variant twice, each time with different cost functions. After that, treatment learning is applied to each set of data in finding the most/least favorable treatments. The effects of these treatments are compared with the control situations (i.e., no treatment) in terms of costs and benefits. Results from the experimentations described above are presented in two groups: table 8.2, 8.3, and 8.4 for weak quality assurance; and table 8.5, 8.6, and 8.7 for “pragmatic” rigorous quality assurance.

Recall that the class ranking function defined in table 8.1 accounted on both benefit and cost with slight preference towards lower cost (e.g., *Cost=vlow, Benefit=high* has a higher ranking than *Cost=low, Benefit=vhigh*). Because of this setting, treatment learner would always recommend treatments that sacrifice a lower benefit for a lower cost. All out result sets reflected this particular class setting.

	Benefit				
Cost	vlow	low	high	vhigh	Total
vlow	34.15				34.15
low		4.02	6.26	5.58	15.86
high		6.2	9.98	8.82	25
vhigh		5.64	8.76	10.59	24.99
Total	34.15	15.86	25	24.99	100

Table 8.2: **SR-1 framework 1** Percentage distributions of $benefit$s and $cost$s seen in 10,000 runs of fig 8.8; no treatment

	Benefit				
Cost	vlow	low	high	vhigh	Total
vlow					
low		4.70	7.74	7.30	19.7
high		9.95	16.0	14.2	40.1
vhigh		9.05	14.1	17.0	40.1
total		23.7	37.8	38.5	100

Table 8.3: **More preferred system: SR-1 framework 1** Percentage distributions of $benefit$s and $cost$s seen after applying treatments ($tav09\ of\ tal = y$) for a more desirable system

	Benefit				
Cost	vlow	low	high	vhigh	Total
vlow					
low					
high		5.16	9.52	8.6	23.27
vhigh		17.32	26.9	32.51	76.73
Total		22.47	36.41	41.11	100

Table 8.4: **Less preferred system: SR-1 framework 1** Percentage distributions of $benefit$s and $cost$s seen after applying treatments ($cav10\ of\ cal = y$) for a less desirable system

	Benefit				
Cost	vlow	low	high	vhigh	Total
vlow	17.63	2.21	2.67	2.5	25.01
low	3.84	8.76	6.16	6.24	25
high	2.48	8	7.12	7.4	25
vhigh	1.06	6.03	9.05	8.85	24.99
Total	25.01	25	25	24.99	100

Table 8.5: **SR-1 framework 2** Percentage distributions of $benefit$s and $cost$s seen in 10,000 runs of fig 8.9; no treatment

	Benefit				
Cost	vlow	low	high	vhigh	Total
vlow	25.35	3.13	3.83	3.6	35.91
low	4.88	11.58	8.4	8.66	33.52
high	2.26	7.65	6.67	7.61	24.19
vhigh		1.56	2.16	2.32	6.38
Total	32.84	23.91	21.06	22.18	100

Table 8.6: **More preferred system: SR-1 framework 2** Percentage distributions of $benefit$s and $cost$s seen after applying treatments ($dav12\ of\ dal = n$) for a more desirable system

	Benefit				
Cost	vlow	low	high	vhigh	Total
vlow					
low	3.4	5.69	1.65	2.11	12.86
high	5.51	15.52	7.71	6.98	35.72
vhigh	4.13	12.95	16.71	17.17	50.96
Total	13.22	34.44	26.08	26.26	100

Table 8.7: **Less preferred system: SR-1 framework 2** Percentage distributions of $benefit$s and $cost$s seen after applying treatments ($cav07\ of\ cal = y$) for a less desirable system

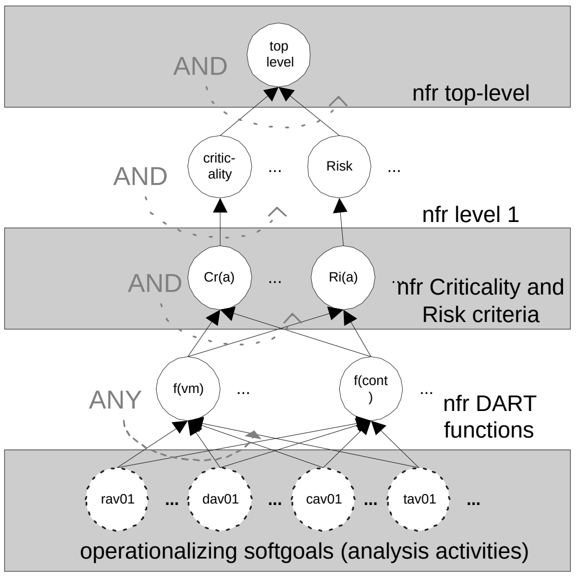


Figure 8.10: SR-1 framework: 3

Several features of these results deserve comment. Consider the results of the experiment on SR-1 framework 1 (weak quality assurance): firstly, treatment eliminated samples within the *Cost=vlow, Benefit=vlow range*, from >34% before treatment to 0% after treatment. Secondly, for the MORE PREFERRED system (where *tav09 of tal=y*), treatment learning drove the sample distributions towards a higher benefit range (*Benefit=high* and *Benefit=vhigh* occupied >76%, as opposed to <50% with no treatment). Third, the distributions of total benefit received after treatments were roughly the same for both MOST PREFERRED and LEAST PREFERRED system. However, the LEAST PREFERRED system (where *cav10 of cal=y*) suffered from very high cost (77% of the sample was classified as *Cost=vhigh*), compared to the MOST PREFERRED system (41% of the sample). This effect could be explained by the way the class ranking function was defined. Treatment learning for the MORE PREFERRED system would give recommendations that yield lower cost over lower benefit, whereas it would identify treatments that result in higher cost for the LEAST PREFERRED system.

The result for the “pragmatic” rigorous quality assurance (figure 8.9) scheme is presented in a similar fashion as the weak one described above. The experimental data is shown in table 8.5, 8.6 and 8.7. First of all, treatments for the MORE PREFERRED system (*dav12 of dal=n*) resulted in an increase of samples in the *Cost=vlow* and *Cost=low* range, from >50% to >69%. Nonetheless, the samples within the range *Benefit=vlow* and *Benefit=low* also increased (from 50% to <57%), a clear indication on the proportionality of cost and benefit. On the other hand, treatment for the LEAST PREFERRED system (*cav07 of cal=y*) resulted in very high cost (>85% in the

Cost=high and *Cost=vhigh* range) compared to that of no treatment ($<$50%). However, the benefit did not significant increase correspondingly, as $>$52% in the *Benefit=high* and *Benefit=vhigh* range after treatment, where it was $<$50% before treatment.

To conclude the SR-1 case study, the following points summarize the results and our observations:

- The use of logic components significantly affects the treatments that TAR2 recommends.
- Variation on cost functions was found to have no observable effect to the resulting treatment recommendations from all SR-1 framework variants studied.
- For all experimentations on the SR-1 framework, TAR2's treatment recommendations have been 10-way cross-validated[2] and their trustworthiness ensured. In other words, all of TAR2's treatment recommendation remained stable, in spite of the uncertainties and imprecise factors (as discussed in section IV B) lay within the framework.
- For the results shown in table 8.6 (on SR-1 framework 2), TAR2 suggested not to do *dav12* would be beneficial. Our treatment learning method can give advice on which activities not to be done in order to receive the most preferred outcome.

[2]Cross-validation is a method of estimating generalization error based on "re-sampling". In 10-way cross validation, the entire dataset is randomly split into 10 mutually exclusive subsets for approximately equal size. Each subset is being tested on the inducer that is trained by the other 9 subsets of data [112].

Chapter 9

Conclusion

Early prediction of software development is essential to organization in order to enhance software quality and avoid unacceptable risks. This thesis presented an investigation on software quality improvement, and proposed the *Softgoal Simulation Tool* which assists in decision making and quality evaluation during the very early life cycle of software development. It begins with the introduction and related work chapters, which identifies the problems with achieving quality software and the current state-of-the-art of system modeling and simulation. Then, the requirement analysis technique proposed in this paper is presented: first, a model is needed to capture the tradeoffs/synergy between software quality attributes and applicable design alternatives. We have adopted the softgoal framework by *Chung, Nixon, Yu and Mylopoulous* [28] in modeling such knowledge for analysis. Second, an inference engine is built to automatically execute the text-encoded softgoal framework. Thirdly, we incorporate *Monte Carlo* simulation to explore the wide range of behaviors in the model, and summarize these behaviors with a treatment-learning tool named TAR2. Five case studies have been presented through chapter 3 to chapter 8 to demonstrate the usage of the *Softgoal Simulation Tool* to model and simulate various systems. This thesis concludes with a summary of research results, contributions to the related software engineering field, and further research directions.

From what we have observed in our experiments, our simulation tool successfully discovered consistent behaviors within each framework despite various uncertainty factors, and provided treatment recommendations relevant to business concerns. As this approach does not require much concrete domain knowledge, time and expenses dedicated to data collection (e.g. appointments with domain experts, gathering surveys) can be minimized. Moreover, TAR2's treatment results pinpointed the most critical decision towards the problem domain, thus users can focus on this key issue and allot less time in discussing the non-critical ones.

So far there has not been any documented successful estimation of this modeling tool at the present moment; nonetheless, with the case studies presented throughout this thesis, it is believed that such tool can identify key attributes over a large space of options, and pinpoint the critical issues of the target modeling system. This research also developed a methodology of quality estimation when data is scarce and information is unavailable - a common situation during the early phase of software development.

9.1 Future Work

This research is one of the first to incorporate a graphical modeling method with a new data mining technique (TAR2) into an automated simulation tool. The requirement analysis technique presented in this thesis is in its preliminary state, and research is in progress. Much remains to be done to investigate the softgoal framework behavior and refine the proposed technique with more real-world case studies.

One way to understand the behavior of the softgoal framework is to perform sensitivity analysis. Sensitivity analysis is a very useful technique involving simulation models. It explores the effects on the key result variables, of varying selected parameters over a plausible range of values. This allows the modeler to determine the likely range of results due to uncertainties in key parameters. It also allows the modeler to identify which parameters have the most significant effects on results, suggesting that those be measured and/or controlled more carefully. Details are discussed in the related work section (chapter 2.7).

Sensitivity analysis is a huge field that includes many techniques and a variety of definitions. This research refers to the context of “true” sensitivity analysis, i.e. the analysis of the effect of extreme changes to a model using minimum and maximum values for each variable or changing the number of connections between components. These results may be summarized using mathematical regression. The goal is to design the least number of extreme experiments to sample the largest number of model variables. Despite its labor-intensiveness, such model validation process is unavoidable and should be performed onto the *Softgoal Simulation Tool*, specifically on the sensitivity to benefit and cost functions, as one of the future research activities.

Another research goal is to make the decision making more interactive to users. Currently the *Softgoal Simulation Tool* does not have *graphical user interface* support: execution commands are to be typed at prompt, and file destinations have to be changed at various text files. Developing a user interface may help improve interactivity by locating the necessary set up files and showing the analysis report in a graphical format. The analysis report may be implemented such that it shows the immediate change of the model behavior when user modifies input variables. Besides interactivity, a user interface also helps promote understandability and ease of development.

This research motivates work into quality evaluation at the very early stage of the software life cycle, when data is scarce and knowledge is uncertain. It also initiates further study into qualitative evaluation of system modeling and simulation, and how it is compared with quantitative methods. Though case studies are presented to demonstrate observations and results obtainable from this modeling simulation technique, more data needs to be collected, and more research needs to be performed to broaden these findings to additional application domains and establish a sound basis of software measurement.

Bibliography

[1] IEEE Std. 610.12-1990. Ieee standard glossary of software engineering terminology, ieee, 1990.

[2] IEEE Std. 982.1-1988. Ieee standard dictionary of measures to produce reliable software, ieee, 1988.

[3] IEEE Std. 982.2-1988. Ieee guide for the use of ieee standard dictionary of measures to produce reliable software, ieee, 1988.

[4] T. Abdel-Hamid and S. Madnick. *Software Project Dynamics: An Integrated Approach*. Prentice-Hall Software Series, 1991.

[5] S.T. Acuna and N. Juristo. Modelling human competencies in the software process. 2003.

[6] R. Ahmed, T. Hall, and P. Wernick. A proposed framework for evaluating software process simulation models. May 2003.

[7] M. Akhavi and W. Wilson. Dynamic simulation of software process models. In *In Proceedings of the 5th Software Engineering Process Group National Meeting*, Costa Mesa, California April 26 - 29, 1993. Software engineering Institute, Carnegie Mellon University.

[8] Apppled Decision Analysis. Dpl.

[9] D. Andreu, J. Pascal, and R. Valette. Fuzzy petri net-based programmable logic controller. *IEEE Transactions on System, Man, and Cybernetics*, 27(6):952–961, 1997.

[10] ANSI/IEEE. Ansi/ieee, standard glossory of software engineering terminology, 1991.

[11] S. Bandinelli, A. Fuggetta, and C. Ghezzi. Software processes as real-time systems: A case study using high-level petri nets. In *In Proceedings of the First European Workshop on Software Process Modeling*, Milan, 4/91, 1991. AICA Press.

[12] B. Boehm. Prentice-Hall, 1981.

[13] B. Boehm. Software risk management: Principles and practices. *IEEE Software*, pages 32–40, January 1991.

[14] G. Boetticher. An assessment of metric contribution in the construction of a neural network-based effort estimator. In *Second International Workshop on Soft Computing Applied to Software Engineering, Enschade, NL*, 2001. Available from: `http://nas.cl.uh.edu/boetticher/publications.html`.

[15] I. Bratko. *Prolog Programming for Artificial Intelligence. (third edition)*. Addison-Wesley, 2001.

[16] L.C. Briand, W.M. Thomas, and C.J. Hetsmanski. Modeling and managing risk early in software development. pages 55–65, 1993.

[17] Mozilla browser web site. `http://www.mozilla.org`.

[18] Neoplanet browser web site. `http://www.neoplanet.com`.

[19] Netscape browser web site. `http://www.netscape.com`.

[20] Opera browser web site. `http://www.opera.com`.

[21] Burgess, C.J. Dattani, I. Hughes, J.H.R. G. May, and Rees K. Using influence diagrams to aid the management of software change, requirements engineering. 6(3):173–182, 2001.

[22] GeNIe (Graphical Network Interface) by DSL. `http://www2.sis.pitt.edu/˜genie`.

[23] Tom Bylander, Dean Allemang, Michael C. Tanner, and John R. Josephson. The computational complexity of abduction. *Artificial Intelligence*, 49(1-3):25–60, 1991.

[24] T. Cao and A. C. Sanderson. A fuzzy petri approach to reasoning about uncertainty in robotic systems. In *Proc. of IEEE International Conference on Robotics and Automation*, pages 317–322, 1993.

[25] J. Castro, M. Kolp, and J. Mylopoulos. Towards requirements-driven information systems engineering: The tropos project, 2002. `citeseer.nj.nec.com/article/castro02towards.html`.

[26] Christie. Simulation in support of cmm-based proces improvement. *The Journal if Systems and Software*, 46:91–105, 1999.

[27] Steece Chulani, Boehm. Bayesian analysis of empirical software engineering cost models. *IEEE Transaction on Software Engineering*, 25(4), 1999.

[28] Chung, Nixon, Yu, and Mylopoulos. Kluwer Academic Publishers, 1999.

[29] W. Clancey, P. Sachs, M. Sierhuis, , and R. van Hoof. Brahms. Simulating practice for work systems design. In P. Compton, R. Mizoguchi, H. Motoda, and T. Menzies, editors, *Proceedings PKAW '96: Pacific Knowledge Acquisition Workshop*. Department of Artificial Intelligence, 1996.

[30] CNNfn. February 24, 1998, 1998. `cnnfn.com/digitaljam/9802/24/robber`.

[31] President's Informaiton Technology Advisory Committee. Information technoloty research: Investing in our future, feburary 24, 1999. `http://www.hpcc.gov/ac/report`.

[32] J. Conallen. *Building Web Applications with UML*. Addison Wesley Professional, 1999.

[33] TITAN System Corporation. *IV&V Catastrophic/Critical/High Risk Function List*, 2002.

[34] Geoff Coyle. Qualitative and quantitative modelling in system dynamics: some research questions. *System Dynamics Review*, 16(3):225–244, 2000.

[35] J.M. Crawford, A. Farquhar, and B.J. Kuipers. Qpc: a compiler from physical models into qualitative differential equations. AAAI/MIT Press, 1990.

[36] B. Curtis, M. Keelner, and J. Over. Process modeling. *Communications of the ACM*, 35(9):75–90, 1992.

[37] J. de Kleer and J.S. Brown. A qualitative physics based on confluences. *Artificial Intelligence*, 24:7–83, 1984.

[38] Dorothy Denning. Reading, MA: Addison-Wesley, 1998.

[39] J.A. Denton. Robust, an integrated software reliability tool. Technical report, 1997.

[40] A. Drappa and J. Ledewig. Quantitative modeling for the interactive simulation of software projects. *The Journal if Systems and Software*, 46:113–122, 1999.

[41] DSL. Smile (structural modeling, inference, and learning engine) by dsl. `http://www2.sis.pitt.edu/~genie`.

[42] M. Feather and T. Menzies. Converging on the optimal attainment of requirements, international conference on requirements engineering. 2002.

[43] E. Folmer and J. Bosch. Usability patterns in software architecture. *HCII 2003*, 2003.

[44] Kenneth D. Forbus. Qualitative process theory. *Artificial Intelligence*, 24:85–168, 1984.

[45] J. Forrester. *Industrial Dynamics*. Pegasus Communications, Waltham, MA, 1961.

[46] J. Forrester. Productivity Press, Portland, OR, 1969.

[47] D.W. Franke. Representing and acquiring telelogical descriptions. Detroit, Michigan, August 1989.

[48] E.G. Frankel. Kluwer Academic Publisher, 1988.

[49] D. Garlan, G.E. Kaiser, and D. Notkin. Using tool abstraction to compose systems. *IEEE Computer*, 25(6), 1992.

[50] G. Gigerenzer and D. G. Goldstein. Reasoning the fast and frugal way: Models of bounded rationality. *Psychological Review*, (103):650–669, 1996.

[51] R.L. Glass. Upper Saddle River, NJ: Prentice Hall PTR, 1998.

[52] A.L. Goel and K. Okumoto. Time-dependent error detection rate model for software reliability and other performance measures. *IEEE Transaction on Reliability*, R-28(3):206–211, 1979.

[53] D.E. Goldberg. *Genetic Algorithms in Search, Optimization, and Machine Learning*. Addison-Wesley, 1989.

[54] D. Harel. Statemate: A working environment for the development of complex reactive systems. *IEEE Transactions on Software Engineering*, 16(4):403–414, April 1990.

[55] H. Harrell, L. Ghosh, , and S. Bowden. *Simulation Using ProModel*. McGraw-Hill, 2000.

[56] the company that develops STELLA software High Performance Systems. `http://www.hps-inc.com`.

[57] Ying Hu. Treatment learning: Implementation and application. Master's thesis, Department of Electrical and Computer Engineering, University of British Columbia, Canada, 2003.

[58] W. Hunfeld. Dynasys. Available at: `http://www.uni-klu.ac.at/˜gossimit/sw/dynasys.zip`.

[59] I. Jacobson, M. Ericsson, and A. Jacobson. Addison-Wesley, Reading, Massachusetts, 1995.

[60] J. R. Josephson, M.C. Tanner, J. Smith, J. Svirbely, and P. Strohm. Red: Integrating generic tasks to identify red-cell antibodies. pages 524–531. IEEE Computer Society Press, 1985.

[61] Antonis C. Kakas, Robert A. Kowalski, and Francesca Toni. Abductive logic programming. *Journal of Logic and Computation*, 2(6):719–770, 1992.

[62] M. Kalos and P. Whitlock. *Monte Carlo Methods, Volume 1: Basics.* J. Wiley, New York, 1986.

[63] M.I. Kellner, R.J. Madachy, and D.M. Raffo. Software process simulation modeling: Why? what? how. *The Journal if Systems and Software*, 46:91–105, 1999.

[64] M.I. Kellner and D.M. Raffo. Measurement issues in quantitative simulations of process models. pages 33–37, Boston, MA, May 1997.

[65] D. Kelton, R. Sadowski, and D. Sadowski. Simulation with arena, second edition. 2002.

[66] Chris F. Kemerer. An empirical validation of software cost estimation models. *Communications of the ACM*, 30:416–429, 1987.

[67] T.M. Khoshgoftaar and J.C. Munson. Predicting software development errors using software complexity metrics. *IEEE Journal on Selected Areas in Communications*, 8(2):253–261, 1990.

[68] S. Kirkpatrick, C.D. Gelatt, and M.P. Vecchi. Optimization by simulated annealing. *Science*, 220(4598):671–680, 1983.

[69] J.P.C. Kliijnen. Sensitivity analysis and related analyses: a survey of statistical techniques. *Journal Statistical Computation and Simulation*, 57(1-4):111–142, 1987.

[70] M. Kolp, J. Castro, and J. Mylopoulos. Organizational patterns for early requirements analysis. *Tropos Working Paper, University of Toronto, Department of Computer Science*, February 2001.

[71] M. Kolp, P. Giorgini, and J. Mylopoulos. A goal-based organizational perspective on multi-agents architectures. August 2001.

[72] K. Konolige. Abduction versus closure in causal theories. *Articificial Intelligence*, 53(2-3):255–272, February 1992.

[73] B. J. Kuipers. Qualitative simulation. *Artificial Intelligence*, 29:289–338, 1986.

[74] S. Kusumoto, O. Mizuno, T. Kikuno, Y. Hirayama, Y. Takago, and K. Sakamoto. A new software project simulator based on generalized stochastic petri-net. pages 293–302, Baston, Massachusetts, May 1997. IEEE Computer Society Press, Los Alamitos, CA.

[75] A. Law and B. Kelton. *Simulation Modeling and Analysis*. McGraw Hill, 2000.

[76] J. Lee, K.F.R. Liu, and W.L. Chiang. A fuzzy petri net-based expert system and its application to damage assessment of bridges. *IEEE Transaction on Systems, Man and Cybernetics - part B: Cybernetics*, 29(3):350–370, 1999.

[77] M. M. Lehman and J.F. Ramil. The impact of feedback in the global software process. *The Journal if Systems and Software*, 46:123–134, 1999.

[78] C. Lin, T. Abdel-Hamid, and J. Sherif. Software-engineering process simulation model. *TDA Progress Report*, pages 42–108, 1992.

[79] Raymond J. Madachy. System dynamics moeling of an inspection-based process. pages 376–386, 1996.

[80] Computer World Magazine, September 1999.

[81] R. Martin and D. M. Raffo. A model of the software development process using both continuous and discrete models. *International Journal of Software Process Improvement and Practice*, June/July 2000.

[82] H. Matheson. Influence diagram. *The Principles and Applications of Decision Analysis; Strategic Decisions Group, Menlo Park, California, USA*, 2, 1984.

[83] I.R. McChesney. Toward a classification scheme for software process modelling approaches. *Information and Software Technology*, 37(7):363–374, 1995.

[84] Kiper Menzies. Better reasoning about software engineering activities. *IEEE International Conference on Automated Software Engineering*, 2001.

[85] T. Menzies. Practical machine learning for software engineering and knowledge engineering. In *Handbook of Software Engineering and Knowledge Engineering*. World-Scientific, 981-02-4973-X, December 2001.

[86] T. Menzies, E. Chiang, M. Feather, Y. Hu, and J.D. Kiper. Condensing uncertainty via incremental treatment learning. *Annals of Software Engineering; special issue on Computational Intelligence*, 2002.

[87] T. Menzies and Y. Hu. Constraining discussions in requirements engineering via models. In *International Workshop on Model-based Requirements Engineering*, 2001.

[88] T. Menzies and Y. Hu. Data mining for very busy people. *IEEE Computer*, 2003.

[89] T. Menzies and J.D. Kiper. How to argue less. In *Submitted to the ACM CIKM 2001: the Tenth International Conference on Information and Knowledge Management*, 2001. Available from `http://tim.menzies.com/pdf/01jane.pdf`.

[90] T. Menzies and H. Singh. How AI can help SE; or: Randomized search not considered harmful. In *AI'2001: the Fourteenth Canadian Conference on Artificial Intelligence, June 7-9, Ottawa, Canada*, 2001. Available from `http://tim.menzies.com/pdf/00funnel.pdf`.

[91] P. Mi and W. Scacchi. A knowledge-based environment for modeling and simulation software engineering processes. *IEEE Transactions on Knowledge and Data Engineering*, pages 283–294, September 1990.

[92] K. Moller. *Increasing Software Quality by Objectives and Residual Fault Prognosis*. Chapman and Hall, 1991.

[93] T. Mukhopadhyay, S.S. Vicinanza, and M.J. Prietula. Examining the feasibility of a case-based reasoning tool for software effort estimation. *MIS Quarterly*, pages 155–171, June 1992.

[94] T. Murata. Petri nets: Properties, analysis and application. volume 77, pages 541–580, April 1989.

[95] J.D. Musa. Operational profiles in software-reliability engineering. *IEEE Software*, 12(1):14–22, March 1993.

[96] J.D. Musa and W.W. Evernett. Software-reliability engineering: Technology for the 1990s. *IEEE Software*, 12:36–43, November 1990.

[97] J.D. Musa, A. Iannino, and K. Okumoto. McGraw-Hill, New York, 1987.

[98] Roger Naill. Managing the discovery life cycle of a finite resource: A case study of u.s. natural gas. Master's thesis, Alfred P. Sloan School of Management. Massachusetts Institute of Technology. Cambridge, MA 02139, 1972.

[99] Peter (ed.) NAUR. Revised report on the algorithmic language algol 60. *Communications of the ACM*, 3(5):229–314, May 1960.

[100] D. Parnas. On the criteria to be used in decomposing systems into modules. *Communications of the ACM*, 5(12):1053–1058, 1972.

[101] Simon Parsons. Institute of Technology, Massachusetts, 2001.

[102] G. Paul. Approaches to abductive reasoning - an overview, 1993.

[103] C.S. Peirce. Collected papers of charles sanders perice, 1931.

[104] Pfahl and Lebsanft. Integration of system dynamics modelling with descriptive process modelling and goal oriented measurement. *The Journal of Systems and Software*, 46:91–105, 1999.

[105] Poole, Mackworth, and Goebel. *Computational Intelligence: A Logical Approach*. Oxford University Press, 1997.

[106] D. Poole. Explanation and prediction: An architecture for default and abductive reasoning. *Computational Intelligence*, 5(2):97–110, 1989.

[107] Powell, Mander, and Brown. Strategies for lifecycle concurrency and iteration: A system dynamic approach. *The Journal of Systems and Software*, 46:91–105, 1999.

[108] Powersim. `http://www.dpsnet.com/powersim`.

[109] W. F. Punch. *A Diagnosis System Using a Task Integrated Problem Solver Architecture (TIPS), Including Causal Reasoning*. PhD thesis, Department of Computer and Information Science, Ohio State University, Columbus, 1989.

[110] J. R. Quinlan. Induction of decision trees. *Machine Learning*, 1:81–106, 1986.

[111] R. Quinlan. *C4.5: Programs for Machine Learning*. Morgan Kaufman, 1992.

[112] Kohavi R. A study of cross-validation and bootstrap for accuracy estimation and model selection. *IJCAI-95*, 1995.

[113] Raffo, Vandeville, and Martin. Software process simulation to achieve higher cmm level. *The Journal if Systems and Software*, 46:91–105, 1999.

[114] D. Raffo, G. Spehar, and U. Nayak. Generalized simulation models: What, why and how. May 2003.

[115] J.F. Ramil and N. Smith. Qualitative simulation of software evolution process. pages 41–47, 2002.

[116] J.F. Ramil and N. Smith. Qualitative simulation of models of software evolution. *To appear Software process improvement and practice*, 2003.

[117] G.P. Richardson. *System dynamics: simulation for policy analysis from a feedback perspective. Qualitative Simulation Modeling and Analysis Ch.7*. Springer-Verlag, 1991.

[118] P. Runeson and C. Wohlin. A dynamic usage modelling approach to software reliability engineering, 1998.

[119] S. Schafer. Mci worldcom data network is up; businesses hit by outages offered 20 days of free service. *Washington Post*, pages E1–E3, August 1999.

[120] Seachhi. Experience with software process simulation and modelling. *The Journal if Systems and Software*, 46:91–105, 1999.

[121] Ross D. Shachter. Evaluating influence diagram. *Operations Research*, 34(6):871–882, Nov. - Dec. 1986.

[122] M. Shaw and D. Garlan. *Software Architecture: Perspectives on an Emerging Discipline*. Prentice Hall, 1996.

[123] H.A Simon. Rational choice and the structure of the environment. In H.A Simon, editor, *Models of Man*. John Wiley, NY, 1957.

[124] B. Smith, N. Nguyen, and R. Vidale. Death of a software manager: how to avoid career suicide through dynamic process modeling. *American Programmer, May*, 1993.

[125] Connie U. Smith. Independent general principles for constructing responsive software systems. *ACM Transactions on Computer Systems*, 4(1), February 1986.

[126] J.W. Smith, B. Chandrasekaran, and T. Bylander. Pathex/liver: Structure-function models for causal reasoning.

[127] Inc Software Inventions. Decide.

[128] H. Sterman. Business dynamics: Systems thinking and modeling for a complex world. 2000.

[129] J. Sticklen. Distributed abduction in mdx2. *Artificial Intellignece in Scientific Computation: Towards Second Generation Systems*, 1989.

[130] A.J. Suarez, P.J. Abed, R.M. Gasca, and J.A. Ortega. Qualitative simulation of human resources subsystem in software development projects. pages 169–176, 2002.

[131] T.J.McCabe. A complexity measure. *IEEE Transaction of Software Engineering, SE-2*, 4:308–320, 1976.

[132] Inc TreeAge Software. Data.

[133] Vensim. `http://www.vensim.com`.

[134] M.A. Vouk. Software reliability engineering. 2000.

[135] T.J. Walsh. Software reliability study using a complexity measure. pages 761–768, New York, 1979. AFIPS.

[136] Microsoft Internet Explorer web site. `http://www.microsoft.com/windows/ie/default.asp`.

[137] NASA IV&V web site. `http://www.ivv.nasa.gov/`.

[138] NASA Jet Propulsion Laboratory web site. `http://www.jpl.nasa.gov`.

[139] SWI-Prolog web site. `http://www.swi-prolog.org/`.

[140] Information Week, August 1999.

[141] M.A. Weintraub and T. Bylander. Quawds: A composite diagnostic system for gait analysis. *Proceedings of the Thirteenth Annual SYmposium on Computer Applications in Medical Care*, pages 145–151, 1989.

[142] P. Wernick and M.M. Lehman. Software process white box modelling for feast. *The Journal if Systems and Software*, 46:193–201, 1999.

[143] A. Wesslen and C. Wohlin. Early estimation of software reliability through dynamic analysis. *Achieving Quality in Software*, pages 175–186, 1996.

[144] Williford and Chang. Modeling fedex's it division: A system dynamics approach to strategic it planning. *The Journal if Systems and Software*, 46:91–105, 1999.

[145] I. H. Witten and E. Frank. *Data Mining: Practical Machine Learning Tools and Techniques with Java Implementations*. Morgan Kaufmann, 1999.

[146] C. Wohlin, P. Runeson, and A. Wesslen. Software reliability estimations through usage analysis of specifications and designs. *International Journal of Reliability, Quality and Safety Engineering*, 3(2):101–117, 1996.

Appendix A

Appendix

A.1 Code Segments

A.1.1 Logic Implementation

```
lurch_logic(and,  [seq=l2r,y=all,n=dc,ytries=all,ntries=0]).
lurch_logic(rand,  [seq=rone,y=all,n=dc,ytries=all,ntries=0]).
lurch_logic(or,  [seq=l2r,y=1,n=dc,ytries=1,ntries=0]).
lurch_logic(ror,  [seq=rone,y=1,n=dc,ytries=1,ntries=0]).
lurch_logic(any,  [seq=l2r,y=1,n=dc,ytries=all,ntries=0]).
lurch_logic(rany,  [seq=rone,y=1,n=dc,ytries=all,ntries=0]).
lurch_logic(xor,  [seq=l2r,y=1,n=xc(1),ytries=1,ntries=xc(1)]).
lurch_logic(rxor,  [seq=rone,y=1,n=xc(1),ytries=1,ntries=xc(1)]).
lurch_logic(nand,  [seq=l2r,y=dc,n=all,ytries=0,ntries=all]).
lurch_logic(rnand,  [seq=rone,y=dc,n=all,ytries=0,ntries=all]).
lurch_logic(nor,  [seq=l2r,y=dc,n=1,ytries=0,ntries=1]).
lurch_logic(rnor,  [seq=rone,y=dc,n=1,ytries=0,ntries=1]).
lurch_logic(nany,  [seq=l2r,y=dc,n=1,ytries=0,ntries=all]).
lurch_logic(nrany,[seq=rone,y=dc,n=1,ytries=0,ntries=all]).
lurch_logic(dxor1,  [seq=l2r,y=2,n=dc,ytries=2,ntries=0]).
lurch_logic(rdxor1,  [seq=rone,y=2,n=dc,ytries=2,ntries=0]).

lurch_logic(dxor, L) :- lurch_logic(dxor1, L); lurch_logic(nand, L).
lurch_logic(rdxor, L) :- lurch_logic(rdxor1, L); lurch_logic(rnand, L).

de_morgan(and, nor).
de_morgan(rand, rnor).
de_morgan(or, nand).
de_morgan(ror, rnand).
de_morgan(any, nand).
de_morgan(rany, rnand).
de_morgan(nand, or).
de_morgan(rnand, ror).
de_morgan(nor, and).
de_morgan(rnor, rand).
de_morgan(nany, and).
de_morgan(xor, dxor).
de_morgan(rxor, rdxor).
```

A.1.2 Main Inference Engine

```
lurch(X) :- once(lurch0(X,Y)), lurch1(Y).

lurch0(X, bad(var,X)) :- var(X).
```

```
lurch0(not not X, Y) :- lurch0(X,Y).
lurch0(not claim2parent(effect(CEffect=Claim),effect(Effect=Goal)),
    claim2parent(effect(InvC=Claim),effect(Inv=Goal))) :-
  invert_effect(CEffect, InvC), invert_effect(Effect, Inv).

% specific for effects that are randomly picked
lurch0(not effect(Roulettable=G), effect(InvE=G)) :-
  roulette(Roulettable, _),
  lurching_roulette(effect(Roulettable=G),effect(REffect=G)),
  invert_effect(REffect,InvE).

lurch0(not effect(E=G), effect(InvE=G)) :- invert_effect(E,InvE).

% if X is logic(Blah) and it is negated, de_morgan the logic
lurch0(not X, logic(DLogic, Terms)) :-
  lurch0(X, logic(Type, Terms)),
  check(internal_logics, Type),
  de_morgan(Type, DLogic).

lurch0(not (X,Y), (not X;not Y)).
lurch0(not (X;Y), (not X,not Y)).
lurch0(not (call(X)), call(not X)).

lurch0(not X, old(not X,Index)) :-
  choices(not X,Index),
  % check that neither X nor not X is memo-ed
  (\+ \+ memo(Index,not X,_); \+ \+ memo(Index,X,_)).
lurch0(not X, assume(not X,Index)) :-
  choices(not X,Index), \+ clause(X,_).
lurch0(not X, new(not, X,Index)) :- choices(not X,Index).

lurch0(claim2parent(A,B),claim2parent(A,B)).

% specific for effects that are randomly picked
lurch0(effect(Roulettable=G), effect(REffect=G)):-
  roulette(Roulettable, _),
  lurching_roulette(effect(Roulettable=G),effect(REffect=G)).

lurch0(effect(E=G), effect(E=G)). % softgoal app specific stuff

lurch0(call(X), call(X)).
lurch0(true, true).
lurch0((X->Y;Z), (X->Y;Z)).
lurch0(once(X), once(X)).
lurch0((X,Y), (X,Y)).
lurch0(X, logic(InferType, Terms)) :-
  X =.. [Logic,Terms],  check(logics, Logic),
  infer(Logic, InferType).

lurch0(X, old(X,Index)) :-
  choices(X,Index),   % check that neither X nor not X is memo-ed
  (\+ \+ memo(Index,X,_); \+ \+ memo(Index,not X,_)).
lurch0(X, assume(X,Index)) :- choices(X,Index), \+ clause(X,_).
lurch0(X, new(X,Index)) :- choices(X,Index).
lurch0(X, call(X)) :- knownBuiltIn(X).
lurch0(X, sub(X)) :- clause(X,_).
lurch0(X, bad(unknown, X)).
```

```
% lurch1:  processed arg 2 of lurch0
% softgoal specific.
% first find out whether Claim is true or not
% if Claim is true and it supports the Goal, lurch Goal

%  choices(Claim, Index), bassert(rouletted(Index, Claim, REffect)).

lurch1(claim2parent(effect(CEffect=Claim), Goal)):-
  lurch(Claim), good_effect(CEffect), lurch(Goal).

% if claim refute an effect on goal, and if claim is false, than
% do the goal.
lurch1(claim2parent(effect(CEffect=Claim),Goal)):-
  lurch(not Claim), bad_effect(CEffect), lurch(Goal).

% otherwise, don’t ever lurch goal ’cuz it won’t have any effect anyways
lurch1(claim2parent(_,_)).

%  choices(G, Index), bassert(rouletted(Index, G, R)).

lurch1(effect(E=G)) :- good_effect(E), lurch(G).
lurch1(effect(E=G)) :- bad_effect(E), lurch(not G).

lurch1(true).
lurch1(bad(Type, X))       :- complain(Type, X).
% changed barph to complain - just a matter of terminology...
lurch1(call(X))      :- X.
lurch1(sub(X))       :- \+ X=not(_), clause(X,Y), lurch(Y).
lurch1(once(X))      :- once(lurch(X)).
lurch1(old(X,Index)) :- memo(Index,X,_).

% conjunction:  no logics such as rors, rany etc involved
lurch1((X;Y)):- once(rone([X,Y],Z,[A])), (lurch(Z); lurch(A)).
lurch1((X,Y)):- once(rone([X,Y],Z,[A])), lurch(Z), lurch(A).

% Eliza hasn’t use this.
lurch1((X->Y;Z))     :- lurch(X) -> lurch(Y) ; lurch(Z).

lurch1(logic(Logic, Terms)) :-
  lurch_logic(Logic, Rules), lurching( Rules, Terms).

lurch1(assume(X,Index)) :- bassert(memo(Index,X,0)).
lurch1(new(not, X,Index)) :-
oneof(X/Y/R,clause(X,Y,R), X/Y1/R1),
brecorda(current_head, R1), %format(’lurching not ~w ~n’, [X]),
lurch(not Y1), %format(’lurched not ~w ~n’, [X]),
recorded(current_head, R1, Ref), erase(Ref),
bassert(memo(Index,not X,R1)).
lurch1(new(X,Index)) :-
oneof(X/Y/R,clause(X,Y,R), X/Y1/R1),
brecorda(current_head, R1), %format(’lurching ~w ~n’, [X]),
lurch(Y1), %format(’lurched ~w ~n’, [X]),
recorded(current_head, R1, Ref), erase(Ref),
bassert(memo(Index,X,R1)).

% ------logics and logics-------
```

```
lurching(Rules, Terms) :-
  decode_rules(Rules, Sequence, Ys, Ns, YTries, NTries, Terms),
  lurching0(Sequence, 'y', Ys, YTries, Terms),
  lurching0(Sequence, 'n', Ns, NTries, Terms).

decode_rules(Rules, S, Y, N, YT, NT, Terms) :- length(Terms, L),
  maplist(decode(Rules, L), [seq,y,n,ytries,ntries], [S,Y,N,YT,NT]).
decode(A,B,C,D) :- once(decode1(A,B,C,D)).
decode1(Rules, Len, A, Len):- memberchk(A=all, Rules).
decode1(Rules, Len, A, L):- memberchk(A=xc(X), Rules), L is Len - X.
decode1(Rules, _, A, B):- memberchk(A=B, Rules).

lurching0(A,B,C,D,E) :- once(lurching1(A,B,C,D,E)).

lurching1(_,_,Var,_,_) :- not integer(Var).
% do lurching only if we care about the threshold
lurching1(_,_,_,0,_).
% stop lurching when no more tries

lurching1(Seq, YOrN, Threshold, OTries, Terms):-
  %Seq is either l2r or rone
  GetOne =.. [Seq, Terms, H, T], call(GetOne),
  lurching2(H, YOrN, Threshold, NThreshold),
  Tries is OTries - 1, !,
  lurching0(Seq, YOrN, NThreshold, Tries, T).

lurching2(Term, YesOrNo, 0, 0) :- try(lurching3(Term, YesOrNo)).
lurching2(Term, YesOrNo, Th, NTh) :- lurching3(Term, YesOrNo), NTh is Th -1.
lurching3(Term, 'y') :- lurch(Term).
lurching3(Term, 'n') :- lurch(not Term).
% ----get rouletted effects----
lurching_roulette(A,B) :- once(lurching_roulette1(A,B)).
lurching_roulette1(Key, Rouletted) :-
  recorded(current_head, H), recorded(H, Key=Rouletted).
lurching_roulette1(Key, Rouletted) :-
  rouletting(Key, Rouletted), recorded(current_head, H),
  brecorda(H, Key=Rouletted, Ref), brecorda(roulette,Ref).
```

A.1.3 Benefit Calculation

```
benefit(X,Y) :- b(X,Y0), Y is Y0.

b(X,Y) :- once(b0(X,What)), b1(What,Y).

b0(X,      oops(var))    :- var(X).

b0(claim2parent(A,B), claim2parent(A,B)).

b0(effect(Roulette=G), Out):-
  roulette(Roulette,_), rouletted(effect(Roulette=G), Out).

b0(effect(E=G), effect(E=G)).

b0(true,  true).
b0((A,B), (A,B)).
b0((A;B), (A;B)).
```

```
b0(X, Op/V/Terms) :-
  X =.. [Logic,Terms],  check(logics, Logic), score(Logic,Op,V).

b0(not A, stakeholder(not A)) :- node_info(stakeholder, A, _).
b0(A, stakeholder(A)) :- node_info(stakeholder, A, _).

b0(A,     abduced(not A) )  :- choices(A,Ind), memo(Ind,not A,0).
b0(A,     abduced(A) )  :- choices(A,Ind), memo(Ind,A,0).

b0(not A,     claused(not A,R)) :- choices(not A,Ind), memo(Ind,A,R).
b0(A,     claused(not A,R)) :- choices(not A,Ind), memo(Ind,not A,R).
b0(A,     claused(A,R)) :- choices(A,Ind), memo(Ind,A,R).
b0(A,     clause_not_lurched(A,_)) :- clause(A,_).
b0(A,     unclaused(A)).

b1(true,Value) :- score(default, Value).
b1((X,Y), O) :- b(X,O1), b(Y,O2), combine(O1,O2,O).

% without support of claim, effect of child softgoal to parent is
b1(claim2parent(effect(_=Claim), _),V):-
  memo(_,not Claim,_), score(wout_claim_support, V).
b1(claim2parent(Claim, Goal),WWW):-
  b(Goal, W), b(Claim, WW),
  scoring(claim2parent, Arith, V),
  do(Arith, [W,WW], O), do(multiply,[V,O],WWW).

b1(effect(E=G),Out) :-
  scoring(E,Arith,Val), b(G,V), do(Arith,[Val,V],Out).

b1(claused(not A,_), W) :- score(not_done,W),
not (var(A), complain('node is not bounded', A)).
b1(claused(A,R), WW) :-
  weighting(A,W),
  clause(A,B,R),
  brecorda(current_head, R),
  b(B,X),
  recorded(current_head, R, Ref), erase(Ref),
  scoring(c2p, Arith, V),
  do(Arith, [W,X], O),
  do(multiply,[V,O],WW).

b1(clause_not_lurched(_A,_), W) :- score(not_lurched, W).

b1(Op/V/L0,Y) :- maplist(b,L0,L), do(Op,L,Y0), do(multiply,[V,Y0],Y).
b1(stakeholder(_), V):- score(stakeholder, V).
b1(abduced(not _),Y) :- score(not_done,Y).
b1(abduced(A),   Y) :-
  weighting(abduced, A, Y).
b1(unclaused(_), Y) :- score(not_done,Y).
b1(oops(X),      _) :-
complain('Something is wrong at calculating benefit',X).

scoring(A,B,D) :-
  score(A,B,C), callable(C), !, call(C,D).
scoring(A,B,C) :- score(A,B,C).

weighting(abduced,A,V) :- !, once(weighting1(abduced,A,V)).
weighting(A,C) :-
  (severity(A,_,S);S=normal),
  once(weighting1(S,A,C)).
weighting1(S,A,C) :-
  weight(S,A,W), callable(W), !, call(W,C).
weighting1(S,B,C) :- weight(S,B,C).

pricing(A,B,D) :-
  price(A,B,C), callable(C), !, call(C,D), jot_down(price,B,D).
pricing(A,B,C) :- price(A,B,C), jot_down(price,B,C).

combine(V1,V2, Out) :-
  score(combine, C),
  do(C, [V1,V2], Out).

do(add, L, Sum) :- sum(L, Sum).
do(min,L,Min)    :- min(L,Min).
do(max,L,Max)    :- max(L,Max).
do(average,L,Av) :- average(L,Av).
do(multiply,L,O) :- multiply(L, O).

rouletted(A,B) :- once(rouletted1(A,B)).
rouletted1(Roulette, Out) :-
  recorded(current_head, Head),
  recorded(Head, Roulette=Out).
rouletted1(A, clause_not_lurched(A,_)).

jot_down(A,B,C):- once(jot_down1(A,B,C)).
jot_down1(Key, Node, Info) :-
  enable_jot_down(Key),
  bassert(jotted_down(Key, Node, Info)).
jot_down1(_,_,_).
```

A.1.4 Other codes:

```
% check.pl
% check the sanity of the code writer (yup that’s me)

type( nodes, [actor, goal, softgoal, task, resource, claim, stakeholder, god]).
type( edges, [depends_on, means, consists_of, contributions]).
type( logics, [all_of, none_of, any_of, any_one_of, one_of]).

type( effects, [
  made, helped, unhurt, unbroken, positively_influenced,
  catastrophically_rated,
  critically_rated,
%  moderately_rated,
  lowly_rated,
  highly_rated]).

type( roulette, [ moderately_rated, positively_influenced]).

type( internal_logics, [
  and,
  rand,
  or,
  ror,
  any,
  rany,
  xor,
  rxor,
  nand,
  rnand,
  nor,
  rnor,
  nany,
  nrany,
  dxor,
  rdxor
  ]).

type( severity, [critical, veryCritical, extremelyCritical, normal,
nonCritical, mandatory, dominant, nonDominant]).
type( costs, [ extremelyHigh , veryHigh, high, notHigh]).

check(A,B) :- once(check1(A,B)).
check1( Type, Thing) :- type(Type, List), memberchk(Thing, List).

%cheat here
check1(effect, AnotherTypeThing) :- check(roulette, AnotherTypeThing).

complain(wrongSyntax, Type, L) :-
  format('Wrong syntax in defining ~w in ~w ~n', [Type, L]).

complain(Type, X) :-
  format('bad ~w for ~w ~n', [Type, X]).
% choices.pl

% make sure X fits the input syntax
choices(X,Ref) :-
  choice(X,_,_,Y),
  hash_term(Y,Ref).

% define types of goals and how they are memo-ed
% choice(Orig, Conjunction, Type, MemoTerm)
% index not terms as same as terms.
% so choices(not X,Ref) with give out hash_term(X, Ref)

% some stakeholders are granted absolute rightness
choice(not X, _, _, _) :- rightness(A), node_info(A, X, _), !, fail.


choice(not X, A, B, Y) :- choice(X, A, B, Y).

choice(X, nil, system_wide, X).  % may print out stuff here
choice(X of Y, of, Y, X of Y).

% display.pl
% configuration file for all visible stuff

displaying(depends_on=[arrowhead='normal']).
displaying(means=[arrowhead='open']).
displaying(consists_of=[arrowhead='tee']).
displaying(contributions=[arrowhead='inv']).
displaying(and=[label='and']).
displaying(or=[label='or']).
displaying(all_of=[label='all_of']).
displaying(none_of=[label='none_of']).
displaying(any_of=[label='any_of']).
displaying(any_one_of=[label='any_one_of']).
displaying(one_of=[label='one_of']).
displaying(made=[label='++']).
displaying(helped=[label='+']).
displaying(unhurt=[label='-']).
displaying(unbroken=[label='--']).
displaying((with)=[label='claim to effect']).
displaying((intermediate)=[arrowhead='inv']).

% -----------------------------added for cara-----------------------------%
displaying(catastrophically_rated=[label='4']).
displaying(critically_rated=[label='3']).
displaying(highly_rated=[label='3']).
displaying(moderately_rated=[label='2']).
displaying(lowly_rated=[label='1']).
displaying(positively_influenced=[label='?']).
% -----------------------------added for cara-----------------------------%

displaying((softgoal)=[shape=doubleoctagon]).
displaying((goal)=[shape=octagon]).
displaying((task)=[shape=hexagon]).
displaying((resource)=[shape=box]).
displaying((actor)=[shape=circle]).
displaying((claim)=[shape=triangle]).
displaying((stakeholder)=[shape=triangle]).
displaying((intermediate_node)=[label='i']).

display_gen(R, Out):-
  display_gen1(R, Temp),
  display_gen2(Temp, Out).
display_gen1(A^B, [O1,O2]):- !,
  display_gen1(A, O1), display_gen1(B, O2).
display_gen1(A, Out) :- displaying(A=Out).
display_gen2(L, Out) :-
  flatten(L, NewL), setof(T, Junk^(member(T=Junk,NewL)), Ts),
  maplist(display_gen3(NewL), Ts, Out).

display_gen3(All, T, PrettyJunks) :-
  setof(Junk, member(T=Junk, All), Junks),
  display_gen4(T, Junks, PrettyJunks).

display_gen4(label, Raw, label=S) :-
  sformat(S, '~w', [Raw]).
display_gen4(Tag, [Raw], Tag=Raw).
display_gen4(_,_,'label="undefined"').

% expand.pl
expand_relate(Tail depends_on Rest, Tail, depends_on, Rest).
expand_relate(Parent consists_of Rest, Parent, consists_of, Rest).
expand_relate(Parent means Rest, Parent, means, Rest).
```

```
expand_nodes(A, B, C) :- once(expand_nodes0(A,B,C)).

expand_nodes0(Type, Node in Actor is C costed Cost, [cost(Node,Type,Cost)|Out]) :-
  check(costs, Cost), expand_nodes(Type, Node in Actor is C, Out).

expand_nodes0(Type, Node in Actor is C, [severity(Node,Type,C),Out]) :-
  check(severity, C), expand_nodes(Type, Node in Actor, Out).

expand_nodes0(Type, Node in Actor costed Cost, [cost(Node,Type,Cost),Out]) :-
  check(costs, Cost), expand_nodes(Type, Node in Actor, Out).

expand_nodes0(Type, Node is C costed Cost, [cost(Node,Type,Cost)|Out]) :-
  check(costs, Cost), expand_nodes(Type, Node is C, Out).

expand_nodes0(Type, Node is C, [severity(Node,Type,C),Out]) :-
  check(severity, C), expand_nodes(Type, Node, Out).

expand_nodes0(Type, Node costed C, [cost(Node,Type,C),Out]) :-
  check(costs, C), expand_nodes(Type, Node, Out).

expand_nodes0(Type, Node in Actor, node_info(Type, Node, Actor)).

expand_nodes0(Type, Node, node_info(Type, Node, global)).
expand_nodes0(_,_,P,_) :- complain(wrongSyntax, node, P).

term_expansion(A if Blah, Out) :-
  expand_term(everyone says A if Blah, Out).
term_expansion(Man says Goal if Blah,
  [(Goal :- Man, Blah)|NEInfo]) :-
  expand0(Goal, contributions, Blah, _, NEInfo).
term_expansion(R, Out):-
  expand_relate(R, Tail, Type, Rest),
  expand0(Tail, Type, Rest, _, Out).
term_expansion(R, Out):-
  R =.. [NType,Node], check(nodes, NType),
  expand_nodes(NType, Node, Out).

expand0(A, B, C, D, E) :- once(expand00(A, B, C, D, E)).
expand1(A, B, C, D, E) :- once(expand10(A, B, C, D, E)).
find_junks(A, B, C) :- once(find_junks0(A,B,C)).

% Node1 depends_on Node2 for Node3...
expand00(Tail, Type, Mid for Rest, Junks, [O1,O2]):-
  find_junks(Rest, Head, Junks),
  expand1(Tail, Type, Head, Junks, O1),
  expand1(Head, Type, Mid, Junks, O2).

% Node1 consists_of Node2 all_of Node 3 all_of N..
expand00(Parent, Type, Logical, Junks, Out):-
  not Type =.. [ˆ|_], has_logic(Logical, Logic,_,_),
  expand0(Parent, TypeˆLogic, Logical, Junks, Out).
expand00(Parent, TypeˆLogic, Logical, Junks, [O1|O2]):-
  has_logic(Logical, Logic, Child, Rest),
  expand0(Parent, TypeˆLogic, Rest, Junks, O2),
  expand1(Parent, TypeˆLogic, Child, Junks, O1).

expand00(Tail, Type, Rest, Junks, Out):-
```

```
  find_junks(Rest, Head, Junks),
  expand1(Tail, Type, Head, Junks, Out).

has_logic(Child all_of Rest, all_of, Child, Rest).
has_logic(Child none_of Rest, none_of, Child, Rest).
has_logic(Child any_of Rest, any_of, Child, Rest).
has_logic(Child any_one_of Rest, any_one_of, Child, Rest).
has_logic(Child one_of Rest, one_of, Child, Rest).

has_logic(Child and Rest, and, Child, Rest).
has_logic(Child or Rest, or, Child, Rest).

find_junks0(Head, Head, [ref="nil", keywords=[all]]):-
  not compound(Head).
find_junks0(Junks, Head, [JunkShell=JunkMeat|OtherJunks]) :-
  find_junks1(Junks, JunkShell, Head, T),
  find_junks0(T, JunkMeat, OtherJunks).
find_junks0(List, List, []) :- is_list(List).

find_junks1(Head ref T, ref, Head, T).
find_junks1(Head keywords T, keywords, Head, T).

expand10(Tail, Type, C1 with C2, Junks,
    [node_info(intermediate_node, Tail^C1^(intermediate), global),
    O1,O2,O3]) :-
  expand1(Tail, Type^(intermediate), Tail^C1^(intermediate), Junks, O1),
  expand1(Tail^C1^(intermediate), intermediate, C1, [ref="",keywords=[]], O2),
  expand1(Tail^C1^(intermediate), Type^(with), C2, Junks, O3).
expand10(Tail, Type, What by Head, Junks, Out) :-
  expand1(Tail, Type^What, Head, Junks, Out).
expand10(Tail, Type, Head, Junks, Out) :-
  append([edge_info, Type, (Tail), (Head)], Junks, O),
  Out =.. O.

goal_expansion(A, O) :-
  has_logic(A, Logic, B, Rest),
  expand_goal(B, Clause), c2l(Clause, NC),
  expand_my_goal(Rest, Logic, Expanded),
  flatten([NC|Expanded], E),
  O =.. [Logic, E].
% pass Logic to make sure each clause has only one type of logic

expand_my_goal(Rest,Logic,[NB|Expanded]) :-
  has_logic(Rest, Logic, B, ORest),
  expand_goal(B, NewB), c2l(NewB, NB),
  expand_my_goal(ORest, Logic, Expanded).
expand_my_goal(Rest,_,W):-
  expand_goal(Rest, What), c2l(What, W).
expand_my_goal(R,_,[]):- print(R), nl.

%goal_expansion(A with B, (claim2parent(B1,A1),A2)):-
%  expand_goal(A, A1), c2l(A1,A2),
%  expand_goal(B, B1).
goal_expansion(A with B, (claim2parent(B1,A1))):-
  expand_goal(A, A1),
  expand_goal(B, B1).
goal_expansion(not X, \+ X).
goal_expansion(Whatever ref _, Whatever).
goal_expansion(Whatever keywords _, Whatever).
%goal_expansion(A by B, (effect(A=B), B)).
goal_expansion(A by B, effect(A=B)).

% extract.pl
% extract node types from the definition
% mainly deals with extracting specific information
% also contains graphing functions
% handles node types

% graph_type(softgoal, ’softgoal.dot’).
% graph_type(contributions, ’contributions.dot’).

:- ensure_loaded(’graph.pl’).

graph_type(Type, FileName) :-
  ignore(write_head(FileName)),
  graph_type1(Type),
  graph_nodes(FileName),
  ignore(write_edges(FileName)),
  ignore(write_bottom(FileName)),
  clear_node_edge.

graph_type1(Type) :- once(graph_type10(Type, N)), graph_type2(Type, N).
graph_type10(Type, nodes) :- check( nodes, Type).
graph_type10(Type, edges) :- check( edges, Type).

graph_type10(X) :- complain( ’nodes and edges’, X).

graph_type2(Type, edges) :-
  forall(
    (edge_info(T,C,P,_,_), match_type(Type, T)),
     assert_my_edge(T,C,P)).
graph_type2(Type, nodes) :-
  forall(edge_info(_,C,P,_,_),
    (  graph_type_nodes1(Type, C),
       graph_type_nodes1(Type, P))).
graph_type_nodes1(Type, Node) :-
  node_info(Type,Node,_), !,
  ensure_assert(appearing(Node)),
  forall(edge_info(T,Node,P,_,_), assert_my_edge(T,Node,P)).
graph_type_nodes1(_,_).

match_type(T,T):-!.
match_type(Type,T1ˆT2) :-
  match_type(Type,T2);
  match_type(Type,T1).


% graph.pl
% params:
% simplify=[mediaShop, mediA]
% hide everything within actors mediaShop and mediA
% keywords=[early, fig1, fig2]
% show only edge_info with keywords early, fig1, or fig 2
% format:  graph(’file.dot’)  graph everything
% graph([keywords=[early, fig1]] graph edges with these keywords
% graph([simplify=[mediaShop, mediA]] graph with simplified
% graph([keywords=[...], simplify=[...]]

graph(FileName) :-
  graph([], FileName).

graph(Params, FileName) :-
  ignore(write_head(FileName)),
  graph_edges(Params),
  graph_nodes(FileName),
  ignore(write_edges(FileName)),
  ignore(write_bottom(FileName)),
  clear_node_edge.

graph_nodes(FileName) :-
  setof(BN, A^B^node_info(A,B,BN), BNs),
  forall(member(OneBN, BNs), graph_node(OneBN)),
  write_clusters(BNs, FileName).

graph_node(OneBN) :-
  forall(node_info(NType, Name, OneBN),
  assert_my_node(NType, Name, OneBN)).

graph_edges(Params) :-
  get_p(simplify, Params, Simplify),
  get_p(keywords, Params, Keywords),
  bagof( T|C|P|K, W^edge_info(T,C,P,W,K),Es),
  forall(    % check that the edge is to be shown
      (member(E, Es),
      E=(_|_|_|keywords=Keyword),
      check_keywords(Keyword,Keywords)),
    graph_big(E, Simplify)).

graph_big(EdgeT|Child|Parent|_, Simplify):-
   simplify(Child, Simplify, NChild),
   simplify(Parent, Simplify, NParent),
   assert_my_edge(EdgeT, NChild, NParent).

check_keywords(A,B):-once(check_keywords0(A,B)).
check_keywords0(_, []).
check_keywords0([all], _).
check_keywords0(Ks, Keywords):-
  member(One,Keywords), memberchk(One, Ks).

simplify(A,B,C):-once(simplify0(A,B,C)).
simplify0(One,SList, Big):-
  node_info(_,One,Big), memberchk(Big,SList).
simplify0(One,_, One).

%%%%%%%%%%%%%%%%%%%%%%%%
% get params
get_p(T, P, L):- memberchk(T=L, P), !.
get_p(_, _, []).

%%%%%%%%%%%%%%%%%%%%%%%%
% assertion and cleaning here
```

```
assert_my_edge(EdgeT, NChild, NParent) :-
  display_gen(EdgeT,Display),
  sformat(S, '"~w" -> "~w" ~p; ~n', [NChild, NParent, Display]),
  ensure_assert(edge(S)),
  ensure_assert(appearing(NChild)),
  ensure_assert(appearing(NParent)).
assert_my_node(NType, Name, Big) :-
  display_gen(NType,Display),
  sformat(S, '"~w" ~p; ~n', [Name, Display]),
  ensure_assert(node(Big, Name, S)).

ensure_assert(Stuff) :-  clause(Stuff, true).
ensure_assert(Stuff) :- assert(Stuff).

clear_node_edge :- retractall(edge(_)), retractall(node(_,_,_)),
retractall(appearing(_)).
%%%%%%%%%%%%%%%%%%%%%%%%
% write_Whatever
% write stuff on file
write_head(FileName) :-
  tell(FileName), format('digraph G { ~n ranksep=2.0; ~n'), told.
write_clusters(BNs, FileName) :-
  forall(
    member(BN, BNs),
    ignore(write_cluster(BN, FileName))
  ).
write_cluster(BN, FileName) :-
  bagof(S, Name^(node(BN, Name, S), appearing(Name)), Ss),
  append(FileName),
  format('subgraph ~w { ~n', [BN]),
  format('style=filled; ~n label="~w";', [BN]),
  forall(member(OneS, Ss), format('~s', OneS)),
  format('~n} ~n'),
  told.
write_edges(FileName):-
  append(FileName),
  forall( edge(S), format('~s', [S])), told.
write_bottom(FileName):-
  append(FileName), format('} ~n'), told.
%%%%%%%%%%%%%%%%%%%%%%%%


% lib.pl

d2l((X;Y),[X|Rest]) :- !,d2l(Y,Rest).
d2l(X,    [X]).

c2l((X,Y),[X|Rest]) :- !,c2l(Y,Rest).
c2l(X,    [X]).

try(X) :- X.
try(_).

invert([H|T], [not H|T1]) :-
  invert(T,T1).
invert([],[]).
```

```
min([H|T],Min) :- min(T,H,Min).
min([],X,X).
min([H|T],In,Out) :-
  (In < H -> Temp=In ; Temp=H),
  min(T,Temp,Out).

max([H|T],Max) :- max(T,H,Max).
max([],X,X).
max([H|T],In,Out) :-
  (In > H -> Temp=In ; Temp=H),
  max(T,Temp,Out).

average(L,A) :-
  sum(L,S),
  length(L,N),
  A is S/N.

sum([H|T],N) :- sum(T,H,N).
sum([],Z,Z).
sum([H|T],X,Z) :-  Y is X + H,  sum(T,Y,Z).
sums(L0,S) :- flatten(L0,L), sum(L,S).

multiply(L,Out):- multiply(L,1,Out).
multiply([H|L],Temp,Out):-
  T0 is Temp*H, multiply(L,T0,Out).
multiply([],T,T).


:- arithmetic_function(randomize/2).
:- arithmetic_function(randomize/1).

% randomize(normal(0,0.4), [[>=, 0], [=<, 0.2]], P).

randomize(A,B) :- callable(A), call(A,B).

randomize(A,B,C) :- once(randomize1(A,B,C)).
randomize1(ArithFun, CriteriaList, Out) :-
  callable(ArithFun), call(ArithFun, Out),
  checklist(randomize2(Out), CriteriaList).
randomize1(A,B,C) :- randomize(A,B,C).
% recurse until the output passes the criteria

randomize2(Val, [Expr,Num]) :-
  Call =.. [Expr, Val, Num], call(Call).

:- arithmetic_function(range/2).
range(Min,Max,Out) :-
  Out is (Max-Min)*ranf+Min.


:- arithmetic_function(skewed_range/3).

skewed_range(Mean,Min, Max,Out) :-
  O is (Mean-Min)/(Max-Min),
  beta_values(L), skewed_range1(O,L,Beta),
  beta(Beta, 1, O1), Out is O1*(Max-Min)+Min.
```

```
skewed_range1(_O, [Out], Out).
skewed_range1(O, [Out|_], Out) :-
  OO is truncate(O*100), OOut is truncate(Out*100), OO=<OOut.
skewed_range1(O, [_|L], Out) :- skewed_range1(O,L,Out).
/*

=(K,V,Old, [K=V|Less])  :- less1(Old,K=V0,Less),!, V0 = V.
=(K,V,Old, [K=V|Old]).
*/
:- arithmetic_function(normal/2).

normal(M,S,N) :-
        box_muller(M,S,N).

% classical fast method for computing
% normal functions using polar co-ords
% (no- i dont understand it either)
box_muller(M,S,N) :-
        w(W0,X),
        W is sqrt((-2.0 * log(W0))/W0),
        Y1 is X * W,
        N is M + Y1*S.

w(W,X) :-
        X1 is 2.0 * ranf - 1,
        X2 is 2.0 * ranf - 1,
        W0 is X1*X1 + X2*X2,
        % IF -> THEN ; ELSE
        % same as xx :- IF,!, THEN,
        %         xx :- ELSE
        % vars bound in IF not available to ELSE
        % no backtracking within the IF
        % -> ; precendence higher than ,
        (W0  >= 1.0 -> w(W,X) ; W0=W, X = X1).

:- arithmetic_function(ranf/0).

ranf(X) :-
        N =  65536,
        % X is random(N) returns a
        % random number from 0 .. N-1
        X is random(N)/N.
/*
repeats(N,X) :-
  between(1,N,I),
  memo(I),
  once(X),
  fail.
repeats(_,_).

memo(X) :- 0 is X mod 250, !, write(user,X), nl(user), flush_output(user).
memo(_).

printl([X|Y],Sep) :-
  write(X),
  forall(member(Z,Y),format('~w~w',[Sep,Z])).
*/
```

```
 :- arithmetic_function(beta/2).

 beta(B,Param,X) :- beta1(B,Param,X).

 beta1(B,Param,X) :- beta2(B,Param,X),!.
 beta1(B,Param,X) :- B1 is 1.0 - B, beta2(B1,Param,Y),X is 1 - Y.

 beta2(0.33,Param,X) :- X is (1-ranf^(0.5*Param)).
 beta2(0.50,_,X) :- X is ranf.
 beta2(0.60,Param,X) :- X is ranf^(0.67*Param).
 beta2(0.670,Param,X) :- X is ranf^(0.5*Param).
 beta2(0.750,Param,X) :- X is ranf^(0.33*Param).
 beta2(0.80,Param,X) :- X is ranf^(0.25*Param).
 beta2(0.90,Param,X)  :- X is ranf^((1/9)*Param).
 beta2(1,_,1).

gen_betas :-
  bagof(Num, A^B^Whatever^clause(beta2(Num, A,B),Whatever), Nums),
  maplist(gen_betas1, Nums, N1),
  append(Nums, N1, AllBs), sort(AllBs, Sorted),
  assert(beta_values(Sorted)).
gen_betas1(Num, Out) :- Out is 1-Num.

:- gen_betas.

oneof(X,Y,Z) :-
  bagof(X,Y, L),
  rone(L,Z,_).

bassert(X) :- assert(X).
bassert(X) :- retract(X),fail.

bretract(X) :- retract(X), bretract1(X).
bretract1(_).
bretract1(X) :- assert(X),fail.

brecorda(Key,Item) :- recorda(Key, Item).
brecorda(Key,Item) :- recorded(Key, Item, Ref), erase(Ref), fail.

brecorda(Key,Item,Ref) :- recorda(Key,Item,Ref).
brecorda(Key,Item,_) :- recorded(Key, Item, Ref), erase(Ref), fail.

l2r(L,Out, Rest) :- member(Out,L), select(Out, L, Rest).

rone(L,One,Rest) :- length(L,N), rone(L,N,One,Rest,_).

rone([H],_,H,[],0) :- !.
rone([H|T],N0,X,Out,N) :-
        N is N0 - 1,
        Pos is random(N0) + 1,
        less1(Pos,One,[H|T],Rest),
        (X=One,
         Out=Rest
        ; Out=[One|Tail],
          N1 is N0 - 1,
         rone(Rest,N1,X,Tail,_)).


less1(1,H,[H|T],T) :- !.
less1(N0,X,[H|T],[H|L]) :-  N is N0 - 1, less1(N,X,T,L).

:- dynamic knownBuiltIn/1.

 makeBuiltin(X) :- knownBuiltIn(X),!.
 makeBuiltin(X) :- assert(knownBuiltIn(X)).

 :- forall(predicate_property(X,built_in), makeBuiltin(X)).


% main.pl

:- load_files([
  readme,   % notes
  ops,     % operaters
  lib,     % util functions
  setup,     % app setup
  report,     % report
  logic_setup,   % logic inference config
  display,     % graph display config
  check,     % type checking
  expand,   % term expansion
  graph,     % graphing module
  extract,     % graphing sub-module
  choices,   % clause-head calling syntax
  lurch,     % meta-interpreter
  utils,     % utils for tar2
  benefits,  % heuristic
  score
  ],
  [silent(true)]).
%    [silent(false)]).

:- format('~s', [
  "Please modify score.pl for the type of scoring."
  ]).


% ops.pl
% all non-prolog convension resides between 701-899
% dependencies...

:- op(911, xfx, says).
:- op(910, xfx, if).

:- op(900, fy, not).
:- op(890, xfx, [depends_on, consists_of, means]).
:- op(880, xfx, for).

:- op(850, fx, [actor, goal, softgoal, task, resource, claim, stakeholder, god]).
:- op(800, xfx, in).
:- op(795, xfx, costed).
:- op(790, xfy, [all_of, none_of, one_of, any_one_of, any_of]).
:- op(780, xfy, [and, or]).

:- op(100, xfx, ref).
:- op(99, xfx, keywords).
:- op(98, xfy, with).
:- op(10, xfx, by).
:- op(9, xfx, of).
:- op(1, xfx, at).


% readme.pl

readme :-
  show_me_types,
  show_me_all_notes.

show_me_all_notes :- forall( note(Type,String), show_me('general notes', Type, String)).

show_me_types:-forall( type(Type, List), show_me('type', Type, List)).
show_me(Head, Type, List) :-
  is_list(List), format('~w: ~t~w has ~w ~n', [Head, Type, List]).
show_me(Head, Type, String) :-
  format('~w --- ~t[~w] :~t ~w ~n', [Head, Type, String]).

note(edge_def,
  'edges with different logical relations should be defined in individual
  sentences. i.e. a consists_of b and c.  a consists_of d or e. ').

note(tar2,
  'functions for easy tar2-ing can be found in utils.pl').

note(lurch,
  ' if SG :- rule1, A by C1 any_of B by C2. and SG :- rule2,
  X by C3 one_of Y by C4. either rule1 or rule2 will be true in one inference').

note(lurch,
  ' the trick is to set the top-level goal is composed of any_of children').

note(softgoal,
  ' assumption:  goals after with are claims to an edge').

note(cara,
  ' cara graphs are located under folder /graphs').

note(test,
  ' test cases are under folder tests').
note(test,
  ' load file test.pl for running test cases').

% report.pl

report(X) :-  quickReport, write('\n===> '), scoreReport(X).

scoreReport(X) :-
  costs(C),
  benefit(X,B),
  format('cost=~w benefit=~w\n',[C,B]).
```

```
quickReport :- forall(memo(_,Y,0),format('assumption=~w\n',[Y])).

% find max/min for one data at a time to avoid outta stack

findMinMax([],_).
findMinMax([H|T],PosAtL) :-
  [H],
  findMinMax1(PosAtL),
  findMinMax(T,PosAtL).

findMinMax1(PosAtL) :-
  result(_, L), nth0(PosAtL, L, Elem),
  findM(>=, max, Elem), findM(=<, min, Elem), fail.
findMinMax1(_).

findM(Arith, M, Elem) :- data(M,CurrM),
  Call =.. [Arith, Elem, CurrM], Call, !,
  retract(data(M,CurrM)), assert(data(M,Elem)).
findM(_,_,_).

% make sure the directories have only one pl file
get_boundries_for_all_files(Wildcard, PosAtList):-
   expand_file_name(Wildcard, ListOfFileNames), retractall(data(_,_)),
   assert(data(max, -1)), assert(data(min, 100000)),
   findMinMax(ListOfFileNames,PosAtList).


tables(Selector, [FileName|ListOfFiles], [OneT|ListOfTitles],
  XLabels, YLabels, XUBound, XLBound, YUBound, YLBound, IndexX, IndexY, String) :-
  retractall(data_classify(_,_,_,_)),
  classify_cell_elem(x,XLabels, XUBound, XLBound),
  classify_cell_elem(y,YLabels,YUBound,YLBound),
  tables1(Selector, [FileName|ListOfFiles], [OneT|ListOfTitles],
  XLabels, YLabels, IndexX, IndexY, "", String).

tables1(_, [],[],_,_,_,_,S,S).
tables1(Selector,[FileName|ListOfFiles], [OneT|ListOfTitles],
  XLabels, YLabels, IndexX, IndexY, CS, String) :-
  tablize(Selector,FileName, XLabels, YLabels, IndexX, IndexY, S),
  tables2(Selector, CS, OneT, S, TS),
  tables1(Selector,ListOfFiles, ListOfTitles,
  XLabels, YLabels, IndexX, IndexY, TS, String).

tables2(html, CS, OneT, S, TS):-
sformat(TS, '~s<p>Title: ~w ~n ~s', [CS,OneT,S]).
tables2(latex, CS, OneT, S, TS):-
sformat(TS, '~s ~n \\footnotesize{~w} ~n ~s', [CS,OneT,S]).

tablize(Selector,FileName, XLabels, YLabels, IndexX, IndexY, String) :-
  data_classifying(FileName, IndexX, IndexY),
  tablize_cells(Selector,[''|XLabels], na, "", XSs), length(XLabels, XLength),
  tablize2(Selector, XSs, XLength, NS),
  tablize_rows(Selector, XLabels,YLabels,NS,S),
  tablize3(Selector, S, String).

tablize2(html, XSs, _, NS):-
```

```
sformat(NS, '<table border=0><tr>~s</tr>~n', [XSs]).
tablize2(latex, XSs, XLength, NS):- sformat(NS,
 '  ~n ~n \\medskip ~n ~n \\begin{minipage}{2.5in}\\begin{tiny} ~n
 \\begin{center}
 ~n \\begin{tabular}[t]{|c@{ }c@{}c@{}c@{}c@{}c@{}c@{}|} \\hline ~n &
 \\multicolumn{~w}{c}{Benefit} & \\\\ ~n Cost ~s &  \\\\ \\hline ~n',
  [XLength, XSs]).

tablize3(html, S, String):-  sformat(String, '~s</table>~n',[S]) .
tablize3(latex, S, String):-  sformat(String, '~s \\\\ \\hline ~n  \\end{tabular}
 ~n \\end{center} ~n \\end{tiny}~n\\end{minipage} ~n',[S]) .

tablize_rows(Selector,_,[],S,NewS) :-
  setof(C, Num^reportSlot(C, Num), Cs), max(Cs, Max),
  bagof(SumBagNum,
  BagNum^Num^Count^(between(1, Max, Count),
    (bagof(Num, reportSlot(Count, Num), BagNum),
    sum(BagNum, SumBagNum) )
  ) , WouldItWork),
  tablize_rows1(Selector, S, S0),
  tablize_cells(Selector, WouldItWork, na, S0, NewS),
  retractall(reportSlot(_,_)).

tablize_rows(Selector, X,[Y|Ys],CS,S) :-
  tablize_row(Selector, X,Y,Ss),
  tablize_rows2(Selector, CS, Ss, NS),
  tablize_rows(Selector, X,Ys,NS,S).

tablize_rows1(html, S, S0):- sformat(S0, '~s<td width=30 bgcolor=#ffffff>
&nbsp</td>~n',[S]).
tablize_rows1(latex, S, S0):- sformat(S0, '~s total & ',[S]).


tablize_rows2(html, CS, Ss, NS):- sformat(NS, '~s<tr>~s</tr>~n', [CS,Ss]).
tablize_rows2(latex, CS, Ss, NS):- sformat(NS, '~s ~s \\\\ ~n', [CS,Ss]).

tablize_row(Selector, XLabels, Y, String) :-
  bagof(Count,
    X^(member(X,XLabels), (class_count(X,Y,Count)->true;Count=0)),
    Bag),
  sum(Bag, SumBag), append(Bag, [SumBag], NBag),
  tablize_cells(Selector, [Y|NBag], 0, "", String).

tablize_cells(Selector, [Elem|List], Count, CS, Ss) :-
  cell_color(Selector, Elem, Color),
  (\+ Count=na
   ->(assert(reportSlot(Count, Elem)),NewCount is Count+1)
  ;true),
  tablize_cells1(Selector, CS, Color, Elem, S),
  tablize_cells(Selector, List, NewCount, S,Ss).
tablize_cells(_,[],_,S,S).

tablize_cells1(html, CS, Color, Elem, S):- sformat(S, '~s<td width=30
bgcolor=~w>~w</td>~n', [CS,Color,Elem]).
tablize_cells1(latex, CS, '', Elem, S):- number(Elem),
P is integer(Elem/100),
P==0, sformat(S, '~s & ', [CS]).
```

```
tablize_cells1(latex, CS, '', Elem, S):- sformat(S, '~s ~w & ', [CS,Elem]).
tablize_cells1(latex, CS, Color, Elem, S):- number(Elem),
P is integer(Elem/100),
sformat(S, '~s ~w{~w} & ', [CS,Color,P]).

classify_cell_elem(Axis,Labels,U,L) :-
  length(Labels,Len), Div is (U-L+1)/Len,
  classify_cell_elem1(Axis,Labels,L,Div,U).

classify_cell_elem1(Axis,A,B,C,D):-once(classify_cell_elem2(Axis,A,B,C,D)).
classify_cell_elem2(Axis, [H],L,_,U):- assert(data_classify(Axis,H,L,U)).
classify_cell_elem2(Axis, [H|T],L,Div,U):-
  LL is L+Div, assert(data_classify(Axis, H,L,LL)),
  classify_cell_elem1(Axis, T,LL,Div,U).

data_classifying(FileName, IndexX, IndexY) :-
  retractall(class_count(_X,_Y,_Count)),
  [FileName], result(_,L),
  data_classifying1(L, IndexX, IndexY), fail.
data_classifying(_,_,_).

data_classifying1(L, IndexX, IndexY):-
  nth0(IndexX,L,ElemX),
  fit_class(x,ElemX,ClassX),
  nth0(IndexY,L,ElemY),
  fit_class(y,ElemY,ClassY),
  set_count(ClassX,ClassY).

fit_class(A,B,C) :- once(fit_class1(A,B,C)).

fit_class1(Axis, Elem,Label) :- data_classify(Axis, Label, L, U), Elem>=L,
Elem<U.
fit_class1(Axis, Elem,Label) :- data_classify(Axis, Label, L, L), Elem=L.
fit_class1(_, Elem, blah) :- print(Elem), nl.

set_count(ClassX,ClassY) :-
  clause(class_count(ClassX,ClassY,_), true), !,
  class_count(ClassX,ClassY,Num),
  N is Num+1, retractall(class_count(ClassX,ClassY,Num)),
  assert(class_count(ClassX,ClassY,N)).
set_count(ClassX,ClassY) :-
  assert(class_count(ClassX,ClassY,1)).

cell_color(Selector,A,B) :- once(cell_color1(Selector,A,B)).
cell_color1(html, Elem, '#6699cc') :- \+ number(Elem).
cell_color1(html, Elem, '#663300') :- Elem>=1000.
cell_color1(html, Elem, '#996633') :- Elem>=800, Elem<1000.
cell_color1(html, Elem, '#cc9966') :- Elem>=600, Elem<800.
cell_color1(html, Elem, '#ff9966') :- Elem>=400, Elem<600.
cell_color1(html, Elem, '#ffcc99') :- Elem>=200, Elem<400.
cell_color1(html, Elem, '#ffffcc') :- Elem>=50, Elem<200.
cell_color1(html, Elem, '#fffffc') :- Elem>=1, Elem<50.
cell_color1(html, _Elem, '#ffffff').

cell_color1(latex, Elem, '') :- \+ number(Elem).
cell_color1(latex, 0, '').
cell_color1(latex, Elem, '') :- Num is integer((10000 - Elem)/100), Num==100.
```

```
cell_color1(latex, Elem, A) :- Num is integer((10000 - Elem)/100),
sformat(S, '\\sq{~w}', [Num]), string_to_atom(S, A).

% setup.pl
% stuff for program setup
% changing the content of this file may have
% reduce your lifespan by half.  Beware.

:- multifile
  node_info/3,
  edge_info/5,
  severity/3,
  cost/3,
  of/2,
  (says)/2,
  good_effect/1,
  bad_effect/1.

:- dynamic
  memo/3,
  choice/4,
  roulette/2,
  price/3.

:- dynamic
  enable_jot_down/1, % record things like cost, weight etc for a node
  jotted_down/3.

go(X) :-  reset, reset(records), time(run(X)), !, report(X).
run(X) :- once(lurch(X)).
reset  :- retractall(memo(_,_,_)), retractall(jotted_down(_,_,_)),
retractall(price(_,_,_)).
reset(tar2) :- retractall(curr_class(_,_,_,_)).

reset(records) :-  reset_records(roulette).
reset_records(A) :- once(reset_records1(A)).
reset_records1(PointerKey) :-
  recorded(PointerKey,_),
  bagof([Ref,OwnRef], recorded(PointerKey, Ref, OwnRef), AllRefs),
  flatten(AllRefs, F), checklist(reset_records2, F).
reset_records1(_).

reset_records2(Ref):- catch(erase(Ref), _, true).

fake_memo(Node) :-
   choices(Node, Index), bassert(memo(Index, Node, 0)).

welcome :-
  format('~s~n~s~n~s~n~s~n~s~n~s~n~s~n~s~n', [
"---->'?- readme' --- for notes; ",
"---->'?- demo(d)' and demo(a)
(please halt and restart prolog for different demo) for examples; ",
"---->'?- go(TopLevelGoalName)' for inference after loading definition file;",
"---->......definition files are under folder /defs; ",
"---->......refer to main.pl for descriptions of all pl files;",
"---->......refer to todos for a list of (my) todos",
"---->......graph will be opened by your windows' default viewer;"
]).

% tar2_macro.pl
% all about running macro-cycle and generating corresponding report

:- load_files([main],
  [silent(true)]).

 macro_print_type(text).
 macro_output_filename('macro_result.out').
% input:
% 1. list of tar2 recommanded actions in this format: [[a1, a2], [b1,b2]|Rest]
% 2. # lurches for each recommanded action(s)
% 3.  attribute type (e.g. activities) to be presented
% output:
% 1. present frequency counts for each bottom leaves of the graph

 :- [defs/cara_analysis], [defs/cara_analysis_utils].

macro_lurches(X, Constraints, Runs, Bs,Cs,Outputs) :-
  macro_lurches(X, Constraints, 0, Runs, [], Bs,[], Cs,[], Outputs).
macro_lurches(_,_, R,R, Bs,Bs,Cs,Cs,Outputs,Outputs):-!.
macro_lurches(X, Constraints, Curr, Runs, CurrBs, NewBs,CurrCs,
NewCs, CurrOutputs, NewOutputs):-
  macro_lurch(X,B,C,Out),
  once(macro_lurch_case(Constraints, Curr, NewCurr, B, Bs, NewBs,
  C, Cs, NewCs, Out, Outputs, NewOutputs)),
  macro_lurches(X, Constraints, NewCurr, Runs, CurrBs, Bs,CurrCs,
  Cs, CurrOutputs, Outputs).

macro_lurch_case(Constraints, Curr, NewCurr, B, Bs, [B|Bs], C, Cs,
[C|Cs], Out, Outputs, [Out|Outputs]):-
  forall(member(O,Constraints), memo(_,O,0)), !, NewCurr is Curr +1.
macro_lurch_case(_,Curr,Curr, _, Bs, Bs, _, Cs, Cs,_,Outputs,Outputs).

macro_lurch(X,B,C,Out) :-
  reset, reset(records),
  run(X), !,
  costs(C), benefit(X,B),
  bagof(One, Aˆmemo(A,One,0), Out).

macro_getAttrFreqCount(AllOuts,AttrFreqCount):-
  flatten(AllOuts, Flattened),
  sort(Flattened, SortedEmperical),
  maplist(macro_getfcount(Flattened), SortedEmperical, AttrFreqCount).

macro_getfcount(List, Attr, fcount(Count,Attr)) :-
  bagof(I, nth0(I, List,Attr), Total), length(Total, Count).

macro_printOne(Bs, Cs, Outs) :-
  average(Bs, AvegB), average(Cs, AvegC),
  macro_getAttrFreqCount(Outs, AttrFreqCountO),
  msort(AttrFreqCountO, AttrFreqCount),
  % sieve out rules and stuff
  Wanted=fcount(C,Attr),
  findall(Wanted,
  AˆBˆ(member(Wanted, AttrFreqCount), (Attr=of(A,B);Attr=not(of(A,B)))),
  WantedAttrFreqCounts),
  macro_output_filename(FileName),
  append(FileName),
  ignore((
    macro_print('Average Benefits for all runs', AvegB),
    macro_print('Average Costs for all runs', AvegC),
    checklist(macro_print_fcount, WantedAttrFreqCounts)
  )),
  told.

macro_print(Text) :-
   macro_print_type(text),
   format('~t~w: ~t~40|~n', [Text]).
macro_print(Text, Thing) :-
   macro_print_type(text),
   format('~t~w: ~t~40|~w~n', [Text, Thing]).

macro_print_fcount(fcount(Count, Attr)):-
  macro_print('Frequency Count for:', Attr=Count).

tests:-
  macro_lurches_all(riskMinimization of system,
  [
  [dav04 of dal] ,
  [dav05 of dal] ,
  [dav06 of dal] ,
  [dav07 of dal] ,
  [dav08 of dal] ,
  [dav09 of dal] ,
  [cav06 of cal] ,
  [cav01 of cal] ,
  [cav02 of cal] ,
  [cav03 of cal] ,
  [cav04 of cal] ,
  [cav05 of cal] ,
  [dav01 of dal] ,
  [dav02 of dal] ,
  [tav08 of tal] ,
  [tav09 of tal] ,
  [dav03 of dal]
     ],
    100).

macro_lurches_all(X, ConstraintsLists, RunsForEachConstraints):-
  macro_output_filename(FileName) ,
  tell(FileName),
  macro_print('Report: lurching',X),
  macro_print('============================================'),
  told,
  forall(member(Constraints, ConstraintsLists),
  (
    append(FileName),
    macro_print('lurching: ',Constraints), told,
    macro_lurches(X,Constraints,RunsForEachConstraints,B,C,O),
    macro_printOne(B,C,O),
    append(FileName),
    macro_print('------------------------------------------------------'), told
  )
  ).

% utils.pl
% misc utilities including textual output
% for TAR2

% -------------------- all about intermediate files to be manipulated ---%

make_results(A,B,C):- make_results(A,[],B,C).
make_results(Args, ConstraintList, NumOfTimes, File):-
  reset(tar2),
  gen_tar2_filename(File, '.pl', F),
  tell(F), told, % initialize file... any better way to do this??
  forall(
    between(1,NumOfTimes, N),
    (make_result(Args,ConstraintList, Out), append(F),
    format('result(~w,~w).~n', [N, Out]),  told)
  ).

make_result(X,ConstraintList, [B,C|Out]) :-
  reset, reset(records),
  checklist(fake_memo, ConstraintList),
  run(X), !,
  costs(C), benefit(X,B),
  make_result_list(Out).
make_result_list(Out) :-
  attributes(L), maplist(done, L, Out).

% -------------------- all about writing tar2 files ------------------------------%

tar2_analyse_pl(File) :-
  gen_tar2_filename(File, '.pl', F),[F],
  tar2_analyse_pl1( 0, benefit),
  tar2_analyse_pl1( 1, cost).
%  tar2_analyse_finishup.

tar2_analyse_pldata :-
  tar2_pencentile_chop(Function),
  once(tar2_analyse_pldata1(Function)).

tar2_analyse_pldata1('seperate'):-
  tar2_analyse_pl1( 0, benefit),
  tar2_analyse_pl1( 1, cost).

tar2_analyse_pldata1(Function):-
  tar2_analyse_pldata1_findAllClassNames(AllClassNames),
  tar2_analyse_pldata1_calculateResults(AllOuts),
  msort(AllOuts, SortedAllOuts), last(L,SortedAllOuts), print(L), nl,
  length(AllOuts, Len), length(AllClassNames, NumClasses),
  Div is truncate(Len/NumClasses),
  tar2_analyse_assertClass( Function, AllClassNames, SortedAllOuts, Div).

tar2_analyse_pldata1_findAllClassNames(FormattedAll) :-
  bagof(ClassNames,
  ClassType^(bagof(ClassName, (class(ClassType, V, ClassName), var(V)),
  ClassNames)),
  Attrs),
  tar2_analyse_find_variations(Attrs, AllClassNames),
  maplist(tar2_analyse_format_classname, AllClassNames, FormattedAll).

tar2_analyse_pldata1_calculateResults(AllOuts):-
  classify_function(Function, ArithList),
  function_param_location(Function, Locations),
  bagof(
  OneOut,
    Count^ParamsForOneResult^(maplist(tar2_analyse_pldata1_getone(Count),
    Locations, ParamsForOneResult),
    tar2_analyse_calculate(ParamsForOneResult, ArithList, OneOut)),
  AllOuts).

tar2_analyse_pldata1_getone(Count, Name=Location, Name=Param) :-
  result(Count,L), nth0(Location, L, Param).

tar2_analyse_pl1( Index, ClassType) :-
  tar2_analyse_sortClass(Index, ClassType, ClassNames, MSortedData, Div),
  tar2_analyse_assertClass( ClassType, ClassNames, MSortedData, Div).

tar2_analyse_sortClass(Index, BorC, Classes, MSBs, Div):-
  bagof(Elem, Count^L^(result(Count,L), nth0(Index, L, Elem)), Elems),
  msort(Elems, MSBs), % get sorted benefits
  findall(Class, V^(class(BorC,_,Class), var(V)), Classes),
  % get all benefit class types
  length(Elems, Len), length(Classes, NumClasses),
  Div is truncate(Len/NumClasses).

tar2_analyse_assertClass( Name, A,B,C):-
  tar2_analyse_assertClass( Name, A,B,C,0,C).
tar2_analyse_assertClass( Name, [One],MSBs,_,MinP,_) :-
  last(Max,MSBs), nth0(MinP,MSBs,Min),
  bassert(curr_class(Name, Min, Max, One)).
tar2_analyse_assertClass( Name, [BC|BCs], MSBs, Div, MinP, MaxP):-
  nth0(MinP,MSBs,Min), nth0(MaxP,MSBs,Max),
  bassert(curr_class(Name, Min, Max, BC)),
  NMinP is MaxP+1, NMaxP is MaxP+Div,
  tar2_analyse_assertClass( Name, BCs, MSBs, Div, NMinP, NMaxP).
tar2_analyse_assertClass(_,[],_,_,_,_).

tar2_analyse_finishup :- tar2_pencentile_chop(A),
once(tar2_analyse_finishup1(A)).

tar2_analyse_finishup1('seperate').

tar2_analyse_finishup1(ClassifyType):-
  setof(ClassType, A^B^C^curr_class(ClassType,A,B,C), ClassTypes),
  bagof(Attr,
    (member(OneClassType, ClassTypes),
    setof(
      OneClassType/Min/Max/ClassName,
      OneClassType^curr_class(OneClassType, Min, Max, ClassName),
      Attr)
    ),
    Attrs),
  tar2_analyse_finishup2(ClassifyType, Attrs).
```

```
tar2_analyse_finishup2(ClassifyType, Attrs):-
  tar2_analyse_find_variations(Attrs, Variations),
  classify_function(ClassifyType, List),
  forall(member(V, Variations), tar2_analyse_finishup3(ClassifyType, List, V)).

tar2_analyse_find_variations(Attrs, Variations):-
  bagof(V, tar2_analyse_find_variation(Attrs,V), Variations).

tar2_analyse_find_variation(Attrs, V):-
    tar2_analyse_find_variation(Attrs, [], V).
 tar2_analyse_find_variation([],O,O).
 tar2_analyse_find_variation([Attr|Attrs], Vs, [One|Out]) :-
   member(One, Attr), tar2_analyse_find_variation(Attrs, Vs, Out).

tar2_analyse_finishup3(ClassifyType, List, ParamSets) :-
  bagof(
    ClassType=Min,
    Max^ClassName^member(ClassType/Min/Max/ClassName, ParamSets),
    ParamMin),
  bagof(
    ClassType=Max,
    Min^ClassName^member(ClassType/Min/Max/ClassName, ParamSets),
    ParamMax),
  setof(ClassName, A^B^C^member(A/B/C/ClassName, ParamSets),
  ClassNames),
  msort(ClassNames, CNS),
  tar2_analyse_format_classname(CNS, FormattedCNS),
  tar2_analyse_calculate(ParamMin,List,Min),
  tar2_analyse_calculate(ParamMax,List,Max),
  msort([Min,Max], [NMin,NMax]),
  bassert(curr_class(ClassifyType, NMin, NMax,FormattedCNS)).

tar2_analyse_format_classname(ClassName, Formatted) :-
  tar2_analyse_format_classname(ClassName, '', Formatted).

tar2_analyse_format_classname([OneC], Curr, Out) :-
  atom_concat(OneC, Curr, Out).
tar2_analyse_format_classname([OneC|Cs], Curr, Out) :-
  tar2_analyse_format_classname(Cs, Curr, O1),
  atom_concat('=',O1,O2), atom_concat(OneC,O2,Out).

tar2_analyse_calculate(A,B,C):-
  once(tar2_analyse_calculate1(A,B,C)).
tar2_analyse_calculate1(_, List, List):- \+ is_list(List).
tar2_analyse_calculate1(Params, List, Out) :-
% Params=[cost=1, benefit=2.34]
  checklist(atomic, List),
  tar2_analyse_calculate3(Params, List, Out).

tar2_analyse_calculate1(Params, List, Out):-
  maplist(tar2_analyse_calculate1(Params), List, NewList),
  tar2_analyse_calculate3(Params, NewList, Out).

tar2_analyse_calculate3(Params, List, Result):-
  maplist(tar2_analyse_calculate4(Params), List, Callable),
  Unif =.. Callable, call(Unif,Result).
```

```
tar2_analyse_calculate4(Params, ClassType, Number):-
  member(ClassType=Number, Params).

tar2_analyse_calculate4(_, Others, Others).

tar2_write_name_file(File) :-
  gen_tar2_filename(File, '.names', FileName),
  tar2_pencentile_chop(FunctionName),
  bagof(C, X0^X1^(classify(FunctionName,X0,X1,C)), [HC|Cs]),
  attributes(L),
  bagof(One=AType, (member(One, L), attribute_type(One,AType)), LL),
  tell(FileName), ignore(( % absolute laziness...
    format('~w', [HC]), forall(member(OneC,Cs), format(',~w',[OneC])),
    format('~n'), forall(member(O=A, LL), format('~w: ~t~10|~w~n',[O,A]))
  )), told.

tar2_write_data_file(File) :-
  gen_tar2_filename(File, '.data', FileName),
  tell(FileName), told,
  tar2_pencentile_chop(FunctionName),
  forall(
    result(_,[B,C|R]),
    (  classify(FunctionName,C,B,CB),
      append(FileName),
      checklist(tar2_write_data_file1, R),
      format('~w.~n', [CB]),
      told
    )
  ).

tar2_write_data_file1(B) :- format('~w, ~t', [B]).

gen_tar2_filename(File, Extention, FileName):-
  tar2_directory(D), (\+ exists_directory(D)->make_directory(D);true),
  atom_concat(D, '/', DD),
  atom_concat(DD,File,DF), atom_concat(DF, Extention, FileName).
```

A.2 Sample Frameworks

A.2.1 Web Browser Selection

```
softgoal goodBrowser.
softgoal performance of browser is critical.
softgoal userFriendly of browser.
softgoal security of browser is critical.
softgoal speed of browser.
softgoal stability of browser.
softgoal opera7.
softgoal msie6.
softgoal netscape6.

claim 'frequently crashes my PC'.

stakeholder rule1.
stakeholder rule2.
stakeholder rule3.
stakeholder rule4.
stakeholder rule5.
stakeholder rule6.

rule1 says goodBrowser if
  performance of browser
  any_of userFriendly of browser
  any_of security of browser
  ref "I made it up" keywords [eliza].

rule2 says performance of browser if
  speed of browser
  any_of stability of browser
  ref "I made it up" keywords [eliza].

rule3 says speed of browser if
  made by opera7
  any_of unhurt by msie6
  any_of unbroken by netscape6
  ref "I made it up" keywords [eliza].

rule4 says stability of browser if
  helped by opera7
  any_of unhurt by msie6
    with helped by ’frequently crashes my PC’
  any_of unbroken by netscape6
  ref "I made it up" keywords [eliza].

rule5 says userFriendly of browser if
  unhurt by opera7
  any_of made by msie6
  any_of helped by netscape6
  ref "I made it up" keywords [eliza].

rule6 says security of browser if
  unhurt by opera7
  any_of unbroken by msie6
  any_of unbroken by netscape6
  ref "I made it up" keywords [eliza].
```

A.2.2 *KWIC*

(Rigorous Quality Assurance)

```
softgoal coherence of system.
softgoal comprehensibility of system.
softgoal simplicity of system.
softgoal deletability of function is critical.
softgoal extensibility of function is critical.
softgoal modifiability of process is critical.
softgoal modifiability of system is critical.
softgoal modifiability of function is critical.
softgoal performance of system is critical.
softgoal reusability of system is critical.
softgoal spacePerformance of system is critical.
softgoal timePerformance of system is critical.
softgoal updatability of function is critical.
softgoal modifiability of dataRep is veryCritical.
softgoal abstractDataType of targetSystem.
softgoal implicitInvocation of targetSystem.
softgoal pipeAndFilter of targetSystem.
softgoal sharedData of targetSystem.

claim c1.
claim c2.
claim c3.
claim c4.
claim c5.

stakeholder rule0.
stakeholder rule1.
stakeholder rule2.
stakeholder rule3.
stakeholder rule4.
stakeholder rule5.
stakeholder rule6.
stakeholder rule7.
stakeholder rule8.
stakeholder rule9.
stakeholder rule10.
stakeholder rule11.
stakeholder rule12.
stakeholder rule13.

rule0 says goodness of system if
modifiability of system
any_of comprehensibility of system
any_of performance of system
any_of reusability of system
ref "L Chung’s book" keywords [chung].

rule1 says modifiability of system if
helped by c1
all_of modifiability of process
all_of modifiability of dataRep
all_of modifiability of function
ref "L Chung’s book" keywords [chung].

rule2 says comprehensibility of system if
coherence of system
all_of simplicity of system
ref "L Chung’s book" keywords [chung].

rule3 says performance of system if
helped by c1
all_of spacePerformance of system
all_of  timePerformance of system
ref "L Chung’s book" keywords [chung].

rule4 says modifiability of function if
extensibility of function
all_of deletability of function
all_of updatability of function
ref "L Chung's book" keywords [chung].

rule5 says coherence of system if
helped by sharedData of targetSystem
ref "L Chung's book" keywords [chung].

rule6 says extensibility of function if
made by implicitInvocation of targetSystem
any_of helped by sharedData of targetSystem
any_of unhurt by abstractDataType of targetSystem
with unhurt by c5
ref "L Chung's book" keywords [chung].

rule7 says simplicity of system if
made by pipeAndFilter of targetSystem
ref "L Chung's book" keywords [chung].

rule8 says modifiability of process if
unhurt by abstractDataType of targetSystem
any_of unbroken by sharedData of targetSystem
with helped by c2
ref "L Chung's book" keywords [chung].

rule9 says modifiability of dataRep if
unhurt by implicitInvocation of targetSystem
any_of helped by abstractDataType of targetSystem
any_of unbroken by pipeAndFilter of targetSystem
any_of unhurt by sharedData of targetSystem
with helped by c2
ref "L Chung's book" keywords [chung].

rule10 says reusability of system if
helped by pipeAndFilter of targetSystem
any_of helped by implicitInvocation of targetSystem
any_of unhurt by sharedData of targetSystem
any_of helped by abstractDataType of targetSystem
with helped by c3
ref "L Chung's book" keywords [chung].

rule11 says spacePerformance of system if
unhurt by implicitInvocation of targetSystem
any_of made by sharedData of targetSystem
any_of unbroken by pipeAndFilter of targetSystem
with made by c4
ref "L Chung's book" keywords [chung].

rule12 says timePerformance of system if
unbroken by implicitInvocation of targetSystem
any_of unhurt by abstractDataType of targetSystem
ref "L Chung's book" keywords [chung].

rule13 says updatability of function if
helped by pipeAndFilter of targetSystem
any_of helped by abstractDataType of targetSystem
ref "L Chung's book" keywords [chung].
```

A.2.3 Choice of Software Organization Styles

Rigorous Quality Assurance

```
softgoal goodness of system.

softgoal availability of system.
softgoal security of system is critical.
softgoal adaptability of system.
softgoal integrity of system is critical.
softgoal authorization of system.
softgoal dynamicity of system.
softgoal evolvability of system.

softgoal accuracy of system.
softgoal completness of system.
softgoal usability of system.
softgoal responseTime of system.
softgoal identification of system.
softgoal authentication of system.
softgoal validation of system.
softgoal confidentiality of system.
softgoal externalConsistency of system.
softgoal run_timeMaintainability of system.
softgoal run_timeModifiability of system.
softgoal extensibility of system.
softgoal updatability of system.

softgoal pyramid of archPattern costed high.
softgoal jointVenture of archPattern costed high.
softgoal co_optation of archPattern costed high.

claim ’external agents can aquire trusted information’.
claim ’possible conflicts between responseTime and security’.
claim ’possible conflicts between security and adaptability’.

stakeholder rule1.
stakeholder rule2.
stakeholder rule3.
stakeholder rule4.
stakeholder rule5.
stakeholder rule6.
stakeholder rule7.
stakeholder rule8.
stakeholder rule_a.
stakeholder rule_b.
stakeholder rule_c.
stakeholder rule_d.
stakeholder rule_e.
stakeholder rule_f.
stakeholder rule_g.
stakeholder rule_h.
stakeholder rule_i.
stakeholder rule_j.
stakeholder rule_k.
stakeholder rule_l.
stakeholder rule_m.
```

```
rule1 says goodness of system if
  availability of system
  any_of security of system
  any_of adaptability of system
  ref "Figure 8 in  [Jaelson02]" keywords [level_0].

rule2 says availability of system if
  integrity of system
  any_of usability of system
  any_of responseTime of system
  ref "Figure 8 in  [Jaelson02]" keywords [level_1, level_2].

rule3 says integrity of system if
  accuracy of system
  all_of completness of system
  ref "Figure 8 in  [Jaelson02]" keywords [level_2].

rule4 says security of system if
  authorization of system
  any_of confidentiality of system
  any_of externalConsistency of system
  any_of made by availability of system
  any_of helped by 'possible conflicts between responseTime and security'
  any_of helped by 'possible conflicts between security and adaptability'
  ref "Figure 8 in  [Jaelson02]" keywords [level_1, level_2].

rule5 says authorization of system if
  identification of system
  all_of authentication of system
  all_of validation of system
  ref "Figure 8 in  [Jaelson02]" keywords [level_2].

rule6 says adaptability of system if
  dynamicity of system
  any_of updatability of system
  any_of helped by 'possible conflicts between security and adaptability'
  ref "Figure 8 in  [Jaelson02]" keywords [level_1, level_2].

rule7 says dynamicity of system if
  run_timeMaintainability of system
  all_of evolvability of system
  ref "Figure 8 in  [Jaelson02]" keywords [level_1, level_2].

rule8 says evolvability of system if
  run_timeModifiability of system
  all_of extensibility of system
  ref "Figure 8 in  [Jaelson02]" keywords [level_2].

rule_a says accuracy of system if
  made by pyramid of archPattern
  any_one_of helped by jointVenture of archPattern
  any_one_of helped by authorization of system
  ref "Figure 8 in  [Jaelson02]" keywords [level_3].

rule_b says completness of system if
  made by jointVenture of archPattern
  any_one_of unbroken by co_optation of archPattern
  ref "Figure 8 in  [Jaelson02]" keywords [level_3].

rule_c says usability of system if
  helped by pyramid of archPattern
  any_one_of made by jointVenture of archPattern
  any_one_of unhurt by co_optation of archPattern
  ref "Figure 8 in  [Jaelson02]" keywords [level_3].

rule_d says responseTime of system if
  helped by ’possible conflicts between responseTime and security’
  any_one_of helped by jointVenture of archPattern
  any_one_of helped by co_optation of archPattern
  ref "Figure 8 in  [Jaelson02]" keywords [level_3].

rule_e says identification of system if
  helped by pyramid of archPattern
  any_one_of helped by jointVenture of archPattern
    with made by ’external agents can aquire trusted information’
  ref "Figure 8 in  [Jaelson02]" keywords [level_3].

rule_f says authentication of system if
  helped by jointVenture of archPattern
  any_one_of unbroken by co_optation of archPattern
  ref "Figure 8 in  [Jaelson02]" keywords [level_3].

rule_g says validation of system if
  helped by jointVenture of archPattern
  ref "Figure 8 in  [Jaelson02]" keywords [level_3].

rule_h says confidentiality of system if
  made by pyramid of archPattern
    with made by ’external agents can aquire trusted information’
  any_one_of helped by jointVenture of archPattern
  any_one_of unbroken by co_optation of archPattern
  ref "Figure 8 in  [Jaelson02]" keywords [level_3].

rule_i says externalConsistency of system if
  unhurt by co_optation of archPattern
    with unhurt by ’external agents can aquire trusted information’
  ref "Figure 8 in  [Jaelson02]" keywords [level_3].

rule_j says run_timeMaintainability of system if
  helped by jointVenture of archPattern
  ref "Figure 8 in  [Jaelson02]" keywords [level_3].

rule_k says run_timeModifiability of system if
  helped by pyramid of archPattern
  any_one_of helped by co_optation of archPattern
  ref "Figure 8 in  [Jaelson02]; note:  stripped out
  arc between run_timeModifiability and usability
  because it seemed to be an error" keywords [level_3].

rule_l says extensibility of system if
  made by pyramid of archPattern
  any_one_of made by co_optation of archPattern
  ref "Figure 8 in  [Jaelson02]" keywords [level_3].
```

```
rule_m says updatability of system if
  made by jointVenture of archPattern
  ref "Figure 8 in  [Jaelson02]" keywords [level_3].

pick_one_in([
  pyramid of archPattern,
  jointVenture of archPattern,
  co_optation of archPattern
  ]).
```

Weak Quality Assurance

```
softgoal goodness of system.

softgoal availability of system.
softgoal security of system is critical.
softgoal adaptability of system.
softgoal integrity of system is critical.
softgoal authorization of system.
softgoal dynamicity of system.
softgoal evolvability of system.

softgoal accuracy of system.
softgoal completness of system.
softgoal usability of system.
softgoal responseTime of system.
softgoal identification of system.
softgoal authentication of system.
softgoal validation of system.
softgoal confidentiality of system.
softgoal externalConsistency of system.
softgoal run_timeMaintainability of system.
softgoal run_timeModifiability of system.
softgoal extensibility of system.
softgoal updatability of system.

softgoal pyramid of archPattern costed high.
softgoal jointVenture of archPattern costed high.
softgoal co_optation of archPattern costed high.

claim ’external agents can aquire trusted information’.
claim ’possible conflicts between responseTime and security’.
claim ’possible conflicts between security and adaptability’.

stakeholder rule1.
stakeholder rule2.
stakeholder rule3.
stakeholder rule4.
stakeholder rule5.
stakeholder rule6.
stakeholder rule7.
stakeholder rule8.
stakeholder rule_a.
stakeholder rule_b.
stakeholder rule_c.
stakeholder rule_d.
stakeholder rule_e.
stakeholder rule_f.
stakeholder rule_g.
stakeholder rule_h.
stakeholder rule_i.
stakeholder rule_j.
stakeholder rule_k.
stakeholder rule_l.
stakeholder rule_m.

rule1 says goodness of system if
  availability of system
  any_one_of security of system
  any_one_of adaptability of system
  ref "Figure 8 in  [Jaelson02]" keywords [level_0].

rule2 says availability of system if
  integrity of system
  any_one_of usability of system
  any_one_of responseTime of system
  ref "Figure 8 in  [Jaelson02]" keywords [level_1, level_2].

rule3 says integrity of system if
  accuracy of system
  any_one_of completness of system
  ref "Figure 8 in  [Jaelson02]" keywords [level_2].

rule4 says security of system if
  authorization of system
  any_one_of confidentiality of system
  any_one_of externalConsistency of system
  any_one_of made by availability of system
  any_one_of helped by ’possible conflicts between responseTime and security’
  any_one_of helped by ’possible conflicts between security and adaptability’
  ref "Figure 8 in  [Jaelson02]" keywords [level_1, level_2].

rule5 says authorization of system if
  identification of system
  any_one_of authentication of system
  any_one_of validation of system
  ref "Figure 8 in  [Jaelson02]" keywords [level_2].

rule6 says adaptability of system if
  dynamicity of system
  any_one_of updatability of system
  any_one_of helped by ’possible conflicts between security and adaptability’
  ref "Figure 8 in  [Jaelson02]" keywords [level_1, level_2].

rule7 says dynamicity of system if
  run_timeMaintainability of system
  any_one_of evolvability of system
  ref "Figure 8 in  [Jaelson02]" keywords [level_1, level_2].

rule8 says evolvability of system if
```

```
  run_timeModifiability of system
  any_one_of extensibility of system
  ref "Figure 8 in  [Jaelson02]" keywords [level_2].

rule_a says accuracy of system if
  made by pyramid of archPattern
  any_one_of helped by jointVenture of archPattern
  any_one_of helped by authorization of system
  ref "Figure 8 in  [Jaelson02]" keywords [level_3].

rule_b says completness of system if
  made by jointVenture of archPattern
  any_one_of unbroken by co_optation of archPattern
  ref "Figure 8 in  [Jaelson02]" keywords [level_3].

rule_c says usability of system if
  helped by pyramid of archPattern
  any_one_of made by jointVenture of archPattern
  any_one_of unhurt by co_optation of archPattern
  ref "Figure 8 in  [Jaelson02]" keywords [level_3].

rule_d says responseTime of system if
  helped by ’possible conflicts between responseTime and security’
  any_one_of helped by jointVenture of archPattern
  any_one_of helped by co_optation of archPattern
  ref "Figure 8 in  [Jaelson02]" keywords [level_3].

rule_e says identification of system if
  helped by pyramid of archPattern
  any_one_of helped by jointVenture of archPattern
    with made by ’external agents can aquire trusted information’
  ref "Figure 8 in  [Jaelson02]" keywords [level_3].

rule_f says authentication of system if
  helped by jointVenture of archPattern
  any_one_of unbroken by co_optation of archPattern
  ref "Figure 8 in  [Jaelson02]" keywords [level_3].

rule_g says validation of system if
  helped by jointVenture of archPattern
  ref "Figure 8 in  [Jaelson02]" keywords [level_3].

rule_h says confidentiality of system if
  made by pyramid of archPattern
    with made by ’external agents can aquire trusted information’
  any_one_of helped by jointVenture of archPattern
  any_one_of unbroken by co_optation of archPattern
  ref "Figure 8 in  [Jaelson02]" keywords [level_3].

rule_i says externalConsistency of system if
  unhurt by co_optation of archPattern
    with unhurt by ’external agents can aquire trusted information’
  ref "Figure 8 in  [Jaelson02]" keywords [level_3].

rule_j says run_timeMaintainability of system if
  helped by jointVenture of archPattern
  ref "Figure 8 in  [Jaelson02]" keywords [level_3].
```

```
rule_k says run_timeModifiability of system if
  helped by pyramid of archPattern
  any_one_of helped by co_optation of archPattern
  ref "Figure 8 in  [Jaelson02]; note:  stripped out arc
  between run_timeModifiability and usability because
  it seemed to be an error" keywords [level_3].

rule_l says extensibility of system if
  made by pyramid of archPattern
  any_one_of made by co_optation of archPattern
  ref "Figure 8 in  [Jaelson02]" keywords [level_3].

rule_m says updatability of system if
  made by jointVenture of archPattern
  ref "Figure 8 in  [Jaelson02]" keywords [level_3].

pick_one_in([
  pyramid of archPattern,
  jointVenture of archPattern,
  co_optation of archPattern
  ]).
```

A.3 TAR2 Output

```
-------------------------------------------------------------------
                Welcome to TARZAN (Version 2.2)
        Copyright (c) 2001 Tim Menzies (tim@menzies.com)
              Copy policy: GPL-2 (see www.gnu.org)

...while on high, our hero watches for the right chance to strike!
-------------------------------------------------------------------

NOW: specified
CHANGES: specified
default: SCORE for Classes not specified

Read 10000 cases (16 attributes) from browser.data

 Worth=1.000000
 Granularity=3 Promising=-10000000.000000 Useful=1.000000 nChanges=1
 Treatment:[No Treatment]

 three_cost=vlow_benefit:                                [     0 -  0%]
 three_cost=low_benefit:                                [     0 -  0%]
 two_cost=vlow_benefit:                                [     0 -  0%]
 three_cost=high_benefit:                                [     0 -  0%]
 one_cost=vlow_benefit:                                [     0 -  0%]
 three_cost=vhigh_benefit:                                [     0 -  0%]
 zero_cost=vlow_benefit:~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~  [  3357 - 34%]
 two_cost=low_benefit:~~~~~~~~~                       [  1086 - 11%]
 two_cost=high_benefit:~~~~~~~~~~~~                    [  1381 - 14%]
 one_cost=low_benefit:~~~~                            [   558 -  6%]
```

```
 two_cost=vhigh_benefit:~~~~~~~                         [   833 -  8%]
 zero_cost=low_benefit:                                [     0 -  0%]
 one_cost=high_benefit:~~~~~~~~~~                      [  1119 - 11%]
 one_cost=high_benefit:                                [     0 -  0%]
 one_cost=vhigh_benefit:~~~~~~~~~~~~~~                  [  1666 - 17%]
 zero_cost=high_benefit:                                [     0 -  0%]
 zero_cost=vhigh_benefit:                                [     0 -  0%]

Confidence1 Distribution:

 -130601:~~~~~~~~~~                       [     1 - 13%]
 -130255:~~~~~~~~~~~~~~~~~~~~             [     2 - 25%]
 -121404:~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~   [     3 - 38%]
 -111827:~~~~~~~~~~                       [     1 - 13%]
   0:~~~~~~~~~~                       [     1 - 13%]

Treatments learned on browser.data:
worth=1.432990 [msie6=n]
worth=1.432990 [netscape6=n]
worth=1.432990 [frequently crashes my PC=y]
worth=2.852924 [opera7=y]

 Worth=1.432990
 Granularity=3 Promising=-10000000.000000 Useful=1.000000 nChanges=1
 Treatment:[
msie6=n]

 three_cost=vlow_benefit:                                [     0 -  0%]
 three_cost=low_benefit:                                [     0 -  0%]
 two_cost=vlow_benefit:                                [     0 -  0%]
 three_cost=high_benefit:                                [     0 -  0%]
 one_cost=vlow_benefit:                                [     0 -  0%]
 three_cost=vhigh_benefit:                                [     0 -  0%]
 zero_cost=vlow_benefit:~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~  [  3357 - 50%]
 two_cost=low_benefit:                                [     0 -  0%]
 two_cost=high_benefit:                                [     0 -  0%]
 one_cost=low_benefit:~~~~                            [   558 -  8%]
 two_cost=vhigh_benefit:                                [     0 -  0%]
 zero_cost=low_benefit:                                [     0 -  0%]
 one_cost=high_benefit:~~~~~~~~~~                      [  1119 - 17%]
 one_cost=high_benefit:                                [     0 -  0%]
 one_cost=vhigh_benefit:~~~~~~~~~~~~~~                  [  1666 - 25%]
 zero_cost=high_benefit:                                [     0 -  0%]
 zero_cost=vhigh_benefit:                                [     0 -  0%]

 Worth=1.432990
 Granularity=3 Promising=-10000000.000000 Useful=1.000000 nChanges=1
 Treatment:[
netscape6=n]

 three_cost=vlow_benefit:                                [     0 -  0%]
 three_cost=low_benefit:                                [     0 -  0%]
 two_cost=vlow_benefit:                                [     0 -  0%]
 three_cost=high_benefit:                                [     0 -  0%]
 one_cost=vlow_benefit:                                [     0 -  0%]
 three_cost=vhigh_benefit:                                [     0 -  0%]
 zero_cost=vlow_benefit:~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~  [  3357 - 50%]
```

```
 two_cost=low_benefit:                               [     0 -  0%]
 two_cost=high_benefit:                               [     0 -  0%]
 one_cost=low_benefit:~~~~                           [   558 -  8%]
 two_cost=vhigh_benefit:                               [     0 -  0%]
 zero_cost=low_benefit:                               [     0 -  0%]
 one_cost=high_benefit:~~~~~~~~~~                     [  1119 - 17%]
 one_cost=high_benefit:                               [     0 -  0%]
 one_cost=vhigh_benefit:~~~~~~~~~~~~~~~                [  1666 - 25%]
 zero_cost=high_benefit:                               [     0 -  0%]
 zero_cost=vhigh_benefit:                               [     0 -  0%]

 Worth=1.432990
 Granularity=3 Promising=-10000000.000000 Useful=1.000000 nChanges=1
 Treatment:[
frequently crashes my PC=y]

 three_cost=vlow_benefit:                               [     0 -  0%]
 three_cost=low_benefit:                               [     0 -  0%]
 two_cost=vlow_benefit:                               [     0 -  0%]
 three_cost=high_benefit:                               [     0 -  0%]
 one_cost=vlow_benefit:                               [     0 -  0%]
 three_cost=vhigh_benefit:                               [     0 -  0%]
 zero_cost=vlow_benefit:~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~  [  3357 - 50%]
 two_cost=low_benefit:                               [     0 -  0%]
 two_cost=high_benefit:                               [     0 -  0%]
 one_cost=low_benefit:~~~~                           [   558 -  8%]
 two_cost=vhigh_benefit:                               [     0 -  0%]
 zero_cost=low_benefit:                               [     0 -  0%]
 one_cost=high_benefit:~~~~~~~~~~                     [  1119 - 17%]
 one_cost=high_benefit:                               [     0 -  0%]
 one_cost=vhigh_benefit:~~~~~~~~~~~~~~~                [  1666 - 25%]
 zero_cost=high_benefit:                               [     0 -  0%]
 zero_cost=vhigh_benefit:                               [     0 -  0%]

 Worth=2.852924
 Granularity=3 Promising=-10000000.000000 Useful=1.000000 nChanges=1
 Treatment:[
opera7=y]

 three_cost=vlow_benefit:                               [     0 -  0%]
 three_cost=low_benefit:                               [     0 -  0%]
 two_cost=vlow_benefit:                               [     0 -  0%]
 three_cost=high_benefit:                               [     0 -  0%]
 one_cost=vlow_benefit:                               [     0 -  0%]
 three_cost=vhigh_benefit:                               [     0 -  0%]
 zero_cost=vlow_benefit:                               [     0 -  0%]
 two_cost=low_benefit:                               [     0 -  0%]
 two_cost=high_benefit:                               [     0 -  0%]
 one_cost=low_benefit:~~~~~~~~~~                     [   558 - 17%]
 two_cost=vhigh_benefit:                               [     0 -  0%]
 zero_cost=low_benefit:                               [     0 -  0%]
 one_cost=high_benefit:~~~~~~~~~~~~~~~~~~~~           [  1119 - 33%]
 one_cost=high_benefit:                               [     0 -  0%]
 one_cost=vhigh_benefit:~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~  [  1666 - 50%]
 zero_cost=high_benefit:                               [     0 -  0%]
 zero_cost=vhigh_benefit:                               [     0 -  0%]
```

```
---- 06/23/03 00:20:20 ---
  Setup: 0 sec
Compute: 0 sec
  Apply: 0 sec
  Total: 0 sec
--------------------------------------
```

A.4 Softgoal Framework-Influence Diagram Transformation

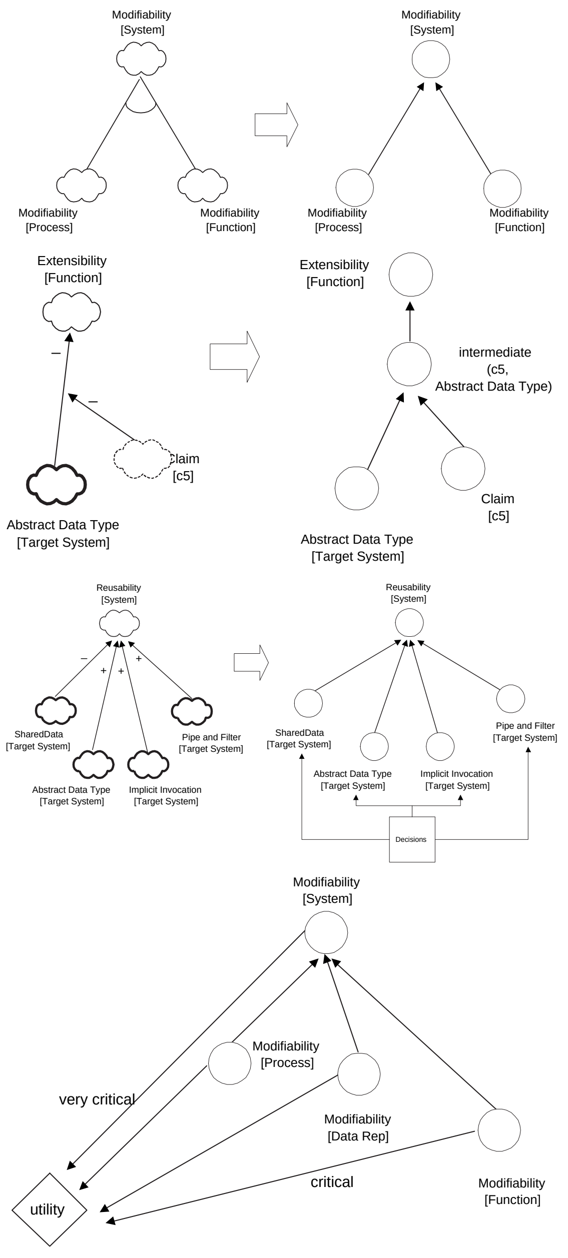


Figure A.1: Transforming Softgoal Framework into Influence Diagram