2006

A new metric for detecting conflict in functional software requirements

Adegboyega Oladayo Sanni
Iowa State University

A new metric for detecting conflict in functional software requirements

by

Adegboyega Oladayo Sanni

A dissertation submitted to the graduate faculty

in partial fulfillment of the requirements for the degree of

DOCTOR OF PHILOSOPHY

Major: Industrial Engineering

Iowa State University

Ames, Iowa

2006

Graduate College

Iowa State University

This is to certify that the doctoral dissertation of

Adegboyega Oladayo Sanni

has met the dissertation requirements of Iowa State University

Signature was redacted for privacy.

Committee Member

Signature was redacted for privacy.

Committee Member

Signature was redacted for privacy.

Committee Member

Signature was redacted for privacy.

Committee Member

Signature was redacted for privacy.

Major Professor

Signature was redacted for privacy.

For the Major Program

Table of Contents

List of Tables

List of Figures

Acknowledgements

I would like to thank God for giving me the grace and wisdom to pursue this new line of research in the year 2005 Anno Domini. May all the glory, honor, adoration and wonderful majestic splendor be unto Him.

I would like to thank members of my committee my major Professor Dr. Jackman, Dr. Ryan, Dr. Olafsson, Dr. Shrader and Dr. Townsend. Dr. Jackman really contributed immensely to my research without him this research would not have been possible, thank you once again.

I would to extend my appreciation to former committee members Dr. Ranga, Dr. Dickerson and Dr. Lutz for their kind support. Thanks to Professors Dr. Sannier and Dr. Kothari, they might not realize it but they helped improve my understanding of Software Systems.

Many thanks to my parents Dr. Mr. and Dr. Mrs. Sanni for granting me the opportunity to travel here to the States and fulfill my dreams. My Aunty and sister for their firm support. Many thanks to my Uncle and his family for supporting me in the States.

I would like to thank my friends here at ISU from Canada, Brazil, Panama, France, Germany, Greece, Hungary, Turkey, Denmark, Africa, Jordan, China, Korea, Japan, Burma, Laos, Taiwan, Thailand, Russia, India and other countries.

Abstract

Typically, the development of software system starts with a goal. The goal is implemented by following a methodology consisting of phases. Initially, the goal is formulated as functional requirements when stakeholders of the software system meet and discuss what the software system should do in order fulfill satisfy needs. The functional software requirement document is then converted to software design document either through conceptual model, software code or both. After, the system is tested rigorously before it is implemented. Since the development of a software system consists of phases with each phase depending on prior stages, an inconsistency made in the initial phase of development such as in the requirement specification phase, may be propagated into other phases. A methodology for detecting conflict in functional software requirements through level of Potential Structural Inconsistency (PSI) is presented in this research. This is accomplished, by representing functional software requirements stated in natural language as structural model (i.e. conceptual model) and similarities between these models are obtained as a level of potential structural inconsistency. Sample functional software requirements are analyzed using this methodology and the inconsistency is compared with a particular type of conflict. In conclusion, various inferences are made based on the new methodology and recommendations are given for further improvements and future research.

Chapter 1 Introduction

Organizations, business units, governments and institutions use multiple software systems to support their operations. For example, the National Aerospace Space Agency (NASA) spent billions of dollars developing high quality software systems to support space exploration. Some of these systems contributed to major losses in their space programs. An unmanned Mars Polar Lander crashed in 1999. The crash was attributed to a software bug (Blackburn 2002). Similarly, NASA's counterpart in Europe, the European Space Agency suffered a loss of over $7 billion when the Ariane 5 rocket exploded during launch (Gleick 1996). The explosion was blamed on an attempt by Ariane's software to convert a 64-bit floating-point value into a 16-bit integer value without adequate checks for overflow.

Failures such as these can be traced to multiple possible sources. One of those sources is the set of software requirements used during the software development process. Undetected inconsistencies between requirements could result in the implementation of inconsistencies in software system modules that may lead to embedded errors in the final assembled software system.

Most organizations use a structured process for developing software in order to mitigate risks and minimize errors. A common example of this structured process is the Software Development Life Cycle (SDLC). The phases of the SDLC include specification of requirements, design of a software system based on the requirements, implementation of the design (i.e. software coding), testing, deployment, and system maintenance. Early detection of inconsistencies in the SDLC can help prevent the propagation of errors through the development process. For example, Nelson et al (1999) found that software developers typically design systems based on requirements that have embedded errors. Apart from not being able to satisfy user needs, errors in requirements are costly, increase development time and cause maintenance problems. Nelson stated that

"A software error costing a mere $1 when caught early in the life cycle, costs $5 to correct at midpoint and $100 to correct later in the life cycle".

From a cost perspective, there is certainly a strong case for early error detection in order to reduce overall SDLC related costs.

The requirements phase consists of a set of iterative activities including elicitation, specification, analysis, negotiation, revision and documentation (Lamsweerde 1998). Elicitation is a data collection activity in which stakeholders (i.e., those who have a stake in the software system) define the elements of the problem and the associated requirements. The requirements are specified in natural language and analyzed. If problems arise, such as, ambiguity or inconsistency, the requirements are negotiated among the stakeholders and revised. After multiple iterations, the requirements are finally documented. The requirements phase involves extensive human interactions that can result in numerous inconsistencies. The sheer volume of information and lack of supporting analytical tools makes it difficult to detect inconsistencies.

Functional requirements specify the desired properties in the application domain that are satisfied using the software system. Inconsistencies created by the stakeholders, will propagate into other phases. Therefore, inconsistencies occurring in the requirements phase (even if detected later) can be expensive to resolve. Methods for detecting inconsistencies in the requirements phase can help prevent major problems in downstream phases.

Stakeholders have varying concepts of what constitutes a software requirement. Terms such as requirements, specification and requirements specifications are often used loosely and interchangeably. The IEEE (IEEE 1998) definition of software and system requirements specifications defines a System Requirements Specification (SyRS) as

"A description of what the system's customers expect it to do for them, the system's expected environment, the system's usage profile, its performance parameters, and its expected quality and effectiveness."

A Software Requirements Specification (SRS) (IEEE 1998) is defined as

"A specification for a particular software product, program, or set of programs that performs certain functions in a specific environment."

Based on the preceding definitions, a requirement in this research context will be defined as

"A description, represented in some form, which captures the needs of one or more customers for a software system."

An inconsistency occurs when there is a disagreement or conflict between two or more functional requirement statements.

1.1 Problem Description

Stakeholders elicit functional software requirements for a requirements document. Multiple stakeholders could specify one or more statements for inclusion in the requirements document. The stakeholders may meet from time to time, reviewing and revising these requirements. For a requirements phase, it is reasonable to assume that there exists an ideal set of requirements, **R***, that would completely satisfy the users' needs. As stakeholders progress through the requirements phase, the set of requirements can be defined as **R**. The expectation is that **R** over some period of time will converge to **R***.

To an observer, **R** appears to be stochastic in nature as its cardinality and accuracy increase and decrease over time. **R** may contain inconsistencies which cause it to deviate from **R***. In this study, we focus on examining **R** at an instant in time for some unknown **R***. Suppose, we form a subset of two requirements statements from **R**. Individually, the statements may be correct. However, when examined together they may be in conflict. The difficulty lies in detecting the conflict.

Requirements are usually specified in natural language. These natural language statements can be understood by a wider audience but can be ambiguous and unstructured as well. Therefore, other types of representations have been used. There are three basic types of representations, namely, informal, formal, and semi-formal models (Nuseibeh et al 2000). Natural language is an informal model. It is often classified as informal because almost anyone could use natural language to specify needs.

The common use of natural language was noted by Berry (03/25/04) who observed that

"Virtually every initial conception for a system is written in natural language."

In order to reason about inconsistencies in natural language, researchers have introduced formal models for detecting logical inconsistencies in natural language requirements. Formal models include formal requirements specification languages such as Software Cost Reduction (SCR) (Chechik 2001), predicate logic (Zowghi 2001), quasi-logic (Hunter 1995) and others.

The motivation for formal models is that they are well defined and provide a structure for reasoning about requirements. For example, if predicate logic statements can be created from a requirement expressed in natural language, then the requirement can be automatically assessed for logical inconsistencies using a theorem prover. However, the reasoning framework provided by the theorem prover is based solely on a classical logic model, which is an abstraction of the actual requirement. A formal model of natural language requirements is by no means an all-expressive and encompassing model since it is an abstraction of a requirement and some fidelity is lost. A theorem prover combined with predicate logic uses a Boolean value to detect the presence of an inconsistency. This approach is unable to assess the degree or significance of an inconsistency.

The third form of representing requirements is a semi-formal model. Our study is concerned with this form of representation that essentially captures a graph view of the requirements. These views describe structural, behavioral, or temporal properties of a requirement.

Practitioners have used semi-formal models such as, Entity Relationship (ER) model, Unified Modeling Language (UML) Class Diagram, UML Sequence Diagram, Data Flow Diagrams (DFD), Jackson Frame Diagrams, Flow charts, State Transition Diagram and others to describe problem domains. A number of researchers have studied the issue of inconsistency within a semi-formal model (Robinson 1999; Spanoudakis 1999), but did not consider using structural models to detect potential inconsistencies in natural language requirements. This may be due to the lack of a suitable structure for reasoning about inconsistencies as in formal models.

1.2 Motivation

Prior to this study, most of the research on detecting inconsistencies has focused primarily on logical inconsistencies (Hunter 1995; Zowghi 2001). These methods are Boolean assessments that indicate the existence of a logical inconsistency. In some cases, there may be uncertainty as to the existence of an inconsistency because the logical method only represents a logical view of inconsistency. The author believes that inconsistency has many forms and representations. Therefore, other views (e.g., structural representation of functional software requirements) should be investigated. A structural view emphasizes the relationships between requirement elements and provides a different perspective for the assessment of inconsistencies. Previous researchers have studied the existence of inconsistencies, but further investigations are warranted to assess the degree of inconsistencies and their severity.

1.3 Research Roadmap

Chapter 2 discusses the previous research and the relevance to the current research to the detection of inconsistencies in requirements. Chapter 3 states the research objectives and formulates the problem in terms of the problem elements and structure. Chapter 4 is the terminology for the research. Symbols and definitions are given in this section. Chapter 5, states the methodology based on formalized theories. In conclusion, Chapter 6 gives background information on the case studies to be investigated. Chapter 7 gives the results of analyzing software requirements from various case studies. In Chapter 8, a discussion of the research findings is presented along with conclusions about detecting conflict in functional software requirements via the level of potential structural inconsistency.

Chapter 2 Literature Review

Hausmann (2002) used a semi-formal model to detect conflicts in functional requirements. A use-case diagram was used to analyze requirement specifications from different stakeholders. Use-cases are part of the Unified Modeling Language framework (UML) ["Unified Modeling Language"].The use-case approach captures functional requirements through symbols consisting of objects and actions. Hausmann refers to the process of gathering and structuring information for the development of complex software system as often resulting in a set of overlapping and partly conflicting requirements models. Hausmann recommends that the requirements should be integrated into a consistent model. Conflict was related to the problem of potential consistency inconsistency in functional requirements. In a case study, conflicts were detected in a pair-wise manner based on the use-case diagram and set analysis. This research indicates that semi-formal models show promise for detecting conflicts in software requirements specification.

Lamsweerde (1998) described the starting point of requirements elicitation as a set of goals specified in a high-level language. He introduced a goal specification language (KAOS) which contained limited constructs in the form of an ontology. These constructs included goals, agents, operations and objects. Once the goals were specified in KAOS a number of heuristic methods were used to detect and resolve inconsistencies in logic, structure, and designations. This ontology demonstrated the importance of classifying requirements based on some set of defined objects (similar to semi-formal models) before detecting inconsistency. This approach is similar to other researchers Aircraft European Contractors

Manufacturers Association (AECMA) who simplified the problems of ambiguity and complexity of natural language requirements by creating a small manageable subset of language for requirements.

Robinson et al (1999) proposed a requirements ontology called Conflict-Oriented Requirements Analysis (CORA) for stakeholder conflict resolution through requirements restructuring. The study identifies three steps in restructuring stakeholder requirements through CORA, namely, conflict identification, resolution generation and resolution selection. CORA is in the form of a semi-formal model based on the UML class diagram. This approach reinforces the notion of reasoning about inconsistency in requirements through semi-formal models. It is important to note that semi-formal models are often used to represent an ontology.

The ontology of CORA serves as a structural template that models stakeholder requirements. The models show the interaction of classes and relationships. The goal of this approach is to resolve stakeholder conflicts. A structural template for requirements suggests that $\boldsymbol{R}^*$ and $\boldsymbol{R}(t)$ have some inherent structure and that resolutions are based on some modification of the structure or its contents.

A requirement conflict is defined as a structural interaction in CORA. An interesting part of Robinson's research is that the structural conflict analysis is composed of a recursive pair-wise comparison of the requirements in CORA for structural differences (i.e. similar to Hausmann's observations). The structural differences form a tree of structural differences. Based on CORA, some of these structural differences form a part of a requirement interaction that define a structural conflict. This provides strong support for using a methodology that is iterative, compares pairs of structures and collects information about structural differences. The author suggests that the concept of a structural difference lends itself to the methodology of gathering some form of structural metrics on functional software requirements in order to determine the occurrence of a conflict.

Spanoudakis et al (1997; 1999) studied potential inconsistencies based on the detection of ontological overlaps in object-oriented specifications (i.e., semi-formal models) and metric functions related to semantic and similarity analysis. A reconciliation method was used for detecting ontological overlaps through the analysis of similarities between object-oriented viewpoints. These viewpoints represent the partial specifications of the overall requirement for object-oriented specifications. The method helped to establish a common understanding of the potential for inconsistency between specification owners through assessment and verification of the detected ontological overlaps. Their assumption is that the existence of an overlap in requirement specifications is a precondition for detecting inconsistencies. The method uses a similarity analysis that measures the distances between specifications based on three metrics, namely, classification, generalization and attribution. The distances represent differences in properties of classes, semantic differences and semantic homogeneity. It is important to note that classes are often part of an ontology and therefore, it is in agreement with Robinson's approach. The result of the similarity analysis is the overall distance measures, weighted graph and lists showing properties of classes weighted by importance. As a result, specification owners can assess the result and vary parameters in order to discover new overlaps. An example of the method is provided in detecting overlaps between two object-oriented specifications of library borrowers, items and their relations. Most of the similarities are referred to as structural but show semantic overlaps. For example, a borrower can be semantically mapped onto Student. This research provides an excellent foundation for detecting potential inconsistency in semi-formal model though it does not address the problem of natural language requirements related to structural representations such as the ER model.

Simple natural language requirements in Italian were transformed into graphical representations of certain classes of information (Ambriola 1997). The interpretation of the natural language requirement was performed using a domain dictionary and a set of fuzzy-logic rules. The graphical representations were data flow diagrams and the ER model (Thalheim 2000). This methodology provides a basis for generating ER models from natural language requirements. However, the method does not check for inconsistency in the ER models. In similar research, Tjoa (1993) presented a tool that can be used to transform requirement specifications expressed in natural language into a conceptual model. The tool is called the Data Model Generator (DMG). The DMG is based on the assumption that syntactic structures of natural language requirement specifications can be translated into a conceptual model, Extended Entity Relationship Model (EER). Their approach was a set of rules and heuristics for extracting syntactic information and capturing static aspects such as entities, entity types, attributes and relationship types. It also showed the degree and connectivity between (mandatory, optional) of relationship types (such as, mandatory or optional). The research is significant because it addressed the issue of creating an ER model from natural language requirements. They noted that one advantage of the system is the automated extraction of models when a user is inundated with a large amount of textual requirements. One limitation though is that Ambriola (1997) and Tjoa (1993) did not address the detection of inconsistencies in the semi-formal models.

In related work, Palmer (1992) proposed an integrated environment for requirements engineering. It supports a broad range of requirements engineering activities with the inclusion of requirements elicitation, classification, analysis, traceability and design. Of interest are the Lexscan and Knowledge-Based Requirements System (KBRS) modules. The Lexscan is an automated tool that can be used to analyze natural language requirements based on syntax. It classifies natural language requirements through indexing and clustering methods as opposed to the reconciliation method, which uses a set-based method. The indexing and clustering methods are used to distinguish between similar requirements.

Classified requirements are passed to the KBRS for detecting a number of problems such as conflict, inconsistencies, incompleteness and others. The KBRS is a knowledge-based tool. The KBRS detects conflict based on captured knowledge, which may be subjective. This research showed the importance of clustering algorithms and suggests that they can be used to detect inconsistencies in clusters generated from natural language requirements. It also shows that similarity can be used to analyze requirements based on some form of classification (e.g., clusters).

Hunter et al (1995) adapted classical logic (Gries 1993) to create a "quasi-classical logic" (QC logic). This was used to represent partial specifications as a "ViewPoints framework" and detect logical inconsistency. Theorem proving was proposed as a framework for reasoning about logical inconsistencies represented in QC logic. Using a case study of an order processing system for a wine warehouse, the authors manually translated the requirements document into QC logic. The authors reasoned about certain scenarios using the QC logic and made inconsistency inferences that were related back to the requirements document. Among some of the inferences was that there was some inconsistency in issuing warehouse request was probably caused by the conflicting rules laid down by the Logistics Manager and the Chief Wine Taster. The inference was made in a rule that states that "If there is an inconsistency in the specification data, the likely source of the inconsistency is a conflict between two development participants". This research is important because it shows how classical logic from the field of mathematics can be used in the detection of logical inconsistencies caused by conflicts in the field of requirements engineering.

Zowghi (2001) investigated the detection of logical inconsistency in natural language requirements using a similar approach as Hunter (1995). The process of detecting logical inconsistencies consists of parsing natural language requirements using a parser named CICO (Gervasi 2000) that produces a parse tree. This tree is a hierarchical tree structure (consisting of nodes and branches) that shows the relationship between the words and their parts of speech. The resulting parse tree is translated into predicate logic (i.e., a formal model) and submitted to a system called Computer-Assisted Requirements Evolution Toolset (CARET). CARET takes the predicate logic as an input and checks for logical inconsistency using different scenarios and a theorem prover. This method is significant because it utilizes an automated theorem-proving framework for detecting logical inconsistencies through the translation of natural language requirements. They suggested an expressive form of logic that could capture complex requirements as an extension to their work, indicating that additional views may be needed that are not supported by a logical model.

Genero (2003; 2002; 2000) investigated the application of metrics to ER models. Genero (2003) used metrics in defining and validating a conceptual model. The ER model was used as a conceptual model. In an empirical study, participants detected similarities between ER models in a certain amount of time. This was used as an understandability attribute of quality. In order to measure the modifiability, the participants evaluated two ER models to determine if they had the same conceptual meaning. The metrics used in correlating the response times include the number of entities, number of binary relationships and others. These types of metrics can be considered as base metrics. The results show high correlation between the metrics and selected quality attributes. This research is important because it provides a probable set of base metrics that could be used to analyze an ER model (Jones 2000) and a correlation method for validating metrics.

A metrics system built on similar base metrics was used to detect the level of quality in an Extended Entity Relationship (EER) (Cherfi 2002). Metrics were derived for assessing the quality of EER Model or UML class diagrams. Using base metrics, composite metrics were derived to measure attributes such as, legibility, expressiveness, and simplicity. For example, measuring legibility was based on the number of inheritance links and the number of line-crossings (i.e., base metrics) in a model. One limitation of these metrics is that, they are used to measure quality (Wilson 1996) as opposed to inconsistency and rely on specialized relationships through the concept of inheritance.

Lorenz (1973), a Nobel laureate discussed concepts of evolution in relation to similarity of forms. He notes that as evolution occurs, two forms that exist in life may take similar paths in parallel for the purposes of adapting to their environment. This may be related to the concept of conflict, if the two forms existing in life are so similar they compete for the same resources. Lorenz states, "The improbability of coincidental similarity is proportional to the number of independent traits of similarity and is, for ***n*** such characters, equal to $\boldsymbol{2}^{n-1}$". For example, Lorenz mentions that an airplane, torpedo and shark amongst other forms bear a notable resemblance because of they need to reduce friction in order to function in their environment. A similar line of reasoning can be extended to software requirements specifications, if two functional requirement statement existing a document are competing for the same section of code to be implemented, this may lead to conflict. The concept of similarity can be viewed as an analogy that can be used to reason about inconsistencies in functional software requirements.

Tverksy's (1977) similarity model consists of a feature-based set for similarity measure. It is predicated on the assumption that a similarity measure between two or more objects is proportional to the number of common features. The common and distinct features are modeled as set operations on the features of the objects in comparison. Tversky gives numerous examples to demonstrate the measure. In one example, the letter "E" is more similar to "F" then "I" based on the more common features. In the same way, the letter "I" is more similar to "F" than "E" based on distinct features. Tversky suggests that although people assume that classifications are determined by similarity, "The similarity of objects is modified by the manner in which they are classified; it serves as a basis of classification of objects, but it is also influenced by the adopted classification." This statement implies that though we proposed the use of the similarity measure to classifying objects, there is an unconscious effort when a specific classification method is adopted which will in turn influence the measure of similarity. It is important to state at this juncture, that there are other possible measures of similarity, but Tversky's measure was used for this study because of its transferability to the requirements context and its basis in set theory based representation.

Chapter 3 Research Objectives

Tversky's similarity measure is central to this research because it was used by Rodriguez et al (2003) to determine the similarity between ontologies using entity classes. For example, two different ontological representations of the word "stadium" are compared with one another based on their respective hierarchical set definitions. The entity class for "stadium" represented an ontology containing elements such as possible names, descriptions, super class definitions, sub class definitions, parts (i.e. objects), functions and attributes. This research provides some indication that Tversky's similarity measure could be useful in exploring structural overlap between pairs of ER Models (which are similar to entity classes). A limited ontology for ER Models would consist of cardinalities, entities, relationships and attributes.

Support for pairwise comparison of requirements is based on the approaches described by Hausmann (2002) and Robinson (1999) where requirements or structures were compared pairwise. This approach simplifies the analysis by considering conflicts between each pair of requirements. The role of overlap is important for detecting conflicts as seen in Nuseibeh (2001) and Spanoudakis et al (1997; 1999) where the condition for the existence of an inconsistency depends on an overlap. In this research, we consider structural overlap between a pair of requirements. Hunter et al (1995) states that "If there is an inconsistency in the specification data, the likely source of the inconsistency is a conflict between two development participants." This statement implies that based on structural overlap, a possible conflict can be detected via the measure of potential structural inconsistency between a pair of functional software requirements.

The need to transform natural language statements into a form suitable for analysis has been recognized by Hunter (1995) and Zowghi (2001). They transformed functional software requirements into logical statements and then analyzed the statements via a theorem prover to identify logical inconsistencies. This suggests that the type of inconsistency detected is closely related to the form of representation. Similarly, if we transform requirements into a structural form, then one would expect to see possible "structural inconsistencies." A structure is defined by a set of related elements.

A structural inconsistency can be defined as a condition in which "*one or more elements of two structures disagree with each other*" while a conflict can be defined as "*a state in which two or more requirements statements cannot co-exist together.*" A disagreement may exist if there is an overlap between two structures. This may occur because the two structures share elements that should not appear in both sets or the relationship between elements is defined differently in the two structures. Overlap (i.e., similarity) between two structures can be defined as the set of common elements found in both structures. Previous research has shown that the existence of an overlap is a precondition for detecting inconsistency in functional software requirements (Nuseibeh 2001; Spanoudakis 1997, 1999). If we use similarity (i.e., the relatedness between structures taking into account common and unique elements to both structures) as a measure of overlap between structures, then the amount of structural overlap for a set of functional software requirements can be determined.

3.1 Primary Research Question

When does structural overlap indicate that a conflict is likely for a pair of functional software requirement statements?

3.2 Research Proposition

When a high numeric value of potential structural inconsistency is observed for a pair of functional software requirements, then a conflict is likely.

3.3 Inferences

When a potential structural inconsistency value is high for a pair of requirements, this indicates that a conflict is likely and there is also a high degree of structural overlap associated with some redundancy. The numeric value for a pair of requirements is assumed to be high irrespective of whether the value is above or under a threshold value. If the pair of requirements is revised in such a way that the potential structural inconsistency value is significantly reduced, then the conflict should be resolved. Possible revisions would include elimination of statements, adding elements, or removing elements.

3.4 Verify with test cases

In these test cases, a set of requirement statements **R with known conflicts** are transformed into corresponding structural representations. Numeric values of structural overlaps were computed and the level of potential structural inconsistency for **R** was determined. If a pair of requirement statements having a known conflict exhibits a high level of potential structural inconsistency, then the embedded conflict will have been successfully predicted and the proposition would have some support. If known conflicts exhibit low values for the metric, then potential structural inconsistency would be a poor indicator of conflicts.

3.4.1 Conditions

Multiple sets of requirement statements were used from different case studies used in previous research (Flowers 1996; Lewerentz et al 1995; Neufville 1994).

3.4.2 Expected Results

The author expects that a high level of potential structural inconsistency indicates conflict to some acceptable degree and that different case studies can be used to show this.

Chapter 4 Terminology

4.1 Notation

The following notations are used in subsequent sections.

$\mathbf{R}$:	a set of requirement statements in a natural language
$\mathbf{R}'$:	an abstraction of $\mathbf{R}$ in a controlled language
r_i :	requirement statement i
Π :	Entity Relationship Model set
π_i :	the entity relationship model for requirement i
ε :	an entity in π_i
α :	an attribute for an entity
c:	the value in terms of number of entity instances in a relationship
ω:	a relationship between two entities
ν:	cardinality for an entity in a relationship
$\Psi(\pi_i, \pi_j)$:	the potential structural inconsistency for π_i and π_j

$\Phi_K(\pi_i, \pi_j)$: set operation on π_i and π_j

4.2 Definitions

4.2.1 Attribute

An attribute is the property that describes of gives more information about an Entity. An attribute set for Entity type i is represented by $\mathbf{A}_i = \{\alpha_{i1}, \alpha_{i2}, \alpha_{i3}, \ldots, \alpha_{ip}\}$.

4.2.2 Cardinality

The cardinality indicates the number of entities participating in a relationship. It is a three-tuple, $v_k = (c, \varepsilon_i, \omega_j)$ consisting of a value (in terms of number of instances), an entity, and one relationship.

4.2.3 Modified Controlled Language Lexicon

The controlled language, ACEMA CL (AECMA 2004), was modified to create a lexicon having only one possible part of speech for each word. This was done to simplify the transformation process. Let $\mathrm{R}'(t)$ be a representation of $\mathrm{R}(t)$ based on the modified CL lexicon.

4.2.4 Entity Relationship Model for a Requirement

The entity relationship model is the structure being used in this research. It is a conceptual model described by entities, relationships, attributes and cardinalities. . An ER model π_i for requirement statement i, is defined as a three-tuple, $\pi_i(\mathrm{E}, \Omega, \mathrm{N})$. The entity set $\mathrm{E} = \{\varepsilon_1, \varepsilon_2, \varepsilon_3,$

…, $\varepsilon_n\}$, represents objects in the system. For each entity ε_i, there is a set $\mathbf{A}_i=\{\alpha_{i1}, \alpha_{i2}, \alpha_{i3}, \ldots, \alpha_{ip}\}$ of attributes that describe the entity. Relationships between entities in **E** are defined by $\Omega = \{\omega_1, \omega_2, \omega_3, \ldots, \omega_m\}$, a pairwise set of relationships where, $m \le \binom{n}{2}$. The cardinality set N = $\{\nu_1, \nu_2, \nu_3, \ldots, \nu_{2m}\}$, defines the number of instances for each entity in a relationship.

4.2.5 Entity

An entity ε_i, is an object that is involved in some task or action. The entity set E = $\{\varepsilon_1, \varepsilon_2, \varepsilon_3, \ldots, \varepsilon_n\}$ represents all entities in a statement

4.2.6 Functional Software Requirements

A collection of statements in natural language that specify what the software must do in order to satisfy customer requirements is given by

$$\mathbf{R} = \{r_1, r_2, r_3, \ldots, r_n\}.$$

4.2.7 Part Of Speech (POS)

The complete part of speech list can be found in the Appendix A. The POS reflects the role of a word in the statement. The basic types are article, noun and verb (i.e. ART, N and V respectively). Appendix A shows the Penn Treebank tag set used in this research. When a statement is parsed by the Stanford parser, the tags of each word in the statement are actually from the Penn Treebank tag set.

4.2.8 Parse Tree

A parse tree describes the structural relationship for the POS set in a requirement statement. The tree is a hierarchical display of nodes consisting of parent and leaf nodes. The Stanford parser produces the parse tree from a statement.

4.2.9 Parent Node

A parent node is a node in the parse tree that is either a statement or a phrase. A phrase may either contain other parent nodes or leaf nodes. A sample of parent nodes are given in Table 1. A detailed description can be found in Appendix A.

Table 1 Parse Tree Parent Nodes

Parent Node	Designation
Statement	S
Noun phrase	NP
Verb phrase	VP
Prepositional phrase	PP
Adjectival phrase	ADJP
Adverbial phrase	ADVP

4.2.10 Leaf Node

Leaf nodes are nodes at the lowest level in the parse tree. They have no child nodes. A leaf node represents a POS for each one word in the requirement statement.

4.2.11 Phrases

When a statement is incomplete it is a phrase. Based on the POS, we can have different types of phrases as shown in Table 1.

4.2.12 Potential Structural Inconsistency

An inconsistency is defined as "*If a reason, idea, opinion, etc. is inconsistent, different parts of it do not agree or it does not agree with something else*" [Cambridge online dictionary]. We define a potential structural inconsistency between π_i and π_j as $\Psi(\pi_i, \pi_j)$. This represents the possibility that elements of structures do not agree with each other

4.2.13 Relationship

A relationship ω_k, is some form of association between two or more entities. The relationship set for π_i is defined by $\Omega = \{\omega_1, \omega_2, \omega_3, \ldots, \omega_m\}$. If there are no relationships between entities, then this would be a null set.

4.2.14 Total Elements

$\Phi_K(\pi_i \cup \pi_j)$ is the set of all elements of type K (e.g., entities, attributes, relationships) present in π_i and π_j.

4.2.15 Common Elements

$\Phi_K(\pi_i \cap \pi_j)$ is the set of all elements of type K (e.g., entities, attributes, relationships) common to both π_i and π_j.

Chapter 5 Research Methodology

5.1 Methodology Overview

The main components of the research methodology include:

1. Derivation of the inconsistency metric, $\Psi(\pi_i, \pi_j)$
2. Selection of suitable datasets for requirements based on known case studies
3. Comparison of the results for $\Psi(\pi_i, \pi_j)$ and known inconsistencies in the datasets
4. Comparison with Predicate Logic; provide some lightweight validation for inconsistency detected

$\Psi(\pi_i, \pi_j)$ is derived from **R** (i.e., the set of requirements) based on Tversky's measure of similarity with normalization. The datasets for the requirement statements are taken from four existing software system development case studies. Each dataset is then analyzed and the value of $\Psi(\pi_i, \pi_j)$ for each pair of requirements is determined. Based on known conflicts in the datasets, a comparison is made between the existence of conflicts and $\Psi(\pi_i, \pi_j)$. These results are compared with the predicate logic approach used in previous research to evaluate the similarities and differences in detecting conflict.

5.2 Derivation of $\Psi(\pi_i, \pi_j)$ from (r_i, r_j)

The major steps in finding $\Psi(\pi_i, \pi_j)$ are as follows.

1. Manually rewrite each r_i as r_i' based on the modified CL lexicon (see section 4.2.3). Words that do not exist in the CL are replaced by looking for words with similar meaning. The meaning of the word within the context of the statement is also taken into account when selecting the replacement word. It is possible that a conflict could be artificially introduced due to this replacement.

2. Parse r_i' using the Stanford Parser (Klein 2003) to produce a parse tree.

3. Manually modify the parse tree by checking the POS for each leaf node in the tree versus the modified CL lexicon. If the POS does not agree with the lexicon, then the POS is replaced with the one in the lexicon and the parse tree structure is modified if necessary.

4. Transform the modified parse tree into π_i, the Entity Relationship Model (ER Model) using Chen's rules.

5. Calculate $\Psi(\pi_i, \pi_j)$ from π_i and π_j.

5.2.1 Transform r_i into r_i'

Natural language is ambiguous because words used in natural language can have different meanings in different contexts resulting in multiple possible parts of speech. The parts of speech are one component of determining the semantics of a statement. For example, consider the word "state" used as follows.

1. *The man states that the project will take a long time.*

2. *The states claim that the project will take a long time.*

In the first statement the word "state" is a verb while in the second statement it functions as a noun. Identifying the different parts of speech in each statement is relatively easy for someone who knows the language. However, using an algorithm that relies on word sense disambiguation rules is a non-trivial problem (Kamsties 2001) beyond the scope of this research.

In order to overcome the problem of ambiguity in natural language, a Controlled Language (CL) is used. A CL is a subset of the English language containing a smaller lexicon (i.e., set of words). The CL used is the Aircraft European Contractors Manufacturers Association (AECMA) Simplified English (SE). AECMA was selected over other CLs (such as, Attempto Controlled English ACE, Processable English PENG, and Caterpillar Fundamental English (CFE)) because it has been in existence for quite some time and is readily accessible.

AECMA SE has been used for the specification of aircraft maintenance manuals in order to improve the readability for aircraft maintenance workers. In a similar manner, by specifying the functional software requirements in a CL, ambiguity in the language is reduced, simplifying the process of producing a parse tree. The AECMA SE consists of a limited lexicon of 956 words including nouns, verbs, adjectives, and prepositions. Some words in the lexicon have more than one possible POS. This lexicon was modified by reducing the possible parts of speech for each word to one. This provided a unique POS for each word.

Each requirement statement, r_i, is rewritten as r_i' using the modified CL lexicon.

5.2.2 Parsing r_i' using the Stanford Parser

The Stanford parser was used to produce a parse tree for each r_i'. Suppose, we have the following requirement statement from a simple case of developing a web based system that monitors Internet traffic.

The system monitors the traffic.

The POS tags from the modified CL lexicon for each word are shown in the Table 2. The combination of each word and its POS tag are the leaf nodes for the parse tree.

Table 2 POS tags for the example

Word	POS tag	Meaning
The	DT	Determiner
system	N	Noun
monitors	V	Verb
the	DT	Determiner
traffic	N	Noun

If all the words in a statement is tagged then it is verified as being wholly contained in the CL. The CL statements help to reduce the possibility of ambiguity. The Stanford parser produces a parse tree using the Penn Tree Bank tags none-the-less if the CL parts of speech help to ensure consistency. The Stanford parser uses a set of grammar rules that contain relationships between the phrases and the POS tag.

The final parser output using the Stanford parser is shown in Figure 1.

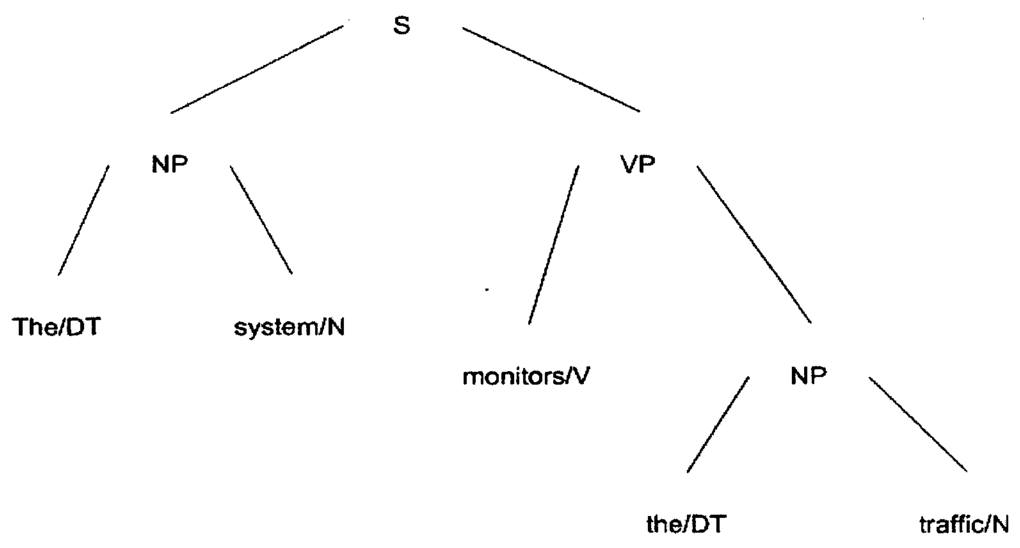

Figure 1: Parser Output

5.2.3 Modify the Parse Tree Using the Modified CL Lexicon

The parse tree is based on a different lexicon (i.e., the Penn Tree Bank tagset). Therefore, the resulting POS tags in the tree may not be consistent with the modified CL lexicon. The leaf nodes in the tree are manually compared with entries in the modified CL lexicon. If they do not agree, then the POS tag from the modified CL lexicon is substituted into the tree. In some cases, this may change the nature of the parent nodes and then the tree is manually re-structured.

5.2.4 Transform the modified parse tree into π_i

Chen (1983) studied the relationship between the parts of speech in English sentences and the associated ER Model. Following the general guidelines for creating ER Models proposed by

Chen, the parse tree nodes were transformed into an ER Model, π_i. The following rules proposed by Chen were used.

Rule 1: If the word is a common noun (e.g., "person", "Chair"), then add ε_k (corresponding to the noun) to **E** if ε_k does not already exist.

Rule 2: If the word is a transitive verb (or verb phrase), then add ω_k (corresponding to the verb) to **Ω**. The following example is given by Chen.

A person may own a car and may belong to a political party.

The transitive verbs, "own" and "belong to", are relationships between (person, car) and (person, party), respectively.

Rule 3: If the word is an adjective, then add α_{ik} (corresponding to the adjective) to $\mathbf{A}_i$ for entity ε_i.

Rule 4: If the word is an adverb, then it is considered to be a modifier of a verb phrase corresponds to an attribute of a relationship in an ER Diagram. The author of this research assumes that this attribute is part of a verb phrase as such a relationship.

Rule 5: If a parent node has the form "The X of Y is Z" and if Z is a proper noun (i.e., a specific instance of a noun), then treat X as a relationship between Y and Z and add ω_k (corresponding to X) to **Ω**. In this case both Y and Z represent entities.

Rule 6: If a parent node has the form "The X of Y is Z" and if Z is not a proper noun, then treat X as an attribute of Y. In this case Y, represents an entity (or group of entities) while Z represents a value. Chen gives the following statement as an example.

The color of the desk is blue.

Rule 7: If a clause is a high-level entities then it is abstracted from a group of interconnected low-level entities and relationships.

Rule 8: A sentence corresponds to one or more entities connected by a relationship, in which each entity can be decomposed (recursively) into low-level entities connected by relationships.

Based on the POS tags, the corresponding ER model components are shown in Table 3.

Table 3 ER Model Components

POS Tag	**ER Model Component**
N	ε
V	ω
ADJ	α
ADV	ω
DT	ν

To define the sets associated with π_i (i.e., **E**, **A**, **Ω**, **N**), the tree is traversed depth-first, from left to right and the components of the ER Model are identified based on the rules.

The general guidelines for extraction are as follows.

Entity Extraction: Traverse the parse tree from left to right until a noun leaf node is encountered. The first noun leaf node encountered is "system/N" with the parent node "NP". Based on Rule 1 we can add "system" to entity set E. In a similar manner we can add "traffic" to the entity set E. In the general case, we can traverse the parse tree from left to right until we find "NP" parent nodes then we look for "N" child nodes , add corresponding words to the entity set until we reach the end of the statement.

Relationship Extraction: Traverse the parse tree from left to right until a verb or adverb is encountered. The first verb encountered is "monitors/V" with a parent node "VP". Based on Rule 1, we can add "monitors" to the relationship set but in order to do that we require two entities. The two entities can be obtained by finding the noun leaf nodes of neighboring NP parent nodes to VP. In this example, "system" and "traffic" are selected and then the relationship is added to the relationship set **Ω**. In the general case, we can traverse the parse tree from left to right until we find "VP" parent nodes then we look for "V" and "RB" (i.e. from the Penn Tree Bank Tagset) child nodes, add corresponding words to the relationship set until we reach the end of the statement.

Attribute Extraction: Traverse the parse tree from left to right and search for neighboring leaf nodes that qualify noun leafs (i.e. providing more information about noun leafs). The search will typical stop for a noun leaf node once a relationship or the end of the statement is encountered.

Cardinality Extraction: Find the noun leaf nodes and look for another leaf node having the same parent node NP and a part of a "Determiner" as its POS. This step was omitted in the research because the use of cardinality introduced a greater complexity in the research and other metric concerns as such it is not used in the similarity analysis.

Using the extraction guidelines for extracting ER model components, we can produce π_i for the example as shown in Figure 2.

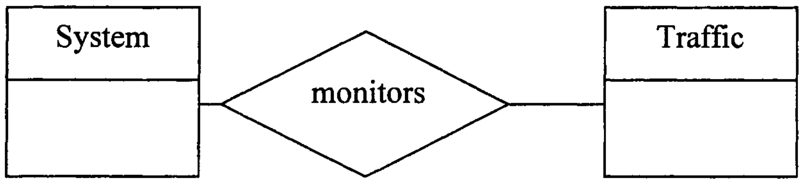

Figure 2: ER Model

The corresponding set members are shown below. These sets represent the structural form of the functional software requirement statement.

E = {system, traffic}

Ω = {monitors (system, traffic)}

A = {null}

N = {the, the}

5.2.5 Inconsistency metric $\Psi(\pi_i, \pi_j)$

The research proposition states that if $\Psi(\pi_i, \pi_j)$ is high, then a conflict is likely to exist between r_i and r_j. $\Psi(\pi_i, \pi_j)$ represents the possibility that elements of two structures do not agree with each other. A set-based similarity measure between two structures can be used to assess the overlapping elements between the structures.

Two measures of similarity were considered, namely, Tversky's measure of similarity (Tversky 1977) and Edit Distances (Ristad 1998). Edit distances is used in genetics research. Tversky's measure of similarity was used because of the set based representation and that was developed irrespective of the field of research (i.e. genetics), essentially both measures are different. Tversky's measure of similarity measure is also readily adaptable to the enumeration of common and unique elements. Rodriquez (2003) used Tversky's measure of similarity in finding semantic similarity among entity classes. Tversky's measure applied to entity classes provides a good starting point for measuring similarity in ER Models.

Given two ER models, π_i and π_j, we can perform set operations on elements of π_I and π_j to determine which elements the models have in common and which ones are different. $\Phi_K\ (\pi_i \cup \pi_j)$ is the set of all elements of set type K (e.g., entities, attributes, or relationships) and $\Phi_K\ (\pi_i \cap \pi_j)$ is the set of common elements of set type K. We can define those elements found in π_i but not in π_j as $\Phi_K\ (\pi_i - \pi_j) = \Phi_K(\pi_i \cup \pi_j) - \Phi_K(\pi_j)$. Similarly for π_j we obtain $\Phi_K\ (\pi_j - \pi_i) = \Phi_K(\pi_i \cup \pi_j) - \Phi_K(\pi_i)$. Using the elements from sets **E**, **R**, and **A**, we can obtain the cardinality of the sets based on equal weights for the three sets giving us,

$$F(\pi_i \cap \pi_j) = |\Phi_E(\pi_i \cap \pi_j)| + |\Phi_R(\pi_i \cap \pi_j)| + |\Phi_A(\pi_i \cap \pi_j)|\,,$$

$$F(\pi_i - \pi_j) = |\Phi_E(\pi_i - \pi_j)| + |\Phi_R(\pi_i - \pi_j)| + |\Phi_A(\pi_i - \pi_j)| \text{ and}$$

$$F(\pi_j - \pi_i) = |\Phi_E(\pi_j - \pi_i)| + |\Phi_R(\pi_j - \pi_i)| + |\Phi_A(\pi_j - \pi_i)|.$$

We can then define the similarity of π_i and π_j as the difference between the number of common elements and those elements that are not in common, giving us

$$S(\pi_i, \pi_j) = \gamma_1 F(\pi_i \cap \pi_j) - \gamma_2 F(\pi_i - \pi_j) - \gamma_3 F(\pi_j - \pi_i),$$

where, γ_1, γ_2 and $\gamma_3 \geq 0$, are assignable weights assumed to be $\gamma_1 = \gamma_2 = \gamma_3 = 1$. The weights are used to assign importance to each function depending on the context of similarity. To normalize $S(\pi_i, \pi_j)$ between 0 and 1, it is necessary to determine the upper and lower bounds of $S(\pi_i, \pi_j)$. The upper bound corresponds to min $(|\pi_i|, |\pi_j|)$, which would be the largest number of elements that π_i and π_j could have in common. Likewise, the lower bound corresponds to $-(|\pi_i| + |\pi_j|)$ representing the condition when all elements are different. The normalized similarity $\Psi(\pi_i, \pi_j)$, is given by

$$\Psi(\pi_i, \pi_j) = \frac{S(\pi_i, \pi_j) - (-(|\pi_i| + |\pi_j|))}{\min(|\pi_i|, |\pi_j|) - (-(|\pi_i| + |\pi_j|))}, \text{ which reduces to}$$

$$\Psi(\pi_i, \pi_j) = \frac{S(\pi_i, \pi_j) + |\pi_i| + |\pi_j|}{\min(|\pi_i|, |\pi_j|) + |\pi_i| + |\pi_j|}.$$

5.3 Selection of suitable datasets for requirements based on known case studies

Four case studies were selected to produce four separate datasets. Some case studies were selected with a small set of requirement statements in order to show the basic application of the methodology. For example, the first test case had 3 requirements. The second test case had 5 requirements and had been studied previously by two researchers. This test case involved a project with known conflicts and inconsistencies that had been previously analyzed using logical methods. Other case studies were considered either because the nature of the known conflict is apparent, there were known problems in the software, or the model size was limited in scope. The small number of statements is important because natural language can be difficult to analyze on a large scale. Therefore, a limited number of statements were used to confirm the author's proposition.

5.4 Comparison of the results for $\Psi(\pi_i, \pi_j)$ and known inconsistencies in the datasets

By comparing the results for $\Psi(\pi_i, \pi_j)$ with known inconsistencies in the datasets, the reliability of the metric can be assessed. This comparison will indicate whether different inconsistencies exist in the document and provide a better understanding of the nature of the underlying conflict(s).

5.5 Comparison with Predicate Logic

Predicate logic was also used to detect inconsistencies in the datasets from the test cases.. The use of predicate logic helps to clarify the different types of inconsistency detected in the case studies (i.e., any differences between logical and structural inconsistency). This approach was used because previous researchers have used this method to identify known inconsistencies and conflict from a logical view. These results were compared with those obtained for $\Psi(\pi_i, \pi_j)$ to determine if there are differences. Another reason for using predicate logic is that predicate logic has been an accepted method for detecting inconsistency.

Chapter 6 Test Cases

In all, 4 test cases (i.e. containing a total number of 41 statements) are presented covering multiple software system domains. The studies include the controlling a pump, an automated dispatch system, robots in a factory, and an airport baggage scheduling system. These studies all have some degree of dependency between objects in their respective systems. Dependency is important because the proposed metric uses similarity analysis as such some degree of dependency is required.

In brief, the Sump Pump test case has a relative small requirement set of 3 statements. The next case of the London Ambulance Service has 5 statements. The Production Cell test case has 8 statements while the DIA Airport test case has 25 statements. The case studies address control scenarios having different levels of complexity. Sump Pump

6.1.1 Description

The Sump Pump test case is a modified pump example from (Hooman 1995; Chechik 2001) that contains 3 statements in all. Though a relatively small data set, it provides a good entry point for detecting inconsistencies (i.e., if $\Psi(\pi_i, \pi_j)$ is not successful for this study, then it is likely a poor metric). This system also shows the core concept of validation in terms of known system concerns that are conflicts. This takes the form of providing a set of ideal requirements R* for analysis. R* is assumed to be true when we have a system that is predefined based on well known implementation. Such systems have standard requirements that have been successfully applied several times. Given R*, we generate an acceptable conflict in the system and introduce R (i.e., requirement statement) that produces a conflict. A diagram of the Sump Pump System is given in Figure 7.

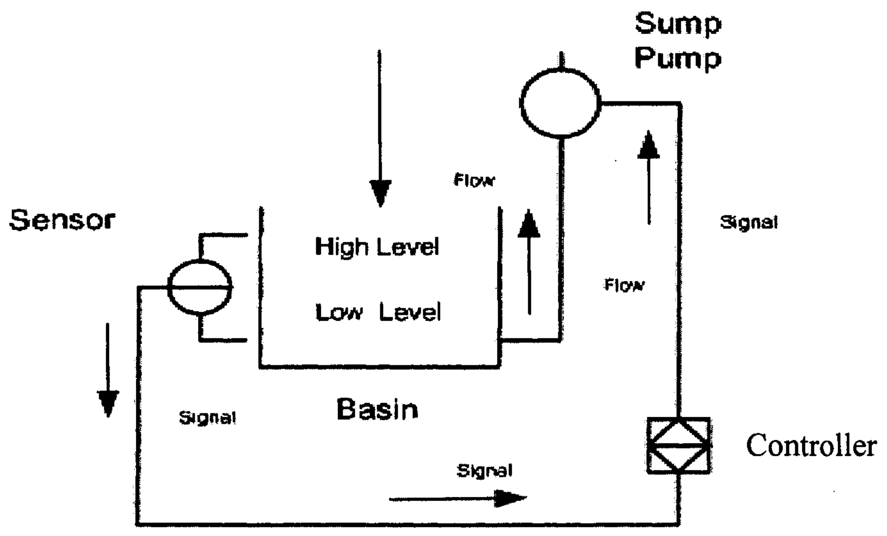

Figure 3: Sump Pump System

When fluid in the system reaches the level associated with a discrete sensor, the change in signal is detected by the Controller. Based on the control logic, the Controller can turn the power for the sump pump on or off.

6.2 London Ambulance Service (LAS)

6.2.1 Description

The London Ambulance Service (LAS) has been used in requirements engineering because of its known inconsistencies (Flowers 1996; Hunter 1995; Zowghi 2001; Sanni 2005). When the LAS system was deployed on October 26, 1992, it suffered from many problems. For example, the Automatic Vehicle Locating System (AVLS) could not track the location and status of some of the dispatch units resulting in the assignment of multiple vehicles to the same incidents (Finkelstein 1996). A fatal incident occurred when an ambulance failed to arrive at its destination on time. At times, the volume of calls and messages overwhelmed the system. After a series of serious problems, the system was shut down and terminated. Finkelstein (1996) found that some areas were not fully defined in the Software Requirements Specifications (SRS). A subset of the original functional software requirements for the LAS is given in Table 2 in Appendix B.

6.3 Production Cell at Karlsruhe

6.3.1 Description

A set of requirements (see Table 3 in Appendix B) are given for an existing/theoretical production cell (Lewerentz 1995). This set of requirements is tested based on the safety concerns and possible conflicts as discussed in the paper.

The author assumes that the CL is representative of the original natural language requirements. A pictorial representation of the system is shown in Figure 4. The words in the diagram are based on the modified CL lexicon. The original words are shown in parentheses.

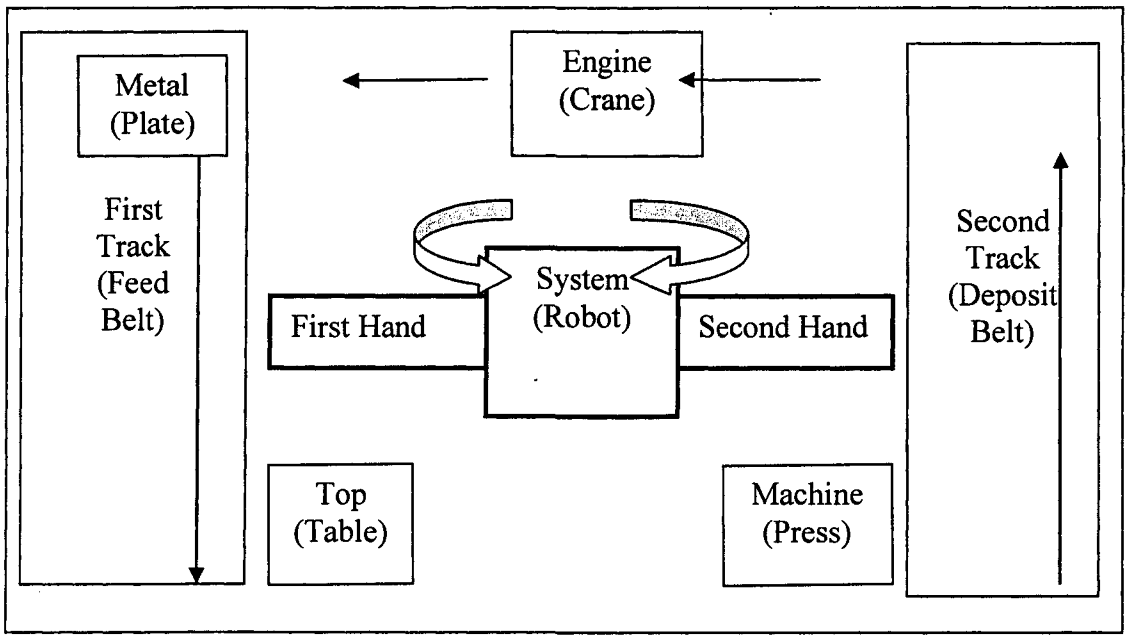

Figure 4: Production Cell System

6.4 Denver International Airport Automated Baggage Handling System (ABHS)

6.4.1 Description

Using similar principles in the general field of investigation such as accident reconstruction, the ABHS requirements were reconstructed by using a collection of information available in the literature and on the web. The ABHS is a complex software system that communicates with a number of baggage handling related devices in the airport (03.25.05 DIA Description). The system experienced numerous problems when it was implemented. The control software played a significant role in these problems. The actual requirements were not available. Therefore, in order to develop the requirements, the nouns in the DIA Description were used as a basis for building the functional software requirements for the system.

Chapter 7 Results

7.1 Sump Pump

The following pair of requirement statements are assumed to form an ideal requirement set, **R***, for the controller and is given by,

$$\mathbf{R}^* = \{r_1, r_2\}$$

where,

r_1: When the fluid level reaches the high threshold, then turn the pump on

r_2: When the fluid level reaches the low threshold, then turn the pump off.

Suppose we introduce a third statement that generates a known conflict of assigning multiple fluid levels to the action of turning the pump off. This conflict as described previously is a known class of conflict. We now have the following requirement set,

$$\mathbf{R} = \{r_1, r_2, r_3\}$$

where,

r_3: When the fluid level is between low and high then the pump is off.

The parse trees for the three requirements are shown in Figures 5 through 7.

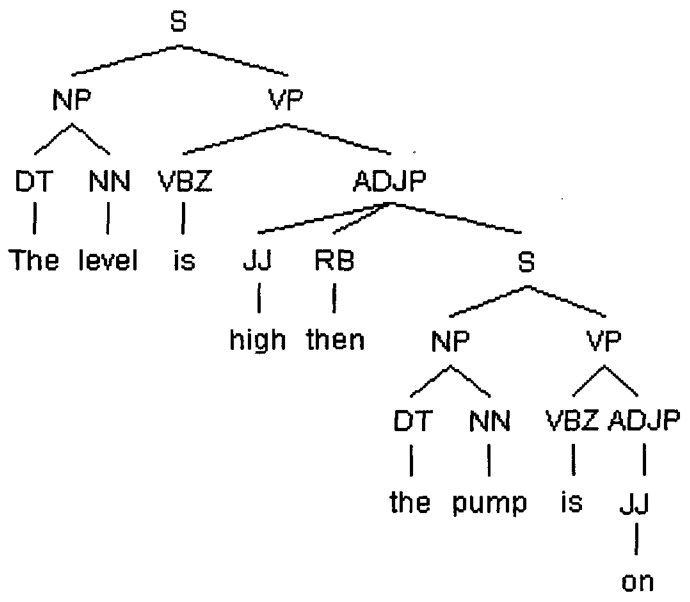

Figure 5: Requirement #1 Parse Tree

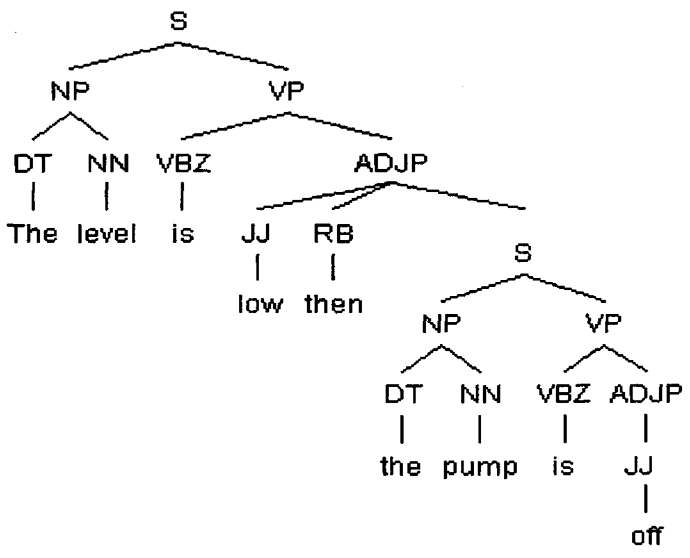

Figure 6: Requirement #2 Parse Tree

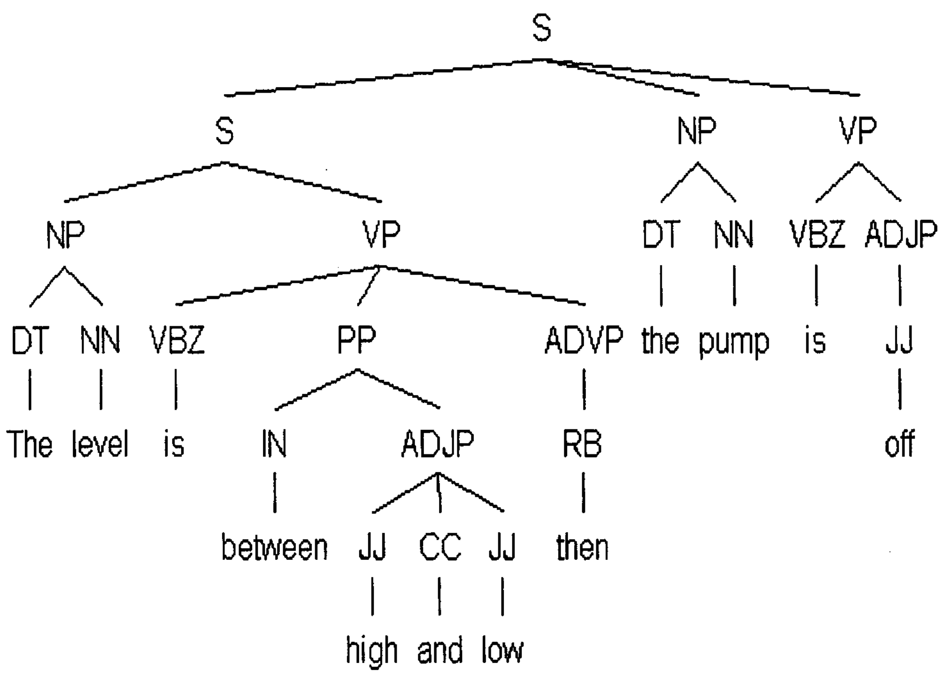

Figure 7: Requirement #3 Parse Tree

By applying Chen's transformation rules, the ER Models in Figures 8 to 10 were produced.

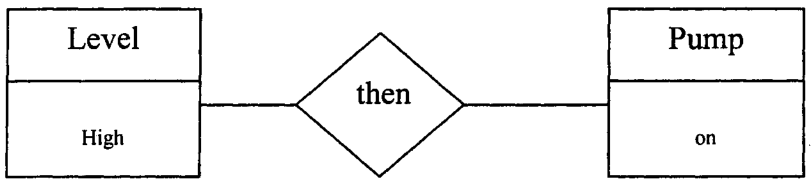

Figure 8: Requirement #1 ER Model

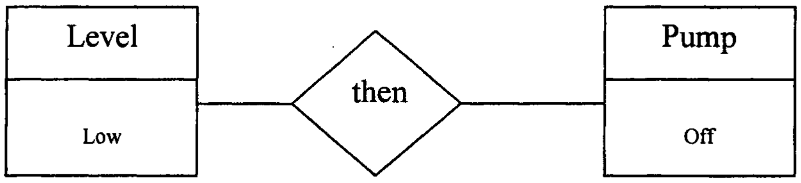

Figure 9: Requirement #2 ER Model

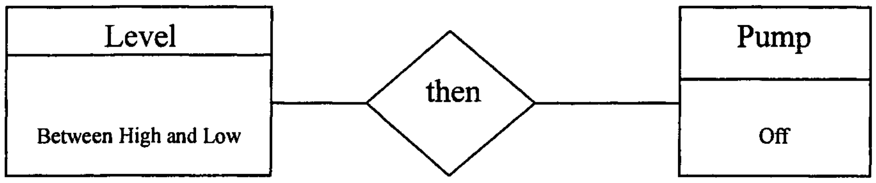

Figure 10: Requirement #3 ER Model

The following calculations describe how $\Psi(\pi_i, \pi_j)$ was determined for requirements 1 and 2.

$$|\Phi_E(\pi_1 \cap \pi_2)| = |\{level, pump\}|$$

$$|\Phi_R(\pi_1 \cap \pi_2)| = |\{then\}|$$

$$|\Phi_A(\pi_1 \cap \pi_2)| = |\{\phi\}|$$

$$F(\pi_1 \cap \pi_2) = 2 + 1 + 0 = 3$$

$F(\pi_1 - \pi_2) = 2$

$F(\pi_2 - \pi_1) = 2$

$S(\pi_1, \pi_2) = 3 - 2 - 2 = -1$

$\min(|\pi_1|, |\pi_2|) = |\pi_2| = 5$

$(|\pi_1| + |\pi_2|) = 5 + 5 = 10$

$$\Psi(\pi_1, \pi_2) = \frac{-1+10}{5+10} = 0.6$$

The value for $\Psi(\pi_i, \pi_j)$ can be calculated in a similar manner for each possible combination of requirements as shown in the table below

Table 4 Sump Pump $\Psi(\pi_i, \pi_j)$ Level

Pair	$\Psi(\pi_i, \pi_j)$
r_3, r_2	0.8
r_1, r_2	0.6
r_3, r_1	0.6

For this system, it is well known that r_1 and r_2 are not in conflict. This suggests that a value of 0.6 for $\Psi(\pi_i, \pi_j)$ is not sufficient to indicate a conflict. The value of $\Psi(\pi_3, \pi_2)$ indicates a potential conflict between r_3 and r_2. This can be attributed to similar actions of turning the pump off. $\Psi(\pi_3, \pi_1)$ does not show a conflict. This is because r_3 describes how the pump enters the off state while r_1 describes how the pump enters the on state.

7.1.1 Sump Pump Predicate Logic Comparison

Based on **R**, we can reason using predicate logic as follows.

A = When the level reaches the high threshold

B = the pump is on

C = When the level reaches the low threshold

D = the pump is off

E = When the level is between low and high

R can be stated as a set of logic statements, $\{\mathbf{A} \Rightarrow \mathbf{B}, \mathbf{C} \Rightarrow \mathbf{D}, \mathbf{E} \Rightarrow \mathbf{D}\}$. Given these logic statements, we can deduce that $\mathbf{C} \wedge \mathbf{E} \Rightarrow \mathbf{D}$. Submitting these statements to a theorem prover using Zowghi's approach (i.e., based on negation) will not return a logical inconsistency. This is true because the methodology relies heavily upon negation in detecting an inconsistency.

Alternatively, the conflict can be explained using partial sets, as follows.

- When fluid levels are high (fluid levels $\Rightarrow$ high)
- When fluid levels are low (fluid levels $\Rightarrow$ low)

- When some fluid levels are high and some fluid levels are low

  (some fluid levels $\Rightarrow$ high ^ some levels $\Rightarrow$ low)

- Pump is off (pump $\Rightarrow$ off)

- Pump is on (pump $\Rightarrow$ on)

- When fluid levels are low then the pump is off

  (fluid levels $\Rightarrow$ low $\Rightarrow$ pump $\Rightarrow$ off)

- When some fluid levels are high and some fluid levels are low then the pump is off

  (some levels $\Rightarrow$ high ^ some levels$\Rightarrow$ low) $\Rightarrow$ (pump $\Rightarrow$off)

This is equivalent to **C ^ E $\Rightarrow$ D.**

The conflict occurs when some fluid levels shut the pump off irrespective of if they are high or low levels. The set of levels contains both high and low (i.e. structurally overlapping requirements), when ideally it should contain high or low (i.e. distinct requirements). Thus, we find the same state generating the same action (i.e., the pump being switched off) when it should be a single state (i.e. as opposed to partial states).

Hausmann (2002) also used a similar technique in detecting conflict in use case diagrams. When he gave an example of a conflict between two use case transformations of "pay bill" and "settle bill", both transformations produce the result of removing the relationship "owns" between the goods and the shop (i.e., they overlap in items that are deleted). In effect, they cancel each other out, creating some form of disagreement. Hausmann's rule can be restated as "If two transformations disable one another, they cannot be part of the same transformation sequence". This is also part of the same class of conflict in terms of "Two life-forms competing for the same limited resource." Ideally they are in conflict and they cannot co-exist in the same space. The author believes this concept to be also true of functional requirement statements.

7.2 London Ambulance Service (LAS)

For requirements IRC.2 and OM.1 $\Psi(\pi_i, \pi_j)$ was found as follows.

$|\Phi_E(\pi_1 \cap \pi_2)| = |\{operator, call, ambulance\}|$

$|\Phi_R(\pi_1 \cap \pi_2)| = |\{receives, should\ dispatch\}|$

$|\Phi_A(\pi_1 \cap \pi_2)| = |\{phone, nearest, available\}|$

$F(\pi_1 \cap \pi_2) = 3 + 2 + 3 = 8$

$F(\pi_1 - \pi_2) = 3$

$F(\pi_2 - \pi_1) = 0$

$S(\pi_1, \pi_2) = 8 - 3 - 0 = 5$

$\min(|\pi_1|, |\pi_2|) = |\pi_2| = 8$

$(|\pi_1| + |\pi_2|) = 11 + 8 = 19$

$$\Psi(\pi_1, \pi_2) = \frac{5+19}{8+19} = .89$$

The results for all pairings are shown in Table 5.

Table 5: LAS $\Psi(\pi_i, \pi_j)$ level

(r_i, r_j)	$\Psi(\pi_i, \pi_j)$
IRC.2|OM.1	0.89
OM.1|OM.2	0.8
IRC.3|OM.1	0.77
IRC.3|OM.2	0.75
IRC.2|IRC.3	0.75
IRC.2|OM.2	0.72
IRC.1|IRC.2	0.29
IRC.1|IRC.3	0.25
IRC.1|OM.1	0
IRC.1|OM.2	0

Values for $\Psi(\pi_i, \pi_j)$ as low as 0.72 indicate potential conflicts in the set. For IRC.2|OM.2, the potential conflict appears to be redundancy. This is also true for the IRC.2|IRC.3 pair where there is unnecessary in formation in IRC.3 about the ambulance. This suggests that there may be a threshold value for $\Psi(\pi_i, \pi_j)$ between redundancies and conflicts. Further research would be necessary to investigate this possible threshold.

The IRC.2|OM.1 pair has the highest $\Psi(\pi_i, \pi_j)$ value, suggesting that this pair of requirements should be reexamined. On closer examination of the two requirements, OM.1 could be interpreted as an ambulance being dispatched whenever a phone call is received (e.g., wrong number or information request). This may relate to the known problem of large call volumes as previously discussed. This also relates to the same class of conflict described by Lorenz (1973), when two life forms that exist are so similar that they compete for the same scarce resources. In this case, we have medical and non-medical related emergency calls competing for the same resource (i.e., the call center).

Another interesting observation is that those pairs with IRC.1 consistently have the lowest values. Therefore, this implies that IRC.1 is the requirement with the least conflicts because its pairings with other requirements has consistently low $\Psi(\pi_i, \pi_j)$ values.

Based on the results thus far, when a structural inconsistency exists, requirements may tend to overlap more than necessary. It appears that when requirement pairs have a $\Psi(\pi_i, \pi_j)$ value near 0.8, then these requirements should be reexamined and revised because a conflict may exist.

7.2.1 London Ambulance Service Predicate Logic Comparison

Since this test case is taken from Zowghi et al (2001), we make a comparison based on our method and their predicate logic method. They reported a logical inconsistency for IRC.3|OM.1. This pair can be found in Table 4 and has a value of 0.77 for $\Psi(\pi_i, \pi_j)$. This suggests a threshold value of 0.77 for $\Psi(\pi_i, \pi_j)$, but this may be context dependent and warrants further research. Zowghi et al did not address the IRC.2 and OM.1 pair in their analysis. This confirms two assumptions made earlier that the logic method checks for the existence of logical inconsistency while the proposed methodology can potentially measure the degree of conflict based on the potential structural inconsistency.

7.3 Production Cell at Karlsruhe

Table 6 shows the results for the requirements set for $\Psi(\pi_i, \pi_j)$ values greater than 0.5. All values can be found in Appendix B.

Table 6: Production Cell $\Psi(\pi_i, \pi_j)$ levels

r_i	r_j	$\Psi(\pi_i, \pi_j)$
R003	R006	0.8
R006	R008	0.79
R003	R008	0.64
R001	R007	0.6

The pair of statements R003 and R006 has the highest value of inconsistency and (R006 , R008 pair is also very close to this value. We can view this in terms of the first and second hand on the robot retrieving the metal from the same location or an unknown location. We know that the first hand of the robot gets the metal from the top. The second hand gets the metal from the open machine. Specification of the requirement hints at under specification and vagueness (i.e. ambiguity). This conflict is related to an instance of logical inconsistency reported by Zowghi (2001) as a pair of tacit requirements. Tacit requirement are the type of requirements that leave designers, developers of systems guessing or assuming and filling in the blanks.

For R006 and R008, we see that the second hand gets the metal seems to be in conflict with the engine gets the metal. These two statements are vague and do not clearly state the location where the "getting" of the metal should occur.

7.3.1 Production Cell Predicate Logic Comparison

The logical comparison is interesting because it not only shows a clear distinction between both methodologies but shows that in order for the logical statement to be clearly understood the requirement statements have to undergo some minor revisions. This is important because the premise of a potential structural inconsistency is that a revision is implied. For this test case, it appears that when a requirement statement requires a minor revision in order for it to be adequately represented in predicate logic, then it may contain some potential structural inconsistency. This can be confirmed from the logical statements as follows.

R001: The first track $\Rightarrow$ metal moved to the top

R002: The top moves to a good position $\Rightarrow$ removal by the first hand of the system

R003: The first hand of the system $\Rightarrow$ the metal is retrieved

R004: The system turns $\Rightarrow$ the first hand points to the open machine

R005: The machine $\Rightarrow$ the metal is changed ^ open for the second hand

R006: The second hand of the system $\Rightarrow$ the retrieved metal ^ turns ^ metal put on second track

R007: The second track $\Rightarrow$ metal moves to the engine

R008: The engine $\Rightarrow$ the retrieved metal got ^ metal put on first track

The production cell is setup so that it can continuously. This implies that each requirement is connected to the next requirement such that $C \Rightarrow B$ and $B \Rightarrow C$. This can be assumed true for R001 through R008. Not withstanding, this additional knowledge based on Zowghi's approach does not improve the detection of a logical inconsistency for this test case. This additional information can be used to obtain new logical statements. Now comparing both methods we have the following.

R003: The first hand of the system $\Rightarrow$ the metal is retrieved

R006: The second hand of the system $\Rightarrow$ the retrieved metal ^ turns ^ metal put on second track

R008: The engine $\Rightarrow$ the retrieved metal ^ metal put on first track

Based on Zowghi's methodology it is difficult to determine whether an inconsistency exists. The logical method does not appear applicable in this instance due to its reliance on negation.

7.4 Denver International Airport (DIA) Automated Baggage Handling System (ABHS)

Based on an initial description given by an online project (DIA Description) and Neufville (1994), the following objects were specified for the system in natural language and controlled language,

Table 7: DIA CL Objects

NL Objects	CL Objects
System	System
Network	Wire
DCV	Container
PLC	Computer
Motor	Engine
Airplane	Airplane
Passenger	Person
Baggage	Bag
Conveyor Belt	Line
Check-in Agent	Agent
Sensor	Device

Table 8 shows the $\Psi(\pi_i, \pi_j)$ values for requirements pairs where $\Psi(\pi_i, \pi_j) \geq 0.6$ The complete set of values can be found in Appendix B.

Table 8: ABHS $\Psi(\pi_i,\pi_j)$ levels

r_i	r_j	$\Psi(\pi_i,\pi_j)$
R008	R009	0.75
R008	R010	0.75
R009	R010	0.75
R011	R012	0.75
R017	R025	0.75
R014	R020	0.6
R016	R025	0.6
R018	R024	0.6

Upon examination, these pairings do not show any apparent conflict though the R008|R009 pair indicates there may be some ambiguity in the signal being sent to possibly different devices. The R017|R025 pair shows no conflict or redundancy. This indicates that there may be missing information in the ER model that would help differentiate these two requirements and lower the $\Psi(\pi_i,\pi_j)$ value. Another possibility is that the semantics of language should be considered.

7.4.1 DIA Automated Baggage System Predicate Logic Comparison

This comparison shows differences in the methodologies. The requirements in Table 7 can be represented as the following logic statements.

R008: The wire $\Rightarrow$ a signal is transmitted to the device

R009: The wire $\Rightarrow$ a signal is transmitted to the small container

R010: The wire $\Rightarrow$ a signal is transmitted to the large container

R011: A person $\Rightarrow$ a small bag is owned

R012: A person $\Rightarrow$ a large bag is owned

R017: The system sends an empty container to a bag

R025: The system sends a full container to a person

It appears that when a requirement has to been considerably modified in order to represent it in logic; an inconsistency can be introduced or it may not be a functional requirement. For example, a person has a small or large bag may not necessarily be requirement though the person interacts with the system. This was also the case based on the LAS results where IRC.1 "A medical emergency is either an illness or an accident". Though this requirement had the lowest PSI, it was a simple classification of a medical emergency. For the ABHS, R011 and R012 describe the function of a person with respect to the function of the system.

Chapter 8 Conclusions

The results of the research show reasonable support for the research proposition that when a high numeric value of potential structural inconsistency is observed for a pair of functional software requirements, then a conflict is likely. The new metric for detecting a possible conflict via potential structural inconsistency uses a normalized Tversky's similarity measure that can be used to compare pairs of requirements using a range of 0 to 1. The normalization makes it easier to analyze and reason about the actual similarity values. One issue for further research is that $\Psi(\pi_i, \pi_j)$ does not differentiate between a conflict, redundancy, and ambiguity (i.e. these concepts appear to be interrelated). Additional information (e.g., semantics) is necessary to differentiate these characteristics of a requirement statement.

This research has addressed the problem of reasoning about inconsistencies and conflict using different representations of software requirements specification. It has been shown that the semi-formal of representation identifies a slightly different set of inconsistencies than formal methods. A potential advantage of the new metric is that it provides a numeric indicator for the degree of conflict versus the Boolean indicators for logical inconsistency.

A high value ranging between 0.7 and 0.8 was repeatedly observed in all the four case studies from different fields. This leads a strong argument for reexamining or revising requirements around this value. The numeric scaling of the requirements may be useful in the prioritization of requirements. The numeric characterization of requirements document through the use of structural metrics, introduces a numerical metric into the detection of conflict that does not exist in the logical method. The implication of this is that requirement specifications can be prioritized based on numeric value of structural metrics. The need for metric based prioritization cannot be overstated due to the large volume of information in a requirements document.

This study provided some insights into the nature of conflict as it relates to the domain of requirements engineering. One observed characteristic was the "the propagation" of conflict through software requirement specifications. A conflict can be considered to be an object that can propagate through the requirements based on the overall structural relationship between the requirement pairings. This was observed in part through the chaining of requirement pairings with high levels of PSI in test cases 3 and 4. Some requirements were reoccurring, forming chains of structural inconsistency. This concept of chained structured pairs of requirements may show some form of conflict propagation through the requirements document.

8.1 Future Research

Results from this study indicate that further differentiation of problems with requirements (e.g., redundancy and ambiguity) may be possible with some revisions to the metric. If multiple thresholds can be found, then this approach could provide a more accurate assessment of requirements.

Additional information from the semantics of the statements could be useful in providing greater differentiation. New methods are needed that can characterize and check the semantics of pairs of requirement statements.

To make this method more suitable for routine use, full automation of the methodology would be advantageous but would be very difficult to achieve. Several steps currently require manual intervention due to the complexities of language.

References

AECMA simplified English "A Guide for the Preparation of aircraft maintenance documentation in the international aerospace maintenance language". AECMA 2004

Ambriola, V. and Gervasi, V. "Processing Natural Language Requirements" International Conference on Automated Software Engineering November 02 - 05, 1997

Ambriola, V. and Gervasi, V. "An Environment for Cooperative Construction of Requirement Bases". Proc. 8th Int. Conf. on Software Engineering Environments, IEEE Computer Society Press, Los Alamitos, 1997, 124-130

Chechik, M., and J. Gannon, "Automated Analysis of Consistency Between Requirements and Design," *IEEE Transactions on Software Engineering, 27*, 7 (July 2001), 651-672

Chen P.P., "The Entity-Relationship Model: Toward a Unified View of Data". ACM TODS 1 (1976) 1-36

Chen P.P., "English Sentence Structures and Entity-Relationship Diagrams". Information Sciences, (1983) 127-149

Cherfi S. Akoka J. and Comyn-Wattiau I., "Conceptual Modeling Quality- From EER to UML Schemas Evaluation". ER2002, LNCS 2503, 2002, pp. 414-428

Easterbrook S. M. and Nuseibeh B. A. "Using ViewPoints for Inconsistency Detection". Software Engineering Journal, Volume 11, No 1, Jan 1996

Flowers, Stephen (1996), "Software Failure: Management Failure", Chichester: John Wiley and Sons.

Finkelstein A. and Dowell J. "A Comedy Of Errors: The London Ambulance Service Case Study". In Proc. Of the 8th International Workshop on software Specification & Design, pages 2-4. IEEE CS Press, 1996.

Genero M., Jimenez L., Piattini M., "Defining and Validating Metrics for Assessing the Maintainability of Entity-Relationship Diagrams". Working Paper 2003.

Genero M., Poels G. and Pianttini M., "Defining and Validating Measures for Conceptual Data Model Quality". CAISE 2002. Toronto. Canada. LNCS 2348. (eds.) Springer-Verlag. 724-727.

Genero M., Jimenez L., Piattini M., "Measuring the Quality of Entity Relationship Diagrams", in the proceedings of ER2000 conference, LNCS 1920, pp. 513-526.

Gervasi V., "Environment Support for Requirements Writing and Analysis". PhD thesis, University of Pisa, March 2000

Gleick, J. (1996) "A bug and a crash," The New York Times Magazine, December 1, 1996.

Gervasi V. and Nuseibeh B, "Lightweight Validation of Natural Language Requirements". Software: Practice & Experience, Feb. 2002, 32(2): 113-133

Gries D. and Schneider B. "A Logical Approach To Discrete Math". Springer-Verlag Publishers, 1993, Pages 59 and 204.

Hausmann J.H, Heckel R., Taentzer G. "Detection of Conflicting Functional Requirements in a Use Case-Driven Approach". ICSE, May 19-25, 2002.

Hooman J. and Vain. J. “An Integrated Technique for Developing Real-Time Systems”. 1995. Proceedings., Seventh Euromicro Workshop on 14-16 June 1995 Page(s):236 - 243

Hunter A. and Nuseibeh B., "Managing Inconsistent Specifications: Reasoning, Analysis and Action", Department of Computing Technical Report, DoC 95/15, Imperial College, London, UK, October 1995.

IEEE Std 830-1998, “IEEE recommended practice for software requirements specifications”, 20 Oct 1998

IEEE Std 1233-1998, “IEEE guide for developing system requirements specifications”, 22 Dec 1998

Jones T.H. and Song I.Y, ”Binary Equivalents of Ternary Relationships in Entity-Relationship Modeling: a Logical Decomposition Approach” Journal of Database Management, April-June, 2000, pp. 12 – 19

Kamsties, E., Berry, D.M. and Paech, B., ``Detecting Ambiguities in Requirements Documents Using Inspections," *Workshop on Inspections in Software Engineering (WISE'01)*, pp. 68-80 Paris, France, Software Quality Research Lab

Klein D. and Manning C.D., "A* Parsing: Fast Exact Viterbi Parse Selection", Proceedings of HLT-NAACL'03, 2003

Lamsweerde A., Darimont R. and Letier E., “Managing Conflicts in Goal-Driven Requirements Engineering”. IEEE Transactions on Software Engineering, Special Issue on Managing Inconsistency in Software Development, November 1998.

Lewerentz Claus and Lindner Thomas, (1995) “Production Cell,”. A Comparative Study in Formal Specification and Verification. KORSO Book 1995, pp. 388-416

Mark R. Blackburn, Robert Busser, Aaron Nauman, Robert Knickerbocker, Richard Kasuda: Mars Polar Lander Fault Identification Using Model-based Testing. ICECCS 2002: 163

Nelson, Clark, and Spurlock. "Curing the Software Requirements And Cost Estimating Blues," PM: Nov-Dec, 1999.

Neufville. "The Baggage System at Denver: Prospects and Lessons," Journal of Air Transport Management, Vol. 1, No. 4, Dec., pp. 229-236, 1994.

Nuseibeh B. A. and Easterbrook S. M., "Requirements Engineering: A Roadmap", In A. C. W. Finkelstein (ed) "The Future of Software Engineering ". (Companion volume to the proceedings of the 22nd International Conference on Software Engineering, ICSE' 2000). IEEE Computer Society Press.

Nuseibeh B. A., Easterbrook S. M. and Russo A., "Making Inconsistency Respectable in Software Development" Journal of Systems and Software, 56(11), November 2001, Elsevier Science Publishers

Palmer J. D. and Fields A. N. "An Integrated Environment for Requirements Engineering". IEEE Software 9, Number 3, May 1992, 80-85

Ristad, E., and Yianilos, P., (1998), "Learning String Edit Distance," IEEE Transactions on Pattern Analysis and Machine Intelligence, Vol. 20(5), pp. 522–532.

Robinson, W.N., Volkov, S., Conflict-Oriented Requirements Restructuring, GSU CIS Working Paper 99-5, Georgia State University, Atlanta, GA, April 9, 1999.

Rodriguez, A., Egenhofer, M., (2003),"Determining Semantic Similarity among Entity Classes from Different Ontologies", IEEE Transactions on Knowledge and Data Engineering, Vol. 15 No. 2., pp. 442-456.

Sanni Gboyega "Detecting Inconsistencies in Functional Software Requirements". Institute of Industrial Engineering Annual Conference, May 15th –19th 2005.

Spanoudakis G. and Finkelstein A. "Reconciling Requirements: A Method for Managing Interference, Inconsistency and Conflict" Automated Software Engineering Volume 3: 1997, 433-457

Spanoudakis G., Finkelstein A. and Till D. "Overlaps in Requirements Engineering". Automated Software Engineering 6(2): (1999) 171-198

Thalheim B., "Entity-Relationship Modeling, Foundations of Database Technology". 2000 Springer Publishers

Tjoa A.M., Berger L, "Transformation of Requirement Specification Expressed in Natural Language into an EER Model. ER 1993. 206-217.

Tversky A., (1977), "Features Of Similarity," Psychological Reviews, Vol. 84 (4), pp. 327-352.

Wilson M., Rosenberg H. and Hyatt E., "Automated Quality Analysis Of Natural Language Requirements Specifications". Fourteenth Annual Pacific Northwest Software Quality Conference, Portland, OR - October 1996

Zowghi D., Gervasi V. and McRae A. "Using default reasoning to discover inconsistencies in natural language requirements." In Proc. of the 8th Asia-Pacific Software Engineering Conference, pages 133-140, Dec. 2001.

Web References

Cambridge Advanced Learner's Dictionary http://dictionary.oed.com, date accessed: 28th of April 2004

Oxford Dictionary http://dictionary.oed.com, date accessed: 28th of April 2004

Berry D.M. http://se.uwaterloo.ca/~dberry/natural.language.html, date accessed: 25th of March 2004

Chen P.P., http://bit.csc.lsu.edu/~chen/, date accessed: 25th of March 2004

Gervasi V. "CICO a fuzzy NL Parser" http://circe.di.unipi.it/Cico/, date accessed: 28th of March 2004

"Stanford Lexicalized Parser" http://www-nlp.stanford.edu/, date accessed: 28th of March 2004

"Penn TreeBank Project" http://www.cis.upenn.edu/~treebank/home.html, date accessed: 31st of March 2004

DIA Description, http://www.csc.calpoly.edu/~dstearns/SchlohProject/function.html, date accessed: 25th of March 2004

"NASA needs better Software" http://www.computerworld.com/softwaretopics/software/story/0,10801,78362,00.html, date accessed: 27th of May 2004

"Unified Modeling Language" , http://www.uml.org/, date accessed: 20th of September 2005

Appendix A: Penn Tree Bank Tagset

POS Tag	Description	Example
CC	coordinating conjunction	and
CD	cardinal number	1, third
DT	Determiner	the
EX	existential there	*there* is
FW	foreign word	d'hoevre
IN	preposition/subordinating conjunction	in, of, like
JJ	Adjective	green
JJR	adjective, comparative	greener
JJS	adjective, superlative	greenest
LS	list marker	1)
MD	Modal	could, will
NN	noun, singular or mass	table
NNS	noun plural	tables
NNP	proper noun, singular	John
NNPS	proper noun, plural	Vikings
PDT	Predeterminer	*both* the boys
POS	Possessive ending	friend*'s*
PRP	personal pronoun	I, he, it
PRP$	Possessive pronoun	my, his
RB	Adverb	however, usually, naturally, here, good
RBR	adverb, comparative	better
RBS	adverb, superlative	best
RP	Particle	give *up*
TO	To	*to* go, *to* him
UH	Interjection	uhhuhhuhh
VB	verb, base form	take
VBD	verb, past tense	took
VBG	verb, gerund/present participle	taking
VBN	verb, past participle	taken
VBP	verb, sing. present, non-3d	take
VBZ	verb, 3rd person sing. Present	takes
WDT	wh-determiner	which
WP	wh-pronoun	who, what
WP$	possessive wh-pronoun	whose
WRB	wh-abverb	where, when

Appendix B: Test Case Requirement

Sump Pump (SP):

Table B.1 SP NL Functional Requirements

Requirements ID	NL Functional Requirements Specifications
1	When the fluid level reaches the high threshold, then turn the pump on
2	When the fluid level reaches the low threshold, then turn the pump off
3	When the fluid level is between low and high then the pump is off

Table B.2 SP CL Functional Requirements

Requirements ID	CL Functional Requirements Specifications
1	When the fluid level is high then the pump is on
2	When the fluid level is low then the pump is off
3	When the fluid level is between low and high then the pump is off

Table B.3 SP PSI Metric

r_i	r_j	$\Psi(\pi_i, \pi_j)$
3	2	0.8
1	2	0.6
3	1	0.6

London Ambulance Service(LAS):

Table B.4 LAS NL Functional Requirements

Requirements ID	**NL Functional Requirements Specifications**
IRC.1	A medical emergency is either an illness or an accident
IRC.2	When an operator receives a phone call concerning a medical emergency, the operator should dispatch the nearest available ambulance
IRC.3	When an operator receives a phone call concerning a non-medical emergency, the operator should not dispatch an ambulance and he should transfer the phone call to another service.
OM.1	When an operator receives a phone call, the operator should dispatch the nearest available ambulance.
OM.2	When an operator receives a phone call, if an ambulance is not the nearest available, then the operator should not dispatch that ambulance.

Table B.5 LAS PSI Metric

r_i	r_j	$\Psi(\pi_i, \pi_j)$
IRC.2	OM.1	0.89
OM.1	OM.2	0.8
IRC.3	OM.1	0.77
IRC.3	OM.2	0.75
IRC.2	IRC.3	0.75
IRC.2	OM.2	0.72
IRC.1	IRC.2	0.29
IRC.1	IRC.3	0.25
IRC.1	OM.1	0
IRC.1	OM.2	0

Production Cell:

Table B.6 PC NL Functional Requirements

Requirements ID	NL Functional Requirement Specification
R001	The feed belts conveys the metal plate to the elevating rotary table
R002	The elevating rotary table is moved to a position adequate for unloading by the first robot arm
R003	The first robot arm picks up the metal plate
R004	The robot rotates counterclockwise so that arm 1 points to the open press, places the metal plate into it and then withdraws from the press
R005	The press forges the metal blank and opens again
R006	The robot retrieves the metal plate with its second arm, rotates further and unloads the plate on the deposit belt
R007	The deposit belt transports the plate to the traveling crane
R008	The traveling crane picks up the metal plate, moves to the feed belt, and unloads the metal plate on it

Table B.7 PC CL Functional Requirements

Requirements ID	CL Functional Requirements Specifications
R001	The first track moves the metal to the top
R002	The top moves to a good position for removal by the first hand of the system
R003	The first hand of the system gets the metal
R004	The system turns so that the first hand points to the open machine
R005	The machine changes the metal and opens for the second hand
R006	The second hand of the system gets the metal and turns to put the metal on the second track
R007	The second track moves the metal to the engine
R008	The engine gets the metal and puts the metal on the first track

Table B.8 PC PSI Metric

r_i	r_j	$\Psi(\pi_i, \pi_j)$
1	2	0
1	3	0.23
1	4	0.18
1	5	0.19
1	6	0.35
1	7	0.60
1	8	0.38
2	3	0.4
2	4	0.14
2	5	0.16
2	6	0.14
2	7	0
2	8	0
3	4	0.2
3	5	0.43
3	6	0.8
3	7	0.23
3	8	0.64
4	5	0.32
4	6	0.14
4	7	0
4	8	0
5	6	0.47
5	7	0.38
5	8	0.33
6	7	0.35
6	8	0.79
7	8	0.38

Denver International Airport (DIA):

Table B.9 DIA CL Functional Requirements

Requirements ID	**CL Requirement Specifications**
R001	The system connects with the database
R002	The system gets the time of a flight from the airport
R003	The system gets the time of a flight to the airport
R004	The system measures the flow of a person through the airport
R005	The system connects a code with a flight
R006	The system connects a flight with a person
R007	The system sends a signal to the wire
R008	The wire transmits a signal to the device
R009	The wire transmits a signal to the small container
R010	The wire transmits a signal to the large container
R011	A person has a small bag
R012	A person has a large bag
R013	An agent puts a tag on a bag
R014	An agent puts a bag on the line
R015	A line holds a bag
R016	The system finds a container
R017	The system sends an empty container to a bag

R018	The container moves on the track
R019	The container has a tag
R020	The equipment gets the bag from the line and puts the bag in the container
R021	The device identifies the code on the tag
R022	The device sends the code to the system
R023	The computer finds the engine
R024	The engine pushes the container on the track
R025	The system sends a full container to a person

Table B.10 DIA PSI Metric

r_i	r_j	$\Psi\ (\pi_1, \pi_2)$
1	2	0.3
1	3	0.3
1	4	0.25
1	5	0.27
1	6	0.25
1	7	0.3
1	8	0
1	9	0
1	10	0
1	11	0
1	12	0
1	13	0
1	14	0
1	15	0
1	16	0.33
1	17	0.3
1	18	0
1	19	0
1	20	0
1	21	0
1	22	0
1	23	0
1	24	0
1	25	0.3
2	3	0.5
2	4	0.21
2	5	0.23
2	6	0.21
2	7	0.25
2	8	0
2	9	0
2	10	0
2	11	0
2	12	0
2	13	0
2	14	0
2	15	0
2	16	0.3

2	17	0.25
2	18	0
2	19	0
2	20	0.2
2	21	0
2	22	0
2	23	0
2	24	0
2	25	0.25
3	4	0.21
3	5	0.23
3	6	0.21
3	7	0.25
3	8	0
3	9	0
3	10	0
3	11	0
3	12	0
3	13	0
3	14	0
3	15	0
3	16	0.3
3	17	0.25
3	18	0
3	19	0
3	20	0.2
3	21	0
3	22	0
3	23	0
3	24	0
3	25	0.25
4	5	0.19
4	6	0.17
4	7	0.21
4	8	0
4	9	0
4	10	0
4	11	0
4	12	0
4	13	0
4	14	0
4	15	0
4	16	0.25
4	17	0.21
4	18	0
4	19	0

4	20	0
4	21	0
4	22	0
4	23	0
4	24	0
4	25	0.21
5	6	0.75
5	7	0.23
5	8	0
5	9	0
5	10	0
5	11	0
5	12	0
5	13	0
5	14	0
5	15	0
5	16	0.27
5	17	0.23
5	18	0
5	19	0
5	20	0
5	21	0.23
5	22	0.23
5	23	0
5	24	0
5	25	0.23
6	7	0.21
6	8	0
6	9	0
6	10	0
6	11	0.21
6	12	0.21
6	13	0
6	14	0
6	15	0
6	16	0.25
6	17	0.21
6	18	0
6	19	0
6	20	0
6	21	0
6	22	0
6	23	0
6	24	0
6	25	0.21
7	8	0.25

7	9	0.25
7	10	0.25
7	11	0
7	12	0
7	13	0
7	14	0
7	15	0
7	16	0.3
7	17	0.5
7	18	0
7	19	0
7	20	0
7	21	0
7	22	0.25
7	23	0
7	24	0
7	25	0.5
8	9	0.75
8	10	0.75
8	11	0
8	12	0
8	13	0
8	14	0
8	15	0
8	16	0
8	17	0
8	18	0
8	19	0
8	20	0
8	21	0
8	22	0
8	23	0
8	24	0
8	25	0
9	10	0.75
9	11	0
9	12	0
9	13	0
9	14	0
9	15	0
9	16	0
9	17	0
9	18	0
9	19	0
9	20	0
9	21	0

9	22	0
9	23	0
9	24	0
9	25	0
10	11	0
10	12	0
10	13	0
10	14	0
10	15	0
10	16	0
10	17	0
10	18	0
10	19	0
10	20	0
10	21	0
10	22	0
10	23	0
10	24	0
10	25	0
11	12	0.75
11	13	0
11	14	0.25
11	15	0.3
11	16	0
11	17	0
11	18	0
11	19	0.3
11	20	0.4
11	21	0
11	22	0
11	23	0
11	24	0
11	25	0
12	13	0
12	14	0.25
12	15	0.3
12	16	0
12	17	0
12	18	0
12	19	0.3
12	20	0.4
12	21	0
12	22	0
12	23	0
12	24	0
12	25	0

13	14	0.5
13	15	0
13	16	0
13	17	0
13	18	0
13	19	0.3
13	20	0.2
13	21	0
13	22	0
13	23	0
13	24	0
13	25	0
14	15	0.3
14	16	0
14	17	0
14	18	0
14	19	0
14	20	0.6
14	21	0
14	22	0
14	23	0
14	24	0
14	25	0
15	16	0
15	17	0
15	18	0
15	19	0
15	20	0.46
15	21	0
15	22	0
15	23	0
15	24	0
15	25	0
16	17	0.6
16	18	0.33
16	19	0.33
16	20	0
16	21	0
16	22	0
16	23	0.33
16	24	0.3
16	25	0.6
17	18	0.3
17	19	0.3
17	20	0
17	21	0

17	22	0.25
17	23	0
17	24	0.25
17	25	0.75
18	19	0.33
18	20	0
18	21	0
18	22	0
18	23	0
18	24	0.6
18	25	0.3
19	20	0
19	21	0
19	22	0
19	23	0
19	24	0.3
19	25	0.3
20	21	0
20	22	0
20	23	0
20	24	0
20	25	0
21	22	0.5
21	23	0
21	24	0
21	25	0
22	23	0
22	24	0
22	25	0.25
23	24	0.3
23	25	0
24	25	0.25