Artificial Intelligence Applications for Improved Software Engineering Development:
New Prospects

Farid Meziane
University of Salford, UK

Sunil Vadera
University of Salford, UK

Information Science REFERENCE

INFORMATION SCIENCE REFERENCE
Hershey · New York

Director of Editorial Content:	Kristin Klinger
Senior Managing Editor:	Jamie Snavely
Assistant Managing Editor:	Michael Brehm
Publishing Assistant:	Sean Woznicki
Typesetter:	Sean Woznicki
Cover Design:	Lisa Tosheff
Printed at:	Yurchak Printing Inc.

Published in the United States of America by
Information Science Reference (an imprint of IGI Global)
701 E. Chocolate Avenue
Hershey PA 17033
Tel: 717-533-8845
Fax: 717-533-8661
E-mail: cust@igi-global.com
Web site: http://www.igi-global.com/reference

Library of Congress Cataloging-in-Publication Data

Artificial intelligence applications for improved software engineering development : new prospects / Farid Meziane and Sunil Vadera, editors.
p. cm.
Includes bibliographical references and index.
Summary: "This book provides an overview of useful techniques in artificial intelligence for future software development along with critical assessment for further advancement"--Provided by publisher.

ISBN 978-1-60566-758-4 (hardcover) -- ISBN 978-1-60566-759-1 (ebook) 1. Computer software--Development. 2. Artificial intelligence. I. Meziane, Farid. II. Vadera, Sunil.
QA76.76.D47A82 2009
006.3--dc22

2009018603

British Cataloguing in Publication Data
A Cataloguing in Publication record for this book is available from the British Library.

Chapter 14
Artificial Intelligence in Software Engineering
Current Developments and Future Prospects

Farid Meziane
University of Salford, UK

Sunil Vadera
University of Salford, UK

ABSTRACT

Artificial intelligences techniques such as knowledge based systems, neural networks, fuzzy logic and data mining have been advocated by many researchers and developers as the way to improve many of the software development activities. As with many other disciplines, software development quality improves with the experience, knowledge of the developers, past projects and expertise. Software also evolves as it operates in changing and volatile environments. Hence, there is significant potential for using AI for improving all phases of the software development life cycle. This chapter provides a survey on the use of AI for software engineering that covers the main software development phases and AI methods such as natural language processing techniques, neural networks, genetic algorithms, fuzzy logic, ant colony optimization, and planning methods.

INTRODUCTION

The software engineering crisis was diagnosed about a half a century ago. Since then a plethora of methods, methodologies, notations, programming languages and environments have been developed and significant progress has been made. However, the nature and complexity of the software being developed has also changed and this has nearly annulled this progress. High rates of software failure continue to be reported, software is still expensive and over budget and it is still difficult to predict the delivery date of software (Fox & Spence, 2005). Parallel to the development of software engineering, there has been visible growth of related disciplines such as Artificial Intelligence having an impact on software development (Pedrycz & Peters, 1997; Rech & Althoff, 2004). As the preceding chapters of the book show, there is significant potential in

DOI: 10.4018/978-1-60566-758-4.ch014

using AI for supporting and enhancing software engineering. The aim of this chapter is to provide a survey of existing research on using AI methods such as natural language processing techniques, neural networks, genetic algorithms, and planning methods for the full software development life cycle. The chapter is broadly structured in a similar way to the parts of the book. There are sections on AI in Planning and Project Effort Estimation, Requirements Engineering and Software Design and Software Testing. Within each section, there are subsections surveying the use of particular AI techniques. The chapter concludes with a summary of the major issues with using AI for enhancing software development and future directions of research.

USE OF AI IN PLANNING AND PROJECT EFFORT ESTIMATION

Good project planning involves many aspects: staff need to be assigned to tasks in a way that takes account of their experience and ability, the dependencies between tasks need to be determined, times of tasks need to be estimated in a way that meets the project completion date and the project plan will inevitably need revision as it progresses. AI has been proposed for most phases of planning software development projects, including assessing feasibility, estimation of cost and resource requirements, risk assessment and scheduling. This section provides pointers to some of the proposed uses of knowledge based systems, genetic algorithms, neural networks and case based reasoning, in project planning and summarizes their effectiveness.

Knowledge Based Systems

It seems reasonable to assume that as we gain experience with projects, our ability to plan new projects improves. There have been several studies that adopt this assumption and aim to capture this experience in a Knowledge Based System (KBS) and attempt to utilise it for planning future software development projects. Sathi, Fox & Greenberg (1985) argue that a well defined representation scheme, with clear semantics for the concepts associated with project planning, such as activity, causation, and time, is essential if attempts to utilise KBS for project planning are to succeed. Hence, they develop a representation scheme and theory based on a frame based language, known as SRL (Wright, Fox, & Adam, 1984). Their theory includes a language for representing project goals, milestones, activities, states, and time, and has all the nice properties one expects, such as completeness, clarity and preciseness. Surprisingly, this neat frame based language and the semantic primitives they develop have been overlooked by others and appear not to have been adopted since their development. Similarly, other proposals that aim to utilise a KBS approach for project management, such as the use of production rules and associative networks (Boardman & Marshall, 1990), which seemed promising at the time have not been widely adopted. When considering whether to adopt a KBS approach, the cost of representing the knowledge seems high and unless this can be done at a level of abstraction that allows reuse, one can imagine that it is unattractive to software developers who are keen and under pressure to commence their projects without delay.

Neural Networks

Neural networks (NNs) have been widely and successfully used for problems that require classification given some predictive input features. They therefore seem ideal for situations in software engineering where one needs to predict outcomes, such as the risks associated with modules in software maintenance (Khoshgoftaar & Lanning, 1995), software risk analysis (Neumann, 2002) and for predicting faults using object oriented metrics (Thwin & Quah, 2002). The study by

Hu, Chen, Rong, Mei & Xie (2006) is typical of this line of research. They first identified the key features in risk assessment based on past classifications such as those presented by Wallace and Keil (2004) and further interviews with project managers. They identified a total of 39 risk factors which they grouped into 5 risk categories: project complexity, cooperation, team work, project management, and software engineering. These were reduced to 19 linearly independent factors using principal component analysis (PCA). Projects were considered to have succeeded, partially failed, or failed. In their experiments, they tried both the use of a back propagation algorithm for training and use of GAs to learn networks, using 35 examples for training and 15 examples for testing. The accuracy they obtained using back propagation was 80% and that with a GA trained NN was over 86%, confirming that use of NNs for predicting risk is a worthy approach, though larger scale studies are needed.

Genetic Algorithms

There have been numerous uses of genetic algorithms for project scheduling in various domains (Cheng & Gen, 1994; Hindi, Hongbo, & Fleszar, 2002; Hooshyar, Tahmani, & Shenasa, 2008; Yujia & Chang, 2006; Zhen-Yu, Wei-Yang, & Qian-Lei, 2008). A survey of their application in manufacturing and operations management can be found in (Kobbacy, Vadera, & Rasmy, 2007; Meziane, Vadera, Kobbacy, & Proudlove, 2000). These typically formulate project planning as a constraint satisfaction problem with an objective that needs optimisation and, which is then transformed into a form suitable for optimisation with a GA.

In the area of software development, Shan, McKay, Lokan & Essam (2002) utilise Genetic Programming to evolve functions for estimating software effort. Two target grammars were adopted for the functions that allowed use of a range of mathematical functions (e.g., exp, log, sqrt) as well as a conditional expressions. The approach was tested on data consisting of 423 software development projects characterised by 32 attributes (e.g. such as intended market, requirements, level of user involvement, application type, etc) from the International Software Benchmarking Standards Group (www.isbsg.org.au) with roughly 50% used for training and 50% used for testing. The results of this study show that the approach performs better than linear and log regression models. An interesting finding of the study was that although the most accurate functions discovered by GP utilised similar parameters to the traditional estimates, a key difference was that it adopted non-linear terms involving team size.

Creating a good assignment of staff to tasks and producing schedules is critical to the success of any software development project. Yujia & Chang (2006) show how it is possible to utilise GAs to produce optimal schedules and task assignments. Their proposal involves a two part chromosome representation. One part includes the assignment of individuals to tasks and another involves representing the topological ordering of the tasks in a way that ensures that the offspring generated using the cross-over operator remain valid schedules. The fitness function is obtained by utilising a systems dynamics simulation to estimate expected task duration given a particular chromosome. The results of their experiments suggest that this is a promising approach, though further work on how to utilise GAs in practice when schedules change is still needed.

An important part of developing an optimal schedule that meets a target completion date is the trade-offs that may occur. For example, attempts at increasing quality can result in increasing cost and possibly compromising completion time but perhaps increasing user satisfaction. Increasing resources on tasks increases the local cost but may result in early completion, higher quality and reduction of overall cost. Hooshyar et al., (2008) propose the use of GAs to optimize schedules to take account of such trade-offs. They represent a schedule by a chromosome consisting of the activity duration and which is ordered based on their dependency. In their experiments, they utilise the standard mutation and two-point cross-over operators and adopt a fitness function that includes the cost and duration. The experimentation is carried out on projects consisting of 10, 20 and 30 activities and conclude that although the well known algorithm due to Siemens (1971) works well for small scale problems, GAs may be more effective for larger scale problems.

Case Based Reasoning

It can be argued that successful project planning and management is heavily based on experience with past cases. It is therefore surprising that there are few studies that propose the use Case Based Reasoning (CBR) for project planning of software development. One of the few exceptions is the study by Heng-Li Yang & Chen-Shu Wang (2008), who explore the combined use of CBR and data mining methods for project planning. They use a structured representation for cases, called Hierarchical Criteria Architecture (HCA), where projects are described in terms of the customer requirements, project resources and keywords describing the domain. The use of HCA enables different weights to be adopted when matching cases, allowing greater flexibility depending on the preferences of the project manager. Given a new project, first similar new cases are retrieved. Then, data mining methods, such as association rule mining, are used to provide further guidance in the form of popular patterns that could aid in project planning. In a trial, based on 43 projects, Yang & Wang (2008), show how the combined use of CBR and data mining can generate useful information, such as “the duration of project implementation was about 26 days and 85% of projects of projects were completed on time”, which can be used to provide guidance when planning a similar project.

REQUIREMENTS ENGINEERING AND SOFTWARE DESIGN

Requirements engineering is often seen as the first stage of a software development project. It is the basis of any development project and this is not restricted only to software engineering. It is a broad and multidisciplinary subject (Zave, 1997). Requirements define the needs of many stakeholders. It is widely acknowledged, that because of the different backgrounds of these stakeholders, requirements are first expressed in natural language within a set of documents. These documents usually represent “the unresolved views of a group of individuals and will, in most cases be fragmentary, inconsistent, contradictory, seldom be prioritized and often be overstated, beyond actual needs” (Smith, 1993). The main activities of this phase are requirements elicitation, gathering and analysis and their transformation into a less ambiguous representation. For a detailed list of activities in requirements see Young (2003, pp 3-5). Requirements form the basis of software design. Ideally, all problems encountered during the requirements phase should be resolved before design starts. Unfortunately, in practice some of the problems inherent to requirements are passed into design, making the late discovery of errors occurring during this phase the most expensive to correct. It is therefore not surprising that requirements engineering is seen as the most problematic phase of the software development life cycle.

Problems Associated with Requirements Engineering

There are many problems that have been identified during the requirements engineering phase of the software development process. These can be summarised as follows:

Requirements are ambiguous: It is widely acknowledged, that because of the different backgrounds of the stakeholders, requirements are first expressed in Natural Language (NL). NL is inherently ambiguous and contributes to the incompleteness of requirements as many assumptions are made on some issues. Detecting ambiguities in NL is an old and major research issue in requirements engineering (Presland, 1986). Research in this area will be further discussed later in this chapter.

Requirements are incomplete, vague and imprecise: Requirements are usually incomplete, vague and imprecise in nature. It has been reported that customers do not really know what they want, or have difficulties in articulating their requirements (Yang, Xia, Zhang, Xiao, Li & Li, 2008). It has also been reported that there is a lack of user involvement during requirements (Hull, Jackson & Dick, 2005). In addition, some of these requirements are vague and cannot be easily validated. This includes statements related to system security (what is a secure system?), user interface (what is a user friendly system?) and reliability (what is a reliable system). Yen & Liu (1995) defined an imprecise functional requirement as "a requirement that can be satisfied to a degree". Therefore, there is a need to improve the quality of these requirements before the modelling phase.

Requirements are conflicting: Conflicts in requirements engineering occur when two different requirements compete for the same resources or when the satisfaction of one requirement precludes that of another. Yen & Liu (1995) stated that "Two requirements are conflicting if an increase in the degree to which one requirement is satisfied often decreases the degree to which another requirement is satisfied".

Requirements are volatile: User needs evolve over time. It is not unusual that during the time it takes to develop a system, user requirements have already changed. The causes of these changes may vary from the increasing understanding of the user about the capabilities of a computer system to some unforeseen organisational or environmental pressures. If the changes are not accommodated, the original requirements set will become incomplete and inconsistent with the new situation or in the worst case useless (Meziane, 1994).

There are communication problems between the stakeholders: During the requirements engineering phase, developers have to talk to a wide range of stakeholders with different backgrounds, interests, and personal goals (Zave, 1997). Communication with and understanding all these stakeholders is an extremely difficult and challenging task.

Requirements are difficult to manage: One of the main problems associated with requirements is that of traceability (Hull, Jackson & Dick, 2005). Traceability is the process of following a requirement from its elicitation to implementation and verification and validation. Linking the different phases of requirements validation is often omitted. Other management issues related to software management are: project management, software cost, development time, resources management and managing the changing environment.

The main contribution of AI in the requirements engineering phase are in the following areas:

- Disambiguating natural language requirements by developing tools that attempt to understands the natural language requirements and transform them into less ambiguous representations.
- Developing knowledge based systems and ontologies to manage the requirements and model problem domains.
- The use of computational intelligence to

solve some of the problems associated with requirements such as incompleteness and prioritisation.

In the following sections, we review and discuss some of the systems developed in these areas.

Processing Natural Language Requirements

The idea of transforming NL requirements automatically into specifications and design goes as far back as the early 80s. In his paper, "Program Design by Informal English Description", Abbott (1983), drew an analogy between the noun phrases used in NL descriptions and the data types used in programming languages. It is true that in those days requirements and modelling were not as distinct activities as they are now, but he did nevertheless associate the noun phrases found in NL descriptions to data types and concluded that these data types "divide the word into classes of object". Abbott stated that "associating common nouns with data types, makes the notion of data types more intuitive". He has also highlighted that this is not a straight forward mechanical approach but requires some tacit knowledge related to the problem domain. Booch (1986) further developed this approach when describing his Object-Oriented analysis and design method. It was later noted that verb phrases and to some extent adjectives describe relationships between these entities, operations and functions (Saeki, Horai & Enomoto, 1989; Vadera & Meziane, 1994; Poo & Lee, 1995). Since then and over the last twenty five years, most research in transforming NL requirements into various modelling languages adopted the same approach. Early systems that attempted to transform NL specifications were relatively simple as the NL understanding field was still in its infancy; however most researchers have taken advantage of recent developments in NL processing systems to develop more robust systems. It is not our intension to review all systems that have been developed to transform NL requirements into software models, but we highlight some systems that have attempted to produce formal specification and OO oriented models from NL Requirements.

From NL to Formal Specifications

Saeki, Horai & Enomoto (1989) proposed a framework to translate specifications written in NL (English) into formal specifications (TELL). Their framework suggests the extraction of four tables from the NL requirements that contain verbs, nouns, actions and action relations. However, they noticed that simple extraction of nouns and verbs was not sufficient and deeper semantical analysis is needed. Indeed, nouns can denote objects but also their attributes as verbs can denote relationships and actions. This has been one of the major challenges since in the automatic transformations of NL requirements into other specification languages. Hence, they suggested a human classification of nouns and verbs. They have identified four classes for nouns; class noun, value noun, attribute noun, action noun and 4 verb classes; relational verb, state verb, action verb and action related verb. In the action table, for each action verb, its agent and target object are identified. In the action related table, messages together with their senders and receivers have been identified. A class template is then used to gather the information produced by the four tables and is suggested to be used for the development of the formal specifications. However, their system was not implemented but set the foundations for future systems.

The NL2ACTL system (Fantechi, Gnesi, Ristori, Carenini, Vanocchi, & Moreschini, 1994) aims to translate NL sentences, written to express properties of a reactive system, to statements of an action based temporal logic. It takes each sentence, parses it, and attempts to complete it by identifying any implicit information that is required to produce a well-formed expression in the action based logic called ACTL. First, NL2ACTL shows that it is possible to utilise existing NL processing tools to develop grammars that are useful for analysing English sentences and for producing formal specifications in a given domain. Second, NL2ACTL demonstrates that when there is a specific application domain and target formal specification language in mind, one can develop a system that can help to identify incompleteness at a detailed level.

Vadera & Meziane (1994) developed the FORSEN system which aims to translate NL requirements into the Formal specifications language VDM (Jones, 1990). They used Logic Grammars (LG) to translate the requirements into McCord's logical form language (LFL) (McCord, 1990). This allowed the detection of ambiguities in the NL requirements. If a sentence is ambiguous, the system displays all possible interpretations and the user is then required to select the intended meaning. At the end of the first phase, each sentence has a single associated meaning represented in the LFL. In the second phase, an entity relationship model is developed using nouns and verbs extracted from the LFL. The developed entity relationship model is then translated to a VDM data type. The last phase of the FORSEN system is the generation of VDM specifications. FORSEN generates specifications by filling in pre-defined schemas for a common range of operation specifications such as adding items, deleting items, and listing items that satisfy some conditions. FORSEN was different from previously developed systems as it did not rely on structures or domain specific requirements. The input was a free English text representing the requirements of the system to be developed. However, it was limited in the range of specifications it could generate. A good review of systems that produce formal specifications can be found in Vadera & Meziane (1997).

From NL to OO Specification

Juristo, Moreno & López (2000), defined a general framework for the automatic development of OO models from NL requirements using linguistics instruments. Their framework is composed of the following nine phases: (i) extraction of essential information; (ii) identification of synonyms and polysemies; (iii) separation of static and dynamic information; (iv) static requirements structuring; (v) dynamic requirements structuring; (vi) object model construction; (vii) behaviour model construction; (viii) object model and behaviour model integration; (ix) object model and behaviour model verifications. They argue that these steps will allow the identifications of all the components required by an OO model and will help in the development and validation of these OO models. They stressed, that inputs should be in NL to avoid unnecessary user interaction with the system, a view earlier supported by Mich (1996). The system should also use rules and grammars rather than heuristics to identify entities, attributes and relations (including specialised relation such as generalisations). These steps will also allow the detection of ambiguities in NL requirements, separate the static model from the dynamic model and develop the object model and the behaviour model. As the following reviews show, few of the systems that have been developed exhibit most of these components.

Mich (1996) used the Large –scale Object-based Linguistic Interactor Translator Analyser (LOLITA) NLP system to develop the NL-OOPS which aims to produce OO specifications from NL requirements. In LOLITA knowledge is represented using conceptual graphs SemNet where nodes represent concepts and arcs relationships between concepts. The NL requirements are analysed and corresponding SemNets produced. During this phase, ambiguities are either resolved or flagged to the user. An algorithm is then used to translate the SemNets into an OO model.

Moreno, Juristo & Van de Riet (2000), developed an approach that linked the linguistic world and the conceptual world through a set of linguistic patterns. The patterns have been divided into two categories: static utility language (SUL) that describes the elements of the problem domain, hence seen as the static part of the object model and the dynamic utility language (DUL) that describes the changes to the elements of the static model. Each language has been described using context-free grammars. For example a conceptual patter would be the definition of a relationship between two classes by means of a verb. This allowed this approach to define formal relations between the linguistic world and conceptual world via the mathematical world.

Harmain & Gaizauskas (2003) developed the Class-Model Builder (CM-Builder), a NL based CASE tools that builds class diagrams specified in UML from NL requirements documents. CM Builder is developed using the GATE environment (Gaizauskas,Cunningham, Wilks,Rodgers & Humphreys, 1996). CM-Builder takes as an input a software requirement text in English and produce as an output an OO model in the CASE Data Interchange Format (CDIF) file that is used as an input to a CASE tool that supports UML (Entreprise Modeler in this case). The output file contains the identified classes, their attributes and the relationships among them. The systems used a parser based on feature-based Phrase Structure Grammar that relies on unification. The output from the parser is semantically represented as a predicate-argument structure. They also make use of a simple ontology concept to represent the world knowledge. The strength of GATE as a specialised language engineering tool allowed a deep analysis of the NL requirements and allowed. Further manipulations of the UML are required using Entreprise modeller to complete the model produced by CM-Builder.

Knowledge Based Systems

Requirements engineering is knowledge intensive and include activities such as "Knowledge Elicitation" and "Knowledge Acquisition". It is not surprising that knowledge-based software and requirement engineering received a wide attention since the early 1980. Lubars & Harandi (1987) stated that "The reuse of experts design knowledge can play a significant role in improving the quality and efficiency of the software development process".

The READS tool (Smith, 1993) is developed at the Paramax Systems Corporation, a software contractor to US and foreign government agencies and directly supports the U. S. Government system engineering process. READS supports both the front end activities such as requirement discovery, analysis and decomposition and requirements traceability, allocation, testing, and documentation. READS is composed of many components to achieve its goals. It starts with the windows document where the requirements documents are displayed and requirements are then discovered manually or automatically. It has been reported that the automatic identification of the requirements hits an 80%-90% rate. The identified requirements are saved in the project's database and displayed in the requirements inspection window. During this phase, the requirements are edited, reviewed, allocated and decomposed; "the goal of decomposition is the development of a set of designable requirements: precise, unambiguous, testable statements of a need or condition that can be directly mapped to a physical solution" (Smith, 1993). Children are attached to each requirement denoting the different interpretation if they are ambiguous. The derived requirements are then aggregated into a single non ambiguous requirements document. Requirements are organised into different views using allocation categories (Smith, 1993).

KBS are used in the design phase by storing design patterns or design families. Lubars & Harandi (1987), used a KBS to store design families, upon the development of the requirements, input and outputs of the system's functionality. The Idea system searches the KB and proposes a design schema. This becomes then the top design level of the system. The users need to refine this schema to fully satisfy the user requirements.

Ontologies

An ontology is an explicit specification of a conceptualisation. Ontologies and techniques used for the semantic web have been investigated in the last few years as a way to improve requirements engineering. Ontologies enable the sharing of common information and knowledge within specific domains. "An ontology can be viewed as a special kind of semantic network representing the terminology, concepts, and the relationships among these concepts related to a particular application domain. Semantic web and ontological techniques provide solutions for representing, organizing and reasoning over the complex sets of requirements knowledge and information." (Yang, Xia, Zhang, Xiao, Li & Li, 2008). Ontologies are developed by many organisations to reuse, integrate, merge data and knowledge and to achieve interoperability and communication among their software systems. Reuse has been a hot issue in software design for many years now. It was one of the main strengths of the OO oriented methods and programming languages introduced in the last three decades. Indeed, there are similarities between the classes in an ontology and classes in OO (Vongdoiwang & Batanov, 2005). Ontologies enhance the semantics by providing richer relationships between the terms of concepts/classes (Siricharoen, 2008).

In their research, Yang Yang, Xia, Zhang, Xiao, Li & Li (2008) use semantic web and ontological techniques to elicit, represent, model, analyze and reason about knowledge and information involved in requirements engineering processes. They argue that the use of semantic representation could improve some of the activities involved in the requirements phase such as filling the communication gap between different stakeholders, effectively support automatic requirements elicitation, detecting incompleteness and inconsistency in requirements, evaluate the quality of requirements, and predict possible requirements changes. Their system uses three ontologies namely: the user ontology provides flexible mechanisms to describe a variety of assumptions about end-users (or customers), and to infer domain-dependent requirements. It is used to support user requirements modelling and elicitation. The enterprise ontology describes business context, structure, rules, goals, tasks, responsibilities, and resources available, for requirements analysts to understand and grasp high-level requirements. The Domain ontology serves as a shared knowledge background among different stakeholders. It is used for consistency and reusability of knowledge accumulated during the project development. The inference rules in the contextual ontologies can be used to elicit implicit requirements, detect incompleteness and inconsistency in requirements description. Automated validation and consistency checking of requirements, to some degree, offer an opportunity for the management of requirement evolution.

Kossmann, Wong, Odeh, Gillies, (2008) developed the OntoREM (Ontology-driven Requirements Engineering Methodology) an ontology-based solution to enable knowledge driven requirements engineering. A metamodel, which is an ontology is designed taking into account the different requirements engineering artifacts, activities, and interrelationships with other domains and disciplines. "The intended application of the OntoREM metamodel ontology is to capture and manage reference knowledge and concepts in the domain of requirements engineering, supported by decision engines that rely on other problem and solution domain ontologies so as to develop high quality requirements for the related domains." (Kossmann, Wong, Odeh, Gillies, 2008). OntoREM supports activities such as 'elicitation', 'analysis and negotiation', 'documentation' and 'validation'. The system produces requirements that are complete and consistent.

Ontologies are also used to develop software Engineering environments. Falbo et al., (2002) have developed the Ontology-based software Development Environment (ODE) based on a software process ontology. Tools for process definition and project tracking were also built based on this ontology. ODE's architectural is composed of two levels: the application level concerns application classes, which model the objects that address some software engineering activity and the meta-level (or knowledge level) that defines classes that describe knowledge about objects in the application base level. The classes in the meta-level are derived directly from the ontology.

In order to capture, represent and structure the domain knowledge about specific model and platform properties Geihs, et al. (2008) use ontologies as a machine-readable formal description technique that supports semantic annotations and reasoning about models and platforms. They developed a systematic support for the automation of model transformations based on domain specific knowledge formally represented by an ontology. "Entities and concepts defined in the ontology are referenced in the platform-independent model (PIM) as well as in a semantic annotation of the target platforms' API. This allows an automatic matching of modelling elements of the PIM to variables, objects and interfaces of the involved target platforms" Geihs et al. (2008). The system ontology links the abstract concepts of the PIM to the concrete platform-specific model (PSM) concepts for objects of the real world. The main benefit of their approach is the reuse of the PIM as well as the reuse of the transformation. Finding the right classes for an object oriented model (often the class diagram) is not an easy task. Siricharoen (2008).proposed the use of ontologies as the inputs of a semi-automatic object model construction program. He attempted to build semi-automatic object model by using and comparing the concepts in the ontology as objects, slot or properties as attributes, and some properties can act as functions or operation.

Intelligence Computing for Requirements Engineering

Pedrycz & Peters (1997) stated that "The emerging area of Computational Intelligence (CI) provides a system developer with a unique opportunity of taking advantage of the currently developed and highly mature technologies". They argue that each of the techniques developed in CI can play an important role in solving the traditional problems found in software engineering. In these sections we review some of the systems developed using CI techniques to support requirements engineering.

The SPECIFIER system (Miriyala & Harandi, 1991) can best be viewed as a case based system that takes as input an informal specification of an operation where the pre and post-conditions are given as English sentences. The verbs in the sentences are used to identify the concepts. The identified concepts are then used to retrieve associated structure templates (represented as frames). These structure templates have slots that define the expected semantic form of the concepts and have associated rules that can be used to fill in the slots by using the informal specification. A set of rules is used to select specification schemas based on the identified concepts. The specification schemas are then filled by using the rules associated with the slots and the structures of the concepts. Once filled, the specification schemas produce formal specifications in a Larch-like language

When dealing with conflicts in requirements, we often drop one of the requirements or modify it to avoid the conflict. However, Yen & Liu (1995) stated that it is desirable "to achieve an effective trade off among conflicting requirements so that each conflicting requirement can be satisfied to some degrees, while the total satisfaction degree is maximized". Hence they suggested that it is necessary to identify and assess requirements priorities. In their approach they use imprecise conflicting requirements to assess requirements priorities. Users are required to relatively order requirements and to decide how much important a requirement is with regards to other conflicting requirements. They then used fuzzy logic and possibility theory to develop an approximate reasoning schema for inferring relative priority of requirements under uncertainty.

TESTING

Despite the wealth of research in the last two decades, software testing remains an area where, as cases of reported failures and numerous releases of software suggest, we cannot claim to have mastered. Bertolino (2007) presents a useful framework for summarising the challenges that we face in addressing the problems of ensuring that systems are fit for purpose, suggesting further research on: (i) developing a universal theory of testing, (ii) fully automatic testing, (iii) design to facilitate testing and (iv) development of integrated strategies that minimise the cost of repeated testing. This section presents some pointers to attempts at using AI techniques to support particular aspects of the testing process, which has the potential to contribute towards a more integrated dream testing environment of the kind proposed by Bertolino (2007).

Knowledge Based Systems

One of the earliest studies to suggest adoption of a Knowledge Based System (KBS) for testing was by Bering and Crawford (1988) who describe a Prolog based expert system that takes a Cobol program as input, parses the input to identify relevant conditions and then aims to generate test data based on the conditions. DeMasie and Muratore (1991) demonstrated the value of this approach by developing an expert system to assist in the testing of software for the Space Shuttle. The software testing process for the Space Shuttle had previously involved running it on a simulated environment and use of manual checks to identify errors, which could take more than 40 people over 77 days of testing. Since the criteria for analysing the performance data were well documented, a rule base was developed, enabling automatic identification of potential problems and resulting in a reduction to around 56 days.

Both the above studies are quite application specific. In contrast, Samson (1990, 1993) proposes a generic environment, called REQSPERT, that has, at its heart, a knowledge base that supports the development of test plans from requirements. REQSPERT takes a list of requirements as input, classifies them into particular types of functional and non-functional requirements, identifies suitable test metrics and then proposes a test plan together with the test tools that could be utilized. Although the approach proposed by REQSPERT is interesting, there has been limited adoption of the model in practice, perhaps because of the investment of effort required in instantiating the model for particular applications. Indeed, this might be the reason why progress on the use of KBS for testing appears to have stalled in the late 1990's, since papers that successfully build further on this idea are hard to find in the literature.

AI Planning

A more active area of research since the mid-1990s has been the use of AI planning for testing. von Mayrhauser, Scheetz, Dahlman & Howe (2000) point out that a major disadvantage of white box testing is that we have to wait until the code is developed before commencing the process of producing the tests. An alternative to the use of white-box testing is to model the domain and produce tests from the model. To be able to generate tests, the model should be rich enough to generate a sequence of commands, where each command may include parameters. The primary example of this approach is the Sleuth system (von Mayrhauser, Walls, & Mraz, 1994a, 1994b), which aims to do this by defining three layers, where the top layer aims to define how to sequence commands, the next layer defines individual commands and the bottom layer defines how to instantiate the parameters required by commands. This idea was explored by applying it to generate tests for an interface to a system capable of storing a large number of tape cartridges, known as StorageTek.

The experiences with Sleuth suggest that although it can be effective, considerable effort may be required to develop a model for particular applications. A research group at Colorado State University (Howe, von Mayrhauser, & Mraz, 1995) explored an alternative approach in which they utilize AI planning methods (Ghallab, Nau, & Traverso, 2004). AI planning methods allow the specification of operators, where each operator can be defined by providing a precondition that must hold in order for the operator to be applicable and post-conditions that define the valid states following application of the operator. An AI planner can take an initial state and a goal and then generate a plan that consists of the sequence of operators that transform the initial state to achieve a given goal. Howe et al. (1995) recognized that by representing commands as operators, providing initial states and setting the goal as testing for correct system behaviour, an AI planner could generate test cases, consisting of a sequence of commands (Howe, et al., 1995; Mraz, Howe, von Mayrhauser, & Li, 1995). To evaluate the idea, they modeled the interface to StorageTex, that was used to illustrate Sleuth and conclude that it was easier to represent the domain using the planner based approach, that the test cases generated are provably correct and the different combinations of initial and goal states can result in a wider and more novel range of test cases. However, they acknowledge that test case generation can be slow, though this might have been because of the particular planner they employed in their study. In a follow up study, they develop their ideas further by showing how it is possible to utilize UML together with constraints on parameters and state transition diagrams to model the domain (Scheetz, von Mayrhauser, & France, 1999; von Mayrhauser, Scheetz, & Dahlman, 1999). The class methods of the UML model are mapped to operators and state transition diagrams together with propositional constraints provide information to define the preconditions and effects of operators. High level test objectives, derived from the UML models, can then be mapped to an initial state and goals for the planner which generates tests based on the objectives (Von Mayrhauser, France, Scheetz, & Dahlman, 2000).

Von Mayrhauser et al. (2000) also shows that the use of AI planning for test generation has the advantage that one can mutate plans to mimic potential errors in the use of a system, for example when a user attempts an incorrect sequence of commands. This mutation of the plans then leads to generation of cases that test the error recovery capabilities of applications.

An important part of testing distributed systems is to check whether it is possible for it to end up in insecure or unsafe states. Gupta, Bastani, Khan & Yen (2004) take advantage of the goal oriented properties of Means-Ends planning by defining potential system actions as operators so that generating tests becomes equivalent to the goal of finding a plan from the current state to specified unsafe or near unsafe states.

Memon, Pollack & Soffa (1999) argue that human generation of test cases for graphical user interfaces requires enumeration of a large number of possible sequences of user actions, making the process inefficient and likely to be incomplete. Instead, as with the above studies, they propose the use AI planning methods, since once the possible actions are specified using operators, a planner can generate tests since it is capable of finding a sequence of actions to achieve a goal from an initial state. There are two interesting aspect of their work: (i) they don't simply specify a single operator for each possible interface action but use abstraction to develop higher level operators, making the search more efficient, (ii) their approach automatically includes verification conditions in the tests that are capable of detecting failure following intermediate actions in a sequence aimed to achieve some goal. They test the feasibility of the approach by applying it to generate test cases for Microsoft Wordpad

Genetic Algorithms

A study by Kobbacy, Vadera and Rasmy (2007) has shown that the use of Genetic Algorithms (GAs) for optimization has grown substantially since the 1980s and this growth has continued while the use of other AI technologies has declined. This trend is also present in their use in testing, with numerous studies aiming to take advantage of their properties in an attempt to generate optimal test cases (Baresel, Binkley, Harman, & Korel, 2004; Baudry, Fleurey, Jezequel, & Le Traon, 2002a, 2002b; Briand, Feng, & Labiche, 2002; Briand, Labiche, & Shousha, 2005; Harman & McMinn, 2007; Liaskos, Roper, & Wood, 2007; Nguyen, Perini, & Tonella, 2008; Ribeiro, 2008; Tonella, 2004; Wappler & Wegener, 2006) .

For example, Kasik and George (1996) utilize GAs for generating tests for user interfaces. They argue that manual testing of interfaces can be inadequate, pointing out that tests are constructed by systems engineers who have a fixed view of how the designed system is meant to be used and hence generate tests that don't really capture the paths that novice users might follow. To overcome this deficiency, they propose a novel system that aims to model novice behavior by use of GAs. The central idea is to represent a sequence of user actions by a gene. A pool of genes then represents potential tests. A tester can then define a fitness function to reflect the extent to which a particular gene resembles a novice and evolution then leads to the best tests. They experiment with a fitness function that gives greater priority to actions that remain on the same window and attempt three alternative strategies for generating tests. First they give little guidance to the GA and observe that this leads to tests that at "best the resulting scripts seemed more chimpanzee-like than novice-like" (Kasik & George, 1996, p250). Second, they began the tests with a well defined sequence of actions and then allowed the GA to complete the sequence of actions. Although the results were better, they remained unconvinced about the quality of the tests. Thirdly, they provide both the start and end parts of a test and let the GA generate the intermediate actions. Not surprisingly, this approach, which they term pullback mode, results in the most appropriate novice like tests. The most interesting part of their work is that it shows the potential for modeling different types of users which could provide a powerful tool for generating particular types of test suites.

Baudry et al. (2002a, 2002b) present an interesting use of GAs aimed at improving the quality of an initial set of test cases provided by a tester. The central idea is based on mutation testing (Untch, 1992) which involves creation of a set of mutant programs, where each mutant is a version of the original program but with an introduced variation. The introduced variation can be an error or bug or result in behavior that is equivalent to the original program. The effectiveness of a set of test cases can then be measured by calculating the proportion of non-equivalent mutants that can be revealed by the test cases. The task for a tester, then, is to develop a set of tests that maximizes this measure, called a mutation score. Baudry et al. (2002b) explore the use of GAs by taking the fitness function to be the mutation score and each gene to be a test. An initial test set is provided by a tester and evolution using the standard reproduction, mutation, and crossover are utilized where the target application to be tested is a C# parser. Their experience with this approach is not good, and they conclude that the "results are not stable" and that they had to "excessively increase the mutation rate compared to usual application of genetic algorithms" (Baudry, et al. 2002a). Given the target application, the individual genes represent source code and with hindsight, this outcome may not be surprising since the crossover operator may not be particularly suitable on programs. However, they also observe two inherent limitations of using the standard evolution cycle (Baudry, et al., 2002a). First, since GAs focus on the use of the best individuals for reproduction, the mutation operator can lose valuable information and the best new genes may not be as good as those of the previous generation. Second, they point out that a focus on optimizing individual genes doesn't help in minimizing the size of the testing set, which is an important consideration given the time it can take to carry out the tests. To overcome these problems, they propose a model in which new members of the gene pool are obtained by bacterial adaptation, where mutation is used to make gentler improvements to the genes and those genes with a score above a certain threshold are retained. Their experiments on the performance of this revised scheme suggest that it is more stable and it converges more rapidly.

Several authors propose the use of GAs for testing OO programs (Briand, et al., 2002; Ribeiro, 2008; Tonella, 2004; Wappler & Schieferdecker, 2007; Wappler & Wegener, 2006). The main aim of these studies is to construct test cases consisting of a sequence of method calls. Constructing sensible sequences of method calls requires that certain pre-conditions, such as the existence of the target object or parameters required by a method are satisfied. An obvious GA based approach is to code methods as identifiers and to attempt to construct a fitness function. But use of mutation and crossover are bound to result in inappropriate sequences, so how can these be avoided? Wappler and Lammermann (2005) demonstrate that it is possible to devise a fitness function that penalizes erroneous sequences. However, in a subsequent paper, Wappler and Wegener (2006) acknowledge that using a fitness function as the primary means of avoiding illegal sequences is not efficient. Instead they propose a novel use of Genetic Programming (GP), which aims to learn functions or programs by evolution. The underlying representation with most GP systems is a tree instead of a numeric list. In general, a tree represents a function, where leaf nodes represent arguments and non-terminal nodes denote functions. In context of testing, such trees can represent the dependencies between method calls which can then be linearised to produce tests. Use of mutation and crossover on these trees can result in invalid functions and inappropriate arguments in the context of testing object oriented programs. Hence, Wappler and Wegener(2006), suggest the use of strongly typed GP (Montana, 1995), where the types of nodes are utilized to ensure that only trees with appropriate arguments are evolved. This still leaves the issue of how such trees are obtained in the first place. The approach they adopt is to first obtain a method call dependency graph that has links between class nodes and method nodes. The links specify the methods that can be used to create instances of a class and which instances are needed by a particular method. This graph can then be traversed to generate trees to provide the initial population. The required arguments (objects) for the trees are obtained by a second level process that first involves generating linear method call sequences from the trees and then utilizes a GA to find the instantiations that are the fittest. Once this is achieved, the trees are optimized by use of recombination and mutation operators with respect to goals, such as method coverage.

Ribeiro (2008) also adopt a similar approach in their eCrash tool, utilizing strongly typed GP and a dependency graph to generate the trees. However, a significant refinement of their work in comparison to Wappler and Wegener (2006) is that they reduce the search space by removing methods, known as pure methods, that don't have external side effects. A trial of this pruning process on the Stack class in JDK 1.4.2 resulted in about a two-thirds reduction in the number of generations required to achieve full coverage.

Briand et al. (2002) explore the use of GAs for determining an optimal order for integrating and testing classes. A significant problem when determining a suitable order occurs because class dependency cycles have to be broken resulting in the need to utilize stubs. The complexity of the stubs needed varies, depending on the level of coupling that exists between classes, hence different orderings require different levels of effort for creation of the stubs. Briand et al. (2002) take advantage of previous work on scheduling, for example in the use of GAs for the traveling salesman problem, and utilize a permutation encoding of the classes together with a fitness function that measures the extent of coupling between the classes. The fitness measure is defined so as to reduce the number of attributes, the methods that would need handling if a dependency is cut, and the number of stubs created. In addition, they disallow inheritance and composition dependencies from being cut since they lead to expensive stubs. They experiment with this approach on an ATM case study utilizing the Evolver GA system (Palisade, 1998), compare the results with those obtained using a graph-based approach and conclude that the use of GAs can provide a better approach to producing orderings of class integration and testing.

CONCLUSION

The survey conducted in this chapter has highlighted some trends in the use of AI techniques in the software development process. In software project planning, the use of GAs is by far the most popular proposal. Their ability to easily represent schedules and the flexibility they offer for representing different objectives make them very appropriate for adoption in practice. Neural networks have also been adopted for risk assessment, but as the first chapter of the book describes, the use of Bayesian networks is more transparent and is likely to be more appealing in practice since project managers, more than most types of users, need to feel they are in control. Likewise, the use of case based reasoning, as proposed by the work of Heng-Li Yang & Chen-Shu Wang (2008) seems to be an attractive approach because it offers transparency and continuous improvement as experience is gained.

In the requirements and design phase, there is a lot of emphasis on identifying errors occurring in the early stages of software development before moving to design. The use of NLP techniques to understand user requirements and attempt to derive high level software models automatically is still and will remain (Chen, Thalheim and Wong, 1999), a hot research topic although there are some issues that are related to these approaches such as the use of ad hoc case studies and difficulties in comparing the developed systems (Harmain and Gaizauskas, 2003). In addition, having a system that can produce full design by automatically analysing NL requirements is not possible as design is a creative activity requiring skills and reasoning that are hard to include in a computer system. KBS have been used to better manage requirements, the requirements process and decisions taken during the design process. In the last few years there has been a lot of interest in the use of ontologies for requirements and design. The development of domain ontologies is making it possible to encapsulate knowledge and rules governing a specific domain in one single resource. Ontologies encompass both the strengths of NLP based systems and KBS in that they allow a better understanding of the problem domain, the detection of ambiguities and incompleteness, and are able to store tacit knowledge, design decisions, and propose metamodels for specific domains.

A number of authors have attempted to utilise GAs and AI planning methods for generating test cases. The current attempts suggest that use of GAs can run into difficulties with generating appropriate valid test cases and a fully automated approach using GAs seems problematic. Hence, a more promising approach is to use strongly typed genetic programming which is capable of reducing the number of ill-defined test sequences. Use of GAs for generating the order of integration of OO classes seems to be more promising with initial trials suggesting it is better than traditional graph based methods. The use of AI planning methods offers an alternative to use of GAs and GP that can offer greater control. The effort required in defining the operators for different applications can be significant, though some progress has been made in defining a framework that could develop into a usable approach in the future.

The survey suggest that there is now good progress in the use of AI techniques in SE but larger scale evaluation studies are needed and further research is required to understand the effectiveness of different approaches. Furthermore, the development of new areas such as intelligent agents and their use in distributed computing, context aware and secure applications will require closer links between SE and AI in the future.

REFERENCES

Abbott, R. J. (1983). Program design by informal English descriptions. *CACM*, *26*(11), 882–894.

Baresel, A., Binkley, D., Harman, M., & Korel, B. (2004). Evolutionary testing in the presence of loop-assigned flags: a testability transformation approach. In *Proceedings of the ACM SIGSOFT International Symposium on Software Testing and Analysis* (pp. 108-118). Boston: ACM Press.

Baudry, B., Fleurey, F., Jezequel, J.-M., & Le Traon, Y. (2002a). Automatic test case optimization using a bacteriological adaptation model: application to. NET components. In *Proceedings of the Seventeenth IEEE International Conference on Automated Software Engineering* (pp. 253-256), Edinburgh, UK. Washington DC: IEEE Computer Society.

Baudry, B., Fleurey, F., Jezequel, J. M., & Le Traon, Y. (2002b). Genes and Bacteria for Automatic Test Cases Optimization in the. Net Environment. In *Proceedings of the Thirteenth International Symposium on Software Reliability Engineering (ISSRE'02)* (pp. 195- 206), Annapolis, MD. Washington DC: IEEE Computer Society.

Bering, C. A., & Crawford, M. W. (1988). Using an expert system to test a logistics information system. In *Proceedings of the IEEE National Aerospace and Electronics Conference* (pp. 1363-1368), Dayton, OH. Washington DC: IEEE Computer Society.

Bertolino, A. (2007). Software Testing Research: Achievements, Challenges, Dreams. In *Proceedings of the IEEE International Conference on Software Engineering* (pp. 85-103), Minneapolis, MN. Washington DC: IEEE Computer Society.

Boardman, J. T., & Marshall, G. (1990). A knowledge-based architecture for project planning and control. In *Proceedings of the UK Conference on IT* (pp. 125-132), Southampton, UK. Washington DC: IEEE Computer Society.

Booch, G. (1986). Object-Oriented Development. *IEEE Transactions on Software Engineering*, *12*(2), 211–221.

Briand, L. C., Feng, J., & Labiche, Y. (2002). Using genetic algorithms and coupling measures to devise optimal integration test orders. In *Proceedings of the Fourteenth International Conference on Software Engineering and Knowledge Engineering* (pp. 43-50), Ischia, Italy. New York: ACM Press.

Briand, L. C., Labiche, Y., & Shousha, M. (2005). Stress testing real-time systems with genetic algorithms. *In Proceedings of the Conference on Genetic and Evolutionary Computation* (pp. 1021-1028). Washington DC. New York: ACM Press.

Chen, P. P., Thalheim, B., & Wong, L. Y. (1999). Future Directions of Conceptual Modeling. In Chen P. P., Akoka J, Kangassalo H, & Thalheim B. (Eds), *Selected Papers From the Symposium on Conceptual Modeling, Current Issues and Future Directions.* (pp. 287-301), (LNCS vol. 1565). London: Springer-Verlag.

Cheng, R., & Gen, M. (1994). Evolution program for resource constrained project scheduling problem. *In Proceedings of the Proceedings of the 1st First IEEE Conference on Evolutionary Computation* (pp. 736-741), Orlando, FL, USA.

DeMasie, M. P., & Muratore, J. F. (1991). Artificial intelligence and expert systems in-flight software testing. In *Proceedings of the Tenth IEEE Conference on Digital Avionics Systems Conference* (pp. 416-419), Los Angeles, CA. Washington DC: IEEE Computer Society.

Falbo, R. A., Guizzardi, G., Natali, A. C., Bertollo, G., Ruy, F. F., & Mian, P. G. (2002), Towards semantic software engineering environments. *Proceedings of the 14th international Conference on Software Engineering and Knowledge Engineering SEKE '02, vol. 27* (pp. 477-478), Ischia, Italy. New York: ACM.

Fantechi, A., Gnesi, S., Ristori, G., Carenini, M., Vanocchi, M., & Moreschini, P. (1994). Assisting requirement formalization by means of natural language translation. *Formal Methods in System Design*, *4*(3), 243–263. doi:10.1007/BF01384048

Fox, T. L., & Spence, J. W. (2005). The effect of decision style on the use of a project management tool: An empirical laboratory study. *The Data Base for Advances in Information Systems*, *32*(2), 28–42.

Gaizauskas, R., Cunningham, H., Wilks, Y., Rodgers, P., & Humphreys, K. (1996). GATE: an environment to support research and development in natural language engineering. In *Proceedings of the 8th IEEE International Conference on Tools with Artificial Intelligence*. (pp.58-66). Washington DC: IEEE Computer Society.

Geihs, K., Baer, P., Reichle, R., & Wollenhaupt, J. (2008). Ontology-Based Automatic Model Transformations. In *Proceedings of the 6th IEEE International Conference on Software Engineering and Formal Methods* (pp.387-391), Cape Town, South Africa. Washington DC: IEEE Computer Society.

Ghallab, M., Nau, D., & Traverso, P. (2004). *Automated Planning: Theory and Practice.* San Francisco: Morgan Kaufmann.

Gupta, M., Bastani, F., Khan, L., & Yen, I.-L. (2004). Automated test data generation using MEA-graph planning. In *Proceedings of the Sixteenth IEEE Conference on Tools with Artificial Intelligence* (pp. 174-182). Washington, DC: IEEE Computer Society.

Harmain, H. M., & Gaizauskas, R. (2003). CM-Builder: A natural language-based CASE tool for object-oriented analysis. *Automated Software Engineering Journal*, *10*(2), 157–181. doi:10.1023/A:1022916028950

Harman, M., & McMinn, P. (2007). A theoretical & empirical analysis of evolutionary testing and hill climbing for structural test data generation. In *Proceedings of the International Symposium on Software Testing and Analysis* (pp. 73-83). London: ACM.

Hindi, K. S., Hongbo, Y., & Fleszar, K. (2002). An evolutionary algorithm for resource-constrained project scheduling. *IEEE Transactions on Evolutionary Computation*, *6*(5), 512–518. doi:10.1109/TEVC.2002.804914

Hooshyar, B., Tahmani, A., & Shenasa, M. (2008). A Genetic Algorithm to Time-Cost Trade off in project scheduling. In *Proceedings of the IEEE World Congress on Computational Intelligence* (pp. 3081-3086), Hong Kong. Washington DC: IEEE Computer Society.

Howe, A. E., von Mayrhauser, A., & Mraz, R. T. (1995). Test sequences as plans: an experiment in using an AI planner to generate system tests. In *Proceedings of the Tenth Conference on Knowledge-Based Software Engineering* (pp. 184-191), Boston, MA. Washington DC: IEEE Computer Society.

Hu, Y., Chen, J., Rong, Z., Mei, L., & Xie, K. (2006). A Neural Networks Approach for Software Risk Analysis. *In Proceedings of the Sixth IEEE International Conference on Data Mining Workshops* (pp. 722-725), Hong Kong. Washington DC: IEEE Computer Society.

Hull, E., Jackson, K., & Dick, J. (2005). *Requirements Engineering.* Berlin: Springer.

Jones, C. B. (1990). *Systematic Software Development Using VDM*. Upper Saddle River, NJ: Prentice Hall International.

Juristo, N., Moreno, A. M., & López, M. (2000). How to use linguistics instruments for Object-Oriented Analysis. *IEEE Software*, (May/June): 80–89.

Kasik, D. J., & George, H. G. (1996). Towards Automatic Generation of Novice User Test Scripts. In *Proceedings of the Conference on Human Factors in Computing Systems* (pp. 244-251), Vancouver, Canada. New York: ACM Press.

Khoshgoftaar, T. M., & Lanning, D. L. (1995). A Neural Network Approach for Early Detection of Program Modules Having High Risk in the Maintenance Phase. *Journal of Systems and Software*, *29*, 85–91. doi:10.1016/0164-1212(94)00130-F

Kobbacy, K. A., Vadera, S., & Rasmy, M. H. (2007). AI and OR in management of operations: history and trends. *The Journal of the Operational Research Society*, *58*, 10–28. doi:10.1057/palgrave.jors.2602132

Kossmann, M., Wong, R., Odeh, M., & Gillies, A. (2008). Ontology-driven Requirements Engineering: Building the OntoREM Meta Model. *Proceedings of the 3rd International Conference on Information and Communication Technologies: From Theory to Applications, ICTTA 2008,* (pp. 1-6), Damascus, Syria. Washington DC: IEEE Computer Society.

Liaskos, K., Roper, M., & Wood, M. (2007). Investigating data-flow coverage of classes using evolutionary algorithms. In *Proceedings of the Ninth Annual Conference on Genetic and Evolutionary Computation* (pp. 1140-1140), London, England. New York: ACM Press.

Lubars, M. D., & Harandi, M. T. (1987). Knowledge-based software design using design schemas. *In Proceedings of the 9th international Conference on Software Engineering*, (pp. 253-262), Los Alamitos, CA. Washington DC: IEEE Computer Society Press.

McCord, M. (1990). Natural language processing in Prolog. In A. Walker (ed.), *A logical approach to expert systems and natural language processing Knowledge systems and Prolog*, (pp. 391–402). Reading, MA: Addison-Wesley Publishing Company.

Memon, A. M., Pollack, M. E., & Soffa, M. L. (1999). Using a Goal Driven Approach to Generate Test Cases for GUIs. In *Proceedings of the Twenty-first International Conference on Software Engineering* (pp. 257-266), Los Angeles, CA. New York: ACM Press.

Meziane, F. (1994). *From English to Formal Specifications*. PhD Thesis, University of Salford, UK.

Meziane, F., Vadera, S., Kobbacy, K., & Proudlove, N. (2000). Intelligent Systems in Manufacturing: Current Developments and Future Prospects. *The International Journal of Manufacturing Technology Management*, *11*(4), 218–238.

Mich, L. (1996). NL-OOPS: from natural language to object, oriented requirements using the natural language processing system LOLITA. *Natural Language Engineering*, *2*(2), 161–187. doi:10.1017/S1351324996001337

Miriyala, K., & Harandi, M. T. (1991). Automatic derivation of formal software specifications from informal descriptions. *IEEE Transactions on Software Engineering*, *17*(10), 1126–1142. doi:10.1109/32.99198

Montana, D. J. (1995). Strongly Typed Genetic Programming. *Evolutionary Computation*, *3*(2), 199–230. doi:10.1162/evco.1995.3.2.199

Moreno, C. A., Juristo, N., & Van de Riet, R. P. (2000). Formal justification in object-oriented modelling: A linguistic approach. *Data & Knowledge Engineering*, *33*, 25–47. doi:10.1016/S0169-023X(99)00046-4

Mraz, R. T., Howe, A. E., von Mayrhauser, A., & Li, L. (1995). System testing with an AI planner. In *Proceedings of the Sixth International Symposium on Software Reliability Engineering* (pp. 96-105), Toulouse, France. Washington DC: IEEE Computer Society.

Neumann, D. (2002). An Enhanced Neural Network Technique for Software Risk Analysis. *IEEE Transactions on Software Engineering*, *28*(9), 904–912. doi:10.1109/TSE.2002.1033229

Nguyen, C. D., Perini, A., & Tonella, P. (2008). eCAT: a tool for automating test cases generation and execution in testing multi-agent systems. *In Proceedings of the Seventh International Joint Conference on Autonomous Agents and Multiagent Systems* (pp. 1669-1670), Estoril, Portugal. New York: ACM Press.

Palisade. (1998). Evolver, The Genetic Algorithm Super Solver, http://www.palisade.com/evolver/. Newfield, NY: Palisade Corporation

Pedrycz, W., & Peters, J. F. (1997). Computational intelligence in software engineering. *Canadian Conference on Electrical and Computer Engineering*, (pp. 253-256), St. Johns, Nfld., Canada. Washington DC: IEEE Press.

Poo, D. C. C., & Lee, S. Y. (1995). Domain object identification through events and functions. *Information and Software Technology*, *37*(11), 609–621. doi:10.1016/0950-5849(95)98298-T

Presland, S. G. (1986). *The analysis of natural language requirements documents*. PhD Thesis, University of Liverpool, UK.

Rech, J. & Althoff, K.D., (2004). Artificial Intelligence and Software Engineering – Status and Future Trends. *Special Issue on Artificial Intelligence and Software Engineering, KI* (3), 5-11.

Ribeiro, J. C. B. (2008). Search-based test case generation for object-oriented java software using strongly-typed genetic programming. In *Proceedings of the GECCO Conference Companion on Genetic and Evolutionary Computation* (pp. 1819-1822), Atlanta, GA. New York: ACM Press.

Saeki, M., Horai, H., & Enomoto, H. (1989). Software development process from natural language specification. In *Proceedings of the 11th international Conference on Software Engineering.* (pp. 64-73), Pittsburgh, PA. New York: ACM Press.

Samson, D. (1990). REQSPERT: automated test planning from requirements. *In Proceedings of the First International Systems Integration Conference* (pp. 702-708), Morristown, NJ. Piscataway, NJ: IEEE Press.

Samson, D. (1993). Knowledge-based test planning: Framework for a knowledge-based system to prepare a system test plan from system requirements. *Journal of Systems and Software*, *20*(2), 115–124. doi:10.1016/0164-1212(93)90003-G

Sathi, A., Fox, M. S., & Greenberg, M. (1985). Representation of Activity Knowledge for Project Management. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, *PAMI-7*(5), 531–552. doi:10.1109/TPAMI.1985.4767701

Scheetz, M., von Mayrhauser, A., & France, R. (1999). Generating test cases from an OO model with an AI planning system. *In Proceedings of the Tenth International Symposium on Software Reliability Engineering* (pp. 250-259), Boca Raton, Florida. Washington DC: IEEE Computer Society.

Shan, Y., McKay, R. I., Lokan, C. J., & Essam, D. L. (2002). Software project effort estimation using genetic programming. *In Proceedings of the IEEE International Conference on Communications, Circuits and Systems* (pp. 1108-1112), Arizona. Washington DC: IEEE Computer Society.

Siemens, N. (1971). A Simple CPM Time-Cost Tradeoff Algorithm. *Management Science*, *17*(6), 354–363. doi:10.1287/mnsc.17.6.B354

Siricharoen, W. V. (2008). Merging Ontologies for Object Oriented Software Engineering. *International Conference on Networked Computing and Advanced Information Management,* (Volume 2, pp. 525 – 530).

Smith, T. J. (1993). READS: a requirements engineering tool. *Proceedings of IEEE International Symposium on Requirements Engineering,* (pp. 94–97), San Diego. Washington DC: IEEE Computer Society.

Thwin, M. M. T., & Quah, T.-S. (2002). Application of neural network for predicting software development faults using object-oriented design metrics. In *Proceedings of the Ninth International Conference on Neural Information Processing* (pp. 2312-2316), Singapore. Washington DC: IEEE Computer Society.

Tonella, P. (2004). Evolutionary testing of classes. In *Proceedings of the ACM SIGSOFT International Symposium on Software testing and Analysis* (pp. 119-128), Boston, MA. New York: ACM Press.

Untch, R. H. (1992). Mutation-based software testing using program schemata. *In Proceedings of the Thirtieth ACM South Regional Conference* (pp. 285-291), Raleigh, North Carolina. New York: ACM Press.

Vadera, S., & Meziane, F. (1994). From English to Formal Specifications. *The Computer Journal*, *37*(9), 753–763. doi:10.1093/comjnl/37.9.753

Vadera, S., & Meziane, F. (1997). Tools for Producing Formal Specifications: A view of Current Architecture and Future Directions. *Annals of Software Engineering*, *3*, 273–290. doi:10.1023/A:1018950324254

van Lamsweerde, R. Darimont, & Massonet P. (1995). Goal-directed elaboration of requirements for a meeting scheduler: Problems and lessons learnt. In *Proceedings of the 2nd IEEE International Symposium on Requirements Engineering*. (pp. 194-203), York, UK. Washington DC: IEEE Computer Society.

von Mayrhauser, A., France, R., Scheetz, M., & Dahlman, E. (2000). Generating test-cases from an object-oriented model with an artifical-intelligence planning system. *IEEE Transactions on Reliability*, *49*(1), 26–36. doi:10.1109/24.855534 von Mayrhauser, A., Scheetz, M., & Dahlman, E. (1999). Generating goal-oriented test cases. In *Proceedings of the Twenty-Third Annual International Conference on Computer Software and Applications* (pp. 110-115), Phoenix, AZ. Washington DC: IEEE Computer Society.

von Mayrhauser, A., Scheetz, M., Dahlman, E., & Howe, A. E. (2000). Planner based error recovery testing. In *Proceedings of the Eleventh International Symposium on Software Reliability Engineering* (pp. 186-195), San Jose, CA. Washington DC: IEEE Computer Society.

von Mayrhauser, A., Walls, J., & Mraz, R. T. (1994a). Sleuth: A Domain Based Testing Tool. In *Proceedings of the International Test Conference* (pp. 840-849). Washington, DC: IEEE Computer Society.

von Mayrhauser, A., Walls, J., & Mraz, R. T. (1994b, November). Testing Applications Using Domain Based Testing and Sleuth. In *Proceedings of the Fifth International Symposium on Software Reliability Engineering* (pp. 206-215), Monterey, CA. Washington DC: IEEE Computer Society.

Vongdoiwang, W., & Batanov, D. N. (2005). Similarities and Differences between Ontologies and Object Model. In *Proceedings of The 3th International Conference on Computing, Communications and Control Technologies: CCCT 2005*, Austin, TX.

Wallace, L., & Keil, M. (2004). Software project risks and their effect on outcomes. *Communications of the ACM*, *47*(4), 68–73. doi:10.1145/975817.975819

Wappler, S., & Lammermann, F. (2005). Using evolutionary algorithms for the unit testing of object-oriented software. In *Proceedings of the Conference on Genetic and Evolutionary Computation* (pp. 1053-1060), Washington DC. New York: ACM Press.

Wappler, S., & Schieferdecker, I. (2007). Improving evolutionary class testing in the presence of non-public methods. In *Proceedings of the Twenty-second IEEE/ACM International Conference on Automated Software Engineering* (pp. 381-384), Atlanta, Georgia. New York: ACM Press.

Wappler, S., & Wegener, J. (2006). Evolutionary unit testing of object-oriented software using strongly-typed genetic programming. *In Proceedings of the Eighth Annual Conference on Genetic and Evolutionary Computation* (pp. 1925-1932), Seattle, WA. New York: ACM Press.

Wright, J. M., Fox, M. S., & Adam, D. (1984). *SRL/2 User Manual*: Robotic Institute, Carnegie-Mellon University, Pittsburgh, PA.

Yang, H.-L., & Wang, C.-S. (2009). Recommender system for software project planning one application of revised CBR algorithm. *Expert Systems with Applications*, *36*(5), 8938–8945. doi:doi:10.1016/j.eswa.2008.11.050

Yang, Y., Xia, F., Zhang, W., Xiao, X., Li, Y., & Li, X. (2008). Towards Semantic Requirement Engineering. *IEEE International Workshop on Semantic Computing and Systems* (pp. 67-71). Washington, DC: IEEE Computer Society.

Yen, J., & Liu, F. X. (1995). A Formal Approach to the Analysis of Priorities of Imprecise Conflicting Requirements. In *Proceedings of the 7th international Conference on Tools with Artificial intelligence.* Washington DC: IEEE Computer Society.

Young, R. R. (2003). *The requirements Engineering Handbook*. Norwood, MA: Artech House Inc.

Yu, Y., Wang, Y., Mylopoulos, J., Liaskos, S., Lapouchnian, A., & do Prado Leite, J. C. S. (2005). Reverse engineering goal models from legacy code. In *Proceedings of the 2nd IEEE International Symposium on Requirements Engineering*, (pp. 363-372), York, UK. Washington DC: IEEE Computer Society.

Yujia, G., & Chang, C. (2006). Capability-based Project Scheduling with Genetic Algorithms. In *Proceedings of the International Conference on Intelligent Agents, Web Technologies and Internet Commerce* (pp. 161-161), Sydney, Australia. Washington DC: IEEE Computer Society.

Zave, P. (1997). Classification of Research Efforts in Requirements Engineering. *ACM Computing Surveys*, *29*(4), 315–321. doi:10.1145/267580.267581

Zhen-Yu, Z., Wei-Yang, Y., & Qian-Lei, L. (2008). Applications of Fuzzy Critical Chain Method in Project Scheduling. In *Proceedings of the Fourth International Conference on Natural Computation* (pp. 473-477), Jinan, China. Washington DC: IEEE Computer Society.