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
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,
nuclear 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 assess-

ing 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
eperience 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