Engineering artificial intelligence software

D. Partridge
Department of Computer Science, New Mexico State University, USA

Abstract Artificial Intelligence (AI) software is a reality, but only for limited classes of problems. In general, AI problems are significantly different from those of conventional software engineering. The differences suggest a different program development methodology for AI problems: one that does not readily yield programs with the desiderata of practical software (reliability, robustness, etc.). In addition, the problem of machine learning must be solved (to some degree) before the full potential of AI can be realized, but the resultant self-adaptive software is likely to further aggravate the software crisis. Realization of the full potential of AI in practical software awaits some prerequisite breakthroughs in both basic AI problems and an appropriate AI software development methodology.

Introduction

Artificial Intelligence (AI), after years of being almost exclusively an academic concern, is now seen to offer techniques that are applicable to commercial software products. AI software is a reality; it consists largely of expert systems, knowledge bases, and natural language interfaces. Expert systems can, for example, analyse chemical mass spectromgrams (DENDRAL and its derivatives, Buchanan & Feigenbaum, 1978), and configure complex computer systems (RI and its extensions, McDermott, 1980). The INTELLECT system is one such natural language interface that allows a user to communicate with a database in English (Eisenberg & Hill, 1984).

So the outlook for the future of AI in software engineering is bright from the purely technological viewpoint (the desirability of the possible social repercussions is altogether a different question, one that I have considered elsewhere, Partridge, 1986a). The initial problems have been conquered; it is only a matter of refining and honing techniques, and of scaling up into less restricted domains. Duda and Gaschnig (1981) give an optimistic overview of the role of knowledge-based expert systems as practical software.

In this tutorial I shall offer a somewhat different view. I shall suggest that 'yes', expert systems and natural language interfaces are harbingers of things to come
from the introduction of AI technology into commercial software. But I shall further suggest that the current expert systems' technology embodies severe inherent limitations with respect to its potential for practical applications as commercial software. Complex basic problems must be solved, to some extent, before we shall reap the benefits of the full potential of AI in commercial software.

I shall outline the general characteristics of programs that are necessary for practical software, and I shall show how conventional software engineering attempts meet these requirements. I shall then show that many AI problems appear to require a program development methodology that does not yet yield programs that exhibit the necessary characteristics of practical software. I shall attempt to delineate the resultant problems as well as sketch out some potential routes to solutions. A by-product of this analysis will be an outline of the scope and limitations of current expert systems' technology. A general cognizance of these limitations will help to circumscribe the current limits of AI as practical software.

The design of practical software

The design of practical software is both an art and a science. Software engineering is the design and construction of computer programs to perform complete and rigorously-specified functions. As the programs are intended for extended practical use they must be robust, reliable and maintainable. In the absence of proofs of program correctness, these requirements demand that the products of software engineering are comprehensible.

There is more or less general agreement that the complexity of the conventional software engineering task is best tackled by a strict separation of the overall process into a sequence of distinct stages (Boehm, 1979). Four such stages are: requirements analysis, specification, design and implementation.

This is the decomposition scheme that I shall use. The initial stage in the production of practical software is requirements analysis, or requirements engineering: the initial user-defined problem, which may be both inconsistent and incomplete, is transformed, via requirements analysis, into a formalized specification, the self-consistency and completeness of which can be more thoroughly checked. Requirements analysis develops an explicit characterization of the scope and limitations of the problem. A somewhat old, but useful, collection of papers on requirements analysis can be found in Ross (1977).

Ideally, the software engineer would like to construct a statement of the problem and justify it once only — right at the beginning — as a result of the requirements analysis step. Mistakes are less costly if they are discovered early on; Boehm (1984) suggests that errors removed in the requirements analysis step cost approximately 100 times less than removal from the subsequent system when it is operational.

From requirements analysis a rigorous functional specification (RFS). The RFS is a generated, sometimes called a rigorous functional specification (RFS). The RFS is a clear statement of what is to be done; the subsequent design process will expand on the RFS to develop a statement of how it is to be done. This what to how relationship is an important ground rule within conventional software engineering;