International Journal of Contemporary Research in Computer Science and Technology (IJCRCST) e-ISSN: 2395-5325
Volume3, Issue 1 (January'2017)

DEPLOYING ARTIFICIAL INTELLIGENCE TECHNIQUES IN SOFTWARE ENGINEERING

J.Maria Merceline Vijila,
Associate Professor,
Department of Computer Science,
Holy Cross Home Science College,
Thoothukudi,TamilNadu,India.

R.Abinaya,
Student,
Department of Computer Science,
Holy Cross Home Science College,
Thoothukudi,TamilNadu,India.

Abstract: There has been a recent surge in interest in the application of Artificial Intelligence (AI) techniques to Software Engineering (SE) problems. The work is typified by recent advances in Search Based Software Engineering, but also by long established work in Probabilistic reasoning and machine learning for Software Engineering. This paper explores some of the relationships between these strands of closely related work, arguing that they have much in common and sets out some future challenges in the area of AI for SE. Software development is a very complex process that, at present, is primarily a human activity. Programming, in software development, requires the use of different types of knowledge: about the problem domain and the programming domain. It also requires many different steps in combining these types of knowledge into one final solution. This paper intends to review the techniques developed in artificial intelligence (AI) from the standpoint of their application in software engineering. In particular, it focuses on techniques developed (or that are being developed) in artificial intelligence that can be deployed in solving problems associated with software engineering processes.

Keywords: Analysis, Synthesis, Programming, Domain, Conversion, Design, Coding

I.INTRODUCTION

Three fundamental questions are likely to run through your mind as you read this paper.

- What is software engineering?
- What is artificial intelligence (AI)?
- What are the aspects of software engineering that makes it amenable to concepts and techniques in artificial intelligence?

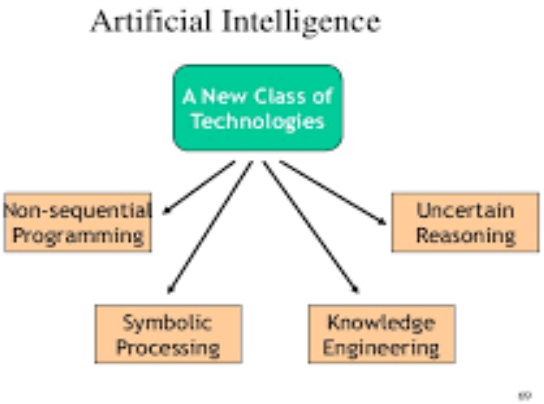

Figure1: Symbolic representation of AI

Software engineering is the act of adopting engineering principles in software development. In this act, the principles of analysis and synthesis are observed . Analysis is the process of breaking something into pieces or components with a view to understanding the individual components. Synthesis, the reverse of analysis, is the putting together of a large structure from small building blocks. Thus, any problem-solving technique must have two parts: analyzing the problem to determine its nature and then synthesizing a solution based on the analysis.

II.SOFTWARE ENGINEERING AND ARTIFICIAL INTELLIGENCE

Software engineering requires two kinds of knowledge: programming knowledge (e.g. data structure construction, control structure, programming language syntax, and how to combine and choose them); and domain knowledge (e.g. concepts, theories and equations characterizing the particular domain). Each domain has its own terms and properties, which must be linked to the programming language in which the software is being developed. Thus, it requires that there be a conversion from one knowledge set to another (e.g. from domain to programming language and vice versa). This obviously requires some method of conversion which requires some techniques in AI. The bottom line in this conversion is that both languages are formal thus making it particularly suitable for AI. There is a similar mapping taking place when going from the requirements specification stage to the design stage, though the initial specification is an informal description of the problem. Thus, the conversion is from an informal statement to a formal one. Software engineering also involves modeling, analysis, and the generation of alternate designs, which entails problem decomposition. Techniques are needed to analyze and evaluate these decompositions in order to choose one design over the others. The kind of decision making needed at this stage is amendable to AI.

First, in the translation from the informal description of requirements into formal descriptions, natural language processing, a sub field in AI, may be used. The task is not so much of direct translation from the natural language to a formal one, but rather to provide help to users since there is no completely automated process that can convert natural language representations to a formal form. However, it can generate questions and elaborations of what the user is writing in natural language. It is the response to these questions that eventuality forms what the system generates. In database management systems, this technique has been adopted in that the naïve user now has a natural English view through which he can write query statements in pure natural English. This sub field of AI encompasses:

- **Voice Communication**: for person-to-computer interaction through vocal inputs with microphone;
- **Speech Synthesis**: for computer-to-person interaction through sound generated by a synthesizer;
- **Language Comprehension**: for person-computer interaction through symbols such as text and words.

Adopting this feature in software engineering can go a long way to solve the problem of extensive documentation.

The traditional view of software development process begins at the requirements specification and ends at testing the software. At each of these stages, different kinds of knowledge (design knowledge at design stage and programming and domain knowledge at the coding stage) are required. At each of the two stages: design and coding, exist a cycle: error recognition and correction. Experience shows that errors can occur at any stage of development . Errors due to coding may occur because of faulty design. Such errors are usually expensive to correct.

Knowledge-based techniques in AI can be used to modify this traditional approach. One strategy is to automatically translate from the requirements specification to program testing. Taking the whole problem as a continuum and makingchanges at the specification stage accomplishes this. The user only provides the requirements and the machine does the translation into codes. This technique is advantageous in that:

- If done correctly, it reduces cost.
- Errors detected in coding will be isolated in the requirements stage.
- Changes need be made only at the requirements stage.

This isolation however is difficult to achieve and has not proven very efficient for large programs.

To generate a system in software engineering, one might find another system with similar requirements. The design of the first system would then be modified until it becomes a reasonable design for the given problem. Although this process looks feasible, it has not been demonstrated in software engineering to any great extent.On the other hand, AI offers a technique called constraint propagation technique which gives rise to a truth maintenance system that can be used for planning. Decisions are made at one place of the planning process and carried to the next level. Then, using analogical reasoning—an offshoot of logical reasoning and problem solving—is used to compare a problem for which a solution is known with another problem to be solved.

A basic problem of software engineering is the long delay between the requirements specification and product delivery. This long development cycle causes requirements to change before product arrival. It therefore becomes imperative to have an automated system which can take the requirements and drive all the way through multiple levels of translation to codes.

In addition, there is the problem of phase independence of requirements, design and codes. Phase independence means that any decision made at one level becomes fixed for the next level. Thus, the coding team is forced to recode whenever there is change in design. However, the AI technique that handles this problem is automated programming which results in reusable code. Thus, when a change is made in the design, that part of the design that does not change remains unaffected. Thus, automated tools for system redesign and reconfiguration resulting from a change in the requirements will serve a useful purpose. This technique, of course, cannot work without employing a constraint propagation technique.

III.TECHNIQUES AND TOOLS OF AUTOMATED PROGRAMMING

Because of the evolutionary nature of software products, by the time coding is completed, requirements would have changed (because of the long processes and stages of development required in software engineering): a situation that results in delay between requirement specification and product delivery. There is therefore a need for design by experimentation, the feasibility of which lies in automated programming. Some of the techniques and tools that have been successfully demonstrated in automated programming environments include:

- **Language Feature**: this technique adopts the concept of late binding (i.e. making data structures very flexible). In late binding, data structures are not finalized into particular implementation structures. Thus, quick prototypes are created which result in efficient codes that can be easily changed. Another important language feature is the packaging of data and procedures together in an object, thus giving rise to object-oriented programming: a notion that has been found useful in environments where codes, data structures and concepts are constantly changing. Lisp provides these facilities.
- **Meta Programming**: this concept is developed in natural language processing (a sub field of AI). It uses automated parser generators and interpreters to generate executable lisp codes. Its use lies in the modeling of transition sequences, user interfaces and data transformations.
- **Program Browsers**: these look at different portions of a code that are still being developed or analyzed, possibly to make changes, thus obviating the need for an ordinary text editor. The browser understands the

structures and declarations of the program and can focus on the portion of the program that is of interest.

- **Automated Data Structuring**: this means going from a high-level specification of data structures to a particular implementation structure.

IV.WHEN DOES AI FOR SE WORK WELL?

The areas in which AI techniques have proved to be useful in software engineering research and practice can be characterized as 'Probabilistic Software Engineering', 'Classification, Learning and Prediction for Software Engineering' and 'Search Based Software Engineering'. In Fuzzy and probabilistic work, the aim is to apply to
Software Engineering, AI techniques developed to handle real world problems which are, by their nature, fuzzy and probabilistic. There is a natural fit here because, increasingly, software engineering needs to cater for fuzzy, ill-defined, noisy and incomplete information, as its applications reach further into our messy, fuzzy and ill-defined lives. This is not only true of the software systems we build, but the processes by which they are built, many of which are based on estimates.

V. RELATIONSHIP BETWEEN APPROACHES TO AI FOR SE

The various ways in which AI techniques have been applied in software engineering reveal considerable overlaps. Forinstance, the distinctions between probabilistic reasoning andprediction for software engineering is extremely blurred, if notrather arbitrary. One can easily think of a prediction system asnothing more than a probabilistic reasoner. One can also thinkof Bayesian models as learners and of classifiers as learners, probabilistic reasoners and/or optimisers.Indeed, all of the ways in which AI has been applied tosoftware engineering can be regarded as ways to optimize either the engineering process or its products and, as such, they are all examples of Search Based Software Engineering. That is, whether we think of our problem as one couched inprobability, formulated as a prediction system or characterized by a need to learn from experience, we are always seeking to optimize the efficiency and effectiveness of our approachand to find good cost-benefit tradeoffs. These optimization goals can usually be formulated as measurable objectives and constraints, the solutions to whichare likely to reside in large spaces, making them ripe for computational search. There is very close interplay between machine learningapproaches to Software Engineering and SBSE approaches. Machinelearning is essentially the study of approaches to computationthat improve with use. In order to improve, we need a wayto measure improvement and, if we have this, then we can use SBSE to optimize according to it. Fortunately, in Software Engineeringsituations we typically have a large number of candidatemeasurements against which we might seek to improve.

VI. CHALLENGES AHEAD IN AI FOR SE

This section outlines some of the open problems in theapplication of AI techniques to Software Engineering.

A.SEARCHING FOR STRATEGIES RATHER THAN INSTANCES

Current approaches to the application of AI to SE tend tofocus on solving specific problem instances: the search for testdata to cover a specific branch or a specific set of requirementsor the fitting of an equation to predict the quality of a specificsystem. There is scope to move up the abstraction chain fromproblem instances to whole classes of problems and, fromthere, to the provision of strategies for finding solutions ratherthan the solutions themselves.

B. EXPLOITATION OF MULTICORE COMPUTATION

A somewhat dated view of AI techniques might considerthem to be highly computationally expensive, making thempotentially unsuited to the large scale problems faced bysoftware engineers. Fortunately, many of the AI techniques thatwe may seek to apply to Software Engineering problems, suchas evolutionary algorithms, are classified as 'embarrassinglyparallel'; they naturally decompose into sub-computations thatcan be carried out in parallel.

C.GIVING INSIGHT TO SOFTWARE ENGINEERS

AI techniques do not merely provide another way to findsolutions to software engineering problems, they also offerways to yield insight into the nature of these problems and thespaces in which their solutions are to be found. For instance,though much work has been able to find good requirementsproject plans , designs ,and test inputs, there is also much work that helps usto gain insight into the nature of these problems. For instance, SBSE has been used to reveal the tradeoffs between requirements' stakeholders and betweenrequirements and their implementations and to bring aesthetic judgments into the software design process. There has also been work on understanding the risks involvedin requirements miss-estimation and in project completion times, while predictive models of faults, quality,performance and effort are naturallyconcerned with the provision of insight rather than solutions. There remain many exciting and interesting ways in which AItechniques can be used to gain insight. For example some open problems concerning program comprehension are described elsewhere. Such work is, of course, harder to evaluatethan work which merely seeks to provide solutions to problems,since it involves measuring the effects of the AI techniques onthe provision of insight, rather than against existing known bestsolutions. This is inherently more demanding, and the refereesof such papers need to understand and allow for this elevatedevaluation challenge. However, there is tremendous scope forprogress; AI techniques have already been shown out to performhumans in several software engineering activities.

D. COMPILING SMART OPTIMISATION INTO DEPLOYED SOFTWARE

Most of the work on AI for SE, such as optimisation, prediction and learning has been applied off-line to improveeither the software process (such as software production, designs and testing) or the software itself (automaticallypatching improving and porting). We might

ask ourselves "If we can optimize a version of the system, why not compile the optimization process into the deployed software so that it becomes dynamically adaptive?"

E. NOVEL AI-FRIENDLY SOFTWARE DEVELOPMENTAND DEPLOYMENT

We cannot expect to simply graft AI techniques into existingSoftware Engineering process and use-cases. We need to adaptthe processes and products to better suit a software engineeringworld rich in the application of AI techniques. AI algorithms arealready giving us intelligent software analysis, development, testingand decision support systems. These smart tools seek to support existing software development methods and processes, asconstructed for largely human-intensive software development.As the use of automated smart AI-inspired tools proliferates,we will need to rethink the best ways in which these can beincorporated into the software development process.

VII.CONCLUSION

Although a very formal theory has been discussed, automated programming still has its own limitations and is sometimes impractical. With the concepts well illustrated, the problem is in the synthesis of big programs. Thus, special cases will need to be identified where these processes are practical.

Besides, the basic concepts for automating program development must be language independent. One of the enhanced language features is object-oriented programming which shortens the requirements-to-code cycle. This object-oriented concept has now been fully developed and is been embedded in a number of object-oriented languages such as C++, Visual C++. Data and procedure packaging (or data binding), another language feature, has also been implanted in visual basic, C++, Visual dbase, Delphi, among others.

VIII.REFERENCES

[1]. Stuart Russell and Peter Norvig, Artificial Intelligence: A Modern Approach (Prentice Hall Publishers, Upper Saddle River, New Jersey USA) 1994.
[2]. Ian Sommerville, Software Engineering (6th Edn.) (Addison Wesley Publishers, New York, New York, USA) 2000.
[3]. Roger S. Pressman, Software Engineering: A Beginner's Guide (McGraw Hill Higher Education Publishers, New York, New York, USA) 1988.
[4]. Seth Hock, Computers and Computing (Houghton Mifflin College Publishers, Boston, Massachusetts, USA) 1989.
[5]. M.L. Emrich, A. Robert Sadlowe, and F. Lloyd Arrowood (Editors), Expert Systems And Advanced Data Processing: Proceedings of the conference on Expert Systems Technology the ADP Environment (Elsevier-North Holland, New York, New York, USA) 1988.
[6]. Shari Lawrence Pfleeger, Software Engineering: theory and Practice (Prentice Hall Publishers, Upper Saddle River, New Jersey, USA) 1998.
[7]. C.S. French, Data Processing and Information Technology (10th edition), (Letts Educational Publishers, London, United Kingdom) 1996.
[8]. 8.American journal of undergraduate research VOL. 1 NO. 1 (2002)
[9]. http://www.cdmasoftware.com/eng.html
[10]. http://www.hackinthebox.org/