ISSN No: 2454-9614

Prospects of Artificial Intelligence in Software Engineering

S. Ragavarthini , B. Vidhya riya*, R. Sangeetha , N.K.G Ragavi, D. Durga lakshmi

Department Of Computer Science and Engineering, Angel College of engineering and technology
Tirupur, Tamilnadu, India.

*Corresponding Author: S. Ragavarthini
E-mail: ragavi473@gmail.com

Received: 2/11/2015, Revised: 20/12/2015 and Accepted: 05/03/2016

Abstract

"Intelligence is the art of good guesswork". Emerging technologies and programming techniques increase our ability to create intelligent software programs. With the advent of viable software engineering solutions, we have come even closer to building artificially intelligent software. Software Engineering (SE) and Knowledge Engineering (KE) are closely related disciplines with goals of turning the development process of software systems and knowledge-based systems, respectively, into engineering disciplines. In particular, they together can provide systematic approaches for engineering intelligent software systems more efficiently and cost-effectively. As there is a large overlap between the two disciplines, the interplay is vital for both to be successful. In this paper, we divide the intersection of SE and KE into three subareas, namely Knowledge-Supported Software Engineering (KSSE), Engineering Knowledge as Software (EKaaS), and Intelligent Software System Engineering (ISSE). For each subarea, we describe the challenges along with the current trends, and predict the future research directions that may have the most potential for success. The projections discussed here are provided both by technology experts and concerned non-experts. Computer systems will continue to get more powerful, and will become increasingly ubiquitous in the future, making the standards of development of artificial intelligence a salient topic in modern engineering. Despite a socially ingrained fear of intelligent machines, there is no governing body to oversee the continued development of AI systems. The disciplines of Artificial Intelligence and Software Engineering have many commonalities. Both deal with modeling real world objects from the real world like business processes, expert knowledge, or process models Development of a strong artificial intelligence would surely call into question (for some)that which we define as "alive." It is yet unclear whether an electronic entity would be titled to legal and civil rights. Furthermore, we do not know whether such an entity oracle of entities would be dangerous to society. These problems indicate a strong ethical component in the development of intelligent software. This paper argues that intelligent machines will be intertwined in our future society, and addresses the lack of a concrete body to govern the development of computer software. The accompanying research further establishes that engineers will have increased ethical and political responsibilities in the development of artificial intelligence systems in the future.

Keywords: *Software engineering; Knowledge engineering; Knowledge-supported software engineering; intelligent software systems; Knowledge as software. *Reviewed by* **ICETSET'16** *organizing committee*

1. Introduction

The disciplines of artificial intelligence and software engineering have developed separately. There is not much exchange of research results between them. AI research techniques make it possible to perceive reason and act. Research in software engineering is concerned with supporting engineers to developed better software in fewer periods. Rech and Altoff (2008) say "The disciplines of artificial intelligences and software engineering have many commonalities. Both deal with modelling real world objects from the real world like business process, expert knowledge, or process models."

Today several research directions of both disciplines come closer together and are beginning to build new research areas. Software agents play an important role as research objects in distributed AI (DAI) as well as in Agent Oriented Software Engineering (AOSE). Knowledge-based System (KBS) are being examined for Learning Software Organizations (LSO) as well as Knowledge Engineering (KE). Ambient intelligence (AmI) a new research area for distributed, non-intrusive, and intelligent software system both from the direction of how to build these system as well as how to design the collaboration between systems. Lastly computational intelligence (CI) plays an important role in research about software analysis or project management as well as knowledge discovery in machine learning or databases **[9].**

Artificial Intelligence techniques, which aim to create software systems that exhibit some form of human intelligence, have been employed to assist or automate the activities in software engineering. Software inspections are been applied with great success to detect defects in different kinds of software documents such as specifications, design, test plans, or source code by many researchers.**[12]**

Automated software engineering is a research area, which is constantly developing new methodologies and technologies. It includes toolsets and frameworks based on mathematical models (theorem proves and model checkers), requirements-driven developments and reverse engineering (design, coding, verification validation), software management (configurations and projects), and code drivers (generators, analyzers, and visualises).

In the following sections we have tried to review some research techniques to automate each phase of software development life cycle using artificial intelligence.

2. Software Engineering And Knowledge Engineering

Figure 1 shows a mapping of the relationship between SE and KE. As shown in the figure, the intersection of SE and KE is identified as SEKE (Software Engineering and Knowledge Engineering), which represents the major topic of this paper. There are many books, journals and proceedings published in the past two decades that are related to SEKE. Among them, the International Conference on Software Engineering and Knowledge Engineering

(SEKE) and its associated International Journal of Software Engineering and Knowledge Engineering (IJSEKE) particularly welcome papers for sharing methods and results between the two disciplines. A central theme of the IJSEKE journal is the interplay between software engineering and knowledge engineering, for example, how knowledge engineering methods can be applied to software engineering, and vice versa **[4].** This emphasis makes the scope of IJSEKE different from that of pure SE journals such as IEEE Transactions on Software Engineering (TSE) and ACM Transactions on Software Engineering and Methodology (TOSEM), and pure KE journals such as IEEE Transactions on Knowledge and Data Engineering (TKDE) and Data and Knowledge Engineering Journal (DKE).

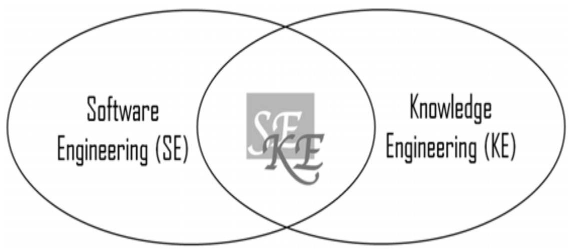

Fig.1 The intersection of software engineering and knowledge engineering Future Research Directions of Software Engineering and Knowledge Engineering 3

In order to study the current trends and the future research directions of SEKE, we divide the intersection of SE and KE into three subareas, namely Knowledge-Supported Software Engineering (KSSE), *Engineering* Knowledge as Software (EKaaS), and Intelligent Software System Engineering (ISSE). As shown in Fig. 2, the top portion of the intersection, *i.e.*, the subarea KSSE, addresses how knowledge engineering methods can be applied to software engineering; in other words, how to make the software engineering activities more efficient and cost-effective using knowledge-based analysis or knowledge-supported systems. A typical example of research in this area is to analyze software engineering data using big data analysis or data mining approaches. The bottom portion of the intersection in Fig. 2 is called EKaaS, which studies the application of software engineering methods to knowledge engineering. Researchers in this subarea view a knowledge-based system as a traditional software system, and study how to make the development process of KBSs more efficient and cost-effective by applying software engineering principles and methodologies. An example of such study is to develop large-scale domain knowledge using modularization principle in SE. In the middle portion of the intersection in Fig. 2, the subarea is identified as ISSE, which focuses on engineering intelligent software systems using both SE, KE methods. There are many different types of intelligent software systems nowadays, among which some typical EX includes knowledge-based software information systems **[5]**, ambient intelligent systems **[6]**, and slow intelligent systems **[7]**.

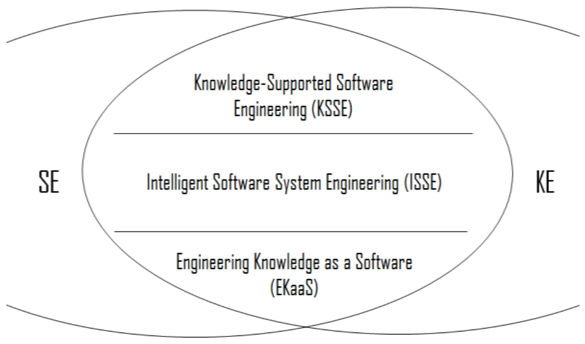
Knowledge-Supported Software Engineering (KSSE)
SE
Intelligent Software System Engineering (ISSE)
KE
Engineering Knowledge as a Software (EKaaS)

Fig.2 Subareas in the intersection of software engineering and knowledge engineering

3. Use of AI in Requirement Specification

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 transforming them into less ambiguous representations.
- Developing knowledge based systems and ontology 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 prioritization **[3].**
- Three aspects may characterize a formal specification:
- Expressiveness: The semantics must cover sufficient details to map real-world descriptions.
- Readability: It helps understanding the specification and thus improves the validation process,
- Structuring: It encourages specification processing. We mean by processing the set of mechanisms that allow refinement, composition, and importation of specifications.

Yoichi Omori and Keijiro Araki propose a developing method that utilizes a formal method to verify the description and feedback to a natural language. A dictionary tool supporting this procedure is developed as an Eclipse plug-in and cooperates with the existing tool for a formal method. Modeling from the description in the natural language into a formal model and feedback from the formal model to the description in the natural language is effectively supported through the dictionary **[5]**.

4. Use of AI in Requirement Tracing

Requirements traceability is an important undertaking as part of ensuring the quality of software in the early stages of the Software Development Life Cycle. Swarm intelligence is applied to the requirements tracing problem using pheromone communication and a focus on the common text around linking terms or words in order to find related textual documents. Through the actions and contributions of each individual member of the swarm, the swarm as a whole exposes relationships between documents in a collective manner. Two techniques have been examined, simple swarm and pheromone swarm. The techniques have been validated using two real-world datasets from two problem domains. The swarm agents mimic and borrow useful behavioural features from communal insects such as ants and bees to identify and promote candidate links between two sets of documents. The simple swarm approach, the heuristic to select a term or traversal path in the search space is based on term or document attributes **[11].**

5. Use of AI in Software Design

Both architecture design and detailed design require designers to apply their technical knowledge and experience to evaluate alternative solutions before making commitments to a definite solution. Normally, a designer starts with a guess of the solution, and then goes back and forth exploring candidate design transformations until arriving to the desired solution (Tekinerdogan, 2000). This exploration of the design space is conceptualized into two main stages: (i) from quality-attribute requirements to (one or more) architectural models - called QAR-to-AM phase, and (ii) from an architectural model to (one or more) object-oriented models - called AM-to-OOM phase. Making the right design decisions for each phase is a complex, time-consuming and error-prone activity for designers. Although tools for specification and analysis of designs exist, these tools do not support the designer in making informed decisions based on quality-attribute considerations.

Along this line, several AI developments have shown the benefits of improving conventional tools with intelligent agents. The metaphor here is that the agent acts like a personal assistant to the user (Maes, 1994). This assistant should be able to monitor the designer’s work, and offer timely guidance on how to carry out design tasks or even perform routine computations on her behalf. For example, given a modifiability scenario, a design assistant could recommend the use of a Client-Server pattern to satisfy that scenario. If the designer agrees to apply such a pattern, the assistant could also take over the assignment of responsibilities to Client and Server components. **[10]**

6. Use of AI in Code Generation

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 International Journal of Advanced Computer Engineering and Communication Technology (IJACECT) 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 modelling 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. When systematic changes need to be made throughout a code, it is more efficient and controllable to do it through another program (i.e., program update manager) than through a manual txt editor. For instance, a change in program X may be required whenever h is being updated by b-1 under the condition that b is less than C. Assume that a program W makes a systematic change in all such places. If another program makes a change in W, then any program changed by W also must be updated. Thus, program update managers propagate changes. Because of this ability, program update managers are useful when prototypes need to be developed quickly **[4].**

Most automatic code generation tools help developers write software from graphical representations of the requirements specifications to visually specify structure and behaviour of the program by means of a modelling language, e.g. the UML. Some of them instead are able to generate software code from textual representations, e.g. a natural language. Few approaches can generate software code from symbolical representations such as mathematical models. Thus, engineering tools build software code based on pre-defined policies and fixed rules, and then developers specify the program logic. State of the art automation of software process mainly deals with automatic code generation. Some solution automate verification process and very few of them are able to code software from requirements. However, these automated software developments are focused on predefined policies and fixed rules to generate code. However, the following approach goes beyond this. The Autonomous Software Code Generation (ASCG) process carried out by the SDA is an agent-oriented approach for automated code generation. It relies on the SDA autonomy to make decisions on how to analyze, design and implement software applications. The approach initially implements only an agent (role as a developer; SDA) who starts dealing with the development of system by reading the requirements specification given as a physical configuration of the Software under Development (SuD). System operations or missions are also specified. The SDA is able to capture this information and queries its own internal knowledge by means of a reasoned in order to make decisions to design the software that realizes the system logic. The system logic is built of interconnected blocks that can exchange information by receiving data from and sending data to other blocks. SA and the OODA loop are the foundation of the ASCG framework. The SDA is able to start the software design with a description of the tanks configuration. Different fuel systems can be graphically described through a visual user interface **[2].**

7. Use of AI in Software Testing

Application of artificial intelligence techniques in engineering and testing of the software is a progressive area of research that leads to the cross-fertilization of ideas in the middle of the two fields. Varieties of AI tools are used to generate test data, research on data suitability, optimization and analysis of the coverage as well as test management. Many automation tasks, such as the generation of test data are developed as constraint solving problems. A well-designed test is expected to reveal software faults. **[8]** Several researchers studied the relevance of search algorithms based on AI such as genetic algorithm, simulated annealing, swarm intelligence as a better alternative for the development of test data generators and have shown promising results. Requirements Based Testing based on particle swarm optimization (PSO), partition testing based on PSO, using PSO and genetic algorithm in the context of evolutionary and structural testing, test case minimization using artificial neural network or data mining info-fuzzy network, using neural network for pruning test cases, software test data generation using ant colony optimization, test optimization using Artificial Bee colony optimization(ABC) are a few techniques where in software testing is made easier using AI.

Frank Padberg et.al.proposed a method for estimating the defect content after an inspection using machine learning. It uses the zero-one matrix of an inspection and an empirical database collected during past inspections as input for computing the estimate. This approach identifies defect content estimation for software inspections as a nonlinear regression problem. A major motivation for our approach was the discovery that some features of an inspection can carry significant nonlinear information about the defect content of the inspected document. Using this information can greatly improve the accuracy of the estimates **[6].**

8. Use of AI in GUI Testing

A growing interest can be seen in use of AI for GUI testing. There has been some research into how GUI testing could be dealt with the help of AI. The various forms of this technique have been found in a quick glance to ACM library search on the topic. Some of these techniques include generating the GUI based on a model, generating tests based on a model, and automating test case generation to make it possible to regenerate the tests each time GUI changes and making automated oracles, which model the behaviour of the user interface. There have also been peeks into generating tests based on artificial intelligence (AI) planning techniques and genetic modelling. **[8]**

9. Use of AI in Software Estimation

Tong SengQuah and Mie MieThetThwin use defect tracking using computational intelligence methods to predict software readiness. Their research extended currently software quality prediction models by including structural/architecture considerations into software quality metrics. The genetic training strategy of NeuroShell Predictor is used in their study. The Genetic Training Strategy uses a "genetic algorithm" or survival of the fittest technique to determine a weighting scheme for the inputs. The genetic algorithm tests many weighting schemes until it finds the one that gives the best predictions for the training data **[7].**

Ricardo de A. Araujo et.al.proposed a hybrid intelligent method, referred to as Morphological-Rank-Linear Hybrid Intelligent Design (MRLHID), using a Modified Genetic Algorithm (MGA) and The Least Mean Squares (LMS) algorithm to design „MRL Perceptions' was proposed to solve the Software Development Cost Estimation (SDCE) problem. The proposed method used the MGA to determine the best particular features to improve the MRL perceptron performance, as well as the initial parameters of MRL preceptor. Furthermore, for each individual of MGA, it is used a gradient steepest descent method (using a Least Mean Squares (LMS) algorithm with a systematic approach to overcome the problem of non-differentiability of the morphological-rank operator) to optimize the MRL perceptron parameters supplied by MGA Two different metrics (PRED (25) and MMRE) were used to measure the performance of the proposed MRLHID model.

A fitness function was designed with these two well-known statistic error measures in order to create a global indicator of the prediction model performance, where the main idea was to maximize the PRED(25) metric and to minimize the MMRE metric. The justification of the inclusion of this metrics to evaluate the proposed method is that most of the existing literature frequently employs the Mean Squared Error (MSE) for estimation evaluation, however it may be used to drive the model in the training process, but it cannot be considered a conclusive measure for comparison of different models. An experimental validation of the method was carried out using the Desharnais and COCOMO databases, showing the robustness of the proposed MRLHID model through a comparison, according to two performance measures and a fitness function, of previous results found in the literature (SVR-Linear, SVR-RBF, Bagging, GA-based with SVR Linear, GA-based with SVR RBF and MRL). This experimental investigation indicates a better, more consistent global performance of the proposed MRLHID model, having around 11% of improvement (Desharnais database) and around 12% of improvement (Cocomo database) regarding the best result reported in the literature. It is possible to notice that the main advantages of the proposed MRLHID model, apart from its superior predictive performance compared to all analyzed models, are
(1) It has both linear and nonlinear components (it means that the model can use a distinct percentage of the linear and nonlinear components according to the linearity or nonlinearity of the problem), and
(2) It is quite attractive International Journal of Advanced Computer Engineering and Communication Technology

(IJACECT) due to its simpler computational complexity, according to analysis presented in **[1].**

9. Conclusion

Software Engineering remains a highly skilled human intensive activity and relies on problem solving skills of human knowledge and experiences. Therefore Artificial intelligence, Expert System, and Knowledge Engineering will continue to play a major role in automating numerous software development activities. The interplay between Artificial Intelligence and Software Engineering is significant and it makes sense to take advantage of their mutual strengths. The survey conducted in this paper has highlighted some trends in the use of AI techniques in the software development process. Further, there is much scope for exploring and evaluating the use different AI techniques in the automation of Software Development Life Cycle in future.

References

[1] Araujo Ricardo de A., Oliveira Adriano L. I. de, and Soares Sergio, "Hybrid Intelligent Design of Morphological-Rank-Linear Perceptrons for Software Development Cost Estimation", 1082-3409/10 © 2010 IEEE 22nd International Conference on Tools with Artificial Intelligence [Pg.No. 160-167].

[2] Insaurralde Carlos C., "Software Programmed by Artificial Agents: Toward an Autonomous Development Process for Code Generation" 2013 IEEE International Conference on Systems, Man, and Cybernetics[Pg.No. 3294-3299]

[3] MezianeFarid, Vadera Sunil, University of Salford, "Artificial Intelligence Applications for Improved Software Engineering Development: New Prospects".[Pg.No. 282]

[4] Onowakpo Jonathan, EbbahGoddey, "Deploying Artificial Intelligence Techniques In Software Engineering", AMERICAN JOURNAL OF UNDERGRADUATE RESEARCH VOL. 1 NO. 1 (2002)[Pg.No.19-23]

[5] Omori Yoichi, Araki Keijiro "Tool Support for Domain Analysis of Software Specification in Natural Language" TENCON 2010, 978-1-4244-6890-4/10/$26.00 ©2010 IEEE[Pg. No. 1065-1070]

[6] Naur and B. Randell (Eds.), *Software Engineering: Report on a Conference Sponsored by the NATO Science Committee*, Garmisch, Germany, October 7-11, 1968.

[7] L. Simons, I. C. Parmee and P. D. Coward, 35 years on: to what extent has software engineering design achieved its goals?,*IEE Proceedings - Software*, **150**(6) (2003) 337-350.

[8] V. R. Benjamins, D. Fensela, Knowledge engineering: principles & methods, *Data & Knowledge Engineering*, **25**(1–2) (1998) 161–197.

[9] K. Chang, Foreword, *International Journal of Software Engineering and Knowledge Engineering (IJSEKE)*, **1**(1) (1991).

[10] Devanbu, P. G. Selfridge, B. W. Ballard and R. J. Brachman, A knowledge-based software information system, in *Proc. of the 12th international conference on Software engineering (ICSE'90)*, Nice, France, March 26-30, 1990, pp. 249-261.