A study on the state of Artificial Intelligence in the Design phase of Software Engineering

Mriganka Shekhar Sarmah[1], Jediael Meshua Sumer[2], Marlom Bey[3], Dr. Bobby Sharma[4]
[1,2,3] Student, Department of Computer Science and Engineering, Assam Don Bosco University, Assam, India
[4]Professor, Department of Computer Science and Engineering, Assam Don Bosco University, Assam, India
mriganka.m2s@gmail.com
jediaelmeshua@gmail.com
marlombey46@gmail.com
bobby.sharma@dbuniversity.ac.in

Abstract-- **The areas of Software Engineering and Artificial Intelligence are among the most sought after fields in the computer science community. This paper assesses the current scenario of implementing artificial intelligence in the design phase of software engineering, and attempts to offer a viable explanation to support the conclusion of the assessment.**

Keywords: Software Engineering, Artificial Intelligence, Natural Language Processing, Domain Ontology, Unified Modelling Language

I. INTRODUCTION

In this day and age, traditional software design has its limits. In traditional SE, the approach towards building a software is manual in nature, which makes it prone to human errors such as oversight and is also inefficient when complicated projects are considered. [6][7]

Another reason for errors may be the ambiguity of the textual requirements which may not be exactly translated according to the perspective of the customer. [8]

Therefore, in order to minimise human error and increase the efficiency of software design, and to deal with ambiguity of textual requirements of the customer, AI techniques can be implemented, mainly in the stages of design, testing and code generation. [1][2]

Hence, the prospects of automation/AI in the phase of software design must be discussed and related problems must be dealt with, which are the reliability and accuracy of automation and how much control it takes away from a software developer.

This paper compares related works regarding how they either tackle this problem or suggests ways to tackle this problem.

The paper is organized into the following sections. Section II discusses works related to the stated topic, section III discusses the analysis and results of existing work, and finally with section IV, the paper is summarized and concluded.

II. RELATED WORKS

In [1], the author discusses the various commonalities between the fields of AI and SE, current status and upcoming trends. Intersections between AI and SE are discussed, especially in the fields of Agent-Oriented Software Engineering, Knowledge Based Engineering Systems, Computational Intelligence and Knowledge Discovery of Data, and Ambient Intelligence. It also gives a short description about AI and software engineering, and the possibilities of interaction with one another through various common points of contact.

In [2], the author discusses the differences between the software engineering process and AI, and the hostilities that exists in software engineers towards automation in developing a software. It further discusses the contribution of AI to the field of SE in fields like automatic programming and some projects like REFINE and Programmer's Apprentice (an MIT project) and the problem areas of SE where AI can be implemented like requirements analysis and definition, process modelling , process support and project planning.

The author states one of the reasons for the existing hostility towards AI is that AI researchers are themselves software engineers, and therefore their method to integrating AI in software engineering follows a typical and traditional approach which precedes the advancements that AI and its endless possibilities can bring to the field of software engineering.

In [3], the author uses the terminology defined in IEEE 12207 standards for software engineering to describe the development processes of requirements analysis, software architecture design and coding, and testing and AI techniques that can be implemented along with them. The paper also talks about open problems in SE that can be solved using AI so as to improve the process of SE in this day and age where software are getting more modular and complex to develop.

In [4], the author deals with a taxonomy called AI in SE Application Levels or AI-SEAL that classifies 15 papers from previous editions of the RAISE workshop. It considers the context in which AI is being applied, i.e. "when" and "on what" AI is being applied. After classification the papers also assigns levels of automation to the papers.

In [5], the author gives a brief overview of SE and expert systems in artificial intelligence and how expert systems can be used to automate the programming process and code generation via the use of genetic programming.

This paper also shows the absence of risk management in AI based systems due to the way they work.

In [6], the author discusses the translation of user requirements into design diagrams and describes it as a daunting task for a designer, as that person has to translate textual requirements into a diagrammatic (UML) form. This paper gives a basic overview of existing systems regarding and how they convert the user requirements using NLP into UML diagrams..

They developed their own tool, DC Builder and presented in the paper a diagrammatic representation and a heuristic set of rules in implementing NLP on user requirements. Finally they evaluated and compared the various tools mentioned and developed, with one another.

In [7], the author describes the extraction of UML diagrams from textual requirements by requirements engineers as a daunting task and the time and effort spent on this justifies a tool to automate this process which brings the author to propose a tool called RAPID which uses NLP in an efficient manner to extract design diagrams from input textual user requirements.

The methodology states various NLP technologies, algorithms and rules to extract class information from the textual requirements.

In [8], the author talks about the ambiguity of textual requirements and the usage of NLP and domain ontology to generate UML diagrams from the said textual requirements.

This paper also talks about existing systems and then approaches the problem using their own method called RAUE to extract UML diagrams from textual requirements.

In [9], the author looks at existing systems which translate textual requirements into UML diagrams. After that the author proposes his own approach called RACE which is an improvement on the TCM system. The methodology of the RACE system is discussed with its algorithms and rules to identify classes, attributes and relationships.

The author concludes the paper stating RACE being an advanced approach towards extraction of UML diagrams from textual requirements using a human-centred UI and what the system did not support.

In [10], the author introduces an automatic test data generator called ATGen. ATGen is based on constraint logic programming and symbolic execution. Developing software include a number of testing methods. The main technique that is currently used in the industry is dynamic software testing where the software is executed using test data. Dynamic testing can be performed using automatic tools, which are - automation of administrative tasks, automation of mechanical tasks and automation of test generation tasks.

ATGen is a prototype testing tool implemented using the ECLiPSe constraint logic programming environment and consists over 5000 lines of commented Prolog code. The current area of application of ATGen is the automatic generation of test data to achieve 100% decision coverage for programs written in SPARK Ada. In decision testing, the aim is to test all decision outcomes in the program. The aim is to generate a test data suite achieving 100% decision coverage. However, initial results of ATGen has proved that at the time this research was conducted it was impossible to experiment with ATGen using industrial SPARK Ada code as such code was not usually made available even for research due to its safety critical aspects. Also, the test results show that ATGen shows wide performance variation between successive runs, particularly for programs with loops. The initial results of ATGen were promising and the overall efficiency of the algorithm is under improvement.

In [11], the author describes an approach towards extracting UML diagrams from natural language using NLP via a tool called CM Builder. CM Builder is a graphical CASE tool that does surface analysis of text to propose candidates for class, attributes and relationship and domain independent semantic analysis to automatically extract the candidates and finally represent them via UML diagrams.

The tool also includes capacity to evaluate its candidate classes. The author also states benefits of object oriented analysis of the tool and the its scope of improvement.

In [12], the author explored the reasons for the lack of impact in important areas in which AI has been expected to significantly affect real world Software Engineering. The session approached the failures of AI in software engineering, looking at the matter through a common cause- reliance on isolationist technology and approaches, rather than upon creating additive technology and approaches that can be integrated with other existing capabilities.

The isolationism has been manifested in several areas, in essence, the market has rejected the isolationist and egocentric approach to implementing AI in software engineering, and that the whole system should be developed and executed in generalized workstations and PCs.

In [13], the author describes recent state-of-the-art approaches to automatically generate regular expressions from natural language specifications. Given that these approaches use only synthetic data in both training datasets and validation/test datasets, a natural question arises: are these approaches effective to address various real-world situations? To explore this question, in this paper, a characteristic study on comparing two synthetic datasets used by the recent research and a real-world dataset collected from the Internet, and an experimental study on applying a state-of-the-art approach on the real-world dataset is conducted. The study results suggest the existence of distinct characteristics between the synthetic datasets and the real-world dataset, and the state-of-the-art approach (based on a model trained from a synthetic dataset) achieves extremely low effectiveness when evaluated on real-world data, much lower than the effectiveness when evaluated on the synthetic dataset.

In [14], the author looks at previous work done in the field of converting natural language requirements into design diagrams and points us the various issues that can be encounter in this process.

The author points out the issues in the context of software engineering, then in the context of natural language and finally proposes a methodology to solve these problems to get good quality UML diagrams.

In [15], the author introduces artificial intelligence to software engineers and conversely, software engineering to artificial intelligence workers. It further highlights the contrast between the two fields and further accentuates their differences in the problems that they attempt to solve, their methodologies and the tools and techniques. The author strongly believes that the work of software engineers and artificial intelligence workers are similar, and the fusion of the two fields is essential for Computer Science as a field to move forward, however, the author also acknowledges the ridge that exists between the two communities. The paper does not however provide the '*how*' on which AI and software engineering can come together, other than details of the core components of both fields.

The paper acknowledges that to move forward with 'user-friendliness', the system needs to allow natural language input and output, and natural language processing is one of the most researched areas in AI. In other words, a system needs AI to further improve its usability.

Conversely, AI is in need of proper models to address its problems. These models are already evident in conventional systems, and it is clear that AI systems need some standardization in order to widen their applications.

Thus, there exists a requirement for both fields to merge on some common ground.

III. RESULTS AND ANALYSIS OF EXISTING WORK

From the study done above a few things are apparent. Firstly, when design diagrams made by a software engineer are considered, three factors come into play-

- Ambiguity and complexity of NL textual requirements. [13]
- Complex nature of the project due to increasing complexity of technology and size of the project.
- Possibility of human error when designing those systems from textual requirements.

The above results in the problem of defining the accuracy of a system, be it designed by a person or by an AI based tool, as there is no point of references. This makes it imperative to create a dataset to compare any diagram to measure its accuracy.

Another way it can be done is to design an algorithmic model that can analyze input textual requirements to give appropriate output, but the amount of work involved in making it foolproof would make it an insurmountable challenge.

Another aspect of translating user requirements into a design diagram is that when the textual requirements are complicated, extracting context requires the use of domain ontology along with defined semantic rules. When using this, problems will be encountered when very complex sentences are considered.

Secondly, when AI in system design is considered, due to its automatic nature, the problem of a system engineer not having enough freedom over the creation of design diagrams, is encountered. Hence, there exists a need to carefully define the level of automation of such tools. This can be done by making the end results of a diagram generator interactive or editable, as can be seen in various smart design diagram tools.

Support	CM-Builder [11]	LIDA	GOOAL	NLOOML	DC-Builder [6]	RACE [9]	RAUE [8]	RAPID [7]
Classes	Yes	User	Yes	Yes	Yes	Yes	Yes	Yes
Attributes	Yes	User	Yes	Yes	Yes	Yes	Yes	No
Methods	No	User	Yes	Yes	No	Yes	No	No
Associations	Yes	User	Semi-NL	No	Yes	Yes	Yes	Yes
Multiplicity	Yes	User	No	No	No	No	Yes	No
Aggregation	No	No	No	No	Yes	Yes	Yes	Yes
Generalization	No	No	No	No	Yes	Yes	Yes	Yes
Instances	No	No	No	No	No	No	No	No

Table I: Evaluation of Tools' functionality (Adapted and modified from [6])

Table I above shows the comparison of the availability of features of various tools that generate design diagrams from user input textual requirements. It can be inferred the table that there are very few tools which fully extract relationships from textual requirements.

Another observation made from the tools mentioned is that although they can create Class diagram they cannot deal with instances to create Object diagrams.

IV. CONCLUSION AND FUTURE WORK

From the above survey on the papers, it is evident that artificial intelligence has a wide scope of application in the field of software engineering, and has the capability to make a software engineer's work easier and faster. However, AI systems have to undergo further improvements to fully exploit their capabilities in the field of software engineering.

The findings of this paper can be further analyzed and utilized to design methodologies that make generation of UML Design diagrams from input user requirements far more efficient. Furthermore, more research has to be done, aimed towards the use of machine learning to automate the task of diagram generation.

REFERENCES

[1] Rech, J., & Althoff, K. D. (2004). Artificial intelligence and software engineering: Status and future trends. *KI*, *18*(3), 5-11.

[2] Sommerville, I. (1993). Artificial intelligence and systems engineering. Prospects for Artificial Intelligence: Proceedings of AISB'93, 29 March-2 April 1993, Birmingham, UK, 17, 48.

[3]Ammar, H. H., Abdelmoez, W., & Hamdi, M. S. (2012, February). Software engineering using artificial intelligence techniques: Current state and open problems. In *Proceedings of the First Taibah University International Conference on Computing and Information Technology (ICCIT 2012), Al-Madinah Al-Munawwarah, Saudi Arabia* (p. 52).

[4] Feldt, R., de Oliveira Neto, F. G., & Torkar, R. (2018, May). Ways of applying artificial intelligence in software engineering. In *2018 IEEE/ACM 6th International Workshop on Realizing Artificial Intelligence Synergies in Software Engineering (RAISE)* (pp. 35-41). IEEE.

[5] Raza, F. N. (2009, March). Artificial intelligence techniques in software engineering (AITSE). In *International MultiConference of Engineers and Computer Scientists (IMECS 2009)* (Vol. 1).

[6] Herchi, H., & Abdessalem, W. B. (2012). From user requirements to UML class diagram. *arXiv preprint arXiv:1211.0713*.

[7] More, P., & Phalnikar, R. (2012). Generating UML diagrams from natural language specifications. *International Journal of Applied Information Systems, Foundation of Computer Science*, *1*(8), 19-23.

[8] Joshi, S. D., & Deshpande, D. (2012). Textual requirement analysis for UML diagram extraction by using NLP. *International journal of computer applications*, *50*(8), 42-46.

[9] Ibrahim, M., & Ahmad, R. (2010, May). Class diagram extraction from textual requirements using natural language processing (NLP) techniques. In *2010 Second International Conference on Computer Research and Development* (pp. 200-204). IEEE.

[10] Meudec, C. (2001). ATGen: automatic test data generation using constraint logic programming and symbolic execution. *Software Testing, Verification and Reliability*, *11*(2), 81-96.

[11] Harmain, H. M., & Gaizauskas, R. (2000, September). CM-Builder: an automated NL-based CASE tool. In *Proceedings ASE 2000. Fifteenth IEEE International Conference on Automated Software Engineering* (pp. 45-53). IEEE.

[12] Balzer, R., Fikes, R., Fox, M., McDermott, J., & Soloway, E. (1990, August). AI and software engineering: will the twain ever meet?. In *AAAI* (pp. 1123-1125).

[13] Zhong, Z., Guo, J., Yang, W., Xie, T., Lou, J. G., Liu, T., & Zhang, D. (2018, June). Generating Regular Expressions from Natural Language Specifications: Are We There Yet?. In *Workshops at the Thirty-Second AAAI Conference on Artificial Intelligence*.

[14] Sharma, N. A. K. U. L., & Yalla, D. P. Issues in Developing UML Diagrams from Natural Language Text. In *Proc. WSEAS Conference, Recent Advances In Telecommunications, Informatics And Educational Technologies* (pp. 139-145).

[15] Ford, L. (1987). Artificial intelligence and software engineering: a tutorial introduction to their relationship. *Artificial intelligence review*, *1*(4), 255-273.