International Journal of System and Software Engineering
Volume 5 Issue 1, June 2017
ISSN.: 2321-6107

Applications of Artificial Intelligence and Data Mining in Optimizing Software Engineering

Upamanyu Chakravarty[1], Rajit Pimpale[2], Raghav Sharma[3], Ramanathan L[4]
[1]3rd Year Student, B. Tech. Computer Science, VIT University, Vellore, Tamil Nadu, India.
E-mail: uchakravarty29@gmail.com
[2]3rd Year Student, B. Tech. Computer Science, VIT University, Vellore, Tamil Nadu, India.
E-mail: rajit.pimpale@gmail.com
[3]3rd Year Student, B. Tech. Computer Science, VIT University, Vellore, Tamil Nadu, India.
E-mail: raghav.sharma204@gmail.com
[4]Assistant Professor (Selection Grade), Department of Software Systems, VIT University, Vellore, Tamil Nadu, India.

Abstract: Software engineering usually deals with a huge amount of data while attempting optimization, consistent cost estimation and other activities. Software engineering uses life cycle models with a number of phases that seem most suitable for development of a particular software. These phases have goals, which when carried out effectively and efficiently, can increase the success rate of the software. This paper focuses on two main ideas which involve the use of data mining and artificial intelligence techniques in software engineering respectively. Data mining techniques such as association, classification and clustering can be used to achieve development goals more efficiently by finding hidden patterns in data. As a result, selection of the appropriate data mining technique for each phase of the life cycle model optimizes the development process. Artificial Intelligence techniques like fuzzy logic, genetic algorithms, neural networks, knowledge bases and machine learning are being increasingly used to improve software development procedures and their functionality. Since a software evolves by exposure to varied environments and expertise of developers, there is a huge scope for applying artificial intelligence techniques to software development and engineering.

Keywords: Software engineering, Data mining, Fuzzy logic, Genetic algorithms, Artificial neural networks.

I. Introduction

Software Engineering is a diverse field concerning the designing and development of software for various electronic devices. This development is associated with a sequence of steps called the software process model or life-cycle model which describes the way the software is developed as a predefined set of steps. There are several process models, each of which have their own pros and cons for different kinds of software. However, the world is evolving at a fast rate and current procedures for software development are slowly becoming obsolete. Thus, there is a need for faster, more efficient software development procedures. As a result, this paper discusses the various evolutions and integrations in the software development process over the years. This paper has divided artificial intelligence into two main factions data mining, soft computing and machine learning. Here, soft computing comprises of neural networks, fuzzy logic and genetic algorithms. Data mining has been found useful in software engineering as a software produces a lot of data. This data, if consolidated and analysed properly, can help programmers in maintenance, predict changes and avoid errors. Since, time and effort is valuable, using data mining techniques can optimize both and thus be cost-effective. Clustering and frequent pattern mining are some of the techniques that are already in use and have been discussed in greater detail in the corresponding section. The section on Soft Computing deals with its integration with software engineering and development so as to improve current development activities. It has been used to improve various phases of the software development cycle by methods such as genetic algorithms (GA), artificial neural networks (ANN) and fuzzy logic. The section on artificial intelligence deals with many of these methods individually. Neural networks are generally learning constructs. In software engineering, they are used to construct learning software that can minimize generalization errors and optimize categorization. It is also used in testing the developed software which is in itself a very time consuming task. Genetic algorithms are mostly used as a software testing technique. Genetic algorithms are evolutionary algorithms that build up and optimize solutions based on natural selection. Genetic algorithms are used to create automated test cases for testing different aspects of the software. Fuzzy logic allows partial participations and acts as a superset of crisp sets. Fuzzy techniques can be used for cost estimation, object-oriented software development and other tasks. It provides simpler models for decision making as it allows more freedom than the traditional two-valued logic constructs. Since software development has inevitable uncertainty, fuzzy logic is inherently capable of dealing with it owing to its allowance of partial membership. Machine learning is used to improve tasks over time through experi-

ence and the same can be applied for software development techniques. For todays world, there are a large number of domains that humans dont have much information about to make an algorithm. For these reasons, machine learning is used so as to construct an algorithm based on past experience. The paper explores recent advances in software development owing to these artificial intelligence techniques and proceeds to evaluate its effectiveness in accordance to current research in a consolidated manner. It provides insight into how these techniques are leading to the evolution of software development by reviewing multiple implementations and concepts proposed. First it reviews some of the general AI integrations with software engineering. Then it reviews soft computing discussing neural nets, genetic algorithms and fuzzy logic in their respective sections. Finally, it discusses machine learning in software engineering.

II. Artificial Intelligence

Software Engineering is a very complex and diverse field that consists a large number of constituent tasks for the development and shipment of efficient software and help people to achieve it faster. These various constituent activities may be highly interdependent. Moreover, software engineering may need the interplay of multiple programming paradigms whose mutual integration may be complex. Both of these aspects can make the development and management of the software that much costlier and harder due to faults and defects. There are also certain aspects of software engineering like risk analysis and testing that are very cumbersome and large tasks to be executed by humans as there are too many alternatives to check. There are many heuristic methods that dont directly work on empirical data and this results in the failure of nearly 20For these difficulties, various improvements have been suggested and researched by the integration of Artificial Intelligence techniques with it. Artificial Intelligence can help current processes be based on more concrete technology and reach a level of detail not attainable by human developers. Software Engineering has seen the use of Neural Networks, Genetic Algorithms, Fuzzy Logic and machine learning. These techniques have been found to be quite efficient and useful for making the current development processes more robust and improve quality of software produced. This section references some research papers on the above-mentioned techniques and discusses some of the current research in software engineering.

A. Current Research in Software Engineering

This paper [1] discusses some of the current research that has come up in software engineering with respect to integrating general artificial intelligence techniques. AI allows current development practices to be based on concrete technology with a higher level of detail than previously possible. Even though such integrations are slow to come and not many in number, they hold great promise. Software engineering has many constituent activities like requirements analysis, project management, source code etc. The first application of AI that it discusses is that of Knowledge-Based systems. Software engineering depends on experts in the field and the tools and methods used currently. These knowledge-based systems use domain modelling that deals with specifying models of real world entities and collectively describe its attributes and relationships with respect to other entities. Domain modelling can describe the associated problem space of a problem. If a software can be seen as a problem to be solved, domain modelling can be used to describe all of its entities and thus be used for requirements analysis with far lower error rates than expert experience which has a hunch factor. The second application discussed is natural language processing (NLP) which deals with the processing of language with semantics closest to ordinary language used by people in everyday life. NLP is highly anticipated as it will revolutionize user interaction in AI. Normal software that depends on constrained language is naturally incomplete and NLP can solve that problem. NLP can be used in software engineering with the view that it will improve user interaction and be more general than current parsers that exist. The third application is data mining which involves the capability of processing large amounts of data to discover interesting patters and associations that can be used to improve development practices. Mining has been used to gain better understanding of several software engineering processes and helps to prevent errors that may occur. Data mining may also help the various changes that may come up during the development of the software and save costs. Clustering and text mining are used often. Clustering deals with an unsupervised method of distinguishing samples and can be used to filter and select data. Its found to be more effective than random sampling. Text mining simply involves the mining of useful information from text.

B. Artificial Intelligence Applications for Improved Software Engineering

Software Engineering is highly dynamic in the sense that software keeps evolving and improves its quality through experience. Moreover, software also evolves as it functions through a changing environment. As a result, every phase of the software life cycle model can be handled by AI methods. This paper [2] discusses some of the main fields in software engineering like Project planning and efforts estimation, Requirements analysis and engineering, and Testing. Project planning involves the planning for the development of the software. It needs to specify the expertise of the employees, requirements specified by the clients, interdependency between the different modules of the software, scheduling tasks, risk assessment etc. It goes hand-in-hand with effort estimation that involves estimating the work required to develop the software. Optimal planning and effort estimation is important for software projects so as to not waste capital and time. In this section, the paper discusses knowledge based systems, neural networks, genetic algorithms and case based reasoning.

1. Knowledge-Based Systems: Software engineering depends heavily on current expert knowledge and a carefully developed set of methods. This knowledge

is important for successfully developing software. Knowledge-based systems can be used to acquire this knowledge and implement it in planning future software projects. Current research says that the cost of developing a high-level knowledge base is very high and unless it can be generalized to multiple projects, its not a very favourable option. Nonetheless, knowledge-based systems have an unparalleled degree of accuracy and completeness.

2. Neural Networks: Neural networks are used in problems that involve classification and prediction. A set of inputs can be provided and a classifying output can be obtained. Such models can be used in risk analysis and predicting errors and changes that come up. Identifying various features in risk assessment and using back propagating networks reached an accuracy of over 80 percent.
3. Genetic Algorithms: Genetic algorithms involve the selection of optimal solution from a population of possible solutions by method of natural selection. It can be used to optimize project scheduling by formulating it as a constraint satisfaction problem. It can also be used to predict very accurate effort functions that work on parameters specified by the particular project. With regards to these two applications, it can create an optimal assignment of staff members for the various tasks and estimate their task duration and minimize development time and cost.
4. Case-based Reasoning: It involves a reasoning method that follows a sequence of nested conditional if-then rules to arrive at a decision. It can be used to plan tasks and estimate their duration. If the project is described in terms of resources, requirements and domain, it can be used with data mining techniques to estimate completion and times. It was found that 85 percent of the 43 projects were completed on time.

   Requirements analysis and Engineering involves defining the domain of the software project and determining the resources required to develop what the client wants. Its the very first step of every software project. However, requirements can sometimes be vague, mutually incompatible and dynamic. Because of these reasons, they are difficult to manage and often lead software projects to failure as a software which doesnt do what its supposed to do is useless. This section uses natural language processing, knowledge based systems and ontologies.

5. Natural Language Processing: The very first step in specifying requirements is to obtain the formal description of the requirements from the current natural language representation of the same. One of the methods to do so is to generate an Object-Oriented specification from the natural language specification. A framework of nine phases was developed that could automatically generate an Object-Oriented model from the natural language requirements. The system was designed to use precise grammars instead of heuristics to identify entities and attributes. This eliminates unnecessary user interaction and gives a functional Object-Oriented model.
6. Knowledge-Based Systems: This simply stores and reuses expert knowledge to improve efficiency. Requirements are identified from the document with over 80 percent accuracy and at a later stage, they are edited, allocated or decomposed. It works by storing design patterns in a system and it proposes a certain design scheme to fully satisfy the clients needs.
7. Ontologies: Ontologies include the specification of concept hierarchies for generalization and conceptualization. On- tologies work like classes in an Object-Oriented model to reuse concepts. It can be used to model and reason about the requirements engineering process with great flexibility and consistency.

   Testing is an intermediate step of the software development process that involves testing the software for its usability and determining faults in it so that it can be improved and made to handle all exceptions. However, bigger the software is, harder it is to find faults in it. Each module may have thousands of cases to test and it is too cumbersome for humans. Thus, for improvement, AI techniques like genetic algorithms have been researched so as to fully automate software testing and determine if its ready for shipping. 8. Genetic Algorithms: These have been used to generate test cases for user interfaces. A sequence of actions can be represented as a gene and a population of these genes represents the possible solutions. Then, a mutation is defined that corresponds to errors and bugs. Natural selection seeks to maximize mutants and thereby, identify erroneous test cases and improve them. It was found that the results are not very stable using this method. Other methods include constructing trees from a dependency graph by genetic algorithms to reduce the search space, determining an optimal order for testing classes and their integration.

C. *Role of Artificial Intelligence in Software Engineering*

This paper [3] outlines the use of fuzzy-based probabilistic model and classification-based learning and prediction. In the past few years, there have been many breakthroughs in the use of AI with software engineering. This paper analyses the development and discusses some of the challenges.

1. Fuzzy-based Probabilistic Models: These models are aimed at modelling real world problems which have too many parameters to keep track of and thus, have a degree of randomness. Probabilistic models have been readily used for such problems. However, many real-world problems become unusually complex with binary logic. As a result, a fuzzy-based probabilistic model is ideal for such problems. Since software involves vague and messy information based on estimates, fuzzy rules can be used to model such problems. Probabilistic models are used to cater for the stochastic behaviour of the user.

2. Classification-based Prediction: There are many fields where interplay of software engineering and machine learning can be fruitful. Using previous data and genetic programming can be used to make an optimal model that learns software behaviour. Thus, machine learning can be used to optimize models used to predict performance.

The second part of the paper discusses some of the challenges of AI and software engineering. The broad challenges have been categorized as

1. Traditionally, AI procedures involved solving problems exactly for a specific instance of a problem instead of generalizing it for a class of similar problems. The challenges here, were to find a strategy as a solution instead of the solutions themselves for specific instances of a problem.
2. Some AI techniques are highly computationally expensive and such algorithms may require a lot of time to come up with an optimal solution. Thus, these algorithms dont scale well with size of problems. As a result, parallel processing can significantly speed up such algorithms. Multi-core computation allows parallel processing and can effectively execute a parallel program. Current systems dont have naturally parallel programming paradigms that can make full efficient use of multi-core architectures.
3. Integration of AI techniques in software engineering is still in the research phase and it might still need time to completely integrate these procedures into industry.

D. Ontologies for Software Engineering

The research [4] explores the use of Sematic Web and the advantages of applying sematic techniques to the software development process. Ontologies specify a concept hierarchy to aid in generalization and conceptualization. Software Engineering has many challenges and there are various procedures that have been developed to cater for those problems. Ontologies have helped to connect and exchange knowledge from Semantic Web and collaboration between fields of research for software engineering.

1. Software Process Ontologies define the life cycle models of software development like Agile process, Waterfall model etc. They define how the development of the software is going to proceed from the first step to the last. Every other task depends on this model.
2. Domain Ontologies describe knowledge pertaining to a certain domain and the relationships that exist between the various concepts that exist in that domain.
3. Requirements Ontologies describes the requirements specified by the customer/client. These ontologies describe both functional and non-functional requirements.
4. System Behaviour Ontologies describe how the system will function under various situations and circumstances. For every action to be taken has a set of conditions to be met, which are also specified by these ontologies. They also specify system status after a certain action.
5. Architecture Ontologies describe the architecture styles of the software system and the relationship between different elements of the system.
6. Pattern Ontologies describe a collection of patterns that are used to create software systems.
7. Implementation Ontologies describe various concepts pertaining to software faults, object oriented concepts like polymorphism and inheritance, and system configuration.
8. Document Ontologies describe all documents related to a software development project and the relationship between each.
9. Quality Ontologies describe quality factors for a software system like consistency, simplicity, robustness etc.
10. Testing Ontologies describe testing factors like identity of tester and properties of testing environment. It also provides information on testing mechanisms for the test being performed.
11. Defect Ontologies describe the defects that are found during testing phase and identifies the person and activity that discovers and rectifies it.
12. Maintenance Ontologies define maintenance type, related activities and modifications done during maintenance. It also specifies concepts on the person who maintains it and the procedure for maintenance.
13. Technology Ontologies is a collection of software development methods and tools that can be used for the development of a class of software.
14. Thus, nearly all the procedures of software development have a related Ontology that specifies concepts related to it. These ontologies can be utilized for all these phases and being generic in scope, can be used in any project. Ontologies have been mainly used in web technology but it has potential opportunities in software engineering.

III. Data Mining

Data mining has been used over the past decade in order to learn more of human behaviours, by extracting valuable information from huge databases stored either automatically or manually. Software Engineering has a lot of important data in the forms of bug logs, project schedules and reports that can help developers starting with a custom project to overcome problems that were already faced and solved by someone before.

Mining Software Engineering Data: While using software engineering techniques to choose a model for a project, there are high chances that a model may not fit according to the requirements for a project. But there are greater possibilities of the required model being tried, tested and implemented before.

To make this knowledge accessible, Xie et al [5] propose in a tutorial using data mining techniques to extract this data from previously completed and successful projects. This includes mining for software engineering data, like chronological code changes, bug logs, code-bases and execution methods and thus charting the problems and evolution of the project. With this, practitioners can explore the valuable data, working on the experience of those before. The tutorial paper breaks this down in several parts: first, the software engineering data sources to be mined. This deals with the various logs and fields that hold the most valuable information, and mention those that are not valuable as well. Second, they discuss the tasks in the project that are assisted by mining software engineering data, addressing technical as well as managerial problems. The various types of tasks include programming methods, defect detection, testing, debugging as well as optimal maintenance techniques that can be employed to finish the project smoothly. Third, the various mining techniques being used specifically for software engineering data are shown. The techniques of classification, pattern association, clustering and learning are used to summarize logs to find patterns. Apart from these, the tutorial discusses the difference of software engineering data mining from other scientific mining, briefing about the current problems in the field.

Applications of Data Mining in Software Engineering: A paper by Taylor et al [6] gives an in depth description of the use of data mining in software development processes. It describes the data mining methods used by large firms and organizations to maintain and use software engineering data, while showcasing the difference between projects that use software engineering methods and those that dont, highlighting its importance in large and small projects alike. Larger organizations employ a system for tracking and fixing defects and bugs and have professional revision control softwares. The major data mining techniques are also described. Association rules and pattern matching are shown to suggest probable future changes, avoid problems because of unfinished changes and to detect unobvious coupling skipped during program analysis. The bug fixing problem is tried to be solved by the Classification method, where it is determined what sort of bug might possibly cause the given input of unexpected outcome. Clustering is employed to pin-point program failure diagnosis and localization. There is also a method mentioned by Runeson et al [21] that is less explored but has vast applications. Text mining is said to have been utilized to fix duplicate bug reports, to figure out incorrect code with correct comments and correct code with incorrect comments. Benefits of data mining are also discussed in the sub-fields of development, management and research, while finally also mentioning that data mining can also cause significant drain of resources if done incorrectly or too extensively.

Software Intelligence: The Future of Mining Software Engineering Data: Hassan, along with Xie gave a new concept of Software Intelligence in their paper [7], which is aimed at giving techniques and concepts to improve business decisions through fact based support systems, supporting not only developers but software practitioners as well. The concept proposed is to be used to answer daily questions in the software development life cycle, including progressive steps, release dates etc., quantifying the factors on facts rather than an intuition. The field of practice of SI is supposed to aid software practitioners, developers and mangers alike primarily to support decision, not to replace it. The SI is being developed and trained by mining open source code repositories and bug repositories which are properly structured thus allowing automatic analysis. For further training, it is meant to mine communication and deployment logs as well in order to be of more assistance that just to code. In conclusion, the paper suggests implementing SI in projects for better efficiency, potentially augmenting the tasks to the software and only to help explain, but not to replace actual developers or managers.

Proposed Application of Data Mining Techniques for Clustering Software Techniques: Here [8], the author proposes the applications of data mining for clustering software techniques. The information can be extracted from the source code repositories or texts in documents by software engineering process. Through that information, knowledge can be obtained by applying different data mining techniques like clustering, classification, etc. as explained in the above paper. That can be used to support software engineering processes like testing, error detection, programming, etc. Clustering is defined as a technique in which an object is grouped with other objects to form a cluster so that the objects within a cluster have high similarity among them whereas they are very different from those in another cluster. Among the best known clustering methods is k-means clustering, wherein n objects are divided into k clusters (k¡n); each cluster is represented by the average value of the objects in that cluster and the value of the cluster is the value of the object that is closest to the center of the cluster. To execute the clustering methods we use open source softwares, like Weka. Here, we need not worry about the type of data that we are inputting because the algorithms transform the data to the required form automatically.

Applications of Data Mining Techniques for Improving Software Engineering: Here the author of this research [9] talks about the different data mining techniques that could be used for improving software engineering. According to the authors, increasing the complexity of the software may lead to poor construction resulting in bugs. The use of iterative mining in overlapping mining techniques can reduce the noise in the data. Data can be collected from various sources, following which it is separated on the basis of increasing complexities and software reliabilities. The data is analyzed and the best data mining technique is thus selected for the SE problem. The data can be divided into three categories namely, graphs, sequence and texts. In sequential data mining, the data to be mined from the source code is divided into smaller classes, tags, blocks so that it could be further mined. Each unit of source code is accessed separately and the wrong source code is separated from each block. Graph data mining methods are mainly used for non- crashing bugs. Non-crashing bugs do not terminate the program but may cause errors in program output

or its execution. Text data mining methods, which happen to be most used, as 80 percent data is of textual form. In these a bug report (BR) is generated. This is further classified into two types, security bug report (SBR) and non-security bug report (NSBR). Depending on several checks, the probability that the BR is an SBR is being found out. These mining techniques have proven to be effective in improving SE quality and reliability.

A Data Warehousing Engineering Process: The author [10] describes the process of data warehouse (DW) construction. Data warehouse is a non-volatile, integrated, structured collection of data. The goal is to design an efficient DW and it can be created by object-oriented (OO) method based on Unified Modelling Language (UML) and the Unified Process (UP). To create an efficient DW, it should fulfil the some premises like it should be based on a standard modelling language, it should provide simple and lucid method for developing a DW, it should be well defined, etc. A DW consists of various layers of data where each layer follows another layer of data and each layer being overlapped. DW can be subsequently analyzed, designed and implemented by the above given premises. To check that the implementation works fine, there is no need to create new diagrams but the old diagrams may be modified so as to obtain the correct output.

IV. Soft Computing

Computation problems and their associated algorithms are often segregated into hard computing and soft computing techniques. Where hard computing focuses on finding exact optimal solutions to problems, soft computing deals with finding approximate near-optimal solutions for complex problems. Its usually classified into neural networks, genetic algorithms, fuzzy systems and hybrid systems that use a combination of any two or three. Each of the three have been included in a section with their corresponding references.

A. Neural Networks

(1) *Artificial Neural Networks for Software Effort Estimation:* Software effort estimation is one of the first steps in a software development process. Due to the dynamic nature of software systems in todays world, effort estimation becomes increasingly complex. As a result, the effort estimation function is modelled as a set of relationships between a dependent variable and other independent variables. Artificial neural networks can be used to model this set of relationships and give a near- optimal prediction model. Effort estimation aims at reducing cost and time of software development while improving its quality. Models like COCOMO are used for effort estimation. Neural networks are modelled after the human nervous system. It consists of neurons in a specific network with connections of varying strengths between each pair of neurons. Input signals fed across a layer of neurons propagate through the network and yield a prediction output. Comparisons between different kinds of networks show that radial basis networks give the best performance, followed by multilayer networks. The paper discusses COCOMO and its performance with respect to neural networks. Most of AI integration with software engineering has focused on project planning, project testing and software measurement. This topic outlines how theres limited research regarding testing efforts of software systems. Software effort estimation has a number of qualitative aspects and cannot be regarded as an exact science. The paper involves the use of multilayer feed-forward neural networks working on a modified back propagation algorithm. Project factors are identified by data analysis and fed into neural network as input to acquire duration of project as output. The paper [11] discusses the use of a neural network to model a small problem and evaluate it. The results showed that neural networks performed a lot better than current estimation techniques.

(2) *Software Measurement using Artificial Neural Network and Support Vector Machine:* The process by which numbers or symbols are being assigned to attributes of entities in a real world so as to characterize them is called measurement. Software measurement is the process which helps an organization to improve a measurement program. Measurement requires entities, attributes and rules to start with. Software measurement can influence the development effort, it also influences the development time and cost, as all these parameters are closely related with each other. By several approaches we can find the software measurement. Some of them are discussed here [12] namely Artificial Neural Networks (ANN), Support Vector Machine (SVM), Radial Basis Function Optimization (RBFM). The artificial neural networks are very similar to the biological neural networks. In these, they have two outermost layers, to process the input and output, namely the input and the output layer. In between these two layers lie one or more hidden layers which link the two outermost layers. These are called hidden output layer weights. Here, each neuron is connected to other by connection links, sending inputs to each other. The weights contain information about the inputs, which is used by the neurons to solve the problem. The SVMs are innovative learning machines that minimize generalization error. In these machines, data is being separated by the construction of (hyper) planes. A kernel function is defined, which maps the problem to a higher dimensional space where the planes are sufficed to define the boundaries.

B. Genetic Algorithms

(1) Application of Genetic Algorithms in Software Testing Techniques: Software testing in software engineering deals with deploying reliable software that can handle exceptions, ensure quality of the service it provides. However, nearly all current software systems are so large and varied that generating

sufficient test cases to test the functionality of the software in every scenario is highly expensive and time-consuming. Due to this, many software systems being deployed to the market have a huge number of bugs and ultimately fail. This paper [13] discusses the use of genetic algorithms in software testing. Software Testing must identify errors that have amassed through all the other activities in the development process. If software testing is an optimization problem to minimize cost, time and number of test cases, it could be solved using genetic algorithms so as to improve quality of software. Software testing is done in two phases called black box testing that tests functionality and white box testing that tests internal structure.

1. Control Flow Testing: It involves identifying the critical path clusters in a program to generate test cases using genetic algorithms. It searches a suitable test case that covers every possible path in a weighted graph called the Control Flow Graph. This is used in white-box testing.
2. Data Flow Testing: Uses an iterative method by constructing paths from previously generated paths evaluated as tests, where each iteration divides the population into two parts and evolved with two different techniques and recombined in the updated population. This is also used in white-box testing.
3. Functional Testing: It involves the generation of test cases. Genetic algorithms can generate improved test plans even though the best plan is unknown. This is used in black box testing. Neural nets can be used to create a model and the output of the neural network can be fed to a genetic algorithm so that test cases can be generated from output domain.

(2) Software Testing using Genetic Algorithms: The process in which runtime quality and quality of a system is tested to maximum limits is called software testing. So we often give some test inputs to the software to check its functionality. We use algorithms, known as genetic algorithms, which generate automatic test case data for software testing. Genetic algorithms offer a good solution in many cases, even in those where very less information is given, as is presented in this paper [14]. They work very well in any search space. The principle of selection and evolution is used by genetic algorithms to solve many complex problems. Genetic algorithms create an initial set of random solutions (population) and the evaluations are done on them. The terms used by genetic algorithm are borrowed from the biological world. The potential solutions are known as chromosomes and the name of processes are derived from nature. Genetic algorithms use following three operations in its working: selection, crossover and mutation. By selection, we can choose individuals for mating from a population based on their fitness. Following selection, the crossover process is applied on the chromosomes selected from the population. It involves swapping of bits or genes between two individuals. After the crossover process, mutation is applied which leads to altercation of chromosomes. This usually results in the formation of good traits and diversity. Genetic algorithms are of major use in software testing. Manual testing is usually not efficient enough, the speed of testing is limited, the cost factor is high, and there is high redundancy involved in test cases. Also the test checking is inefficient and inaccurate. These problems can be avoided, all at once by using genetic algorithms.

(3) Automatic Software Testing by Genetic Algorithm Optimization: A Case Study: While making software, the main aspects that the developer wishes to fulfill are the quality and reliability of the software. There should be a minimum quality in any case. So a developer can check for the quality of the software by two ways, either he can give the test inputs manually or the inputs can be fed automatically. The problems with manual inputs are many; they can take high efforts, increase further cost, and be redundant. So usually we prefer the testing of the software automatically, via genetic algorithms, which being low on effort and cost consumption, can solve problems on different scales, be it easy or highly complex. The methods in this algorithm of this paper [15] are based on recombination of genes; quite similar to those in the nature. GA, being robust, can be more or less suitably applied to most complicated optimization tasks. Any software may be syntactically correct, but it may have unknown number of semantic errors. These can be searched and removed by running the programs using some testing methods. The software testing using GA can yield encouraging results for the user and therefore can cut down 30-50 percent of the total cost of the software. The GA based optimization can be brought into great use while developing software testing automation. However, for the automated testing methods there should be presence of highly trained developers and users.

C. Fuzzy Logic

A Fuzzy Logic Based Software Cost Estimation Model: Incorporating fuzzy sets in estimation models, Ziauddin et al [16] challenge the existing methods of models of software cost estimation methods. Considering the existing models, it is implied that most of the critical estimation by earlier models was done by matching required project model analogically to an existing one, but most often that process is not able to handle the customizations precisely. Newer models used regression analysis as well as numerical derivations. This paper suggests using fuzzy logic in order to estimate costs more accurately. The proposed model works with COCOMO 2, a cost estimating model that uses regression to find the estimates. This utilizes several software attributes including effort multipliers, scale factors, software size in order to give an idea of the effort, schedule and the personnel required for the project. Initially, a crisp set of input is fuzzified separately for each input type into categories very low, low, medium, high and very high, for a common three point membership function. The fuzzified input are then run through an inference engine, which contains predefined rules to give an output for a specific combination of inputs. This is essential because the inputs are

not numerals, but linguistic values. The output obtained after this step is defuzzufied using the centroid method to calculate the centre of gravity under the area of the graph. This gives a certain sharp output in spite of having used fuzzy input set. The model proposed in this paper was found to be better than the original COCOMO model as well as the Alaa Sheeta model to find the actual count of effort. A comparative test found the mean magnitude of relative error to be at 7. Even though the other models may produce slightly more accurate results in some of the cases, the proposed model never gives a huge margin of error with the actual effort, thus concluding fuzzy logic is effective in estimating software costs.

Applying Fuzzy Logic Techniques in Object-Oriented Software Development: Marcelloni et al [17] gave a paper addressing the quantization problem in software development where processes cannot always have two valued logic. Their paper suggested an object oriented fuzzy logic technique for development. Object oriented methods help make adaptable, robust and reusable software systems, while fuzzy logic gives new perspectives to software development like additive software life cycle. The quantization problem, as described in the paper, is explained through an example of an entity object having membership to a class. A problem faced might be that the object faced is only partially the member, but the software engineer is forced upon his perspective bias to give a value among the two logic values. The solution proposed is similar to that of the previous paper discussed, where the membership is sub-divided into more levels like weakly, slightly, fairly, substantially, strongly and extremely, which are crisp sets themselves, but make the transition smoother rather than sudden, thus not forcing the engineer to use his bias. Another problem faced, and which is discussed later in this paper as well, is the contextual bias problem, where the perspective bias of the engineer developing the software may create biased software that may not be useful for all the user bases. This again is partially solved by having higher quantization levels, which reduces the effect of contextual factors on the final output. Further, incorporating more of linguistic values rather than numerical ones to increase the degree of vagueness which is required in some of the software modules is also a solution provided in the paper.

Use of Fuzzy Logic in Software Development: Almost the same issue is discussed in a paper by Shradhanand et al [18] as well, which shows the quantization drawbacks and the need for vagueness in the uncertain field of software engineering. They show the importance of linguistic variables in project progress, but also extrapolate that same logic to the fields of project management and software testing. Project managers who use the numerical inputs face the problem of getting an overestimate or underestimate of commitment by the development team at the beginning of the project due to the non-feasibility to provide any amount of confidence about the various factors of the development process. Fuzzy logic is suggested in these situations where instead of keeping an aim to meet the goals, the aim is to keep making the fuzzy inputs progressively crisper as the project advances. The initial estimate may be taken by going through previous similar projects, thus making it easier for both the developers as well as the manager to complete the project efficiently. The paper goes on to say that the area of software testing is inherently uncertain due to multiple reasons, one main reason being that exhaustive testing cannot be computationally practical for any environment except extremely simple ones, and thus the test input set is to be a finite set of elements in a finite number of environments, thus introducing uncertainty. It is also said that testing may introduce even more error in the results due to the observation, much like the famous Heisenberg Uncertainty Principle.

Practical Application of Fuzzy Logic from Software Engineering Point of View: Among all the papers that give the advantages of using fuzzy logic, a paper by Sram [19] displays the risk of utilizing fuzzy logic without proper knowledge of it in the projects, among other necessities of using it. While fuzzy logic allows the developers to focus on the decision logic of the algorithm, the fuzzy architecture needs to be designed in a bottom up fashion. The risk here is that if the managers and other experts modelling the software have no knowledge of the working of fuzzy sets, it can result in quite some issues. Furthermore, the current implementation tools are not designed for fuzzy logic, while the current developers are used to the pre-existing technologies. If fuzzy logic is still implemented, the entire development team has to be soundly aware of the concepts, and if not, might result in additional time spent to learn it. Even modern softwares that are made with keeping the masses in mind face problems while scaling a project with fuzzy values.

V. Machine Learning In Software Engineering

Machine learning handles the problem of making programs and constantly betters their performance through experience. This learning can mostly be used in technologies in which the problem domains are not understood properly or we have a large amount of data with patterns to be discovered and domains where programs must adapt to changes effectively. The main challenges are developing software systems dynamically adapting to various changes in the environment, the second challenge is that the domains where to apply this are very poorly understood i.e. there might be a lot of raw data but there is no effective way to develop a good algorithm. Although there might be many ways to address these issues, this paper [20] states transformational programming is the best technique where software is made, improved, and maintained according to specification level and is converted to production-quality software using automatic program synthesising and this kind of development will allow software engineering to obtain knowledge of the unknown domains. Some of the important types or class of Machine learning algorithms are concept learning, decision trees, neural networks, reinforcement learning, genetic algorithms, instance based learning, inductive logic based learning and analytical learning. Many of these methods can be and are being used in developing better tools

or software products in many areas such as requirement engineering, software reuse (application generators), testing and validation, and in project management (cost, effort prediction or estimation).

Component Reuse: In reusing software, it is very important to find and extract relevant components from pre-existing programs. This problem can be converted into the form of an instance based learning problem

1. Components are represented in the n-dimensional Euclidean space.
2. Information in a component can be divided into indexed and unhindered information, where they are used for retrieval and contextual purpose respectively.
3. Queries to the repository can be represented as constraints on index able attributes.
4. Retrieval is over using K-nearest neighbour, inductive retrieval, and local weight regression.
5. The adaption can be structural or derivational in nature.

Rapid Prototyping: This is an important tool for understanding and validation of software requirements. In addition, software prototypes can be sued for many purposes such as training and system testing. These techniques can be augmented by including a machine learning approach such as genetic programming.

Requirement Engineering: This refers to the process of setting up services that the system should be able to provide and rules which it must follow. ML algorithms are particularly useful when data from the problem domain exists already and it is known how the system should react to certain inputs. Under this circumstance, requirements can be learnt from the data

Validation: Verification and validation are very important tasks in the life cycle; to check software with its specification we use an analytical leering algorithm.

Software Defect Prediction: Software defect prediction is very important tool in software engineering now we need a holistic model rather than a single-issue model that hinges on size or complexity.

Bayesian Belief Networks: are used it represents joint probability distribution for a given set of variables which could be values for a constraint. A BBN can be used to infer the probability distribution for a target variable. So, the outline is to define the variables define relationship between the variables and finally obtain the probability of the defect occurring.

Project Effort (cost) Prediction: Estimating cost in developing a project is done in many ways. Most of the existing work is based on algorithm models of effort. A viable alternative approach is to use instance based learning.

VI. Conclusion

Thus, these papers give an insight about the Software engineering process using various data mining and AI techniques. We learnt that while using data mining techniques like clustering, classification, management, etc. we can improve the quality and reliability of the software, cut down costs and efforts, while using AI techniques like artificial neural networks, fuzzy logic, generic algorithms we can make our software free of bugs and errors. However the testing of the software has to be carried out with highly trained developers and users so as to avoid any software crashing. These evolutionary techniques have greatly helped in software testing. They have been efficient even on increasing the number of test cases and the amount of data. The data can be easily processed and the knowledge from it can be extracted easily, regardless of its type. In the coming time we can expect more progress in these fields resulting in more automation and simplicity.

VII. References

[1] M. M. Ahamed, "Current research topic In software engineering," 2013.

[2] F. Meziane, "Artificial intelligence applications for improved software engineering development: New prospects," *IGI Global*, 2009.

[3] M. Harman, "The role of artificial intelligence in software engineering," *In Proceedings of the First International Workshop on Realizing AI Synergies in Software Engineering*, pp. 1-6, 2012.

[4] M. P. S. Bhatia, A. Kumar, and R. Beniwal, "Ontologies for software engineering: Past, present and future," *Indian Journal of Science and Technology*, vol. 9, no. 9, pp. 1-16, 2016.

[5] T. Xie, J. Pei, and A. E. Hassan, "Mining software engineering data," *IEEE Computer Society in Companion to the proceedings of the 29th International Conference on Software Engineering*, pp. 172-173, 2007.

[6] Q. Taylor, C. Giraud-Carrier, and C. D. Knutson, "Applications of data mining in software engineering," *International Journal of Data Analysis Techniques and Strategies*, vol. 2, no. 3, pp. 243-257, 2010.

[7] A. E. Hassan, and T. Xie, "Software intelligence: The future of mining software engineering data," *In Proceedings of the FSE/SDP Workshop on Future of Software Engineering Research*, pp. 161-166, 2010

[8] H. R. Rezende, and A. A. A. Esmin, "Proposed application of data mining techniques for clustering software projects," *INFOCOMP Journal of Computer Science*, vol. 9, no. 6, pp. 43-48, 2010.

[9] W. Husain, P. V. Low, L. K. Ng, and Z. L. Ong, "Application of data mining techniques for improving software engineering," *In ICIT 2011 the 5th International Conference on Information Technology*, 2011.

[10] S. Lujn-Mora, and J. Trujillo, "A data warehouse engineering process," *In International Conference on Advances in Information Systems*, Springer Berlin Heidelberg, pp. 14-23, 2004.

[11] D. Santani, M. Bundele, and P. Rijwani, "Artificial neural networks for software effort estimation: A review," *International Journal of Advances in Engineering Sciences and Technology*, vol. 3, no. 3, pp. 193-200, 2014.

[12] B. K. Sinha, A. Sinhal, and B. Verma, "A software measurement using artificial neural network and support vector machine," *International Journal of Software Engineering Applications*, vol. 4, no. 4, pp. 41, 2013.

[13] V. Sangeetha, and T. Ramasundaram, T., "Application of genetic algorithms in software testing techniques," *International Journal of Advanced Research in Computer and Communication Engineering*, vol. 5, no. 10, pp. 2278-1021, October, 2016.

[14] A.Sharma, P. Rishon, and A. Aggarwal, "Software testing using genetic algorithms," *International Journal of Computer Science and Engineering Survey*, vol. 7, no. 2, pp. 21-33, 2016.

[15] J. T. Alander, and T. Mantere, "Automatic software testing by genetic algorithm optimization: A case study," *In Proceedings of the 1st International Workshop on Soft Computing Applied to Software Engineering*, pp. 1-9, April, 1999.

[16] Ziauddin, S. Kamal, S. Khan, and J. A. Nasir, "A fuzzy logic based software cost estimation model," *International Journal of Software Engineering and its Applications*, Vol. 7, No. 2, pp. 7-18, March, 2013.

[17] F. Marcelloni, and M. Aksit, "Applying fuzzy logic techniques in object-oriented software development," *In European Conference on Object-Oriented Programming*, Springer Berlin Heidelberg, pp. 295-298, June, 1997.

[18] A. K.Shradhanand, and D. S. Jain, "Use of fuzzy logic in software development," *Issues in Information Systems*, vol. 8, no. 2, pp. 238-244, 2007.

[19] N. Sram, "Practical application of fuzzy logic from software engineering point of view," *Obuda University e-Bulletin*, vol. 2, no. 1, pp. 285-291, 2011.

[20] D. Zhang, "Applying machine learning algorithms in software development," *In The Proceedings of 2000 Monterey Workshop on Modeling Software System Structures*, pp. 275-285, June, 2000.

[21] P. Runeson, M. Alexandersson, and O. Nyholm, "Detection of duplicate defect reports using natural language processing," *In Proceedings of the 29th International Conference on Software Engineering*, pp. 499-510, 2007.