Software Development Automation: An Approach to Automate the Processes of SDLC

A.R.V. Anthony
Sri Lanka Institute of Information Technology
Sri Lanka

G.M. Dilshan Prasad
Sri Lanka Institute of Information Technology
Sri Lanka

S.U. Randunuge
Sri Lanka Institute of Information Technology
Sri Lanka

S.R.A.M.P.A. Alahakoon
Sri Lanka Institute of Information Technology
Sri Lanka

Dinuka R. Wijendra
Sri Lanka Institute of Information Technology
Sri Lanka

Jenny Krishara
Sri Lanka Institute of Information Technology
Sri Lanka

ABSTRACT
Software development complexity is one of the most important factors that must be determined by clear procedures or methods in software production. It is determined practically by using a quantitative value, which is based into one or more qualitative attributes. These attributes focus on how the codes' internal and external behavior. But the software complexity should be computed beyond, that level since the complexity identifies the effort of determining the internal logic behind the software. Therefore, software complexity should be expressed as a combination of the different phrases of the software development life cycle namely requirement analysis, source code implementation, maintenance, testing and quality checking as well. As a solution, the methodology of reducing the overall software complexity by creating a software development application has been considered, which will automate the requirement analysis, software logic implementation, maintenance and the testing process in the overall software development cycle without restraining the software complexity into one or more quality attributes.

General Terms
Deep Learning

Keywords
Natural Language Processing, Code complexity, UML generation

1. INTRODUCTION
Businesses that revolve around software and software development are very common in the present day. Almost every company has their own mobile application, web sites or internal system. People use those services to get through with our day-to-day life, make our daily work easier. More and more consumers of these software and services are rising and with that more complex and competitive the businesses get. Time it takes to develop a software, the quality of it and the cost will play big roles for the success of these businesses. Software development is not that simple so to achieve high quality and low cost with low time is very hard. The proposed system is used to make the software development process less complex and easier for developers go through several stages of software development life cycle. There are main four components in this proposed system that is used to reduce the complexity of requirement gathering, software logic designing, locating errors and maintenance. Therefore, the proposed system will reduce the overall complexity of software development and help develop high quality software with less human effort.

2. METHODOLOGY

2.1 Generating the internal logic according to the requirements
The main purpose of this component is gathering the user requirements and analyze them to build up a logic to move on with the implementation by targeting the customer requirement fulfilment. When consider about the predictions, they can be divided in to three categories as class identification, relationship between classes identification and the rough prediction on time range for the development of the software.

The Project Proposal document is the main input for this functionality, which specifies the problem caused the necessity for the development of the required software and the expected benefits, outcomes by the customer's point of view. Since the acceptance and the rejection of this document is not confirmative and the document does not follow any formal language techniques or templates, a prediction based on the data which collects from this document, a technique which is more specific for this task must be used. So, by considering these facts the best solution which can used to solve this problem is Machine Learning technique. Since this is a deal between human language and computer, the best technique which can be used to achieve these tasks is Natural Language Processing (NLP), which is very specific area of the Machine Learning. Part of Speech tagging (PoS tagging) is an effective technique available in NLP, which helps to identify the root forms of the words in a document. In programming, there is a special technique to identify the classes. According to that technique there is a huge possibility for a noun to become a class in the implementation. The same technique used for this purpose, therefore the first step of predicting the class names is the identifying the nouns in the functionality area of the document. So, the methodology of that process is tagging each word of the document by using the Pos tagging technique, and to perform that a machine learning model is required. For this functionality, a model is used which is implemented by using Python and specially an open-source library which is known as TensorFlow. First, the model is trained by using some test word sentences and test tags, then after that the through java front end the project proposal document passes to this model

as an array of words. Then, using this tagging technique each word in that array is tagged according to their root form, then by analyzing those root forms the nouns in that array can be identified as the classes in the implementation. After that those nouns can be added to a separate array.

Arr_nouns = [*word1*, *word2*, *word3*, *word4*, ...]

Arr_words = [*word_i*, *word_ii*, *word_iii*, *word_iv*, …]

By doing this task the two arrays can be compared to identify the indexes of the nouns in the Arr_words array then the logic to predict the relationship between two classes can be done by extracting the words between those specific two indexes of the nouns. Then the time series forecasting is going to be used for the prediction of the time range for the implementation. For this process number of development teams and the number of members in a team are going to be the main predictors. To train this model a data set which consists with those predictors are going to be used, the model builds by using the python language. The most important thing is the time series must be stationary all the time, in other words the mean and the variance must constants.

First step of this process is reducing the trend, transformation is used for this purpose. since there is some noise in the trend it should be smoothed, moving average is used to smooth the trend. to build the model differencing technique is used. The technique which use for this prediction making is known as ARIMA (Auto Regressive Integrated Moving Average) when using this model the predictions depend on ARIMA parameters (p,d,q) p stands for number of Auto Regressive terms, q stands for number of Moving Average terms and d stands for number of differences. To identify the values which are suitable for p and q use two popular plotting techniques, such as Auto Correlation Function (ACF) and Partial Auto Correlation Function (PACF). Three prediction models are going to build when using this technique, they are Auto Regressive (AR) model, Moving Average (MA) model and the model that shows the number of differences.

Then after building these models predicted values can be stored as a series and convert the differences to a log scale, then the model can plot the predicted value with the original value.

2.2 Automation of mapping requirements into UML diagrams

UML diagrams play an important role in the developers' level of understanding of the software to be developed. Client requirements which are well understood would lead to a high-quality final product. The success rate according to [17] is only 28%. Understanding the requirements of the system clearly and properly will be crucial in improving the success rate of a software solution. Even though developing UML diagrams is important for a software project, it is a time-consuming procedure. According to [18], a major reason for this process to consume a large amount of time is collection of the requirements and mapping them into diagrams is done manually in many software companies. High time consumption can distract the developers from the focus of developing code. The time to market of a software product could be reduced due to the large amount of time spent for creating UML diagrams for their better understanding. Therefore, automating the procedure of generating diagrams by analyzing the clients' requirements could solve the problem of high consumption of time.

Firstly, an array of class names, which is an output of the first component is used as an input to the machine learning model. This machine learning model analyses the class names and understands the relationships between classes. For the analysis, Global Vectors for Word Representation (GloVe) [14] embeddings are used. This unsupervised learning algorithm maps a word into a vector representation, which can be used to measure the similarity of class names. The mathematical concept of Cosine Similarity was used to calculate the similarity in cosine values of the angles made by vector representations of each word. The cosine similarity between two vectors j and k with respect to a unit vector i can be written by using Eq. (1), which explains the cosine similarity is directly proportional to the angle between the two vectors in question [1].

$$\mathrm{sim}(j,k) = \frac{\sum_{i=1}^{n} j_i k_i}{\sqrt{\sum_{i=1}^{n} j_i^2}\sqrt{\sum_{i=1}^{n} k_i^2}} \qquad (1)$$

By using cosine similarity, the similarity, or the closeness of two words (class names) in the vector space can be calculated as a quantitative value. The cosine similarity value corresponding to each class name pair in the input array is added to a new array with the class name pair. This output is then feed to a Java program to generate the diagrams.

In this program, the cosine similarity value is checked against ranges, and for each range a tag explaining the relationship level between the classes is assigned. The input array is altered by replacing the cosine value in the array by the tag. The tags are "inherits", "implements", "composes of", "aggregates", "associates" and "no relation". The "no relation" tag is given to relationships for the least cosine similarity range.

Then using these tags, the program creates different diagrams by mapping the relationships. PlantUML [2] is used for the creation of diagrams and Graphviz [3] is used to visualize the created diagrams. The application generates four types of diagrams namely Class diagrams, ER diagrams, Object diagrams and Flow diagrams. Flow diagram is a diagram, which illustrates the flow of the software in terms of classes.

2.3 Tracking bugs using a bug tracker

The internal logic and the possible flow diagrams generated by previous functionalities are then referred by the software engineers to implement the source code. Before making the product into its finalized version, the quality and its accuracy must be ensured to increase its stability. Thus, this functionality includes a bug tracker, which identifies the errors and unethical programming practices that software developers have practiced. The importance of tracking bugs is shown in [15].

The main problem of the available bug trackers is [16], not focusing on all the possible bugs. It is not a good user experience if the bug tracker does not show bugs for all the errors that developers make during the coding phrase. Some applications do not show any error message in the error log which can be considered as a negative performance criterion. But, in this component it can be clearly shown that most of the errors are identified, which are not currently identified by the available bug trackers as shown in TABLE I.

TABLE I: Bugs that are focusing by the bug trackers

Bug category	Whether it finds the bug or not
Possible deadlock	Yes
Unreachable code due to constant guard	Yes
String equality with.equals()	Yes
Object overriding	Yes
Stream not closed on all paths	Yes
Unused local variables	Yes
Unnecessary return statements	No
Division by zero	No
Possible unexpected errors	No

It can be clearly observed that unused local variables, unnecessary return statements, division by zero and unexpected errors are not identified by the current applications so that more focus is given to resolve those bugs with the concatenation of the other bugs tracking procedures as well. The user interface has the option to browse the files and then track the bugs by selecting the "track bugs" option. Then it will show the bugs if any of the bugs occurred in the browsed code. It has the ability of showing the code and the error log of it. When selecting the error, it shows the recommendations to the solutions so that the user can easily fix the error. To find the bugs, it is given some conditions to track the errors of the code. Then, it analyzes the code and checks the condition of the similarity. If they mismatch, the system will show errors. The following section describes how the bugs have been discovered.

2.3.1 Multiple return statements

This error occurs when it has many return statements in a single method. It will be detected with their brackets and the count of the return statements. If the count is more than one, the system will track it as an error.

2.3.2 Array storing error

This error occurs when user is trying to store incompatible values with the datatypes of the array. It will be detected by comparing the datatype and the passed values to the array. If any passed value is incompatible with the datatype, the system will track it as an error.

2.3.3 Illegal states error

This error occurs according to the order of calling the next(), set() and remove() methods in an linked list iterator. TABLE II shows the possible combinations and their possibility of being an error or not.

TABLE II: Possible error combinations

Combination	Error / Not an error
next(), set(), remove()	Not an error
next(), remove(), set()	Error
set(), next(), remove()	Error
set(), remove(), next()	Error
remove(), next(), set()	Error
remove(), set(), next()	Error
next(), set()	Not an error
set(), next()	Error
next(), remove()	Not an error
remove(), next()	Error
remove(), set()	Error
set(), remove()	Error

When implementing the procedure, it searches for the order of the occurrences of the methods. For any mismatched error, it will show an error in the error log.

2.3.4 Illegal thread states error

This error occurs when user tries to start the same thread multiple times. Thread name and the occurrences of the start() method for that particular thread are checked and if there are multiple occurrences, the system will track it as an error.

2.3.5 Division by zero

This error can be occurred when there is a division by zero. This error can be risen in two ways. One possible way is directly divide a number by zero. The other way is assign zero to a variable and then divide a number by that variable. When implementing the first option, the division mark and the value of 0 are checked. If it is found in any code, the system will track it as an error. When implementing the second option, it checks the variables assigned to 0 and if there is any division occurred by using that variable. If so, it will track as an error.

2.3.6 Number formatting error

This occurs when incompatible values are passed to the parse methods. First it checks whether there is any parse method occurred, if so, it will check the parameter which has been passed to the parse method. If there is any incompatible value like a string or integer or null, it will track as an error by the system.

2.3.7 Illegal arguments error

This error occurs when null parameter is passed as a parameter in any method. The method structure and the number of parameters is checked. If any parameter has set to null, it will track as an error by the system.

2.4 Evaluate the Code Quality and calculate the Code Complexity

There have been an increase in the amount of research conducted on software development related to code quality improvement and how code complexity calculations have impact on the quality of the codes and the impact at maintenance stage of software development. As the other functionality, the system involves identifying quality issues of codes and provide optimal fixes or recommendations to fix those issues and provide the users the ability to use different metrics to calculate complexity of the code.

In order to detect issues and calculate complexity system needs to identify syntaxes and elements of the given code. For this identification process the code element identification component is used which is a component that was developed for this system. In this component, code will be taken as the input for the system and it will be analyzed line by line in order to identify code elements. In this process all the identified classes, variables and objects are stored temporarily for further use of code complexity calculation and quality analysis.

2.4.1 Complexity calculation component

There are six complexity metrics currently available for the code complexity calculation component.

- Complexity due to Line of Codes - Used to calculate code complexity due to the actual code lines [4] and provides other information such as the amount of comments and percentage of comments compared to actual codes.
- Complexity due to Size – Used to calculate code complexity due to usage of code elements [5] such as identifiers, reserved words, operators, operands, literals, and delimiters.
- Complexity due to Recursive methods – Used to calculate code complexity due to usage of recursive methods [6] in the code. For calculations, results of complexity due to size is used. The size complexity for the recursive method is doubled and added to the total complexity due to size for this metric.
- Complexity due to Control Structures – Used to calculate complexity due to usage of control structures [7] in the code.
- Cyclomatic Complexity – Used to calculate the McCabe's Cyclomatic complexity [8]. Complexity (M) is defined as,

$$M = E - N + 2P \quad (2)$$

  in which E is the number of edges, N is the number of nodes and P is the number of connected components. The standard version [9] that complies with McCabe's definition is used in order to implement this metric in the developed system. In the standard version, methods in the code has a base complexity of 1 and for every control structure and conditional expressions, complexity count gets increased by 1 and boolean operators used in guard conditions complexity count gets increased.

- Cognitive Complexity – Used to calculate the effort needed by the human brain to comprehend its internal logic [10] of the code. This complexity metric covers some areas that cannot be covered by cyclomatic complexity and have a far in-depth analysis [11].

2.4.2 Code Quality Analysis Component

In this component by the assist of code element identification, code quality issues are identified, and users are given the ability to automatically apply fixes for the issues or get recommendations in order to apply fixes manually for them. As an additional function, quality of the code can be predicated using the code complexity metric values of the given code by using machine learning technology. For this, a data set containing complexity values of codes that are having high code quality and codes that have low quality standards will be used to train the machine learning model through the Random Forest [12] supervised algorithm [13].

3. RESULTS AND DISCUSSION

3.1 Logic generation

This covers the class identification and the relationships between classes, which covers more of the logic implementation. As the result, it provides the list of classes and the relationship between them as a separate string arrays by analyzing the data gathered by the project proposal document. There is another outcome from this component, which is predicted by the TSLM, it is a rough prediction on the time range which will take part for the whole implementation. Although the logic implementation is not all about the classes and their relationships, there are some several areas that need to be considered such as the methods, access modifiers and other parameters. There is a limitation of the time forecasting method because of the practical issues with the data gathering regarding on the standardization methods in different organizations. The specialty of the outcomes of this component is the classes, which describe the scope and the structure of a software and the logical diagrams that helps to build up the software. Identification of these basic components of the beginning would exactly reduce the human errors, which can be occurred while analyzing the project proposal as well as within the software implementation process, thus, occurs to have a reliable and accurate software.

3.2 UML diagram generation

The logical diagrams generated using this system are very dependent on the text embedding used for the identification of the relationships between classes. The relationships are identified based on the vector representations given for the words in the GloVe [14] text embeddings and the available dimensions. The possible diagrams generated by the proposed system are shown in Fig. 1, Fig. 2 and Fig. 3.

The best possible relationships according to the text embedding are mapped in the diagrams. The generated diagrams can vary according to the embedding that is used for the vector representations. Furthermore, the relationships in the diagrams are based on semantic similarity of words.

An embedding could be trained for this sole purpose by representing words of specific relationship types as vectors at close proximities in the vector space. This type of a specialized embedding could improve the accuracy and the flexibility of the generated diagrams. This functionality has been implemented using pre-trained GloVe [14] embedding due to the time limitation. Three of the diagrams generated in this component are conventional, but the flow diagram is a new concept, which illustrates the flow of the internal logic in the perspective class. It demonstrates what classes assist the selected class to achieve its tasks in the software.

The functionality of automatically generating diagrams could

be very beneficial for the developers at software companies. The automatic generation of diagrams would take away the burden drawing diagrams and enable them to concentrate on developing high quality code.

3.3 Tracking bugs in the code

The proposed bug tracker would identify more than ten errors that the current bug trackers do not identify. Then, it will be merged with a current available bug tracker to identify all possible errors that can be detected from the software. The proposed bug tacker has the ability of browsing a java file and then detect the bugs by selecting the track bugs button as shown in Fig. 4. The error log will show available. The recommendation to correct the bug will be showed when selecting the error.

3.4 Code Quality Evaluation and Code Complexity Calculation

The proposed component can identify code elements properly and sort them accordingly to the use of quality analysis and complexity calculations. Although there are minor issues due to the different types of syntaxes used different users, the functionality evaluates the quality and perform the complexity calculations as expected as shown in Fig. 5.

Although there is a limit of fixes that can be automatically fixed in the code without the supervision of developer, the issues within the limit that can be fixed automatically can be done reliably. To improve the scale of quality issues that can be identified, this component can be merged with reliable existing system if needed. Overall, both components are very beneficial for the developers in order to manage and maintain quality of the code in the long run.

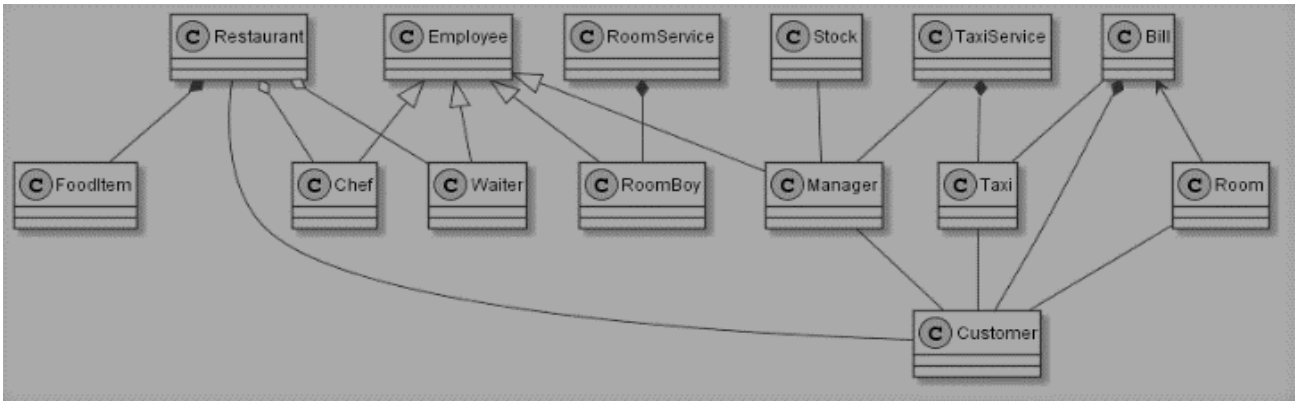

Fig. 1 Class Diagram

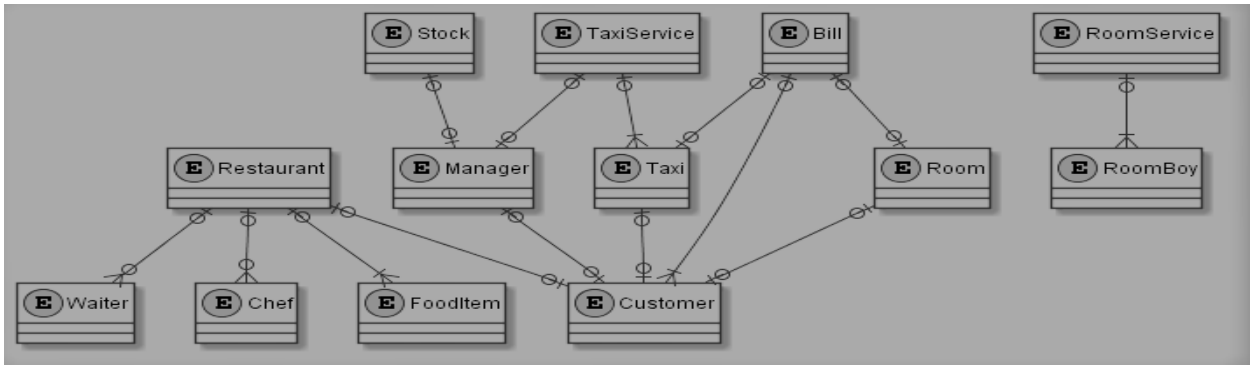

Fig. 2 ER Diagram (Generic)

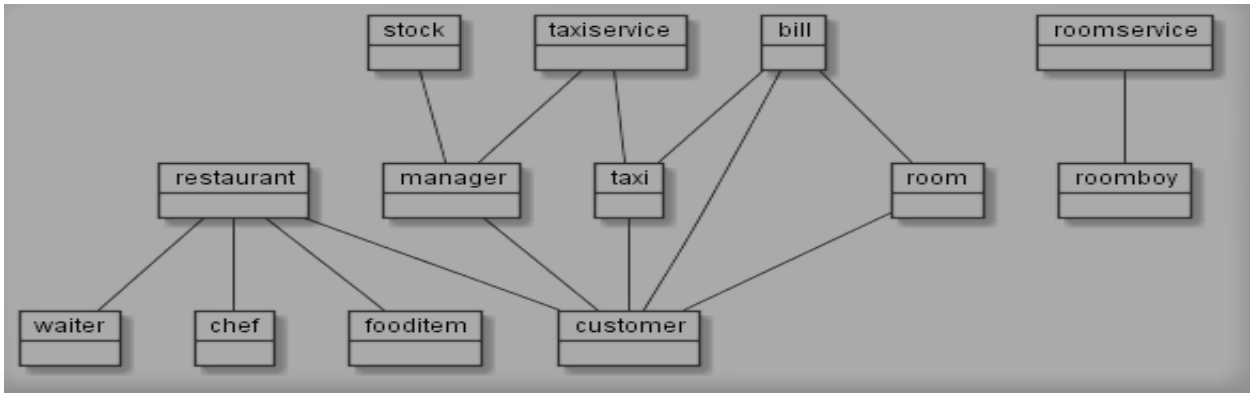

Fig. 3 Object Diagram

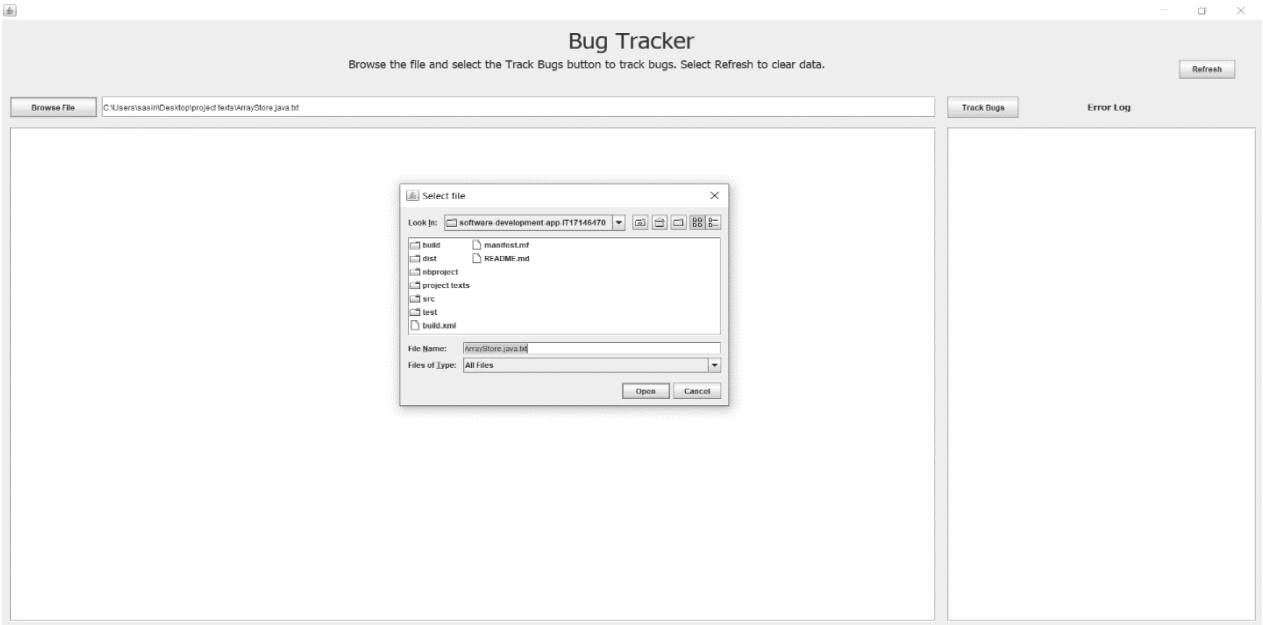

Fig. 4 Basic UI of the Bug Tracker

Cyclomatic Complexity Calculation

Code Lines	Weight
public class FibonacciMain {	0
public static String fibonacci(long number) {	1
if ((number == 0) || (number == 1)){	2
return number;	0
}	0
else {	0
return fibonacci(number -1) + fibonacci(number -2);	0
}	0
}	0
public static void main(String args[]) {	1
for(int count = 0; count <= 10; count++){	1
ystem.out.println("Fibonacci if " + count + " is " + fibonacci(count));	0
}	0
if (a && (y == b)) {	2
if (y == x) {	1
while (true) {	1
if (x++ < 20) {	1
break;	1
}	0
}	0
} else if (y == t && !d) {	2
x = a ? y : x;	1
} else {	0
x = 2;	0
}	0
}	0
}	0

Total complexity of the program 14

Fig. 5 Calculation result of code complexity

With the generated UML diagrams and the identified logic, the system is implemented by the developers. Debugging code consumes a major portion time of the implementation phase in SDLC. The bug tracker in the proposed system, which was developed with the motivation of increasing the efficiency and reducing time consumed for the debugging process was tested for its performance by using 30 different source codes. The bug tracker identified all the errors in the source codes with 100% accuracy. The suggestions provided by the bug tracker cuts down 60% of the total time, in locating the bug and debugging. With the suggestions provided by the bug tracker the developer will only have to make the suggested change in the code to fix the error. The time saved from the suggestions by the bug tracker increases the efficiency of the software development.

The code complexity analysis and code quality analysis of this

study was tested for the performance using the same source codes used for the bug tracker. Similar to the bug tracker, the code quality suggestions and the complexity calculations were made with 100% accuracy. The suggestions provided to fix code quality issues assist the developers in improving the source code with high efficiency. This again results in the increased efficiency and reduced time spent for the software development lifecycle.

Fig. 6 compares the time taken for the completion of phases of the SDLC in the normal context, where most of the work is done manually against the time taken for the completion of the same phases when the proposed system is used to automate the processes in the SDLC phases. The chart was created based on the data found in [19] and the times taken to complete the same tasks using the system proposed by this study. For completion of requirement analysis and designing, the traditional approach takes at least 14 days according to [19], but the time taken for the completion of identifying the logic and generating UML diagrams is only 1 day when the proposed system is used.

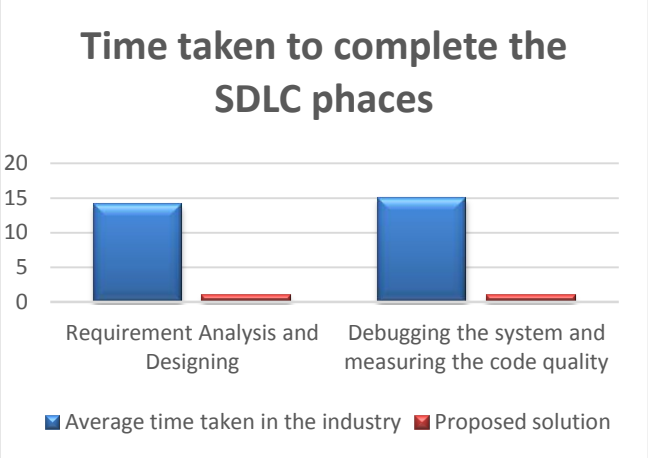

Fig. 6 Performance of the proposed system

4. CONCLUSION

When studying on how to improve overall software development, it became clear that there are plenty of tools used for many purposes in software development throughout the software developments phases. But most of these tools are mostly focused on one or two of tasks in the software development phases. The proposed system by this research focuses on most of the software development phase in order to make the software development process less complex and more efficient. With the proposed system, all the phases like requirement gathering, designing UML diagrams, detecting errors and improving code quality are automated. So, probability of occurring errors is less with the use of this system. And this saves lot of time and money because, all the processes are automated. There are also some limitations of the system that have been identified. The system is currently limited to only java language when it comes to bug tracking, complexity calculations and code issues detections. Also, there are limitations when providing automatic fixes to code issues as it might go against the logic built by the developers. So furthermore, research should be done on in these areas to improve the system. Finally, it is possible to conclude that the system is successful in making the software development process more efficient, less complex and helps greatly to automate the software development process.

5. ACKNOWLEDGMENT

We wish to thank our research supervisor Ms. Dinuka R. Wijendra, Department of Information Technology of Sri Institute of Information Technology for her keen interest, inspiring guidance, and encouragement with our work, throughout the research.

6. REFERENCES

[1] Cosine Similarity- https://www.machinelearningplus.com/nlp/cosine-similarity/

[2] PlantUML – Library to generate diagram. Retrieved June, 30, 2020 from https://plantuml.com/starting

[3] GraphViz – graph visualising software. Retrieved June 30, 2020 from http://www.graphviz.org/.

[4] Non-commenting source statements. Retrieved July, 31, 2020, from https://pmd.github.io/latest/pmd_java_metrics_index.html#non-commenting-source-statements-ncss

[5] Ryan Stansifer, Basics of the Java Language. In Notes about the Java Programming Language. Retrieved July, 27, 2020, from https://cs.fit.edu/~ryan/java/language/basics.html

[6] Recursion. In Wikipedia. Retrieved July, 27, 2020, from https://en.wikipedia.org/wiki/Recursion_(computer_science)

[7] Control Structures. In Wikipedia. Retrieved July, 27, 2020, from https://en.wikiversity.org/wiki/Control_structures

[8] McCabe (December 1976). "A Complexity Measure". IEEE Transactions on Software Engineering (4): 308–320. doi:10.1109/tse.1976.233837.

[9] Java code metrics - Cyclomatic Complexity (CYCLO). Retrieved July, 31, 2020, from https://pmd.github.io/latest/pmd_java_metrics_index.html#McCabe76

[10] G. A. Campbell, "Cognitive Complexity — An Overview and Evaluation," 2018 IEEE/ACM International Conference on Technical Debt (TechDebt), Gothenburg, 2018, pp. 57-58.K. Elissa, “Title of paper if known,” unpublished.

[11] Danny Verpoort, Insights in Cyclomatic and Cognitive Complexity in Your Application https://medium.com/takeaway-tech/insights-in-cyclomatic-and-cognitive-complexity-in-your-application-58922ae59e80

[12] Random Forest. In Wikipedia. Retrieved July, 27, 2020, from https://en.wikipedia.org/wiki/Random_forest

[13] Supervised Learning. In Wikipedia. Retrieved July, 27, 2020, from https://en.wikipedia.org/wiki/Supervised_learning

[14] Jeffrey Pennington, Richard Socher, and Christopher D. Manning. 2014. GloVe: Global Vectors for Word Representation.

[15] Singh, Sandeep. "Analysis of bug tracking tools." *International Journal of Scientific & Engineering Research 4, no. 7 (2013): 134.*

[16] Marko, Trajkov, and Smiljkovic Aleksandar. "A Survey of Bug Tracking Tools: Presentation, Analysis and Trends." *aleksland. com/wp-content/uploads/2011/01/Survey. pdf* (2011).

[17] Muqeem, M., & Beg, M. R. (2014, July). Validation of

requirement elicitation framework using finite state machine. In *2014 International Conference on Control, Instrumentation, Communication and Computational Technologies (ICCICCT)* (pp. 1210-1216). IEEE.

[18] 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.

[19] Average time to develop a custome software. Accessed on September 01, 2019 from https://soltech.net/how-long-does-it-take-to-build-custom-software/