ISSN: 0976-3104
SPECIAL ISSUE: Computer Science

ARTICLE

A STOCHASTIC SOFTWARE DEVELOPMENT PROCESS IMPROVEMENT MODEL TO IDENTIFY AND RESOLVE THE ANOMALIES IN SYSTEM DESIGN

Pratik Rajan Bhore[*], **Shashank D. Joshi, Naveenkumar Jayakumar**

Dept of Computer Engineering, Bharati Vidyapeeth Deemed University College of Engineering, Pune, INDIA

ABSTRACT

Background: *It is observed that the existing research models vary within small scale industries considering the overall system design. Their tool focuses mostly on quality oriented questions and the manual or automated detection using only the source code for finding out the scope of improvement, but it misses out finding the real anomalies in the design phase.* ***Methods:*** *So it is a necessity to develop a model that can use the manual as well as automated detection also parameters such as anomaly density, reverse engineering and also use traceability matrix to trace from testing of a software code to the design requirements for identifying anomalies in the system design and its evolution. In this research, the authors present a semi-automated stochastic software development process improvement model which will identify and resolve the anomalies in system design and its evolution. The proposed model can be realized through the tool for the enhancement of software development process assessment in an economical manner consuming minimum resources. The proposed device encloses a set of questionnaires for phases of SDLC like Testing, Coding, Design and Requirement phases. These questions get designed because of the quality assessment and improvement. The tester answers the questions (KPA), and the result indicates the improvement needed in the considered KPA.* ***Results:*** *Moreover, hence the authors use the model to detect and resolve such software design anomalies and also calculate the anomaly density along with the bug %.* ***Conclusion:*** *The proposed model tackles these design anomalies and enhance the quality of the software by manually as well as automatically testing and finding the more anomalies in the software. Also, it presents a better software development process model integrated with TDD agile model which is novel than the other existing models.*

KEYWORDS
, Design Anomalies; Software Quality; Software Testing; Process Improvement; Software Development; Software Engineering

Received: 5 June 2017
Revised: 14 June 2017
Published: 4 July 2017

***Corresponding Author**
Email:
pratik4u2007@gmail.com

INTRODUCTION

In the previous articles [1] by the authors of this research, the first article consisted of the novel approach that what were the basic parameters [1] that were being considered by the authors to create the model. The basic parameters got refined, and more parameters were added in as the research progressed. Then the next article explained about the unique methodology and solution which was going to be used for tackling with the design anomalies in software [2]. In this paper, the authors present the entire implementation and result in the analysis where the explanation is done about how the semi-automated stochastic model works. These results will fulfill the main objectives of quality enhancement as well as the identification and resolving of anomalies in the system design. Below take a look once again at the proposed stochastic model and the workflow of tasks carried by developer or tester in the architecture.

So before going into the methods for implementation of the model here, the proposed stochastic model architecture is shown here. From the previous article [2] about the proposed architecture, the author integrates that model into the agile model Test Driven Development [3]. In the TDD the focus is mainly on testing and refactoring of coding. The author takes the TDD further into the design phase from the testing and coding phases. It is how the newly integrated semi-automated stochastic model looks when integrated with TDD to give a better software development process improvement model [3]. The author hence proposes a new model for the developers or testers that can be used to test the software and develop it. It consists of four phases namely: Testing, Coding, Design, and Requirement. The earlier models all work only till the refactoring of code while the stochastic model can use reverse engineering to trace back to the design anomalies and also use traceability matrix to trace back from test cases in testing to requirements. It is the novelty of this model which is more than what the earlier models have to offer. Below the stochastic model architecture is seen in [Fig. 1].

Here above it is observed how the entire proposed model works that what is its actual workflow of the model. It is integrated with the TDD agile model to give a better process improvement model to the developer and tester for the software development and testing processes.

To give a proper introspect into the working of this model and to support the research the authors shows exactly how the flow happens.

Firstly the developer/ tester select the software he wants to test. Here the authors have tested three software namely TIC TAC TOE, Pokémon Attack, and ScratchPad. All these are mobile applications which were tested. The testing tool is used for checking if any bugs [4] can be resolved from the outside without going within the system design. Then if the bugs are not fixed from the outside, then coding phase is entered for testing. The code is tested for anomalies firstly. The anomalies are identified here from the code. The functional testing shows that there are code and design anomalies within the software [4] so explicitly testing these phases helps in eliminating these anomalies. The code [5] consists of the anomaly which can be checked using the UML tool [5] that if this anomaly affects the design directly. The anomaly in the code [6] directly makes design anomalies occur. It is a part of the entire design anomaly that occurs. There are various types of these anomalies such as Blob, Functional Decomposition, and Spaghetti Code [6]. The proposed model helps in mapping the anomaly from testing code to the design of the software. It helps in eliminating the actual anomaly from the design phase. The occurring anomaly is code can be mapped in the design diagram of the software. The model supports the elimination from directly the design phase using reverse engineering [7] if the resolving of the anomalies from code [7] does not eliminate the entire anomaly in design. The design consists of the requirements as well in the use case that are gathered before creating the design. The traceability matrix document helps in tracing back and forward from the test cases in testing to the requirements in the design [8]. It helps in mapping the design anomaly as well and determining the relation between the phases. The testing and the mapping using the traceability matrix are all manuals which assist in identifying the testing or the bug % [8] in the software. This manual process is then overtaken by the automated process of finding the anomaly density [9] testing the code and design. In the end, the quality is enhanced by eliminating the anomalies. Having a semi-automated model is better than only having a manual or only an automated model. It is said because manual detection takes alot of time for detection of the anomalies while automated models consume alot of time in calculating false positives. Hence a semi-automated approach is better to have in developing and testing the software.

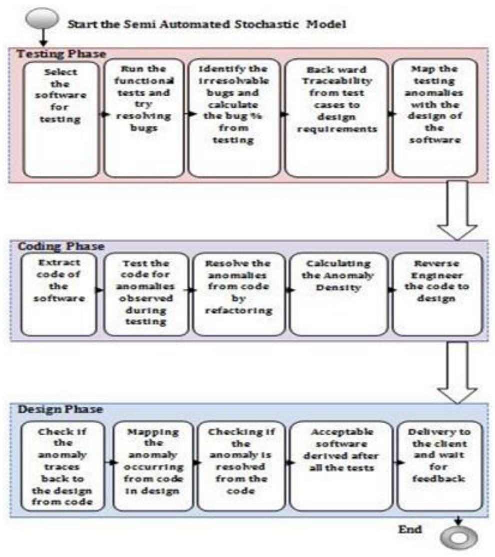

Start the Semi Automated Stochastic Model
Testing Phase
Select the software for testing
Run the functional tests and try resolving bugs
Identify the irresolvable bugs and calculate the bug % from testing
Back ward Traceability from test cases to design requirements
Map the testing anomalies with the design of the software
Coding Phase
Extract code of the software
Test the code for anomalies observed during testing
Resolve the anomalies from code by refactoring
Calculating the Anomaly Density
Reverse Engineer the code to design
Design Phase
Check if the anomaly traces back to the design from code
Mapping the anomaly occurring from code in design
Checking if the anomaly is resolved from the code
Acceptable software derived after all the tests
Delivery to the client and wait for feedback
End

Fig. 1: Stochastic model derived after integration with TDD Agile model

MATERIALS AND METHODS

The authors implement this module on the three different software namely: TIC TAC TOE, Pokémon Attack, and ScratchPad. These three are mobile applications in use. Testing is done on this software firstly to correct the bugs that can be corrected from the outside. If the bugs or errors cannot be corrected from the outside, then the problem lies in the code and design. Testing helps in mapping the anomaly [9] and relating it to the design phase. This module is a semi-automated where the testing part of the code and design of software is all automated while the mapping of the anomalies is all done manually after it is identified by the module.

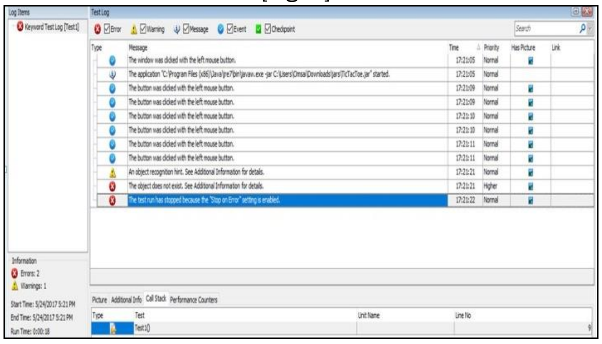
So firstly by using the testing tool, the software are tested for any bugs or errors that cannot be solved. The test cases are created based on these tests that are carried by the tester or developer. The bug % or the testing % is determined by calculating it. The parameters used to calculate it is the bug or error scenarios of the testing carried and the test cases.

Let’s have a proper look at calculating the bug%. Consider n as the total no. of test cases of the software, y as the bug or error scenarios and x as the other scenarios which consist no bugs or resolvable bugs. Hence the equation derived from this is [10]:

$$\text{Bug\% /Error\%} = \frac{\text{the total no. of bug scenarios (y)}}{\text{the total no. of test cases (n)}} \times 100$$

Functional testing the TIC TAC TOE software and finding Bug Percentage

Below here the testing tool shows the no. of test cases done on the TIC TAC TOE software. It consists of 11 test cases (n), and the no. of unresolved bugs [10] using the testing tool are 3 (y) here. It indicates the problem lies within the code and design of the software. The test casesof the software hence can be seen below [Fig. 2].



Log Items
Keyword Test Log [Test1]

Test Log
Error | Warning | Message | Event | Checkpoint | Search

Type	Message	Time	Priority	Has Picture	Link
Event	The window was clicked with the left mouse button.	17:21:05	Normal	✓	
Message	The application "C:\Program Files (x86)\Java\jre7\bin\javaw.exe -jar C:\Users\Omsai\Downloads\jars\TicTacToe.jar" started.	17:21:05	Normal		
Event	The button was clicked with the left mouse button.	17:21:09	Normal	✓	
Event	The button was clicked with the left mouse button.	17:21:09	Normal	✓	
Event	The button was clicked with the left mouse button.	17:21:10	Normal	✓	
Event	The button was clicked with the left mouse button.	17:21:10	Normal	✓	
Event	The button was clicked with the left mouse button.	17:21:11	Normal	✓	
Event	The button was clicked with the left mouse button.	17:21:11	Normal	✓	
Warning	An object recognition hint. See Additional Information for details.	17:21:21	Normal	✓	
Error	The object does not exist. See Additional Information for details.	17:21:21	Higher	✓	
Error	The test run has stopped because the "Stop on Error" setting is enabled.	17:21:22	Normal	✓	

Information
Errors: 2
Warnings: 1
Start Time: 5/24/2017 5:21 PM
End Time: 5/24/2017 5:21 PM
Run Time: 0:00:18

Picture | Additional Info | Call Stack | Performance Counters

Type	Test	Unit Name	Line No
	Test1()		9

Fig. 2: Test cases of TIC TAC TOE software

....

By clicking on the bugs [10], it is given in the design of the software that there is an anomaly affecting one of the buttons of the software. It can be mapped in the design phase as well. Hence, it proves the occurrence of design anomaly in the software. The bugs identified hence can be seen below [Fig. 3].

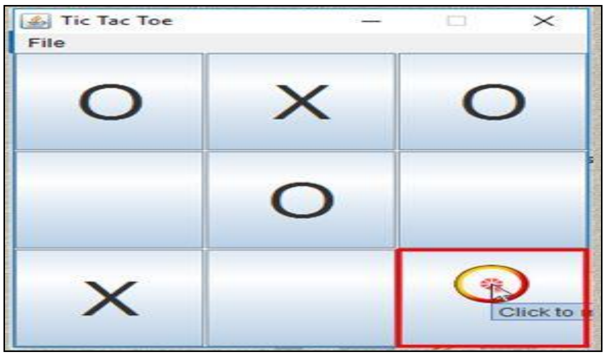


Fig. 3: Bug detected during functional testing in TIC TAC TOE

....

Using the equation mentioned above i.e. y/n x 100 gives the bug or the error % [10] in the software. Hence using it, we calculate the % which gives:

Bug %: 3/11 x 100 = 27.2 %

It is the bug % [10] in the software application which is found.
Similarly, the other two software are tested as well which are Pokémon Attack and ScratchPad.

Testing the Pokémon Attack software

Here similarly to the TIC TAC TOE, the testing tool shows the no. of test cases done on the Pokémon Attack software. It consists of 17 test cases (n), and the no. of unresolved bugs using the testing tool are 2 (y) here. It indicates the problem lies within the code and design of the software. By clicking on the bugs, it is given in the design of the software that there is an anomaly affecting the GUI [11] touch and the run button as well of the software. It can be mapped in the design phase as well. Hence, it proves the occurrence of design anomaly in the software.

Using the equation mentioned above i.e. y/n x 100 gives the bug or the error % in the software. Hence using it, we calculate the % which gives:

Bug %: 2/17 x 100 = 11.7 %

It is the bug % in the software application which is found.

Testing the scratchpad software

Similarly here the testing tool shows the no. of test cases done on the ScratchPad software. It consists of 12 test cases (n), and the no. of unresolved bugs using the testing tool are 4 (y) here. It indicates the problem lies within the code and design of the software. By clicking on the bugs, it is given in the design of the software that there is an anomaly affecting the open file option on the GUI [11] design of the software. It can be mapped in the design phase as well. Hence, it proves the occurrence of design anomaly in the software.

Using the equation mentioned above i.e. y/n x 100 gives the bug or the error % in the software. Hence using it, we calculate the % which gives:

$$\text{Bug \%: } 4/12 \times 100 = 33.3\ \%$$

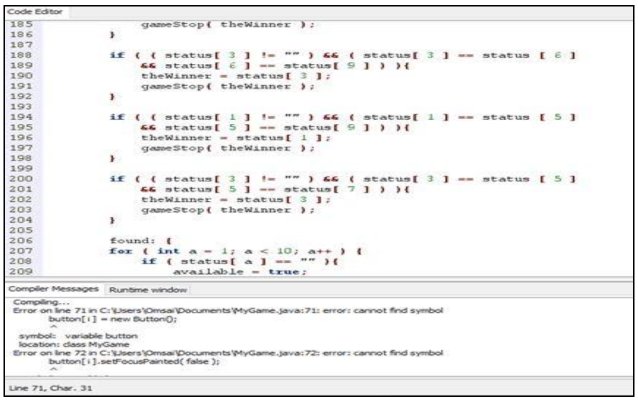
It is the bug % in the software application which is found.

Testing the code and design of the software and finding Anomaly Density

After determining that there are anomalies in the software within then, the tester carries tests on the code and design of the three software. Based on the parameters in the code, the anomaly density is calculated here. Anomaly Density [12] is an important parameter for calculating the quality of the software. The anomaly density is the total no. of anomalies found in all the modules of the software upon the total no. of the line of code. Then, later on, it can be converted to KLOC which thousand line of code [13].

$$\text{Anomaly Density\%} = \frac{\text{Total no. of anomalies}}{\text{Total Size (KLOC)}}$$

Firstly in the TIC TAC TOE software, the code is tested where the error seen in the functional test case was viewed. Below it is shown how the anomaly lies in the design of the software and it is the button of the application. The identified anomalies in the code of the software hence can be seen below [Fig. 4].


Code Editor

```
            gameStop( theWinner );
        }

        if ( ( status[ 3 ] != "" ) && ( status[ 3 ] == status [ 6 ]
            && status[ 6 ] == status[ 9 ] ) ){
            theWinner = status[ 3 ];
            gameStop( theWinner );
        }

        if ( ( status[ 1 ] != "" ) && ( status[ 1 ] == status [ 5 ]
            && status[ 5 ] == status[ 9 ] ) ){
            theWinner = status[ 1 ];
            gameStop( theWinner );
        }

        if ( ( status[ 3 ] != "" ) && ( status[ 3 ] == status [ 5 ]
            && status[ 5 ] == status[ 7 ] ) ){
            theWinner = status[ 3 ];
            gameStop( theWinner );
        }

        found: {
        for ( int a = 1; a < 10; a++ ) {
            if ( status[ a ] == "" ){
                available = true;
```

Compiler Messages | Runtime window

```
Compiling...
Error on line 71 in C:\Users\Omsai\Documents\MyGame.java:71: error: cannot find symbol
        button[ i ] = new Button();
        ^
  symbol:   variable button
  location: class MyGame
Error on line 72 in C:\Users\Omsai\Documents\MyGame.java:72: error: cannot find symbol
        button[ i ].setFocusPainted( false );
        ^
```

Line 71, Char. 31

Fig. 4: Anomalies identified in the code of TIC TAC TOE

....

Further, the code is used to reverse engineer [14] to the design phase of the software where it is seen the design anomaly called as "Blob" has been created. A blob consists of a God class that stores all the attributes and operations and the subclasses are not seen as they are not functional. The identified anomaly blob hence can be seen below [Fig. 5].

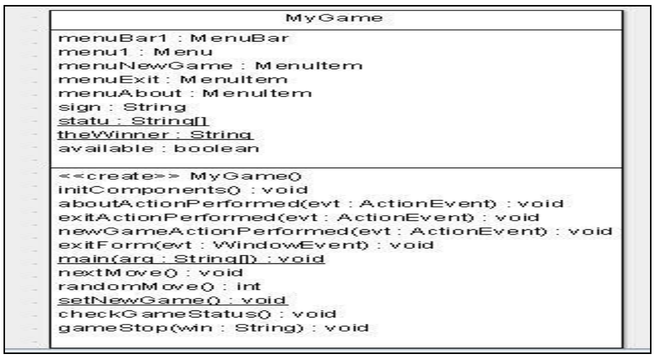
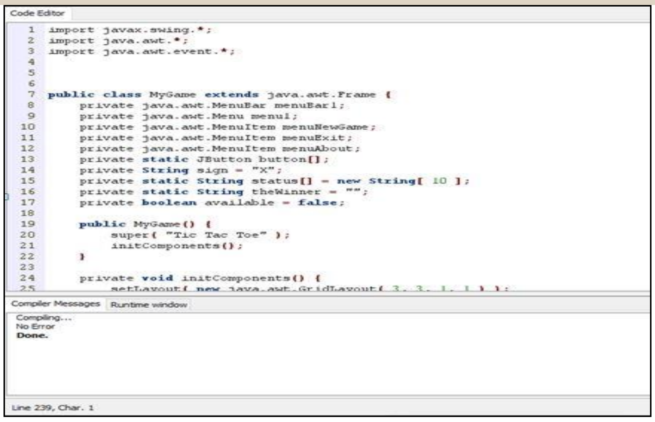

```
MyGame
menuBar1 : MenuBar
menu1 : Menu
menuNewGame : MenuItem
menuExit : MenuItem
menuAbout : MenuItem
sign : String
statu : String[]
theWinner : String
available : boolean
<<create>> MyGame()
initComponents() : void
aboutActionPerformed(evt : ActionEvent) : void
exitActionPerformed(evt : ActionEvent) : void
newGameActionPerformed(evt : ActionEvent) : void
exitForm(evt : WindowEvent) : void
main(arg : String[]) : void
nextMove() : void
randomMove() : int
setNewGame() : void
checkGameStatus() : void
gameStop(win : String) : void
```

Fig. 5: Blob anomaly Identified in the OO Design of TIC TAC TOE

....

After the anomaly is resolved from the code, it is visible there is no anomaly in the code now. The proposed module helps in deriving this conclusion that there is no anomaly in the code now as seen below [Fig. 6].

ISSN: 0976-3104



Code Editor

```
import javax.swing.*;
import java.awt.*;
import java.awt.event.*;



public class MyGame extends java.awt.Frame {
     private java.awt.MenuBar menuBar1;
     private java.awt.Menu menu1;
     private java.awt.MenuItem menuNewGame;
     private java.awt.MenuItem menuExit;
     private java.awt.MenuItem menuAbout;
     private static JButton button[];
     private String sign = "X";
     private static String status[] = new String[ 10 ];
     private static String theWinner = "";
     private boolean available = false;

     public MyGame() {
          super( "Tic Tac Toe" );
          initComponents();
     }

     private void initComponents() {
          setLayout( new java.awt.GridLayout( 3, 3, 1, 1 ) );
```

Compiler Messages | Runtime window

```
Compiling...
No Error
Done.
```

Line 239, Char. 1

Fig. 6: Code refactored using the module

After eliminating the anomalies, the developer uses the reverse engineering tool [15] to check the OOD [15] of the software in design phase if the design anomaly is eliminated and it gets disposed of in the design like seen below [Fig. 7].

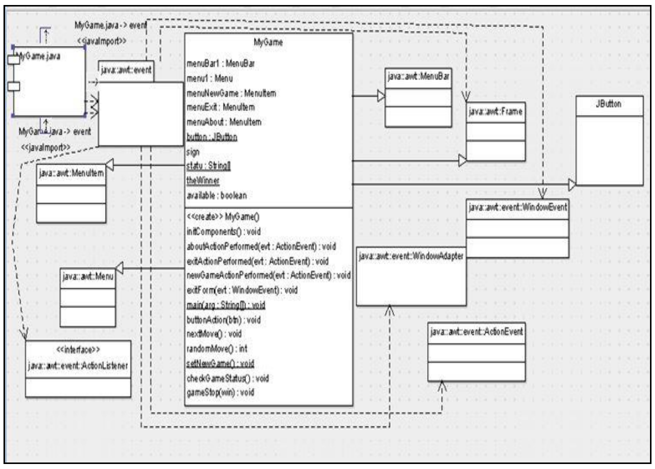


Fig. 7: Design anomaly resolved (Reverse Engineering)

Similarly the other two software namely Pokémon Attack and ScratchPad as well are tested. The anomalies are found in the code and design as well of both the software. In the Pokémon software is the anomalies are found mapped from code to the design also seen during functional testing of the software. The mapping is done manually using these results in testing, coding, and design. The anomaly in the interface [16] touch of the gaming application, as well as the run button, is seen here in the code and design as well. The anomaly occurring in the design is the functional decomposition. None of the functions are seen as there is no relationship [16] determined in the OOD [16] of the software. It shows the design anomaly in the software that is why while playing the game there is a problem in the design interface as there is a functional anomaly in design. It is directly mapped from the test case to the design requirements using the traceability matrix parameter. By using the proposed module, this anomaly is resolved just like in the TIC TAC TOE example shown above.☐

Further, the testing of the final software of ScratchPad is done as well. The code and design are tested here. The anomaly called Spaghetti Code is observed in this software. This anomaly seen during functional testing made the open file option disappear. The code gets tangled due to this and directly affects the main class in the OOD [17] of the software. The code gets tampered with, and major parts are missing. Hence it gets identified in the proposed module. The same anomaly is mapped seen in functional testing to the design. The open file anomaly occurred due to spaghetti code affects the OOD as well. In the main class, this operation and attribute are missing hence affecting the interface of the software as well. The design is the phase the customer comes in contact with hence it is important to keep it anomaly free.

Just like the first software is resolved from anomalies, in the same manner, the other two software are also made anomaly free as all the design anomalies get resolved. The tests result in identifying three major types of anomalies in the software namely Blob, Functional Decomposition, and Spaghetti Code. Also, the manual mapping is done of this in the design of the software from the test cases and code.

Hence proving the semi-automated approach carried out by authors. Further, the anomaly density [18] is calculated by the three software.

Calculating the Anomaly Density for TIC TAC TOE software here,
Anomaly Identified= 11
Line of Code=237

Anomaly Density %: 11/237 LOC x 1000(KLOC) = 46.4%

Calculating the Anomaly Density for Pokémon Attack software here,
Anomaly Identified= 16
Line of Code= 231

Anomaly Density %: 16/231 LOC x 1000(KLOC) = 69.2%

Calculating the Anomaly Density for ScratchPad software here,
Anomaly Identified= 5
Line of Code= 113

Anomaly Density %: 5/113 LOC x 1000(KLOC) = 44.2%

The Anomaly Density % [19] in all the three software is more than the calculated Bug % [20]. It proves that the anomalies need to be addressed by testing the code and design phases of the software then only the functional black box testing of it. The normal outside testing detects few bugs and indicates there are many more anomalies within which cannot be resolved using the testing tool. Hence using the proposed module the code and design are tested, and the anomalies are identified as well as resolved. Hence the Anomaly Density % is more than the Bug %. Anomaly Density is a parameter used to calculate the Quality of the Software. If the anomaly density is more than bug %, then there are more anomalies identified by using the stochastic model. Hence the proposed module is better for testing the software and resolving the anomalies within than the existing models.

The author's module also provides the additional features of HTML editor rather known as IDE Expert which can be used to correct the dynamic code and designs. The tests are carried on this extra feature where the module detects the anomaly from the code. The suggestion is shown where the anomaly lies in the code which can resolve anomalies in the design.

RESULTS

In the result analysis, the expected results have been derived after conducting the necessary tests to support the proposed research. The bug percentage is calculated firstly while functional testing and later the anomaly density is calculated when the tester tests the code and design of the software. Firstly the Software Stats are shown where the software on which the tests are carried is seen, and then the no of classes which are part of the code and no of the line of code is shown. These classes from code are directly related to the classes in Object Oriented diagrams. It is how anomalies like Blob, Functional Decomposition, and Spaghetti Code occur in the design. This model resolves such anomalies in the design. The software statistics hence can be seen below [Table 1].

Table 1: Software statistics

SOFTWARE APPLICATION	CLASSES	LINE OF CODE
Tic Tac Toe	108	237
Pokémon Attack	433	231
ScratchPad	79	113

Later the bug percentage [21] is derived from functional black box testing which is done manually to the each software to solve the bugs that can be solved from outside and also determine the bugs that need to be solved using white box testing from the within of code and design. Here the software functions are tested where unit components and the integration of these components is all tested. Test cases are written here by the developer or tester as the testing is carried. Later the bug percentage is calculated based on the test cases carried and the bug scenario detected that are irresolvable from the outside. The bug percentage results hence can be seen below [Table 2].☐

Table 2: Bug percentage

SOFTWARE APPLICATION	TEST CASES	BUG SCENARIO
Tic Tac Toe	11	3
Pokémon Attack	17	2
ScratchPad	12	4

ISSN: 0976-3104

Next is the calculation of the anomaly density which is the parameter that shows the quality of the software. It indicates that the anomalies which are resolved to give quality enhancement [21]. The LOC as well is the parameter used here. It is carried in an automated way using the model. It is calculated based on code and design of the software. The anomaly density results hence can be seen below [Table 3].

Table 3: Anomaly density

SOFTWARE APPLICATION	ANOMALY IDENTIFIED	LINE OF CODE
Tic Tac Toe	11	237
Pokémon Attack	16	231
ScratchPad	5	113

The final results display the bug % and the anomaly density % which is more than it. It shows that during the testing of code and design more anomalies will be identified and resolved more than by only functional testing. Hence the manual and automated way resolves more design anomalies than by only functional testing. The model combines to give out the semi-automated model which will eliminate all the anomalies manually and automatically to give better results as well as give better quality. It will be beneficial to give the customer a better experience and to get better feedback as well. Also, the design anomalies will be resolved, and the quality will improve as well. The final results hence can be seen below [Table 4].

Table 4: Final results

SOFTWARE APPLICATION	BUG % (TESTING)	ANOMALY DENSITY %
Tic Tac Toe	27.2%	46.4%
Pokémon Attack	11.7%	69.2%
ScratchPad	33.3%	44.2%

Comparing manual and automated testing carried in the semi-automated stochastic model for identifying and resolving design anomalies. The graph hence can be seen below [Fig. 8].

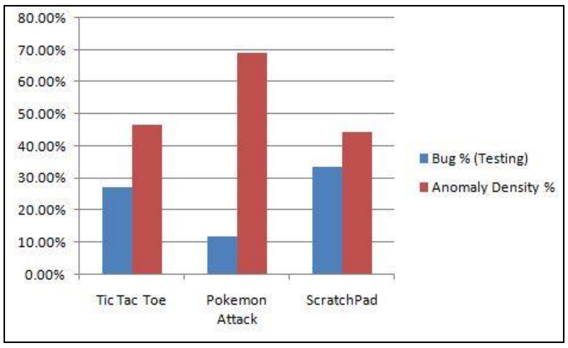


Fig. 8: Graph comparison between manual and automated testing

CONCLUSION AND FUTURE WORK

In the entire research, the authors have proposed a stochastic software development process improvement model the main aim of this system was to enhance quality in various software by identifying the anomalies from the code and design phases and resolving them. In this study, the authors fulfill the objectives by detecting and correcting these anomalies from the code and employ the technique of reverse engineering and traceability matrix [22] to check whether the actual design anomaly has been removed entirely. The customer using the software is in contact directly with the design phase when he uses it. It is the interface between the full software and the customer. That is why this model tries to improve the quality of the system design for the satisfaction of the client and to give a better feedback in return to the developer. These anomalies try to compromise the performance of the software. Also in today's world as there is much competition between software organizations everyone would like their software to be used more than other and to be bug-free. The proposed model also has an additional feature where it helps in achieving this goal where most of the online systems employ the dynamic language of HTML while most of the offline usable software employ the JAVA language source code in implementing their software and also use object oriented programming to form their design. The model deals with such do not and give them a chance to improve their quality with less time wastage and also save a lot of companies' resources. Also, it helps in creating highly reliable software.

Shortly, many researchers can take this work forward by addressing the initial stage of the requirement stage as well. In this research work, the authors address the design and use case requirements as well as its relationship to the coding phase. In the coming days, many researchers can try to establish the relationship of the actual requirement phase with the design phase as it is the first phase of the software development cycle. Here the employed agile models such as TDD and FDD that focus on the changes that can be brought to the code to remove design anomalies but in the future researchers can integrate their model with agile models like XP or Scrum that focus on the requirements more than the code where there are breakpoints to adapt to changes requirements. It can further improve the quality of the software as well as the proposed model further as the technology is changing with the coming days [23]. This further work in this area will be beneficial further to many more organizations as well.

CONFLICT OF INTEREST

There is no conflict of interest.

ACKNOWLEDGEMENTS

None

FINANCIAL DISCLOSURE

None

REFERENCES

[1] Bhore PR, Joshi SD, Jayakumar N. [2016] A Survey on the Anomalies in System Design: A Novel Approach. International Journal of Control Theory and Applications, 9(44):443-455.

[2] Bhore PR, Joshi SD, Jayakumar N. [2017] Handling Anomalies in the System Design: A Unique Methodology and Solution. International Journal of Computer Science Trends and Technology, 5(2):409-413.

[3] Jayakumar N, Bhor M, Joshi SD. [2011] A Self Process Improvement For Achieving High Software Quality. International Journal of Computer Science Trends and Technology, 3(5):306-310.

[4] Patil TB, Joshi SD. [2015] Software Improvement Model for small scale IT Industry. International Journal of Advanced Research in Computer and Communication Engineering, 4(5):601-604.

[5] Fenton N, Pfleeger SL. [1998] Software Metrics: A Rigorous and Practical Approach. ACM Transactions, 15(2):126-139

[6] Fowler M. [1999] Refactoring—improving the design of existing code. 1st edn. Addison-Wesley, Reading IEEE.

[7] Brown WJ, Malveau RC, Brown WH, McCormick HW, Mowbray TJ. [1998] Anti Patterns: Refactoring Software, Architectures, and Projects in Crisis. 1st edn. Wiley, New York IEEE, 15(2):101-112

[8] Kessentini M, Kessentini W, Sahraoui H, Boukadoum M, Ouni A. [2011] Design anomalies Detection and Correction by Example. IEEE.

[9] Yoshida N, Saika T, Choi E, Ouni A, Inoue K. [2016] Revisiting the relationship between code smells and refactoring. IEEE 24th International Conference on Program Comprehension (ICPC), 33(1):1-4.

[10] Mens T, Tourwé T. [2004] A Survey of Software Refactoring. IEEE Transactions on Software Engineering, 30(2):126-139.

[11] Sahraoui H, Abdeen H, Bali K, Dufour B. [2015] Learning dependency-based change impact predictors using independent change histories. Information & Software Technology IEEE, 23(2):220-235.

[12] Moha N, Gueheneuc YG, Duchien L. [2010] DECOR: A Method for the Specification and Detection of Code and Design Smells. IEEE Trans. Softw. Eng.

[13] Martin R. [1994] OO Design Quality Metrics An analysis of dependencies. IEEE.

[14] Chechik M, Gannon J. [2001] Automatic Analysis of Consistency between Requirements and Designs. IEEE Trans. Softw. Eng.

[15] Fenton N, Ohlsson N. [2000] Quanti. Analysis of Fault & Failure in Complex Softw. Sys. IEEE Trans. Softw. Eng.

[16] McCabe TJ. [1976] A Complexity Metric. IEEE Transactions on Software Engineering, 2(4):308-320.

[17] Marinescu R. [2004] Detection Strategies Metrics based rules for detecting design flaws. IEEE Computer Society.

[18] Dhambri K, Sahraoui H, Poulin P. [2008] Visual detection of design anomalies. IEEE Computer Society.

[19] Abdeen H, Shata O. [2012] Metrics for Assessing the Design of Software Interfaces. Int. J. of Adv. Res. Comput. Commun. Eng., 1(10):1-8, IEEE.

[20] Hossain A, Kashem A, Sultana S. [2013] Enhancing Software Quality Using Agile Techniques. Springer.

[21] Gautam AK, Diwekar S. [2012] Automatic Detection of Software Design Patterns from Reverse Engineering. Springer.

[22] Bhore PR. [2016] A Survey on Storage Virtualization and its Levels along with the Benefits and Limitations. International Journal of Computer Sciences and Engineering, 4(7):115-121.

[23] Bhore PR. [2016] A Survey on Nanorobotics Technology. International Journal of Computer Science & Engineering Technology, 7(9):415-422.