RESEARCH ARTICLE OPEN ACCESS

Handling Anomalies in the System Design: A Unique Methodology and Solution

Pratik Rajan Bhore [1], Dr. Shashank D. Joshi [2], Dr. Naveenkumar Jayakumar [3]
Department of Computer Engineering
Bharati Vidyapeeth Deemed University College of Engineering
Pune - India

ABSTRACT
The primary problem in the software these days is the anomalies occurrence in the design and code of the system. These anomalies or rather these bugs cause a huge impact on the quality and reliability of the software. It is a big issue in the software organizations for the developers or the testers creating or developing the software. In this article, we propose a methodology and a solution for handling these constantly transpiring anomalies in the system design. We take forward our research from the earlier article where we surveyed about the various anomalies and the novel approach using the different metrics for handling the anomalies in the system design.
Keywords :— Code Anomalies, Design Anomalies, Software Bugs, Software Development

I. INTRODUCTION

Here to begin with, in this article [2][3] we start by giving a gist of what our approach was from the previous paper [5] that we wrote to begin our research on this area. The approach consisted of various software metrics [6] when integrated together will create a unique process methodology [1] for the developers or the testers in the software organizations [4] for detecting [13] and correcting the bugs in the coding and design phase of the system. To revise back on the metrics [6], we had mentioned before they were [5]:

1. Key Process Area
2. Agile Methodology
3. Quality Gate
4. Traceability Matrix
5. Source Code
6. Reverse Engineering/ Reengineering

When all of these are integrated together and used that is when we get our proposed process model [1] for tackling the anomalies [9] in system design. It will help in boosting the software quality as well. Detection and correction of anomalies [13] are directly proportional to the quality increase in the software and vice versa.

Quality Enhancement $\propto$ Detection and Correction of Anomalies

For achieving this, we further propose a unique methodology and architectural solution to handle the problem with finesse.

II. THE UNIQUE METHODOLOGY

We propose a unique methodology which implies in carrying out the following tasks in its proper order:

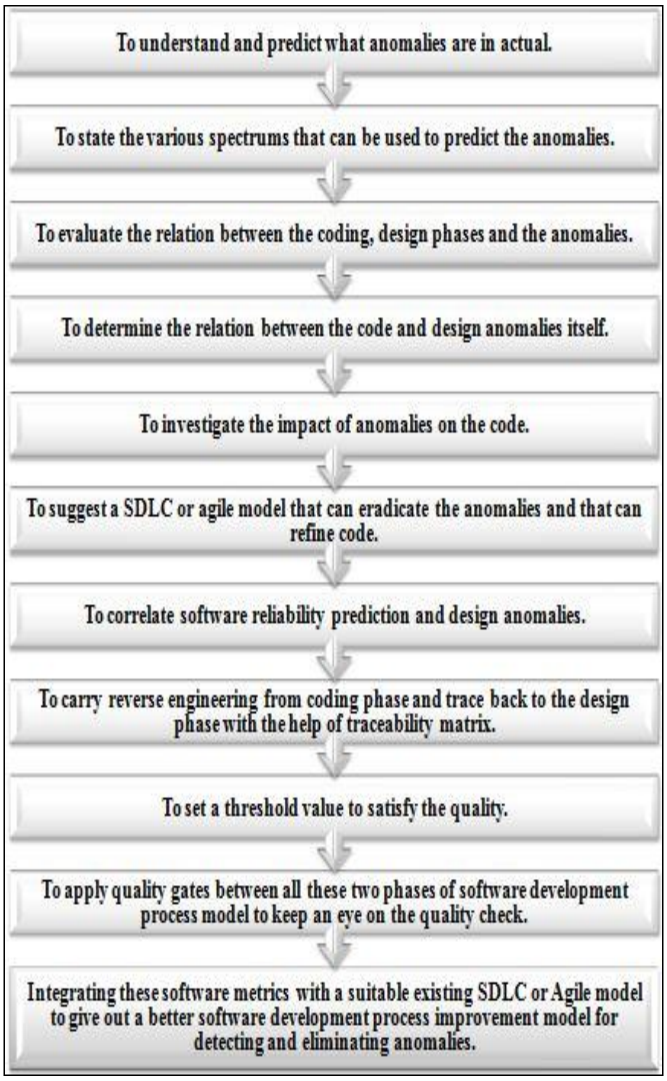
To understand and predict what anomalies are in actual.
To state the various spectrums that can be used to predict the anomalies.
To evaluate the relation between the coding, design phases and the anomalies.
To determine the relation between the code and design anomalies itself.
To investigate the impact of anomalies on the code.
To suggest a SDLC or agile model that can eradicate the anomalies and that can refine code.
To correlate software reliability prediction and design anomalies.
To carry reverse engineering from coding phase and trace back to the design phase with the help of traceability matrix.
To set a threshold value to satisfy the quality.
To apply quality gates between all these two phases of software development process model to keep an eye on the quality check.
Integrating these software metrics with a suitable existing SDLC or Agile model to give out a better software development process improvement model for detecting and eliminating anomalies.

Fig. 1 Proposed Methodology

To be precise about the proposed methodology [2][3]we are going to explain how the method is unique and how it works.

1) Step One:

The model we propose helps the developers and the testers in predicting [12] and showing what real anomalies [9] are. It helps in detecting [13] these anomalies occurring in the design and the coding phases of the software. Humans do not have enough knowledge for the manual detection and correction [13]. The required knowledge to find anomalies [9] in the code or design for humans is immense. Hence auto detection model helps in predicting [12] the actual anomalies showing developers where the problem lies in the code [10] or the design.

2) Step Two:

The model consists of various spectrums which are needed to be brought together for the orderly functioning of it. These spectrums are nothing but the software metrics [6] that function together to achieve the objectives of the model. Here we employ the spectrums such as KPA, Agile Model, Quality Gate, Traceability Matrix, Source Code and Reverse Engineering [5] like we have mentioned in the novel approach we have stated. It helps in predicting [12] the anomalies [9] in the code or the design of software.

3) Step Three:

The model correlates the coding and the design phases. It gets evaluated by using the model. It is crucial knowledge for the developer for determining what effect the code [10] refining can cause on the design of the software or vice versa. The design gets directly affected when the code of the software gets changed. Hence if the anomalies [9] lie in the design, they can be directly resolved using the coding phase. Also, the downside of this is that if there are code anomalies in the code, then it can affect the design of the software in a bad way. Hence evaluating the relation between these two phases is quite important here.

4) Step Four:

The model helps in determining the relation of the code and design anomalies [9]. Like we see in the third phase it is vital for the developers to recognize the relation between the code and design phases as the anomalies get interrelated with each other after their occurrence. The anomaly in code [10] can cause an anomaly to occur in the design of the software that's how much they are dependent on each other. Moreover, hence making it imperative to understand and determine their relation always.

5) Step Five:

The model helps in investigating the impact the anomalies [9] cause on the software. The introduction of bugs in the code will directly affect the design of the software with which the user is in direct contact. The user will experience direct discomfort using the software and will have negative feedback hence due to lack of quality. That is why the developers and testers in the software organizations [4] should investigate these anomalies [9] before delivery to the customer to have better feedback.

6) Step Six:

The model is integrated with an agile methodology to give us a complete model along with the other parameters involved. Our module chooses the agile methodology rather than SDLC because they address the code [10] in the development of the software design swiftly. In such situations FDD or TDD models are crucial. They address the source code as an important metric to develop the quality of the software. They refactor the code [7][8] to give a better design and eliminating the bugs befalling in the code and design. In our model it is based on auto detection [13] hence these models help to achieve our goal rather than the pair programming used in previous models that are manual based.

7) Step Seven:

The model correlated software reliability prediction [12] with the anomalies occurring in the design [9] or the code [10]. It is important for the organization's [4] sake that these anomalies get predicted [12] during the development phases rather than after the delivery to the customer. Hence the occurrence is needed to be corrected by prediction to make the software a reliable one beforehand.

8) Step Eight:

In this model, the two metrics that help to check the changes in the design by the refining of the code are reverse engineering or re-engineering and traceability matrix. They carry out the tracing back from the [10] to the design phase to check if the anomaly is corrected from the code properly. Therefore this helps in elimination of the real anomaly.

9) Step Nine:

The model can help the developers assign a metric threshold value by the anomaly detection [13] to analyze the quality percentage in the software. If a certain threshold value is defined and the quality of the software is less than the defined threshold value [5], then there are bugs in the code or the design. We can correct the bugs in code or design using our model and check the quality enhancement then.

10) Step Ten:

Like we have seen above the model helps in setting a quality gate in between phases to check quality. Suppose all the bugs are detected and eliminated [13], then the quality will be higher than the predefined threshold value. It is the use of the quality gate for the developers to determine the exact quality ratio in the software. It completes the process model [1] for detecting and correcting the anomalies [9].

11) Step Eleven:

Lastly, the model gets supplemented by integrating all the metrics [6] mentioned above and an existing agile method that focuses on the source code [10]. Like FDD and TDD which helps automatically detect anomalies [9] in the code such as JAVA source code or directly the design such as the dynamic pages on the online software which are in HTML. Our model helps in detection of bugs in these areas that are widely used by the software developers. Also using reverse engineering tool, we can check if the design of JAVA code is improving by tracing back. While in the HTML code we can directly see the design in the model which using the reverse engineering algorithm to give the design outlines. It is all achieved by integration of the agile method with the metrics stated. Moreover, in the end, the objective of the anomaly detection and correction [13] is achieved.

III. THE PROPOSED ARCHITECTURE/ SOLUTION

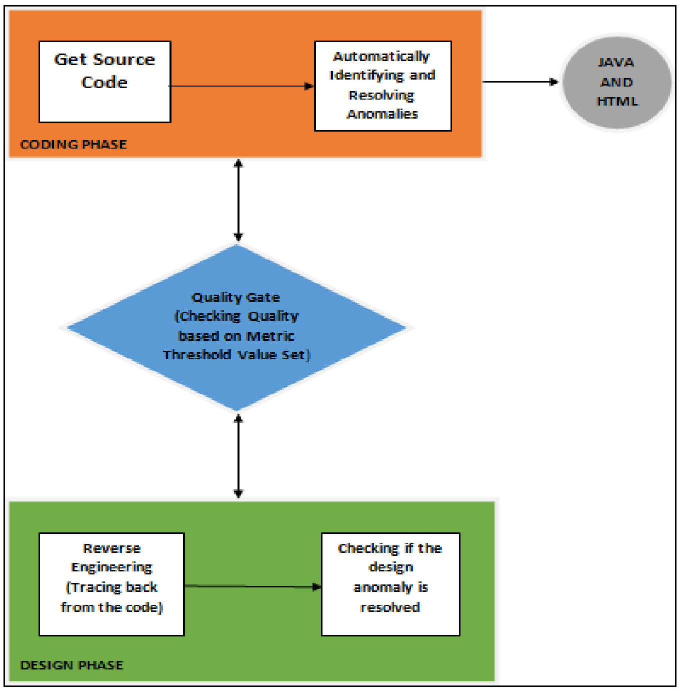
Get Source Code
Automatically Identifying and Resolving Anomalies
JAVA AND HTML
CODING PHASE
Quality Gate (Checking Quality based on Metric Threshold Value Set)
Reverse Engineering (Tracing back from the code)
Checking if the design anomaly is resolved
DESIGN PHASE

Fig. 2 Proposed Architecture/Solution Workflow

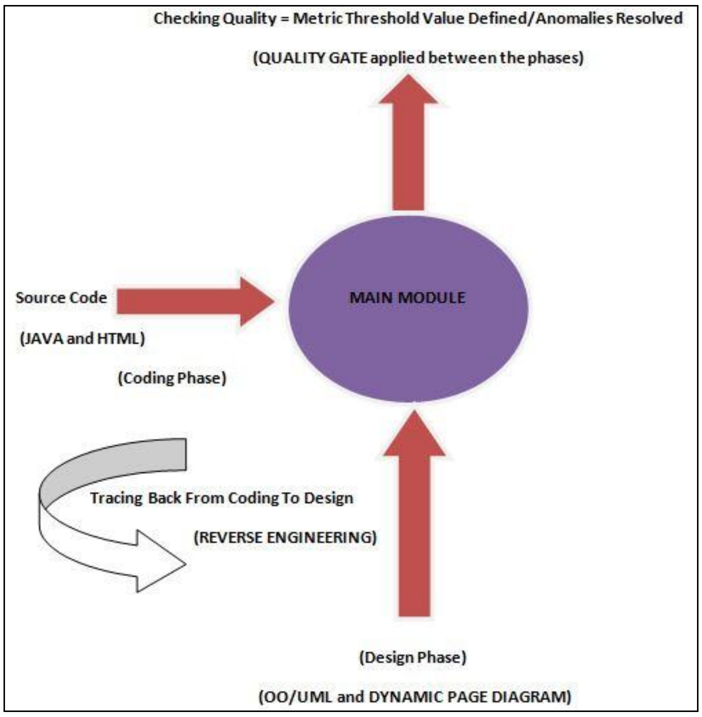
Checking Quality = Metric Threshold Value Defined/Anomalies Resolved
(QUALITY GATE applied between the phases)
Source Code
(JAVA and HTML)
(Coding Phase)
MAIN MODULE
Tracing Back From Coding To Design
(REVERSE ENGINEERING)
(Design Phase)
(OO/UML and DYNAMIC PAGE DIAGRAM)

Fig. 3 Proposed System Architecture/Solution

To get an insight into the working of the proposed architectural module, we explain here how the entities in the diagram mentioned above work. This proposed architectural module is nothing but the solution that deals with the problem of anomalies [9] in the system design. Firstly the module has a login and password. Only the authenticated developer or tester can enter in the module. Hence only the software organization [4] employees can access this module. After authentication, the developers can start to get into the source code of any software that is based on JAVA or HTML code. In today's world, most of the offline software are built in JAVA coding while all the online software gets built in HTML coding. Hence our main focus is improving these software processes [1] to give out very reliable and quality software which gets fully developed. The module after getting the code starts analyzing the correctness of the code. We know that the coding phases and the design phases are interrelated to each other as they occur one after the other. Hence their anomalies are correlated to each other as well. Suppose there exists a code anomaly in the source code of the software, this anomaly will directly affect the design of the software. The effect caused a design anomaly in the design phase as well. It proves that the anomalies [9] in both the phases are interrelated to each when they occur. In our previous article, [5] we have explained what the various types of anomalies in the code or design are and how they occur due to each other. Our module starts finding the bugs directly from the code saving time and resources of the developers. It shows the exact placement of the bug. Metrics [6] such as Line of Codes and the algorithm used in our module helps to detect the exact line where the bugs lie in the code. Below it shows what is missing in the code which resulted to that anomaly occurring in the code. Using the reverse engineering tool integrated into the module we can trace back to the design phase where we can check how the design changes by the introduction of anomalies [9] in the code. By eliminating these detected bugs [13], we can hence enhance the quality and reliability of the software. Our module offers this in JAVA as well as HTML code and design. HTML is used extensively in the dynamic pages of the online software. The introduction of bugs occurs a lot in the online software more than the offline software nowadays. So to have an extra advantage of correcting the module offers the developers to keep in check the online software or the dynamic pages of their software organization [4] in check. The variation can be seen in the design directly if the code is refactored [7][8] or refined. In the end, if the code is in check and the quality is 100% correct [11] after all the anomalies [9] get resolved then the module also helps in compiling the code and gives out the desired output. We can then finally check if the design anomaly is resolved as well to give a perfect design with perfect quality and reliability for the customers. It is how our module is unique from other existing models.

IV. ISSUES IN EXISTING MODELS

There are multiple issues in the existing models that need to get addressed. Hence in our novel model, we try to eliminate all these shortcomings or matters in the existing models to give out a better and developed model for the use of the software developers or testers in the software organization [4]. The issues are:

In the first issue, the existing models fail to address that the developers have a deadline in which the customers should be deployed with the desired software. Before the software is delivered the testing and developing should be fully complete as per the wish of the stakeholders and the customers. The only obstacle that can cause a delay in getting the desired software quality and reliability in time can be the introduction of the anomalies in the software [9]. Hence to eliminate these bugs and resolve those in due course the developers need a model that will remove all the bugs in time and well before the deadline in their organization [4]. The existing models consider dealing with these anomalies manually by using techniques such as pair programming. It can extend the development further than the actual deadline. The human eye is bound to miss a few bugs, and hence the quality will be not improved as per need. Also, there are large types of anomalies that occur in the code or the design, and the humans do not have all the knowledge about them which is needed. Hence it was difficult to determine if it is an anomaly or not. Hence it is necessary to have an automatically detection model which can detect all the bugs in the code of the software as well as in the design and therefore the client gets a high-quality software with zero bugs fully functioning.

The next issue in the existing models is that none of them uses the reverse engineering to check if the code correction eliminates the design anomaly. They use various programming techniques and algorithms but fail to check in the actual design if the anomaly is corrected. As the client comes in contact with the outlet design of the software, it makes it the most important phase in the software. The proper design can attract more clients, and if the quality is high in the design, then the clients will be satisfied as well.

The next issue is that they do not consider the traceability matrix and trace back and forth from code to design. Also, the models fail to keep a record of the anomalies [9] resolved so that next time when the developer faces similar anomaly, then he can handle it in no time.

The next issue is that they cannot eliminate all the bugs in the code or design as the algorithms they use is not bound to pick up a few anomalies, while our model gives a 90 -100% quality [11] enhancement as it detects [13] all the anomalies in the code and design. The detected anomaly is shown below in our model that in which line of code it lies and how it affects the design of the software. It helps the developer to resolve the anomaly from the LOC [11] directly. By doing this the desired quality and reliability is achieved.

The next issue is that none of them consider using the quality gate in their processes [1]. The quality gate is an important milestone which is vital in checking quality in the code or design. It can be applied to the both phases. A metric threshold value is defined where if more anomalies are detected [13] then it is determined that if the quality is lower than set threshold value. This is all considered in our model.

The next issue is none of them can analyze all the bugs in a JAVA or HTML [11] code or design. Our model offers correction in both these languages which get mostly used in the online and offline used software. Hence making this necessary to be used in today’s software. Therefore, this helps in achieving the objective of high quality and reliability in the end. Also, the customer satisfaction gets guaranteed about the software.

FUTURE WORK AND CONCLUSION

In this article, [5] we have explained our unique methodology which consists of sequential tasks and our unique solution consisting of the proposed architectural module which will deal with the anomalies [9] in the coding or the design phase of the software. The main objective hence is completed which is quality enhancement and reliability achievement which can be gained using this model. In the coming days we will present in our next article with the result and implementation of this proposed model to give a better idea to developers or researchers.

REFERENCES

[1] N. Jayakumar, M. Bhor, S.D. Joshi, A Self Process Improvement For Achieving High Software Quality, 2011.

[2] P.R. Bhore, A Survey on Storage Virtualization and its Levels along with the Benefits and Limitations. International Journal of Computer Sciences and Engineering, vol. 4, no. 7, pp. 115-121, July 2016.

[3] P.R. Bhore, A Survey on Nanorobotics Technology, International Journal of Computer Science & Engineering Technology, vol. 7, no. 9, pp. 415-422, September 2016.

[4] T.B. Patil, S.D. Joshi, Software Improvement Model for small scale IT Industry, 2015.

[5] P.R. Bhore, Dr. S.D. Joshi, Dr. N. Jayakumar, A Survey on the Anomalies in System Design: A Novel Approach, International Journal of Control Theory and Applications, vol. 10, May 2017.

[6] N. Fenton, S.L. Pfleeger, Software Metrics: A Rigorous and Practical Approach, ACM DL, 1998.

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

[8] W.J. Brown, R.C. Malveau, W.H. Brown, H.W. McCormick, T.J. Mowbray, Anti Patterns: Refactoring Software, Architectures, and Projects in Crisis. 1st edn. Wiley, New York. 1998.

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

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

[11] T. Mens, T. Tourwé, A Survey of Software Refactoring. IEEE Transactions on Software Engineering., vol. 30, no. 2, 126-139, February 2004.

[12] H. Sahraoui, H. Abdeen, K. Bali, B. Dufour, Learning dependency-based change impact predictors using independent change histories. Information & Software Technology, pp. 220-235, 2015.

[13] N. Moha, Y.G. Gueheneuc, L. Duchien, Le Meur. DECOR: A Method for the Specification and Detection of Code and Design Smells. IEEE Trans. Softw. Eng. 2010.