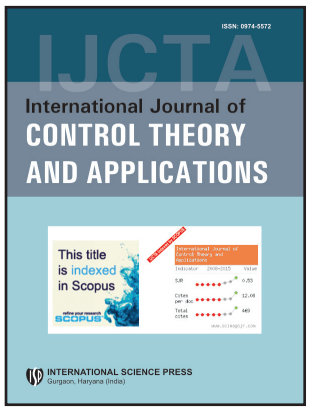

International Journal of Control Theory and Applications

ISSN : 0974-5572

Volume 9 • Number 44 • 2016

A Survey on the Anomalies in System Design: A Novel Approach

Pratik Rajan Bhore[a], Shashank D. Joshi[b] and Naveenkumar Jayakumar[c]

[a-c]Department of Computer Engineering, Bharati Vidyapeeth Deemed University, Pune. 411043, India. Email: [a]pratik4u2007@gmail.com; [b]sdj@live.in; [c]naveenkumar@bvucoep.edu.in

Abstract: Anomalies mostly arise in the design of the software while the process cycle of its evolution is being carried out. This occurrence is due to the factors such as low maintainability and less adaptability to the changes that take place in the future. As the required knowledge about these anomalies is far more than the actual knowledge that is known to humans, a significant subset of code and design anomalies is difficult to manually identify and utilizes many resources as well as the time of people. Here, the design anomalies get handled by trying to eliminate them from the coding phase directly but where the majority of these models go wrong is when they try to correct them manually from the coding phase. As we know that the coding and design phases are interrelated, but this whole process spends the time and resource of the people as we have mentioned earlier. It shows that there are many limitations in these existing manual detection models which exist. That is why this paper focuses on how to overcome these shortcomings in these existing models by proposing a novel approach which will improve the various deficiencies such as pair programming in the existing models and give a better model by integrating it with the FDD or TDD agile models for addressing these anomalies swiftly in the system design in the coming future.

Keyword: Design Anomalies, Code Anomalies, System Design, Software Quality, Software Process Improvement, Software Development.

1. INTRODUCTION

Design anomalies befall during the software development process cycle while or after it has been completed. These anomalies can directly attack the software quality by causing a major setback in the system's evolving stage. Researchers known as Fenton and Pfleeger[1] have mentioned it in their earlier works that design anomalies may cause failures not necessarily directly; they may do it indirectly as well. Identifying the design anomalies is a necessity for developing the software. When these design anomalies get resolved after identifying them, then it will be a big contribution to improving the software quality. However, identifying the design anomalies is far from trivial as manual detection techniques have their limitations. Firstly, the manual detection of design anomalies is time and resource consuming, as it depends on human resource and their working speed. The automatic detection of design anomalies does not depend on human resources or their working speed and is better to manual handling, the reason being that people don't have enough knowledge about the design anomalies which is very necessary if

they want to detect the anomalies from the coding phase during pair programming activities. Furthermore, there are multiple hard tasks other than handling of the design anomalies, such as the relationship between the code and design phases, the capacity of the search space for the optimization of the code, other uncertain definitions, and the defining of a proper metric threshold value. These problems are part of the coding and design phases and they transpire during software evolution obstructing the improvement of software quality.

This article focuses mainly on the following areas:

- The exact definition of design anomalies.
- The several types of these anomalies.
- Discussion on the existing models.
- And a novel approach to overcome the various limitations of the existing models.

2. DESIGN ANOMALIES DEFINITION

The design anomalies are also known as software bugs or the maintainability flaws that occur in the software evolution. These design anomalies are nothing but the situations in the design phase that keep on occurring and affect the software evolution on a large scale. They mostly occur during the maintenance process in the software evolution. Earlier in the year 1997, Fenton and Pfleeger stated in their research that the design anomalies may cause failures in software tasks indirectly not necessarily directly. It will, in fact, make the software quite desperate in trying to bring changes to the system which will, in turn, become the main cause for the introduction of design anomalies. Here we study different categories of the design anomalies. These design anomalies display a diversity of symptoms which are mandatory for studying to get an essential knowledge about the design anomalies. The main aim to acquire this necessary knowledge about the design anomalies is so that we can facilitate their identification and propose their resolving solutions. The earlier research literature suggests there are two main types of the design anomalies namely (1) code smells and (2) anti-patterns. Earlier in the year of 1999, this first type of design anomalies called the code smells was defined by the researchers Fowler and Beck[2]. This code smells had twenty-two sets of symptoms of general constant occurring maintainability flaws in the software. These maintainability flaws consisted of four main categories namely: large and lazy class in the design phase, the longer lists of certain metrics and the code smell known as feature envy smell. Most of these design anomalies are identified and resolved by the most common solution called as a refactoring of the code[3], which restructures the code for the elimination of these flaws in the design. But all of this is done manually. Earlier in the year of 1998, this second type of design anomalies called as the anti-patterns got defined by the researcher Brown et. al.,[4].

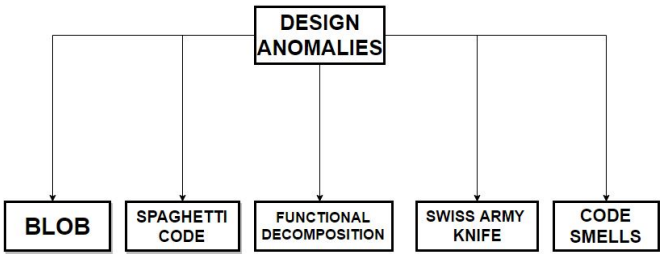

Figure 1: Design Anomalies Taxonomy

3. TYPES OF DESIGN ANOMALIES

Design anomalies are the ones that violate the certain design principles that are known as the defined design rules. The designing of the software system works based on these functional specifications and in restricted to

this regulation. When these rules get violated, then it is known that a design anomaly has occurred in the system. The occurrence of these design anomalies affects the software quality and its characteristics. It has a tremendous impact on the degradation of the quality and affects it negatively. Hence it is a necessity to identify and resolve these design anomalies as soon as we can use the source code or the SRS of the software. While developing the identification approach we need to have the required knowledge of the design anomalies firstly, and then recognize what type of design anomaly it is. Also, we need to consider the difficulties that it may cause if not resolved. Here, we present some of the common occurring types of design anomalies in the software.

A. Blob

The design of software consists of object-oriented diagrams (UML) for Java code or structural diagrams for C language code. These diagrams are made up of various classes. A Blob[5] is a type of design anomaly that gets detected in such software designs where there is just a single large class that operates and controls the entire behavior of the system. The encapsulation of the data occurs in other classes in the design of the software. Blob is understood further to be a type of a design anti-pattern which gets discovered in the software system. It gets identified as this large, complex, non-cohesive controller class that gets connected with other category data classes.

B. Spaghetti Code

Firstly we know that the design phase and the coding phase are related and occur one after the other in the software development cycle. Hence the dependency rate is very high on each other. If the code structures get restructured, it will change the design as well. The Spaghetti code[5] is a type of such code that affects the design to cause anomalies in it by altering the code from the coding phase. It consists of a twisted and nonstructured control structure. The design anomalies here get introduced due to procedural thinking in the object-oriented programs. Due to this, the anomalies get introduced in diagrams such as the UML that display the design of the software. Here the OO design diagrams are made up of various classes as well consisting of (1) significant schemes, (2) processes having no software metrics, (3) having no inheritance and lastly (4) defining global variables that determine the variation in the coding and design phases of the software.

C. Functional Decomposition

Functional decomposition[5] is yet another type of an anomaly in the system design. It falls in the category of an anti-pattern. They occur during a design situation that gets created in an OO diagram which consists of various classes. The main situation of its introduction is when a large complex class from these different classes get produced in such a manner that it will carry out just one goal and operation all the time. It gets identified in the source code in the coding phase of the Software. They get created in the OO structures due to the handling of non-experienced OO developers. The OO diagram here in the design of the software too like the earlier types consists of various classes. However, there's one single large class which consists of a great scheme which carries out the operations while other classes are idle. The large main class carries out a single specific function while other small classes carry out other functions or stay idle. This large main class contains various attributes within itself. These attributes carry out the operation of naming the various function performed inside the main class. This type of design anomaly gets introduced when the OO developers though maybe skilled but have a lack of knowledge about the design anomalies as well as the OO programming itself which is gets performed for detection of the anomalies in the OO programs. Here as the name suggest the large main class, gets decomposed into smaller classes with different functional names like Compute and Display that carry out different functions within the main class. Here the properties like inheritance and polymorphism do not get used more frequently

like expected in the large class and get associated with many other data types which are a similar case in the earlier seen design anti-pattern known as the Blob.

D. Swiss Army Knife

The Swiss Army Knife[5] is yet another design anti-pattern type. This kind of design anomaly consists of a single large complex class. This class carries out multiple functions and satisfies various needs in the class. Their examples are namely the Utility classes. As the name suggests in the real world, a Swiss Army Knife has many knives in it which can get employed for various operations. Similarly, here it can use the one complex class for the purpose of running different functions by using the interfaces that together create the larger main class.

E. Code Smells

Code smells[6] are the other category of design anomaly just like the first type which is the anti-patterns introduced in the design phase of the software. They occur in the coding phase initially and get observed in the source code which directly affects the design of the software. Earlier in the year 1998, the researcher Martin Fowler stated in his research about the code smells that they are the ones that indicate on their surface when there is a more profound or severe issue within the system of the software. On the bright side, we can also consider the code smells as the structures that indicate us whether there is a source code issue or breach that directly affects and violates the design rules and which can hugely degrade the software quality of the system as well. They do not stop the functioning of the system and do not obstruct the programming. Hence they cannot be called as software bugs or flaws as they are technically sound and they keep the functioning going on as well. Instead, they show us that the code structure is in violation of the defined design rules which will reduce the software development as anomalies will be introduced shortly by the changes occurring in the design. In the earlier research in the year 2009, a researcher called Robert Martin[7] looks at the code smells as a valuable entity in the software system, which is integral in the software development. It is due to their indication that the anomalies in the design could get identified and the software quality could be enhanced. The most common solution to handle this code smells is the method of refactoring. Here in this solution, the source code gets restructured for eliminating the anomalies in the design phase. But yet again it is done manually.

4. IDENTIFYING AND RESOLVING ISSUES

For improving the software quality in a system, the anomalies must be discovered and eliminated from the code and design of the software. Here we discuss the issues the identification and resolving tool or model faces while recognizing these design anomalies.

A. Issues with Identifying Design Anomalies

In the system design of software, there is no agreement for recognizing the design anomalies. There isn't enough knowledge about understanding whether the design pattern affects the design quality. It takes great knowledge to figure out if a design pattern violates the design rules and principles that are pre-defined. If such design situation occurs, it is hard to recognize. It is always helpful and beneficial to the system if we can identify the major harmful errors and make a list among various occurring faults and failures. However, among these constant occurring errors within the system what is an actual design anomaly needs to be determined. These design anomalies get measured by false positives, true positives properties of performance in the system. Firstly in the identification process, the detection of false positives is carried out because it causes the creation of rejection. The selected development teams perform this activity. Next step is going through the faults and failures recognized and used them to determining the major anomalies among them. Next is when the real anomaly candidates detected are

understood and how they could be resolved using the true positive measure. This approach can be beneficial for achieving better quality but very time and resource consuming. For example, suppose the identified anomaly candidate is a Blob. It is known to be an anti-design pattern. However, how do we identify it? It is firstly a part of a list multiple candidates of failures. This design anomaly violates the design rules[8] as well as has a God Class in it. This God Class is nothing but the one single large class that controls all the operations of the system while other classes just encapsulate the data. These other classes just sit idle and do nothing else. These all are part of OO diagram[9] (UML) creating a design of the software. Hence this is how the model or the tool has to identify and detect the anomaly candidate from the system design. Many anomalies get refactored from the coding phase itself and eliminated from the coding phase manually. However, the auto detection will save time and resources of the developers if considered. Hence agile methods like FDD and TDD are useful where automatic code correction is in play. In software organizations there is a deadline which the developer has to take care of in such situations such a model like the proposed one are very useful. This helps in enhancing the software quality to the satisfaction of the client as well. But there are a few questions that can arise during identification of an anomaly candidate. A discussion is a key solution for handling these arisen questions. This insightful discussion results in acquiring the required knowledge which would be helpful for the detection and elimination of the design anomalies. These issues need to get discussed here, and it is imperative to answer them.

1. How can a developer recognize from the design phase that if it is an anomaly candidate in real?

   In the design phase of software, there are few rules and principles used for designing the software. If these design rules get violated, then it is known that a situation has occurred where there is a failure in the design. There is no harmony decided for determining how it is an actual design anomaly candidate or rather it is just another minor error. Hence humans need to have enough knowledge about design anomalies to identify which ones are harmful to the system and are anomaly candidates in real. If the detected pattern is indeed an anomaly, then only the time is invested in correcting it which impacts in improving the quality of software.

2. Does making lists of anomaly candidates detect help?

   The detected anomaly candidates may be just minor failures or faults. These errors are all added to a list. The real anomaly candidates also get added to this list. If an anomaly selects the false positive measures, then the development team responds to it and rejects them. While if true positive metrics get decided, then it takes many periods for improving these anomalies also correcting them might not be effective at all on the quality improvement. Other uses of these lists of detected failures[10] are how to utilize them and understand the real detected anomaly candidates from them.

3. What is known as a boundary or boundaries in design?

   In the design phase, there is a certain boundary or limitation while creating a design for the software. If the certain limit gets violated, then it considered being a design anomaly affecting the system and its quality.

Suppose we extract an OO diagram of software design (UML) from Java code. Such diagrams consist of many classes that complete the UML design model. One of these categories is a God Class. The God Class is nothing but the largest and most complex class among the others. This class carries out all the functions while the other classes sit idle consisting of data. After such boundary gets crossed in the design, the developer observes the OO diagram to recognize the antipattern is a Blob. It gets determined from the God class observation. It is the use of the boundaries in the design.

The discussion can continue further and more such questions may arise such as:

1. A metric threshold value is needed to be defined as having a quality check in the design phase. How is it defined?
2. How can the design situations be addressed which are solely responsible for causing the design anomaly occurrence?
3. How does the recovery of data of the application take place for using it for identifying the design anomaly?

B. Issues with Resolving Design Anomalies

There are many concerns even after the design anomalies get identified by using the source code of the software as they are detected manually and a satisfying quality enhancement is not achieved here. The main problem is how to resolve these detected design anomalies. The most common solution used here is the refactoring method. Many existing models use this solution along with different programming algorithms of their choice. They try to eliminate the anomalies from the design phase using the source code which is in the next stage i.e. coding but they try to do it manually with the help of pair programming. The design anomalies occur in the classes that together create the design of the software. The refactoring method gets implemented for the correction of these classes. Restructuring of the code structure goes on in here, and as per the defined design principles, the design anomalies get eliminated. Restructuring means moving the structures in the source code. If the resolving takes place separately of the design anomalies from the classes, then, it might diminish the quality and many more attributes of the software even further. Hence the anomalies in the system design need to be handled in a global manner. Refactoring is an instant method of solution which can be a sometimes quicker solution than most solutions as the code of any software can get extracted for checking the anomalies in it. However, the major issue is that sometimes even after refactoring gets carried then the anomalies are not eliminated, and the quality of the software is still degrading in the system as the human is bound to miss a few anomalies with his eye rather where the automatic detection will not. Hence it causes a waste of all the time and resource invested for resolving of the design anomalies. These real anomalies befall in the system design where it shows the proper relationship between the design phase and the coding phase. All the models focus on the elimination of design anomalies from the code, which is rightly so but the humans participating here cause for lot of time wasting. These issues are our prime focus of concentration for research. They will be resolved in our ongoing research and will be presented to all in the coming future where these shortcomings will be considered to save a lot of time of the developer and give maximum quality enhancement by handling the anomalies from code affecting the design of the software.

5. DISCUSSION ON EXISTING MODELS

Most of the existing models that carry out the detection and correction of the design anomalies propose new solutions to do so. However, there are various shortcomings in them as well that limit them in few aspects for the elimination of anomalies in system design. Hence we carry a comparative analysis between the existing architectures before proposing our novel approach to eliminate their shortcomings.

In the year 1998, there were researches carried on the elimination of the anomalies that occurred in the maintenance processes of the software evolution. This research was proposed through the work by Brown et. al., He emphasized through his research that the maintenance of the software during its developing stage or after it got deployed to the customer for use consumes almost 90% of the total budget of the entire estimated cost employed in the project.

In the year 1997, which is one year prior to Brown's research. Researchers Fenton and Pfleeger mentioned a very important statement in their work it said that design anomalies may not necessarily cause the system failures directly, they may cause it indirectly as well. They both proposed how to detect the anomalies that occur during maintenance and how to correct these detected anomalies. The solution they proposed was quite simple which was carrying changes in the coding phase for eliminating the anomalies for improving the quality. The term code smells reported by them as well. They found out the how the introduction of the code smells in the coding phase of the system would affect the design of the software.

In the year 1999, this is one year after Brown's research. The researcher Fowler coined the term anti-pattern. The anti-design patterns like Blob, Spaghetti Code, and Functional Decomposition were all reported by Fowler in 1999. He stated that blob was the anti-pattern that got identified when large classes were part of the design, Spaghetti code was the anti-pattern that got identified when the code structures get manipulated, and their control was tangled. Moreover, lastly, he stated that Functional Decomposition was the anti-pattern if the large class carried out only one function while others carried other functions on data encapsulation.

In the year 1999, research proposed by Fenton and Neil was based on a prediction of anomalies[11]. It was the solution given out by them that the anomalies got detected in the design or the code, and they could get predicted before with the use of certain parameters. These software metrics used were on the size and complexity which predicted the anomalies from the code or the design of the software. Predictors are also known as the independent variables that along with the dependent variables get used for estimating the relationship between the coding and design phase. Moreover, also using the Line of Code and The McCabe's cyclomatic complexity[12] these parameters help in predicting the anomalies in the development or code of the software.

Three of these equations shown below comprise of the parameters such as LOC and complexity used for the identification of the design anomaly. They have got stated in many of the earlier works of the researchers.

1. Anomaly Detection in the Design = 4.86 + 0.018 LOC[12]
2. Anomaly Detection in the Design = 4.2 + 0.0015 LOC 4/3[12]
3. Density of the Anomaly in the Design = No. of Total Anomalies/LOC[12]

Based on the earlier mentioned researches many researchers have come up with their solution and models of their own in the 21th Century.

In the year 2009, Liu et. al.[13] proposed in his research work that the detection and correction of the anomalies in the maintenance process are tough tasks. Also, it is time-consuming if it gets done manually and also consumes the resources if hand operated. The number of actual anomalies in the design is of more than the real resources used to detect them. In the 21st century, Liu et. al., along with other researchers Moha et. al.[14], Marinescu et. al.[15], Khomh et. al.[16] proposed in their respective research works that had a technique which gets based on specified design rules. When these design rules are violated, then these signs or situations are considered for the detection of the anomalies in the design. They used metrics such as quantitative, structural and lexical for the detection purposes. In the same year, Moha et. al., published his research which proposed that there be detection rules that if violated indicates show there are possible design anomalies. He was the one who created these rules for use for other researchers as well. He used formal definitions for the creation of the detection principles.

In the same 21st century during different years, these same researchers Moha et. al., Liu, Marinescu, Khomh et. al., and Kessentini[17-22] proposed research work that consisted of various automated techniques used for the discovering of the design anomalies.

Researches were presented by O'Keeffe and Cinneide[23] in the year 2008 and also Harman and Tratt[24] in the year 2007 a year earlier that is. They proposed refactoring as their main solution or rather used this technique to enhance the software quality and eliminate the design anomalies using the source code. Refactoring is a technique that works in the coding phase by restructuring the code to remove the correlated design phase anomalies. However, few questions are arising from this research such as:

1. How to define and select the parameters for detecting the anomalies and improving the software quality?
2. How would we combine two software parameters and use it for detecting the design anomalies?
3. What will be the appropriate refactoring exposition which will help in enhancing the quality of the software?

These questions do not get answered in many previous types of research. Some of them are the techniques or models proposed by Fowler et. al., in the year 1999, by Liu in 2009, by Mens and Tourwe[25] in the year 2004, Moha et. al., in the year 2010, Sahraoui et. al.[26] and the other researchers in the 21st century. All these models fail to satisfy the questions mentioned above.

In the 21st century architectures, the solution techniques used for the correction and the detection of the design anomalies get divided into two categories. First was the technique based on the design rules used for detection and screening for correction and second was the visually based method used for detection and correction. The first type consisted of many models that are based on the design rules. In the year 2004, Marinescu et. al., proposed a set of design rules which were made up of various parameters. By using this regulation, the detection, and correction of the anomalies were getting done. It got carried on the OO design diagrams which consisted of the classes and system with sub levels as well.

In the year 2010, Moha et. al., proposed a very popular strategy called as DECOR. This method comprised of the general rule language on its basis the anomaly marks got specified. It was his detection strategy. In the next year Khomh et. al., carried further this DECOR strategy. He proposed that how the DECOR method could be used to sort the detected anomaly candidates and support uncertainty in the system. In the year 2010, the researcher Kessentini et. al., stated a new mechanism that was part of the search technology. This technique detected the design anomalies automatically in the code itself. These were all models based on the first category. The techniques based on the second type of visualization based method get proposed as follows. In the earlier the year of 2004, Kothari et. al., proposed a framework which was pattern based. This framework was used for developing a detection and correction tool that supported the detection of the design anomalies by providing a solution of outlining the design anomalies candidates along with mixed colors. After four years in 2008, this work was carried forward by Dhambri et. al.[27] He proposed a visualization based technique for the detection of the design anomalies. This technique was on the automated, and it detected the anomaly candidates based on the design signs or situations that indicated the presence of an anomaly in the design. A human intervention was also carried here where along with the machine the human also checked if the detected signs are indeed the real anomaly candidates in the design. These techniques of rule-based and visualization based used for detection and correction of the design anomalies is not satisfying as they cannot be helpful in carrying their work in the large-scale software systems. Hence these models can be testing still as they would not detect all the anomalies in the system in such situations. Hence while estimating or predicting anomalies, these models fail. While the models using refactoring solution, don't take into mind the efforts while using this technique as manually detecting is wastage of precious time. All these shortcomings will be dealt with in our research in coming future.

A Novel Approach to Enhance the Software Quality by handling the Design Anomalies

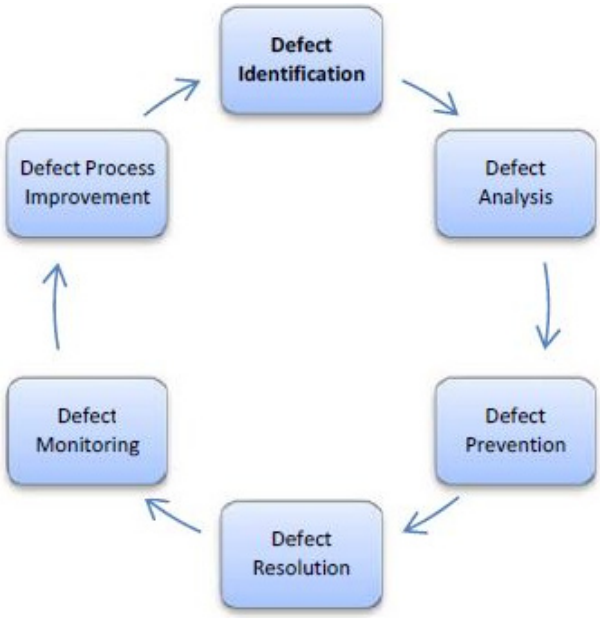

Figure 2: Anomaly Detection Approach Workflow

Earlier we have seen the various shortcomings in the existing models. For the enhancement of the software quality as well as reliability, the anomalies should get eliminated from the system design. For enhancing the quality, the failure rate should get reduced in the software. It can be achieved by using an innovative approach to software development as well as improving the software processes by identifying the anomalies. Hence a novel approach is proposed here which tackles all the existing problems. Many previous approaches try to remove the anomalies directly from code manually, or some try to eradicate it from directly design. However, here we will work in two phases the Coding and Design phases where we try to remove the anomalies directly from the coding phase but the only difference is they get detected automatically unlike the existing models. The following mentioned software metrics[28] boost chances of enhancing the quality of the software as well as the existing models.

1. KPA (Key Process Area)
2. Agile Models (eg.FDD and TDD)
3. Quality Gate
4. Traceability Matrix
5. Source Code and its relationship with design
6. Reverse Engineering

Let us take a better look at our proposed approach and the metrics we will use.

A. KPA (Key Process Area)

The first metric used is KPA. The proper definition of KPA[29] is that it is the Answering Per Criteria % upon the Questions asked and the answers not answered. The KPA carries out the solving of the average success % of each phase namely design and coding in the software process cycle.

$$\text{KPA} = \frac{(\text{No. of answering per criteria})}{(\text{No. of questions} - \text{No. of N/A answer})}$$

The KPA consists of a group of correlated works that are part of the same range. They are put together because their aim is the same, and they have the same purpose or goal to get carried. This purpose can be towards the improvement in a certain field. For example, we try to improve certain areas such as the existing process models and the quality of the software. All key process areas are well-known to both connected and staged illustrations. The organization selects the process areas in which it wants to carry out the improvement. A process model is the one that is used to create the software product. It gets divided into various stages. The company selects the areas or the correlated process operations for improvement that will be beneficial for their business and gain profit to them with the creation of the perfect software product.

The organization has the freedom to choose whatever process areas to improve in a software project. Once the process areas to improve are selected then the company can plan how to improve them accordingly to give out better quality software which will satisfy the customers as well as the stakeholders.

B. Agile Models

Here we try to integrate our proposed metrics with an existing agile model to give out a better model. Our focus is more on enhancing the software quality and reliability[30]. To achieve this result, we use traceability matrix to trace back from the coding phase to the design phase. Why integrate with the agile model and not the SDLC model? It is because our main focus is to eliminate the anomalies even from the coding phase of the software development process model and not only just the design phases. Moreover, the agile models like Feature Driven Development (FDD) and Test Driven Development (TDD) give an extra edge over the SDLC models for concentrating on the coding phase optimizing when needed and eliminate the bugs from the design phase. They consist of refactoring and restructuring of the code to improve the design of the software. We create a auto detection model by integrating with these agile methods to give out a satisfying model for the software developers. Hence we try to integrate our metrics into the agile model to give out a better model different from the earlier models proposed. It will provide us with a better software quality than the metric threshold value set. Moreover, also we achieve improvement in the software processes.

C. Quality Gate

How do we check for consistency in the Coding or the Design? How do we know if there are any anomalies in the code or the design? It is verified using a metric called Quality Gate[31-32]. We apply quality gates between the two phases for checking if the quality satisfies the metric threshold value which gets set for quality checking purposes. If not we trace back to the earlier phase for eliminating the anomalies in that phase. Let's emphasize and learn what Quality Gates are. Quality Gates get applied before a phase to check the quality % in the phase. Quality Gates are more helpful than applying milestones between the phases. They are benchmarks applied between the phases before the software project gets started. Each Quality Gate comprises of the necessary information about the phase and its quality %. In the quality gate, contrary to a software analysis, a quality check is carried which is only formal and is not an in-depth check which gets carried by the contents of the documents based on their relevance. It consists of a set of information preserved about the phase, which gets detailed in a document called as checklist. The document of checklist itself gets used in a meeting with all developers or the organization authorities that make the decision for the benefit of the customers and stake holders. For carrying out the quality audit, a metric threshold value % is set to satisfy the quality % in the phase. If the quality % does not meet the set value, then the tracing back is done using the reverse engineering tool. It all depends on their

decision after checking the quality in the phases. The project gets rejected if the resources and time consumed to correct it will be more than the estimated budget. Alternatively, the project can get put on hold for the possible detection and correction of the design anomalies. Moreover, lastly, it gets approved to continue as per regular schedule if the quality % satisfies set threshold value in the software phase. Here we directly check the quality in the coding phase and eliminate the bugs directly from here to give better code and design quality as well. The quality gate will be applied between these two phases to inter-relate these phases as in when we eliminate bugs from code it will automatically improve quality in the design.

$$\text{Quality Improvement (Anomaly Elimination)} = \frac{\text{Anomalies eliminated during development}}{\text{Anomalies existing in the product}} \times 100\%$$

D. Traceability Matrix

The most important metric other than the Quality gates in our approach is the traceability matrix. We employ the traceability matrix for tracing back from the coding phase to the design phase. We reverse engineer from the coding phase trace back to the design phase, and likewise, we try to eliminate the design anomalies then from each step if the threshold value is not satisfied. So let's understand what this metric truly is. A traceability matrix is nothing but a record. It gets frequently used in the form of a table that stores relationship data. It links the two phases of coding and design along with each other which are helpful to identify the anomalies from them. It also contributes to check whether the current software project quality is getting satisfied according to set metric threshold value or even to keep a quality check during a new software project commences and the software is getting created as per estimated quality.

E. Source Code and its Relationship with the Design Phase

The first phase we come in contact with in our approach is the coding phase. The source code gets refactored, and the optimization of the quality can be traced back towards the design phase. Hence the code of origin is an important metric for detection and the elimination of the design anomalies for quality improvement. Many types of software consist of anomalies that affect the quality and also hence it is working. Its code will get extracted, and from its code, we can reach its design phase for the elimination of the anomalies. Each time we have to check for improvement check we check it with the quality gate where the threshold value % will get set. It is how we can trace back from the source code to the design for maximum complete elimination of the software bugs for improving the software quality and achieving improvement[32] in the existing process models.

F. Reverse Engineering

The most important metric is the software tool which helps from moving from the coding phase towards the design phase. This is known as the Forward engineering or the Reverse engineering tool[33]. It helps in extracting the source code of the software. Then if the quality % does not satisfy the set threshold value on the quality gate then the using this tool and traceability matrix we can trace back to the design phase. It also helps in extracting the Object Oriented diagram (UML) from the Java code.

6. FUTURE WORK AND CONCLUSION

In this article, we completed a survey on the code and design anomalies in the software system and proposed a novel approach for identifying and resolving them. We studied about the code and design anomalies, its various types, the shortcomings of the existing process models and in the end, we proposed a novel approach to improve the current process models using the certain metrics that we stated above and also to enhance the software

quality[34-36]. In the coming days, we will implement this novel approach through our ongoing research where we cover all the limitations of the earlier presented models. The results and implementation will be shown shortly in our next article. Hence it will automatically improve these existing software processes models by integrating these metrics in them. Moreover, also it can be used to develop the software product quality and reliability by controlling the failure rate and the anomaly occurrence in the models.

REFERENCES

[1] Fenton N, Pfleeger SL. Software Metrics: A Rigorous and Practical Approach. ACM DL. 1998.

[2] Fowler M, Beck K, Brant J, Opdyke W, Roberts D. Refactoring: Improving the Design of Existing Code. 1999.

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

[4] Brown WJ, Malveau RC, Brown WH, McCormick HW, Mowbray TJ. Anti Patterns: Refactoring Software, Architectures, and Projects in Crisis. 1st edn. Wiley, New York. 1998.

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

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

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

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

[9] Ragab SR, Ammar HH. Object oriented design metrics and tools a survey. The 7th International Conference on Informatics and Systems (INFOS), Cairo. 2010; 1-7.

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

[11] Fenton N, Neil M. Critique of Softw. Defect Prediction Models. IEEE Trans. Softw. Eng. 1999.

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

[13] Liu H, Yang L, Niu Z, Ma Z, Shao W. Facilitating software refactoring with appropriate resolution order of bad smells. In: Proc. of the ESEC/FSE. 2009; 265–268.

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

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

[16] Romano D, Raila P, Pinzger M, Khomh F. Analyzing the Impact of Antipatterns on Change-Proneness Using Fine-Grained Source Code Changes. IEEE Computer Society, ACM DL. 2012.

[17] Kessentini M, Sahraoui H, Boukadoum M, Wimmer M. Design anomaly Detection Rules Generation: A Music Metaphor. IEEE Computer Society. 2011.

[18] Ghannem A, Kessentini M, El Boussaidi G. Detecting model refactoring opportunities using heuristic search. ACM DL. 2011.

[19] Ouni A, Kessentini M, Sahraoui H, Boukadoum M. Maintainability defects detection and correction: A multi-objective approach. Springer. 2013.

[20] Kessentini W, Kessentini M, Sahraoui H, Bechikh S, Ouni A. A Cooperative Parallel Search Based Software Engineering Approach for Code Smells Detection. IEEE Trans. Softw. Eng. 2014.

[21] Ghannem A, El Boussaidi G, Kessentini M. On the use of design anomaly examples to detect model refactoring opportunities. IEEE, Softw. Qual. J, Springer. 2015.

[22] Ouni A, Kessentini M, Sahraoui H, Inoue K, Deb K. Multi-criteria Code Refactoring Using Search-Based Software Engineering: An Industrial Case Study. ACM Transactions on Software Engineering and Methodology. 2016.

[23] O'Keefe, M, O'Cinneide M. Search-based software maintenance. In Proceedings of the 10th European Conference on Software Maintenance and Reengineering. 2006; 249-260.

[24] Harman M, Tratt L. Pareto Optimal Search-Based Refactoring at the Design Level. Proc. Genetic and Evolutionary Computation Conf. 2007; 1106-1113.

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

[26] Sahraoui H, Abdeen H, Bali K, Dufour B. Learning dependency-based change impact predictors using independent change histories. Information & Software Technology. 2015; 67, 220-235.

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

[28] Abdeen H, Shata O. Metrics for Assessing The Design of Software Interfaces. Int. J. of Adv. Res. Comput. Commun. Eng. 2012 December; 1(10), 1-8.

[29] Patil TB, Joshi SD. Software Improvement Model for small scale IT Industry. 2015.

[30] Kaur R, Sengupta J. Software Process Models and Analysis on Failure of Software Development Projects. 2011.

[31] Koul S, Global J. Enhancing Project Management – A Quality Gate Usage. 2013.

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

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

[34] Naveenkumar J, Bhor M, Joshi SD. A Self Process Improvement For Achieving High Software Quality. 2011.

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

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