Annals of R.S.C.B., ISSN:1583-6258, Vol. 25, Issue 4, 2021, Pages. 2974 – 2989
Received 05 March 2021; Accepted 01 April 2021.

# A Hybrid Framework to Control Software Architecture Erosion for Addressing Maintenance Issues

**Dipra Mitra[1], Mohit Arora[2*], Manik Rakhra[3], Challa Rushith Kumar[4], Mallu Leeladhar Reddy[5], S. Praveen Kumar Reddy[6], Chandan Kumar[7], Mohammad Shabaz[8]**

[1]Department of Computer Science and Engineering, Chandigarh University, India.

[2*]School of Computer Science and Engineering, Lovely Professional University. Phagwara, India.

E-mail: mohit.15980@lpu.co.in

[3]School of Computer Science and Engineering, Lovely Professional University. Phagwara, India.

[4]School of Computer Science and Engineering, Lovely Professional University. Phagwara, India.

[5]School of Computer Science and Engineering, Lovely Professional University. Phagwara, India.

[6]School of Computer Science and Engineering, Lovely Professional University. Phagwara, India.

[7]School of Computer Science and Engineering, Lovely Professional University. Phagwara, India.

[8]School of Computer Science and Engineering, Lovely Professional University. Phagwara, India.

**ABSTRACT**

Software architecture erosion is an alarming issue encountered by many organizations in the software industry. It occurs when 'as-implemented' structure does not conform to the 'as-intended' structure, which leads to low quality, maintainability and complexity in software architecture. Architecture erosion attacks system's skeleton structure and thus making it vulnerable to errors. Two architectural framework paradigms have been articulated in this paper wherein one based on spring Model View Controller hibernate and other on simple Java code. Based on design principles such as separation of concerns, single responsibility principle, principle of least knowledge and CASE tools i.e. SonarQube and JArchitect, these two architectures have been used in this proposed work to find the bugs, vulnerabilities, cyclic-dependencies and architectural violations. The system architecture thus created ensures smooth functioning and least maintenance associated effort and cost.

**KEYWORDS**

Software Architecture, Architecture Erosion, Cyclic Dependencies, Architectural Rule Violations.

## Introduction

Software architecture is the major structures of a software system, and it focuses on how numerous software processes uphold to accomplish their functions or tasks. The architecture of a system describes its extensive constituents, their interconnections, and how they work together with each other. It works as a model and blueprint for a system. It controls the system complications and settle an intercommunication and collocation mechanism amongst constituents. It outlines top down design of big software systems with their complete architecture in an ideal and orderly manner. The main objective of a software architectural representation of a system is categorizing the system's significant constituents, discovering resemblance among the constituents and distinguishing them accordingly. These components are pieced together by adapters that deliver the match or interrelation among unlike constituents. The connectors may turn out to be elements themselves, playing a central role in characterizing one suitable architectural styles amongst other thus having paramount effect on the characteristics of an appropriate style. Software structure comprises of frozen spots and hot spots. Frozen spots define the general architecture of a product framework, in other words its fundamental segments along with the relationship among them. These stay unaltered (frozen) in any instantiation of the application structure. Figure 1 depicts the software architecture lifecycle. Hot spots signify those parts where the programmers using the framework add their own code to add the functionality particular to their personal project. Framework is an application that is finished aside from the real usefulness, you connect to the usefulness and you have an application, they are extremely valuable to designers.

Annals of R.S.C.B., ISSN:1583-6258, Vol. 25, Issue 4, 2021, Pages. 2974 – 2989
Received 05 March 2021; Accepted 01 April 2021.

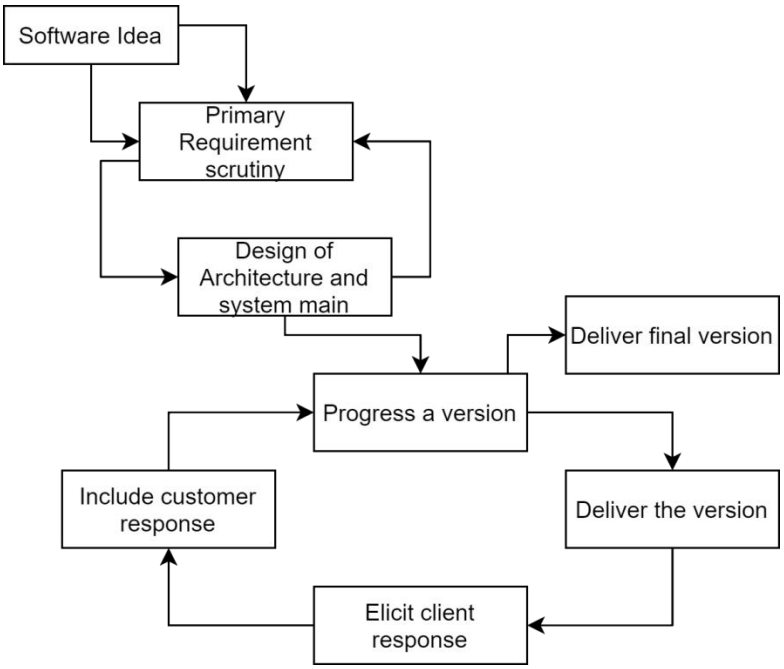


**Fig. 1.** Software Architecture lifecycle

It contains the entire thing you have to make an application. In reality you can frequently insignificantly make an ostensible application with not very many lines of source that does literally nothing yet it gives you window administration, sub-window administration, set of decisions, catch bars, and so forth. In an object-oriented environment, a structure comprises of unique and substantial classes. Instantiation of such a framework includes shaping and sub-classing the present classes. When working up a solid software framework with a software framework, creators utilize the problem areas as per the particular requirements and necessities in accordance with the system. Software frameworks depend upon a known principle called the Hollywood Principle: "Don't call us, we'll call you" which is suggestive of the fact that the client characterized classes (for example, novel subclasses), receive messages from the predefined structure classes. Architects by and large handle this by executing superclass dynamic techniques. For instance, numerous Java frameworks. Architecture Essential activities are Architecturally significant requirements, Analysis, Synthesis and Architecturally evaluation. There are four basic exercises in software architecture plan. These fundamental engineering exercises are accomplished iteratively during various phases in early software improvement life-cycle, and also during the advancement of a framework.

i. **Architectural Analysis** is the strategy of perceiving the environment where the arranged system(s) will work together and deciding the systems prerequisites.
ii. **Architectural Synthesis** or design is the way toward making architecture. Given the necessities controlled by the examination, the present condition of the outline and the results of any assessment exercises, the plan is made and made strides.
iii. **Architecture Evolution** is the procedure of maintaining and adjusting current programming design to meet necessity and ecological changes. As software design gives a software system's critical structure, its advancement and upkeep would basically affect its essential structure. Thusly, design advancement is worried with including novel usefulness and also keeping up current usefulness and framework execution. Architecture requires unsafe supporting exercises. These supporting exercises occur all through the center software architecture process. They comprise of information administration and correspondence, plan thinking and decision making, and documentation.
iv. **Architecture supporting activities** are the approved activities for software architecture through basic software architecture exercises. These exercises bolster a product modeler to finish examination, union, evaluation and movement. For example, a draftsman needs to assemble information, settle on decisions and report amid the investigation stage.

Some present frameworks rule specific territories of use headway, for instance, JavaScript/CSS systems that embed

Annals of R.S.C.B., ISSN:1583-6258, Vol. 25, Issue 4, 2021, Pages. 2974 – 2989
Received 05 March 2021; Accepted 01 April 2021.

the application's view through its presentation layer and there are others that handle a more noteworthy measure of the active parts of the presentation. Instances of structures that are starting at now offered by benchmarks bodies or associations incorporate.

i. **Resource Explanation Framework**, an arrangement of rules from the World Wide Web Association for in what way to characterize some Internet asset, for example, a Web webpage in addition to its substance.
ii. **Internet Business Framework**, a gathering of programs that frame the mechanical establishment for items from SAP, the German organization in line by business sectors to serve a venture asset administration line of items.
iii. **Sender Policy Framework**, a well-defined methodology as well as programming for making e-mail additional safe and sound.

Software architecture erosion or disintegration is determined as the aspect that happens when the executed design of a product framework go amiss from its normal design. The actualized architecture is the model that has been executed or worked in inside low-level plan builds and the source code. The term solid design additionally alludes to the implemented architecture [1]. The proposed architecture is the result of the design configuration handle, otherwise called the conceptual architecture [2] or the arranged design. The deviation itself is not caused by venomous human activities yet rather by routine repair and change work normal for a developing programming framework. The general impact of structural design choices as well as the exchange offs between individual qualities is inspected by a planner. The software architecture ought to just speak to the structure of the framework by concealing the execution subtle elements in addition to controlling both the quality trait as well as the practical necessity.

During the lifespan of some usual software system it experiences advancement plus making of various prescriptive and clear architecture at various circumstances. On the off chance that a man doesn't have sufficient learning about what the actualized and proposed engineering is then the likelihood of the event of software disintegration turns high. The impact of architecture disintegration causes the dis-fulfilment of partner's prerequisites as the progressions wind up noticeably hard to utilize on the product and in the most exceedingly bad, it can even prompt disappointment of programming undertakings. Practically every other venture experience disintegration at some stage in software improvement cycle unless some exertion is done to conquer it. In another normally referred to case of design disintegration, Godfrey and Lee [3] depict their examination of the extricated models of the Mozilla web program (further developed into Firefox) and the VIM content manager. Both product items demonstrated substantial number of undesirable interdependencies existing among their center subsystems. Actually, due to gravely dissolved design of Mozilla, noteworthy postponements delayed the arrival of the item and constrained engineers by having them revise some of its center modules starting with core modules from scratch. Architecture disintegration causes numerous bugs in programming, for example, increment in inner unpredictability with the expansion of new usefulness, developing time to change the product and time-to-market, diminished quality, expanding the test exertion for upkeep of programming and so on in the meantime lessening the designer's efficiency as much period is spent on understanding the difficult present portions of the software. The final product is: costs rises and efficiency falls. It turns out to be yet further costly when software disintegration brings about a "product avalanche", when the measure of disintegration achieves a point where the product can't be kept up or enhanced any further and modify turns into the main arrangement, with all the utilized expenses and dangers. As the circumstance gets most exceedingly terrible, the main conceivable alternative remain is to fabricate the product sans preparation or at the end of the day "Rework" the product however this choice is extraordinarily expensive and dangerous with respect to due dates or spending plan. As revising includes the new programming to accomplish the majority of the usefulness that the current programming has hence no time is left to make a change in the product putting both the venture and association on stake. That is the reason the product, and for the most part its engineering, must have the capacity to manage various solicitations for change to forever remain in working condition. Architecture disintegration prompts the steady of the engineering quality of software systems. With the end goal of this overview we characterize designing quality as an assumption of compositional honesty (i.e. fulfillment, rightness and consistency), conformance to quality characteristic prerequisites, and selection of sound programming building standards. Building nature of a framework may not generally liken to system performance. A well-performing system may be plagued by seriously eroded architecture. Be that as it may, such a framework is amazingly delicate and has a high danger of separating at whatever point alterations are made. The most common types of software architectural disintegration consist of,

Annals of R.S.C.B., ISSN:1583-6258, Vol. 25, Issue 4, 2021, Pages. 2974 – 2989
Received 05 March 2021; Accepted 01 April 2021.

- **Architectural Rule violations-** For re-architecting or encourage advancement a few plan principles ought to be taken after e.g. maintain a strategic distance from strict layering between subsystems.
- **Unreachable Code-** It is also known as the dead code which is not ever executed nor required for any utilization but rather it is as yet messing the code base contributes towards structural disintegration.
- **'Copy & Paste' Codes-** In spite of the fact that code duplication is well known with the end goal of reuse and execution productivity as duplicate glue is the most well-known strategy yet as the size builds the practicality cost increments, for example, a settling a blunder or alteration in one clone case is probably going to must be dispersed to the next clone illustrations.
- **Metric outliers**- Include further class progressive systems, huge bundles and complex code.
- **Dependency-** Between bundles and modules lessens reusability, blocks upkeep, anticipates extensibility, constrain testability and limits a designer ability to comprehend the results of progress.
- **Cyclic Dependencies-** Are the most exceedingly terrible kind of disintegration. Phases tend to snitch into plan. For example, if A and B are put in an alpha bundle, and one is set in a various bundle, a cyclic reliance amongst alpha and numbers exists despite the fact that the class structure is a cyclic. They ought to be repaired or promptly wiped out as they wind up in delicate code.

There are certain side effects that show disintegration in architectural designs. They are:

- **Inflexibility-** It rolls out the product hard to improvement as a change can cause infringement in ward modules hence surpassing an opportunity to play out that change, along these lines the administrator's dread so much that they in the long run deny to permit any adjustments in software.
- **Brittleness-** It is firmly identified with rigidity making the product crack each time it is altered henceforth the administrator's dread that the product will burst in some unforeseen way at whatever point they endorse a fix driving towards exorbitant improve. Such programming is not reasonable to keep up as they turn out to be most exceedingly bad as each alteration and bug settle takes significantly more. In such cases, the designers lose the control on their product and it turns out to be truly difficult for them to work through such software in addition to there is compel to revamp the software.
- **Serenity-** It is the inability to recycle segments from identical or diverse software ventures as the vast majority of the product includes much comparative kind of modules composed by different designers. Tranquility shows up when the engineers discover that the work and hazard important to part the needed parts of the product from the undesirable parts are too huge to acknowledge thus the software is just modified rather than reused.
- **Reduced Effectiveness and Efficiency-** because of deferral in software discharges, spending overwhelms quality bugs and so on.

There are a few dangers connected by programming design for instance in the improvement group where novel contracts might not comprehend the framework as well as old representatives need to buckle down, not contradicting the anxiety, brings about high turnover which is a reason for structural consumption as the learning of engineering is lost when they assent. So also, unbendable software is another issue as it is extremely hard to improve and grow it. Quite possibly an alteration can even origin a presentation of novel bug in software along these lines the product must be exceedingly viable or repairable. The advancement group additionally has not a steady association with the product's life as there is a probability that some part can left the group plus the learning of the architecture and software related with them likewise vanishes. Software Architecture disintegrates increasingly when new contracts commit errors and set aside much opportunity to catch up the venture promptly. Engineering will additionally disintegrate when the new contracts sufficiently lacking learning about design would try to make changes to the framework. Figure 2 represents how to control.

Annals of R.S.C.B., ISSN:1583-6258, Vol. 25, Issue 4, 2021, Pages. 2974 – 2989
Received 05 March 2021; Accepted 01 April 2021.

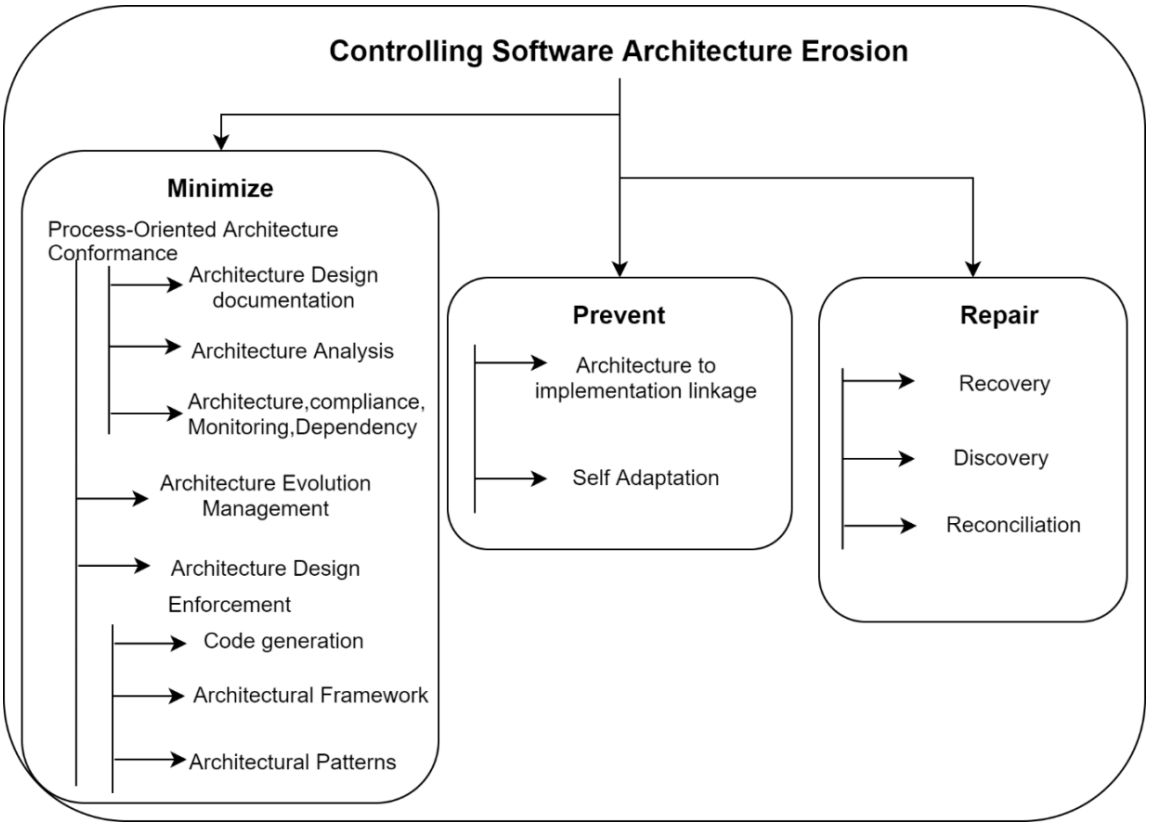


**Fig. 2.** Controlling Software Architecture erosion

## Related Work

De Silva and Perera (2015) propose some of tools which help to overcome from issue of disintegration. The tools which is used to overcome from this issue is static engineering conformance checking device. This tool identifies building limitations pollutions and along these lines help to maintain a strategic distance from the product plan disintegration. This device depends on GRASP ADL. Handle is a printed engineering portrayal dialect fit for taking the "justification" behind building outline decisions. It underpins an arrangement of compositional parts for catching the engineering of framework. e.g. System Element, Layer Element, Component Element, Link, Element, Connector Element, and Interface Element [4]. Terra, Valente et.al (2012) recommends the essential plan of a proposal framework whose rule explanation behind existing is to give refactoring principles to engineers. The paper formally delineates the approach which offers proposals to evacuate architectural infringement recognized by DCL (Dependency Constraint Language) which is space particular dialect and is utilized for depicting the auxiliary limitations among the modules in programming framework. This dialect is straightforward, simple reasonable sentence structure. Conformance apparatus is utilized which is dclcheck and dclfix [5]. Kamran Sartipi (2003) recommends a pattern-based recuperation philosophy whose goals can be determined as far as the structural properties that are very much characterized through an architectural pattern. The proposed architectural pattern is fixated on Architecture Description Language (ADLs). The result of the recuperation can be specifically tried in opposition to the recuperation destinations through: conformance checking with the available documentation that affirms the decay of the fundamental framework usefulness into segments, deciding the particularity nature of the recouped design to affirm the recuperation of a viable framework, conformance with the segment and connector size and sort limitations do by the pattern [6]. Pruijt and Brinkkemper (2013) propose the Architecture Compliance Checking (ACC) is a way to deal with confirm he conformance of executed program code to abnormal state models of engineering outline. ACC is utilized to forestall building disintegration amid the advancement and development of a software system. Static ACC, based on static software analysis techniques, focuses on the modular architecture and especially on rules constraining the modular elements [7]. De Silva and Balasubramaniam (2012), “Controlling Software Architecture Erosion” recommends the methodologies and advancements proposed throughout the year for regulation of software architecture disintegration or to distinguish

Annals of R.S.C.B., ISSN:1583-6258, Vol. 25, Issue 4, 2021, Pages. 2974 – 2989
Received 05 March 2021; Accepted 01 April 2021.

and re-establish design that have been dissolved. These methodologies incorporate the instruments, systems and procedures, which are grouped into three classifications viz., limit, counteract and repair design disintegration. These procedures are as follows – process-oriented architecture congruence, architecture design enforcement, architecture evolution management, self- adaptation, architecture to implementation linkage, and architecture restoration techniques with a recovery compilation [8]. Herold, Counell et.al (2015) "Detection of Violent Causes in Reflection Model" describes that Reflection Model is a technique used to detect architecture violation that occurs during software architecture erosion. This paper focuses on the technique which will help us to automatically detect the causes of violation of reflection model and detection of typical symptoms for these causes. Reflection model tools JITTAC. Common causes for violations in a reflection model: Architecturally misplaced software units and Call-back. Symptoms for violation causes are: Structural Symptoms (It basically describe the structural pattern in reflection model, the mapping and the code) and Measurable Symptoms (It describes the quantifiable properties of the elements in structural symptoms) [9]. Petrov et.al (2011) proposes that another systemic strategy for examination and plan of architecture design that addresses some real constraints in the present best in class. It presents another product engineering strategy that methodologies the product design area with systemic procedures and we demonstrate expressly relevant considers software architecture definition. It presents the idea of example for the relevant condition, which serve close by engineering designs as the essential vocabulary for architecture depiction and examination. Our investigation approach utilizes a probabilistic demonstrating and choice formalism to guide software architecture evolution. Patterns are a fitting device for measuring the dynamic ideas appropriate to architecture, for example, "calculated uprightness" and "wellness for reason" as opposed to breaking down those architectural concepts to a smaller scale level determination with the end goal of estimation and control [11]. Chanda and Liu (2015) proposes that the gathering of partners trading their perspectives with a specific end goal to settle on plan choice since architecture rationale behind different outline choices is not completely caught and henceforth influences the practicality of software system. Keeping in mind the end goal to catch and keep up the Software Architecture rationale for examination a keen software architecture rationale basis catch framework has been composed that enables distinctive member or partners to take an interest in an online discourse to determine configuration issue cooperatively. This paper utilizes three distinct methodologies. Right off the bat, we decide aggregate assessments of a gathering on various perspectives and recognize perspectives which have picked up a huge consideration into the online exchange. Besides, propose a technique to build up a traceability network that connections different software architecture components to its related software necessities. Thirdly, we perform printed investigation of partners' perspectives to decide the points that are generally talked about [13]. Caracciolo, Lungu et.al (2015) suggests that software architecture disintegration can be controlled by documenting design decision and using architecture description language (ADLs) techniques [14]. Maqbool and Babri (2004) proposes the utilization of Bayesian learning technique for automatic recovery of as software system's architecture, given fragmented or obsolete documentation. In this utilize programming modules with known characterizations to prepare the Naive Bayes classifier. After this we utilize the classifier to put new cases, i.e. Software modules, into appropriate sub-systems. At that point they will assess the execution of the classifier by directing investigations on a product framework, and contrast the outcomes got and a manually prepared architecture. To thinking and finishing up results Bayesian learning utilizes the likelihood-based approach. Bayesian learning strategies depend on Bayes hypothesis. One of the Bayesian learning method is Naïve Bayes classifier that has been successfully applied to solve many problems like text classification, speech/image recognition [15]. Dargomir and Lichter (2012) proposes an automatic assessment of software architecture, intended to bolster the architecture-based advancement of software systems at different deliberation levels. It gives recoup, imagine and assess the software architecture of a system. This paper comprises of two noteworthy objectives: Architecture monitoring and representation which starts with mapping the source code artefacts of a software system to architecture components beginning from the system's architecture view, by labelling the source code as needs be Architecture evolution and assessment that address the advancement and assessment of the architecture of a software system. The trigger of any advancement is change ask. The change demand can be caused, e.g., by the expansion of new prerequisites or by the choice to structurally re-consider worsened parts of the system [16]. Budi et.al (2010) suggests that Layered Architecture is a good principle for separating concerns to make systems more maintainable. One example of layered architecture is a separation of classes into three groups boundary, control, and entity these are refer to as three analysis class stereotypes in UML. When the classes of different stereotypes interact with each other, and properly design, the overall system would be maintainable flexible and robust. Whereas poor design results in less maintainable system which is more prone to errors. It provides a frame work which can automatically labelled classes as boundary, control or entity, and detect design flows of rules associated with each stereotype. There are two common rules variants: - Robustness rule and well-formedness rules [17]. Herold et.al (2013) in their paper shows a report about the use of rule-based architecture

Annals of R.S.C.B., ISSN:1583-6258, Vol. 25, Issue 4, 2021, Pages. 2974 – 2989
Received 05 March 2021; Accepted 01 April 2021.

conformance checking approach which examine an industrial reference for the German open organization. The limitations for usage of reference architecture are formalized as architecture guidelines empowering programmed conformance checking device bolster. The above approach can recognize and keep away from design disintegration if connected over the product life cycle. Way to deal with design conformance checking are: (an) indicating the reference engineering as a meta-model to such an extent that the engineering of a framework can be demonstrated and (b) formalization of building elements required to be checked for conformance as intelligent expressions [18]. Eyck, Helleboogh et.al (2011), "Using code analysis tools for architectural conformance checking" In this paper, we examine a couple of code examination devices that can be utilized to check static conformance of a system to its architecture that offer support for Java and consider their abilities for compositional conformance checking: Architecture Rules, Macker, Lattix DSM, SonarJ, Structure101 and XDepend. Checking software architecture conformance is vital to keep that the framework erodes after some time and to defend the quality. At the point when a system is executed or transformed, it is difficult to evaluate whether the real implementation conforms in with the design. Architecture conformance checking is the verification whether a framework conforms in with its planned architecture, which is crucial to protect the quality characters of the system. A few methodologies exist to support architecture conformance checking. One way to deal with accomplish conformance is to combine an Architectural Description Language (ADL) general-purpose programming dialect. A case of this approach for the Java programming dialect is Arch-Java. Since engineering elements are top of the line segments in the tongue and fill in as a starting point for usage, design conformance is approved by the lingo itself. Along these lines compositional information is safeguarded inside the code, be that as it may it requires the utilization of a committed dialect to fabricate the framework [19].

## Proposed Methodology

Architectural level of understanding is crucial for software specifically when the system is to be reformed to meet changing requirements. Here we are focusing about the code smells, technical debt and code duplication majorly and which can be solved with the help of through refactoring of code, placing proper design principles and design patterns. To yield maximum architecture resources and software designers must estimate the size of technical debt. This technical debt can be handled by framework also, so that developer can write the code in better environment.

Our main objective is to correcting an eroded software system structural design by aligning the executed architecture with the proposed architecture or planned architecture. This process consists of the support of architecture discovery (it is used to build the proposed architecture derived through system requirements, documents specifications and system use cases) and recovery (used for recovering the implemented architecture). To diminish the complexity of system in order to increase maintainability, reliability, security and performance of software system.

The present problem manages the issue of programming engineering disintegration which makes the product framework more mind boggling, more prone to blunders and hard to maintain [20]. Engineering disintegration constitutes the most perceptible impact of the debasement of plan since software framework are under consistent weight to adjust to evolving necessities, innovative advancement and social scenes. These frameworks are must require to keep on delivering satisfactory levels of execution to clients [21]. Because of these changes made to the product framework over a timeframe harms its basic respectability and damage its plan standards. With a particular end goal to evacuate mistakes in the product framework the framework should regularly experiences the procedure of re-designing. This should be possible by adjusting the actualized design to planned engineering keeping in mind the end goal to redress a disintegrated framework.

Let's consider a fully layered framework, the layers of which have the ability to just make use of administrations given by the layer instantly beneath it [23]. An architecture violation is encountered when any source code component is unable to perceive these constraints. If remained resolved, such infringement causes the architecture to change into a solid piece, with unfavorable impacts on comprehend capacity, viability, and evolvability. Following is the research methodology:

**Step 1:** Analyze architecture and design the patterns.
**Step 2:** Finalize and implement the architecture based on design principles and patterns.
**Step 3:** Develop the one application without framework and one with spring MVC framework.
**Step 4:** Define the behavior of the classes to check the dependencies and violations rules.

Annals of R.S.C.B., ISSN:1583-6258, Vol. 25, Issue 4, 2021, Pages. 2974 – 2989
Received 05 March 2021; Accepted 01 April 2021.

**Step 5:** Connect the implemented architectures with the JDBC and record their compilation and run time.
**Step 6:** Validate the compliance of both the application by checking violations of rules, validate configuration with the help of tools SonarQube and JArchitect.
**Step 7:** Check for code smells, vulnerabilities, number of issues in application, technical debt using SonarQube and JArchitect.
**Step 8:** Remove the issues manually in order to make application less prone to errors so that new version of application can be produced.

By reviewing number of research papers, we concluded some of the important aspects of software architecture problems which occurs regularly in software development. There are some problems which we identified which are architectural smells, code smells, code duplication, technical debt and possible relations of similar types of problem which leads to Architectural problems [24].

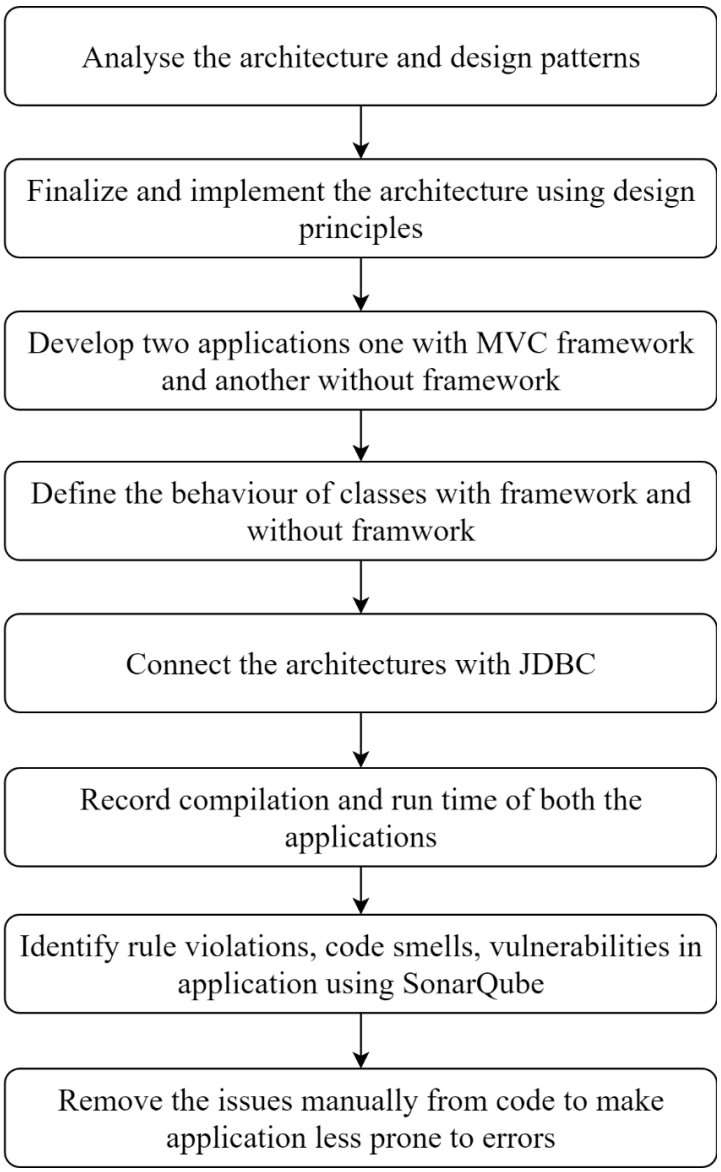


**Fig. 3.** Flowchart of Methodology

In order to control software architecture disintegration or erosion I have developed two projects. One is developed using Spring MVC framework and another one is developed with simple java without beans. The project

Annals of R.S.C.B., ISSN:1583-6258, Vol. 25, Issue 4, 2021, Pages. 2974 – 2989
Received 05 March 2021; Accepted 01 April 2021.

"SpringMVCHibernateCRUD" is developed using springMVC framework and follows all the design principle and patterns. The other project "Project-2" is developed without using design principles and design patterns [25]. These two projects are developed in Eclipse. Both the projects consist of an interface in which user will enter some of its basic information as shown in the figures given below. These projects consist of four different classes UserController.java (contains the servlets), UserDao.java (contains the logic for database operation), User.java (contains the POJO (Plain Old Java Object)), Database.java (contains the class for initiating database connection). The project "SpringMVCHibernateCRUD" which consists of Spring MVC framework is developed using Maven. Apache Maven is a product project administration and understanding tool. In view of the idea of a project object mode (POM), Maven can deal with a project's build, announcing and documentation from a focal snippet of data. Maven takes care of managing dependencies, developing a deployable component, runs application in Tomcat, creating sample report. The comparison of these projects is done with the help of SonarQube and JArchitect. The SonarQube is an open source tool for quality administration and is committed to persistently examining and measuring the technical quality of source code, from project portfolio down to the technique level, and following the introduction of new Bugs, Vulnerabilities, and Code Smells in the Leak Period [26]. Bugs are code that is more expected not providing the intended behavior. Examples such as null-pointer dereferences, memory leakages, and logic errors. Code smells are smelly codes which are difficult to maintain and introduce bugs. For example, complex code, duplicate code, codes which are not covered by unit testers. SonarQube helps to find and track the security Vulnerability in code. For example, SQL injections, passwords, badly managed errors [27]. JArchitect is a java static analysis and code quality tool. JArchitect assist in a large amount of code metrics, permits for visualization of dependencies with the help of directed graphs and dependency matrix [28]. User can also write custom rules and query code by using CQLinq (Code Query over LINQ), calculates Technical debt, checks for Quality Gates, Issues Management, generate custom reports, explore existing architecture, harness test coverage data [29].

## Results and Discussions

The various project screenshots have been shown and discusses in this section. The figure 4 is the architecture which is based on java code which does not consists of beans and spring MVC framework.

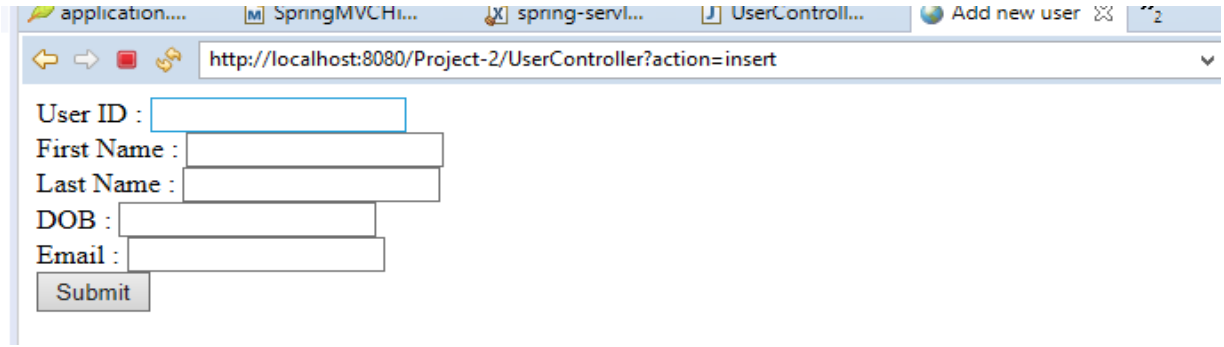


**Fig. 4.** Java code without beans

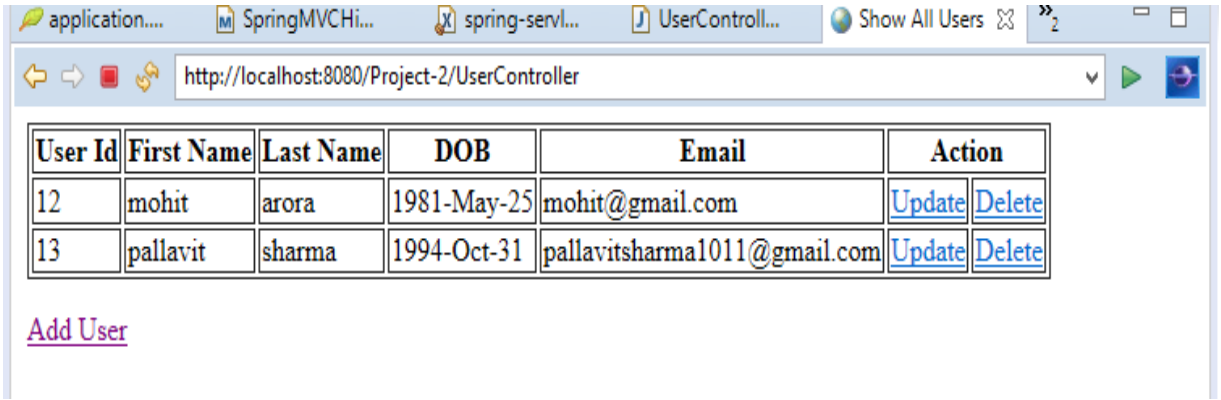


**Fig. 5.** Java code without beans output

Annals of R.S.C.B., ISSN:1583-6258, Vol. 25, Issue 4, 2021, Pages. 2974 – 2989
Received 05 March 2021; Accepted 01 April 2021.


**Fig. 6.** Controller class code

The figure 6 shows the code for the controller class which controls the data taken as input from the user and putting the input into the database.

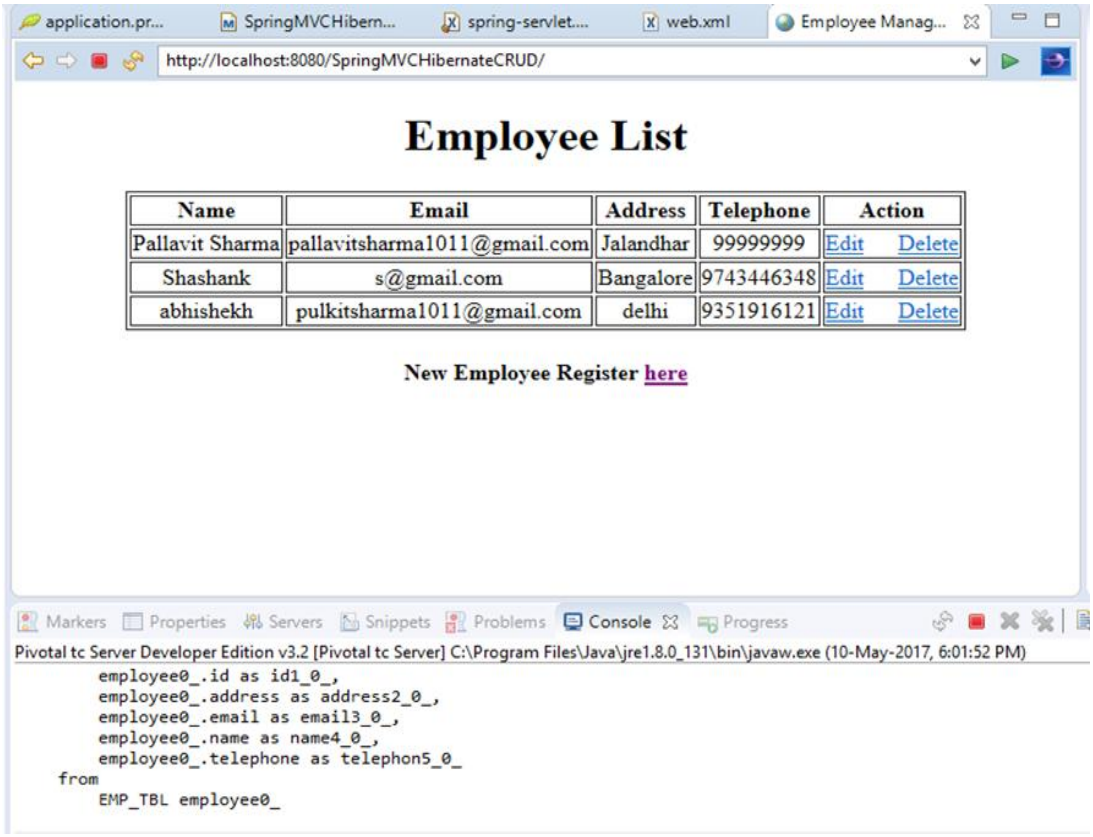


**Fig. 7.** Spring MVC and hibernate output

The Spring Web MVC system gives Model-View-Controller (MVC) engineering in addition to prepared parts that

Annals of R.S.C.B., ISSN:1583-6258, Vol. 25, Issue 4, 2021, Pages. 2974 – 2989
Received 05 March 2021; Accepted 01 April 2021.

can be utilized to construct adaptable as well as approximately coupled web applications. The figure 7 is based on spring MVC framework and consists of beans. The objects that shape the foundation of your application and that are overseen by the Spring IoC holder are called beans [30]. A bean is a protest that is instantiated, unite, and generally overseen by a Spring IoC holder. These beans are made with the setup metadata that you supply to the container. For example, in the form of XML <bean/>.

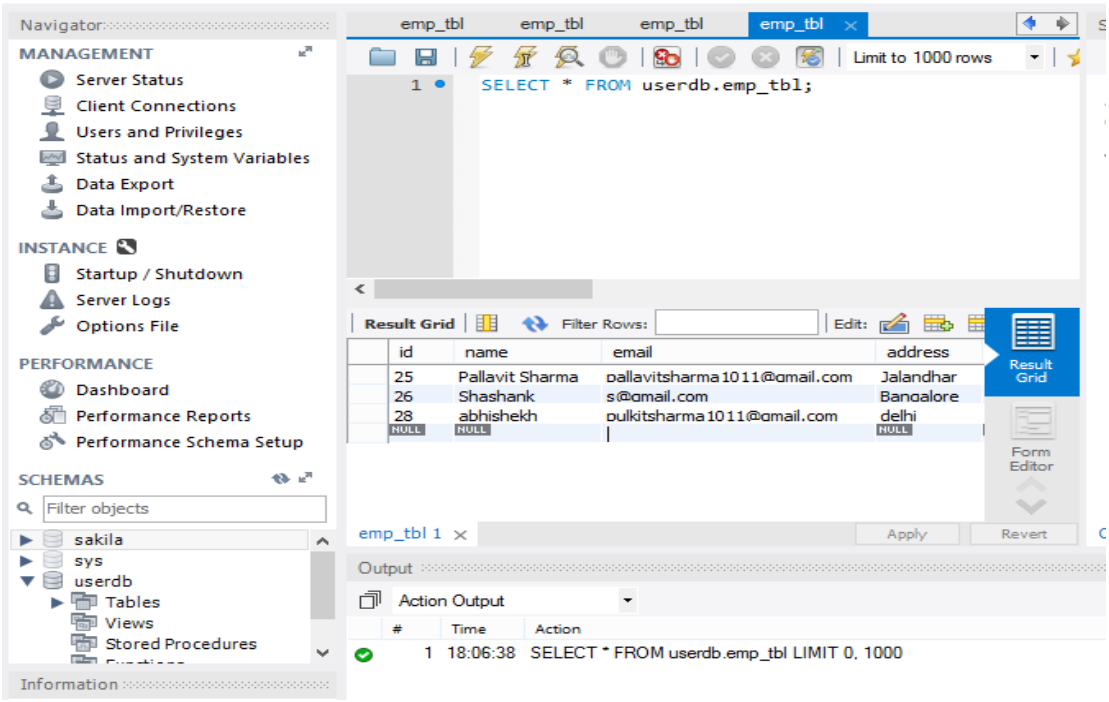


**Fig. 8.** Spring MVC framework database

The figure 8 shows the output of the spring MVC framework in MySQL Workbench. This shows that how the values are entering into the database.

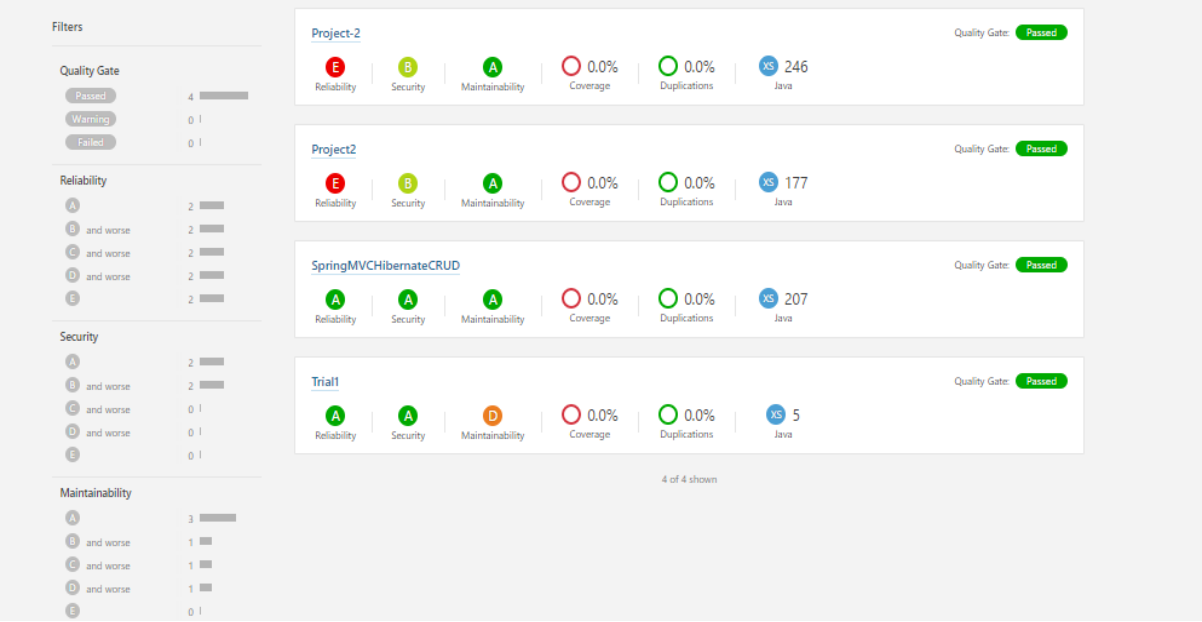


**Fig. 9.** SonarQube output

Annals of R.S.C.B., ISSN:1583-6258, Vol. 25, Issue 4, 2021, Pages. 2974 – 2989
Received 05 March 2021; Accepted 01 April 2021.

Figure 9 shows the rating of the four different projects. The Project-2 which is developed without any architecture pattern and does not follow any design principle shows 'E' rating in reliability, 'B' rating in security and 'A' rating in maintainability. Where the project SpringMVCHibernateCRUD which is developed using MVC (Model-View-Controller) architecture pattern and follows design principles shows 'A' rating in reliability, 'A' rating in security and 'A' rating in maintainability.

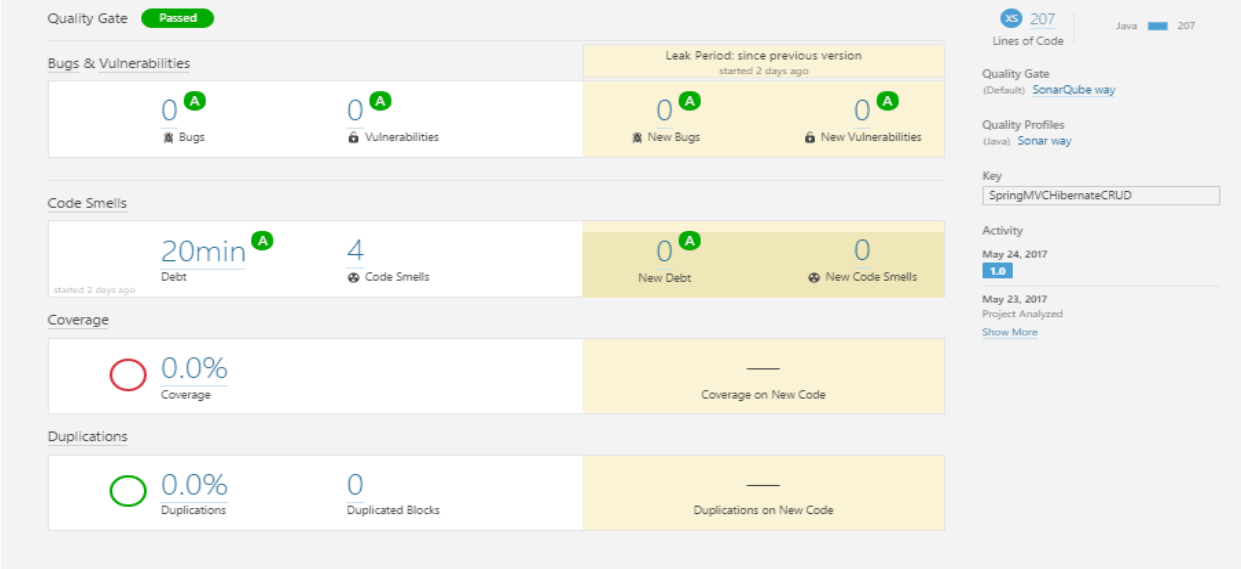


**Fig. 10.** Spring MVC output in SonarQube

Figure 10 shows the number of vulnerabilities, bugs, code smells, and duplicated blocks in SpringMVCHibernateCRUD project which is developed by using MVC architecture pattern and follows design principles. It consists of 4 code smells and also shows the technical debt to remove the code smells.

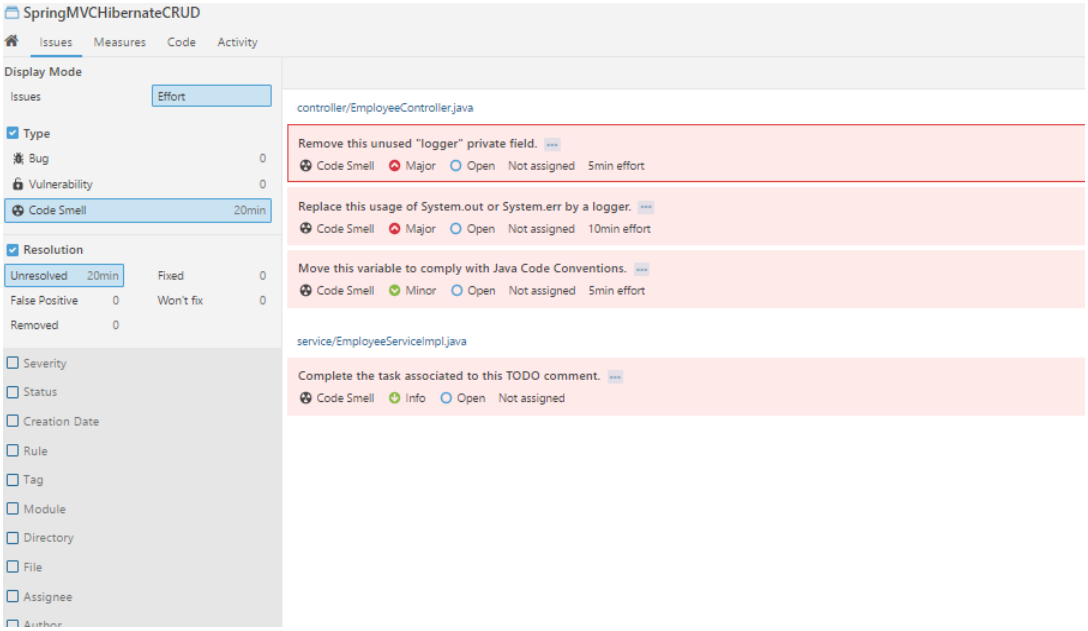


**Fig. 11.** Spring MVC output (SonarQube)

Figure 11 shows the issues in the given project and effort needed to remove single issue and code smell.

Annals of R.S.C.B., ISSN:1583-6258, Vol. 25, Issue 4, 2021, Pages. 2974 – 2989
Received 05 March 2021; Accepted 01 April 2021.

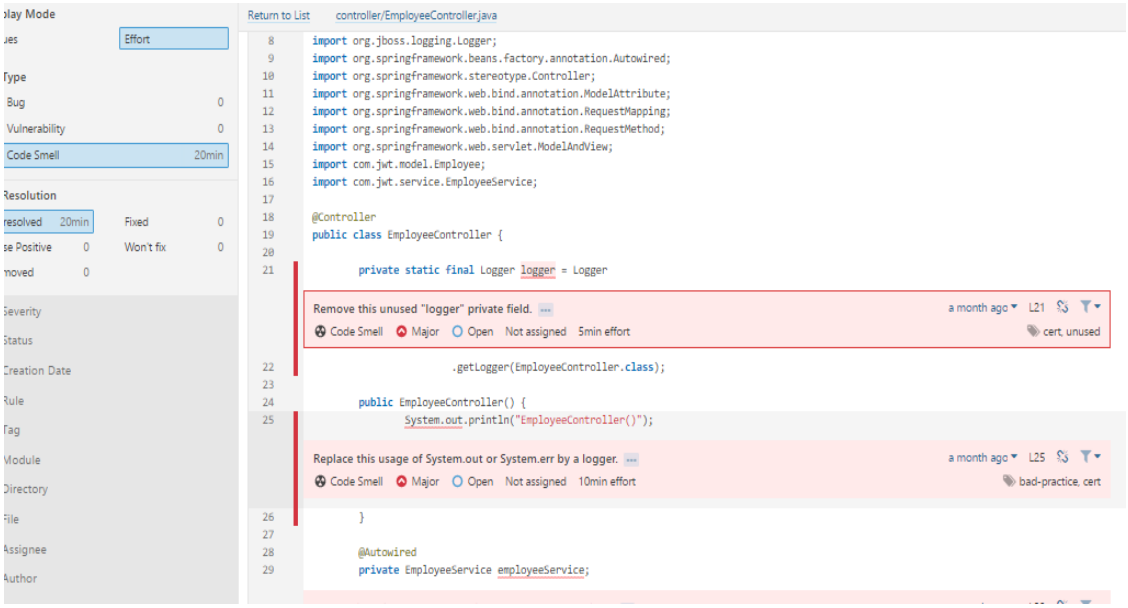


**Figure 12.** Spring MVC output (SonarQube)

Figure 12 shows the location where the error occurs. It will tell us the package and the class where the code smell will occur and also tell us whether it is minor or major.

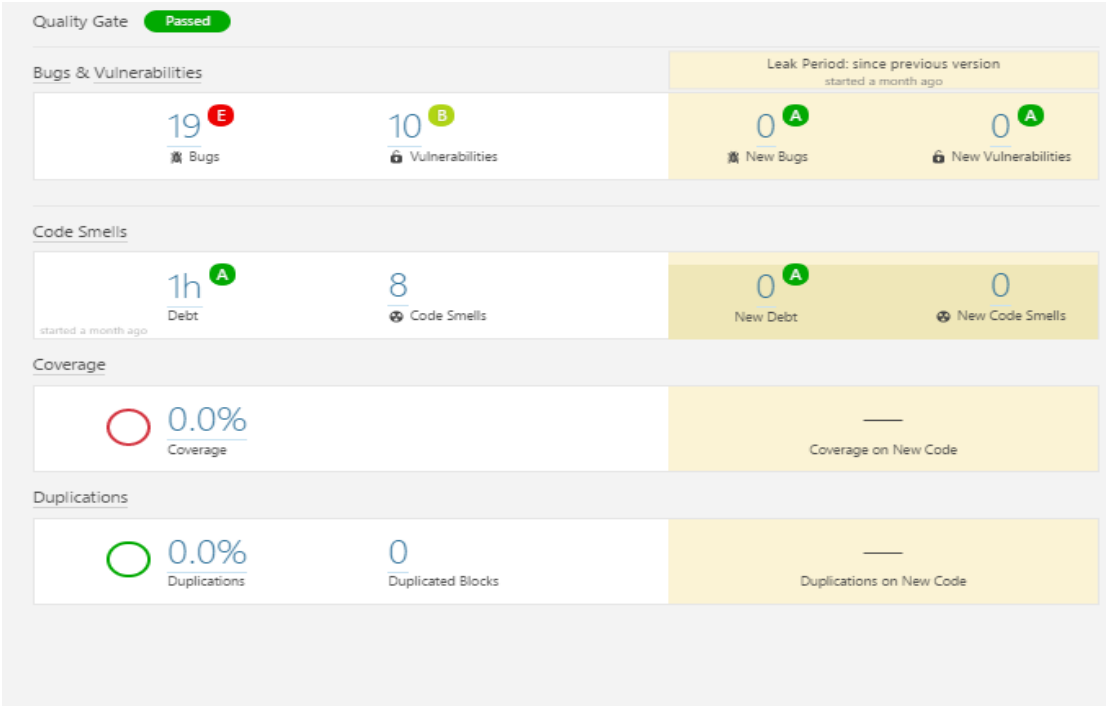


**Fig. 13.** Project-2 output (SonarQube)

Figure 13 shows number of vulnerabilities, bugs, code smells, and duplicated blocks in Project-2 which is developed by using java and does not follows design principles and patterns. It consists of 19 bugs, 10 vulnerabilities, 8 code smells and also shows the technical debt to remove the code smells. This shows that SpringMVCHibernateCRUD architecture is far better than second architecture as it contains a smaller number of bugs or issues.

Annals of R.S.C.B., ISSN:1583-6258, Vol. 25, Issue 4, 2021, Pages. 2974 – 2989
Received 05 March 2021; Accepted 01 April 2021.

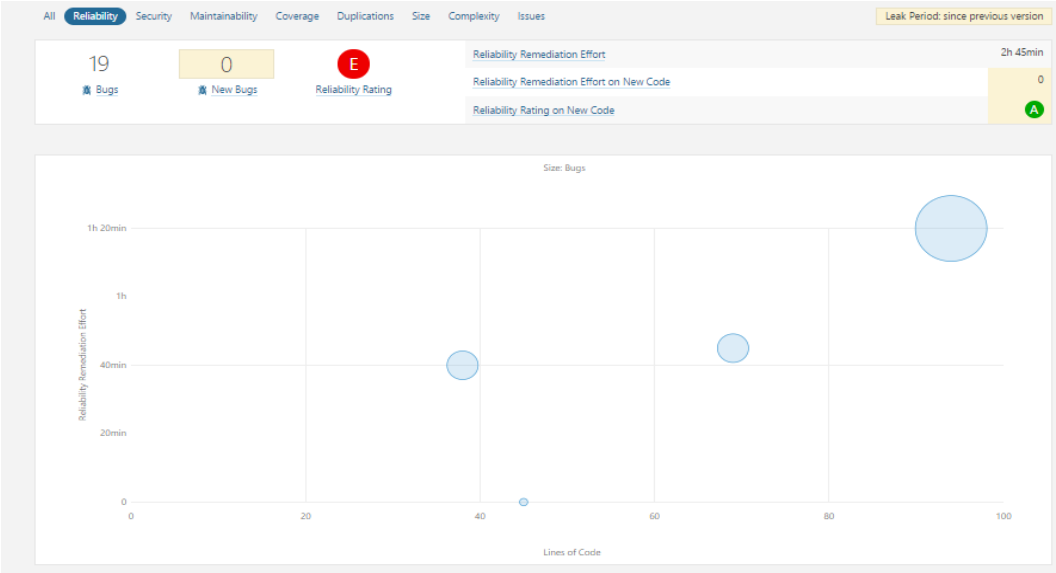


**Fig. 14.** Project-2 output (SonarQube)

Figure 14 shows the bubble graph of reliability of project-2 which is not reliable as its rating is "E".

The above figures show the output of SpringMVCHibernateCRUD architecture and another architecture in JArchitect which shows method complexity and number of rules both the architecture follows. There are some tools such as PMD, Check style, find bugs which is used for static analysis of architecture and these are tools which is used for hard coding analysis [31]. Now a days we are working on design principles, design patterns and frameworks which can generate the whole code dynamically according to your classes and object definition. For that reason, we need a tool that can analyze code in dynamic environment [32]. Hence, we concluded that by using these tools we can negotiate in technical depth complexity, code smells, and vulnerabilities in architecture.

**Table 1.** Comparison between spring MVC and Project-2 in SonarQube

| | **Spring MVC project** | **Project-2** |
|---|---|---|
| **Bugs** | 0 | 19 |
| **Vulnerabilities** | 0 | 10 |
| **Code smells** | 4 | 8 |
| **Technical debt** | 20 minutes | 1 hour |
| **Unresolved issues** | 4 | 37 |
| **Complexity** | 29 | 29 |
| **Size** | 207 | 246 |
| **Maintainability rating** | A (<=5% of time already gone) | A (technical debt ratio is less than 5%) |
| **Reliability rating** | A (0 bug) | E (there is 1 blocker bug) |
| **Security rating** | A (no vulnerabilities) | B (there is 1 minor vulnerability) |
| **Quality gates** | 14 | 14 |
| **Code smell** | 9 | 9 |
| **PMD rule** | 203 | 203 |
| **Byte code instruction** | 204 | 487 |
| **Code difference summary** | 25 | 25 |
| **Naming conventions** | 11 | 11 |
| **Packages** | 4 | 4 |
| **Methods** | 40 | 23 |
| **Fields** | 10 | 11 |
| **Third party element** | 54 | 80 |
| **Method complexity** | 2 (Max), 1.07 (avg.) | 6 (Max), 1.78 (avg.) |

Annals of R.S.C.B., ISSN:1583-6258, Vol. 25, Issue 4, 2021, Pages. 2974 – 2989
Received 05 March 2021; Accepted 01 April 2021.

Spring project is more reliable, less maintainable, more secure and less complex as compare to project-2 because it follows design principles and design patterns. Hence according to SonarQube static tool analyzer spring MVC project is far better than project- 2. In the table given above we concluded that the complexity between the methods reduces in spring MVC project from 6 to 2 (Max) and 1.78 to 1.07 (average). Hence, spring MVC project is much far better than project-2.

## Conclusion

This paper lays foresight on the present methods require to deal the architecture erosion problem. We have proposed technique to control architecture erosion demonstrating through two architectural frameworks This shows that using the springMVC and hibernate framework which is developed by using MVC architecture pattern and follows design principles, one can minimize the architecture erosion.

## Future Scope

We are still comparing the two architectures one is pattern (MVC) without framework and the other is MVC with spring and hibernate framework. The comparison is done based on the architecture design principles such as separation of concerns, single responsibility principle, etc. The comparison will show that using the MVC spring and hibernate framework we can minimize the architecture erosion [33]. More to go with JACOCO, in future, we are going to implement the same thing using JACOCO code analysis. JACOCO should provide the standard technology for code coverage analysis in Java VM based environments. The focus is providing a lightweight, flexible and well documented library for integration with various build and development tools. One of the most interesting things regarding JACOCO is that it works for functional as well non-functional characteristics and it also lightweight and works also for VM. Today's world is belonging to cloud, so everywhere we found virtual machines if we are writing any code in cloud environment then this tool is very useful. Our next focus is analysis of code in cloud environment using these tools.

## References

[1] Herold, S., English, M., Buckley, J., Counsell, S., & Cinnéide, M.Ó. (2015). Detection of violation causes in reflexion models. In 2015 *IEEE 22nd International Conference on Software Analysis, Evolution, and Reengineering (SANER)*, 565-569.

[2] J. Knodel, D. Muthig, M. Naab and M. Lindvall, "Static evaluation of software architecture," in 10th *European Conference on Software Maintenance and Engineering (CSMR),* 2006, pp. 279-294.

[3] N. Sangal, E. Jordan, V. Sinha, and D. Jackson. Using dependency models to manage complex software architecture. *SIGPLAN Not.*, 40(10):167-176, 2005.

[4] M. De Silva and I. Perera, "Preventing software architecture erosion through static architecture conformance checking," 2015 *IEEE 10th Int. Conf. Ind. Inf. Syst. ICIIS 2015 - Conf. Proc.*, pp. 43–48, 2016.

[5] R. Terra, M.T. Valente, K. Czarnecki, and R.S. Bigonha, "Recommending refactoring to reverse software architecture erosion," *Proc. Eur. Conf. Softw. Maint. Reengineering, CSMR,* pp. 335–340, 2012.

[6] K. Sartipi, "Software architecture recovery based on pattern matching," Softw. Maintenance, 2003. *ICSM 2003. Proceedings. Int. Conf.*, pp. 293– 296, 2003.

[7] L. Pruijt, C. Köppe, and S. Brinkkemper, "Architecture compliance checking of semantically rich modular architectures: A comparative study of tool support," *IEEE Int. Conf. Softw. Maintenance, ICSM,* pp. 220–229, 2013.

[8] L. De Silva and D. Balasubramaniam, "Controlling software architecture erosion: A survey," *J. Syst. Softw.*, vol. 85, no. 1, pp. 132–151, 2012.

[9] S. Herold and S. Counsell. "Detection of violation causes in Reflexion models" *IEEE Int. Conf. on Soft. Analy. Evol. And Re-engg.*, 2015.

[10] P. Sharma, M. Arora, S. Chopra. Controlling software architecture erosion to support maintainability" *SSRG Int. Conf. of Comp. Sci. and Eng. (ICETM),* 2017

[11] P. Petrov and U. Buy, "A systemic methodology for software architecture analysis and design," *Proc.* - 2011 8th Int. Conf. Inf. Technol. New Gener. ITNG 2011, pp. 196–200, 2010.

[12] P.U. Chavan, M. Murugan, and P.P. Chavan, "A review on software architecture styles with layered robotic software architecture," *Proc. - 1st Int. Conf. Comput. Commun. Control Autom. ICCUBEA* 2015, pp. 827–

Annals of R.S.C.B., ISSN:1583-6258, Vol. 25, Issue 4, 2021, Pages. 2974 – 2989
Received 05 March 2021; Accepted 01 April 2021.

831, 2015.
[13] N. Chanda and X. (Frank) Liu, "Intelligent Analysis of Software Architecture Rationale for Collaborative Software Design," 2015 *International Conference on Collaboration Technologies and Systems (CTS)*, pp. 287–294, 2015.
[14] A. Caracciolo, M.F. Lungu, and O. Nierstrasz, "A Unified Approach to Architecture Conformance Checking," *Proc. - 12th Work. IEEE/IFIP Conf. Softw. Archit. WICSA* 2015, pp. 41–50, 2015.
[15] O. Maqbool and H.A. Babri, "Bayesian Learning for Software Architecture Recovery," 2007 *Int. Conf. Electr. Eng.*, pp. 1–6, 2007.
[16] A. Dragomir and H. Lichter, "Model-based software architecture evolution and evaluation," in *Proceedings - Asia-Pacific Software Engineering Conference, APSEC*, 2012, vol. 1, pp. 697–700.
[17] A. Budi, D. Lo, and S. Wang, *"Automated Detection of Likely Design Flaws in Layered Architectures,"* no. July, pp. 7–9, 2011.
[18] S. Herold, M. Mair, A. Rausch, and I. Schindler, "Checking conformance with reference architectures: A case study," *Proc. - IEEE Int. Enterp. Distrib. Object Comput. Work. EDOC*, pp. 71–80, 2013.
[19] J. Van Eyck, N. Boucké, A. Helleboogh, and T. Holvoet, *"Using code analysis tools for architectural conformance checking."*, 2011
[20] J. Knodel and D. Popescu, *"A Comparison of Static Architecture Compliance Checking Approaches 1,"* 2007.
[21] ISO/IEC 9126-1, *Software Engineering - Product Quality - Part 1: Quality Model*, 2001.
[22] N.M. Edwin, *"Software Frameworks, Architectural and Design Patterns,"* no. July, pp. 670–678, 2014.
[23] Bosch, J (2000), *"Design and Use of Software Architectures"*, Addison-Wesley Professional.
[24] Microsoft, Microsoft Application Architecture Guide. $2_{nd}$ ed. Microsoft Press, 2009.
[25] Freeman, P. *The Central Role of Design in Software Engineering.* Software Engineering Education Freeman, P. and Wasserman, A. eds. Springer-Verlag: New York, 1976
[26] T. Abdellatif, S. Bensalem, J. Combaz, L. De Silva, and F. Ingrand, "Rigorous design of robot software: A formal component-based approach," *Rob. Auton. Syst.*, vol. 60, no. 12, pp. 1563–1578, 2012.
[27] G.S. Kumar, K. Rameetha, and K.P. Jacob, "A generic software architecture for a domain specific distributed embedded system," *Int. Conf. Softw. Eng. Theory Pract.* 2007, SETP 2007, pp. 41–46, 2007.
[28] P. Iñigo-blasco, F. Diaz-del-rio, M. C. Romero-ternero, D. Cagigas-muñiz, and S. Vicente-diaz, "Robotics software frameworks for multi-agent robotic systems development," *Rob. Auton. Syst.*, vol. 60, no. 6, pp. 803–821, 2012.
[29] W. Michael, *"A Layered software architecture for Hard Real Time (HRT) embedded systems Monterey, California,"* 2002.
[30] J.B. Tran, R.C. Holt, Forward and reverse repair of software architecture, in: *Proceedings of the Conference of the Centre for Advanced Studies on Collaborative Research, IBM*, 1999, pp. 12–20.
[31] M.W. Godfrey, E.H.S. Lee, Secrets from the monster: Extracting Mozilla's software architecture, in: *Proceedings of the International Symposium on Constructing Software Engineering Tools*, pp. 15–23.
[32] Garlan, D.A. & Ockerbloom, J.R., (1995) "Architectural mismatch: Why reuse is so hard". *IEEE Software*, 12(6):17–26, Nov 1995
[33] Rakhra M, Singh R, Lohani TK, Shabaz M. *Metaheuristic and Machine Learning-Based Smart Engine for Renting and Sharing of Agriculture Equipment.* 2021; 2021.