





The architectural erosion problem in Andorid Apps

# Identifying, Detecting and Fixing Architectural Erosion in Android Apps

Juan Camilo Acosta Rojas

February 19, 2025

*This thesis is submitted in partial fulfillment of the requirements  
for a degree of Master in Systems and Computing Engineering  
(MISIS).*

## Thesis Committee:

Prof. Name LAST NAME (Promotor)

Universidad de los Andes, Colombia

Prof. Reviewer 1 REVIEWER 1

Institution, Country

Prof. Reviewer 2 REVIEWER 2

Institution, Country

# Identifying, Detecting and Fixing Architectural Erosion in Android Apps

© 2025 Juan Camilo Acosta Rojas

The Software Design Lab

Systems and Computing Engineering Department

Faculty of Engineering

Universidad de los Andes

Bogotá, Colombia

To my family. The fruit of all their efforts be shown here.

## Abstract

Software engineering involves different efforts with the same purpose: high-quality software development. Among these efforts, we can find the software architecture definition process, which gives preference to the various quality attributes according to the system's needs. However, for different reasons, the solution development can deviate from the original architecture design; it could generate performance problems, and as a consequence, it affects the user experience. This impact on software quality could be represented, at a greater rate, in a mobile environment due to its limited resources. Previous research with this approach has focused on problem study and detection in different areas like security, connectivity, etc. However, in mobile development, the concept of "architectural erosion" has not been deeply studied. The main goal of this research project is to detect and locate architectural erosion bugs in mobile applications with automated static and dynamic analysis based on deep learning models.



## **Acknowledgements**





# Contents

Abstract . . . . .	iv
Acknowledgements . . . . .	vi
List of Figures . . . . .	x
1. Introduction . . . . .	1
1.1. Why make researching efforts in architectural erosion? . . . . .	2
1.2. Machine learning models in architectural erosion . . . . .	2
2. Definitions . . . . .	3
2.1. Development Process of a Software Project . . . . .	3
2.2. Architectural design in software engineering . . . . .	4
2.2.1. Architectural design in Android development . . . . .	4
2.2.2. Quality Attributes . . . . .	6
2.3. Architectural Erosion . . . . .	7
2.3.1. Approaches and Perspectives . . . . .	8
2.3.2. Main Reasons and Symptoms . . . . .	8
2.3.3. Consequences . . . . .	9
2.3.4. Metrics and treatments for architectural erosion . . . . .	10
2.4. Representative set of a sample . . . . .	12
2.5. Natural Language Processing in Software Issues Detection . . . . .	12
2.5.1. Word Embeddings for Words Representation . . . . .	13
2.5.2. Performance and Similarity Metrics in NLP . . . . .	14
2.6. Programming Languages fundamental for Software Analysis . . . . .	15
3. Related Work . . . . .	19
3.1. Research Methodology . . . . .	19
3.1.1. Architectural Erosion: An Initial Overview . . . . .	19
3.2. Architectural Erosion Symptoms Identification . . . . .	21
3.3. Architectural Erosion Symptoms Detection . . . . .	22
3.4. Giving Architectural Erosion Solutions . . . . .	22
4. Finding Architectural Erosion Issues . . . . .	23
4.1. Architectural Erosion Keywords in Commits . . . . .	23
4.1.1. Architectural erosion symptoms Identification . . . . .	23
5. Detecting Architectural Erosion Issues . . . . .	27
5.1. UAST in Android Studio . . . . .	27
5.2. Rules Definition . . . . .	29
5.3. AER Detector implementation . . . . .	29
5.4. AER linter tool testing criteria . . . . .	29
A. TITLE . . . . .	31
Bibliography . . . . .	33
Acronyms . . . . .	35

## CONTENTS

List of Terms . . . . .	35
-------------------------	----

## List of Figures

2.1. Similarities and differences between Android architectural patterns. ] . . . . .	5
2.2. Three-layer framework for mobile development. ] . . . . .	6
2.3. Main Concepts of Architectural Erosion in Software Engineering . . . . .	11
2.4. Tokenization process of one sentence Manning [7] . . . . .	13
2.5. Representation in tow dimensions of Similar Words give a Word Embedding Ju- rafsky [4] . . . . .	14
2.6. Example of an Abstract Syntax Tree (AST) statement. ] . . . . .	16
2.7. Example of an Abstract Syntax Tree (AST) statement. ] . . . . .	17
5.1. Definition of Android Studio UAST structure ] . . . . .	28

## CHAPTER 1

Introduction

---

In the beginning of system building, we might undergo a strong design process, where you define the system components and resources you need to use. This process is named *software architecture definition*. After that, you have to align the development process with those architectural rules to achieve goals in performance, security, availability, and others. However, due to growth in software development, as time goes by, software tends to violate architectural rules, affecting system quality attributes. This deviation is called *architectural erosion*. In mobile development, it can reduce application performance and other system quality attributes, affecting device resources and, ultimately, the user experience. In recent research, there have been significant advances in bug resolution across different approaches. For example, in security, connectivity, and code smells, there are various tools and components in both stacks (Frontend and Backend) that detect and offer solutions for different bugs related to these approaches.

However, in the architectural erosion analysis, despite 73 research studies tackling this concept, some of these propose a toolset; it is not clear how we can resolve architectural erosion insights or a set of directives for facing this problem in the mobile development ecosystem. For this reason, it is important to find the impact of architectural erosion in mobile applications and how we can detect, give alerts, and provide solution recommendations for fixing architectural erosion bugs with static analysis and the help of natural language processing techniques.

## 1.1. Why make researching efforts in architectural erosion?

Due to different researches, the costs in terms of human resources, technological resources, time and money would grow after the first release and deployment of the software project. By this, is necessary to establish the different reasons and artifacts that affect the software sustainability, the software performance in short, middle and long term. For this task, there are many some approaches that analyze different software project components like its design, its code, its architectural rules. One of the most important, and one of the most studied too, has been the static code analysis approach. In this approach, has been created a lot of tools for detect architectural erosion in specific code fragments, showing a analysis along time. This tools has been very helpful for the developers and for the teams, but it has not been enough in mobile development ecosystem, where system resources are mire limited that side-server environment. Is necessary to build a component with specific purpose in Android Apps and show the importance of detect architectural erosion in mobile app code and recover of this issues in a early way.

## 1.2. Machine learning models in architectural erosion

Recent research in Machine Learning models, specifically in Natural Language Processing, has concluded that the use of pre-trained models for the classification or generation of words in a specific language, their performance with programming languages is better than natural languages (in this NLP approach, ex: English, Spanish, e.t.c), due to their syntax. This idea could be recreated in issues detection in source code repository analysis. Issues like masked code, pattern design detection, and other ones have been implemented with pre-trained machine learning models. In architectural erosion, it could be useful for detecting the first symptoms of this and making early detection for, after that, using specific rules of architecture in static code analysis. In resume, machine learning models could detect the first symptoms of architectural erosion in an Android project and create a component with optimized performance for detecting and recovering issues about this.

## Definitions

---

### 2.1. Development Process of a Software Project

When you have to make a software project planning, you have to define the main requirements that achieve the main goals of the business idea inside the software project realization motivation. First, you have to define the functional requirements. In software engineering, functional requirements define the main features of each software component involved in the project. Requirements like design, basic functionalities, and defined flows are built on the functional requirements definition process. After that, you have to define the main requirements that do not directly impact the user experience, called nonfunctional requirements. Nonfunctional requirements indirectly affect your software project; bad planning about those requirements could progressively affect the user performance and the software project architecture, becoming slower, more weight, or less useful for a single application user. In this process, you must design a solution that satisfies the functional and non-functional requirements. This process is called the design process of a software project. Inside this stage, you must define the number of components that give the software functionalities. Furthermore, you have to each component infrastructure, into a process named Software Architecture Definition Laplante [5].

## 2.2. Architectural design in software engineering

For an effective and efficient software project development, we have to realize the respective investment in each stage. From the functional requirements creation to the design of each component and its features and constraints. To do this, it is necessary to make an architectural design. The architecture of a software project. In general, defines the components that require the entire system and the connections and relationships (and their types). Furthermore, each component has to be defined in this design. In that order, different standards ensure the system's quality according to its architectural design.

### 2.2.1. Architectural design in Android development

Architectural Design in the mobile approach has been advancing step by step; this depends on the creation of new technologies in each development layer since the data management layer with the implementation of new database libraries like the room to the User Interface layers with a different way of creating new screens inside the application (with the use of either JetPack Compose or Fragments organization). Depending on what simple should be an application, how many components would have, which of them would be connected, and the reasons for those connections. It is necessary to review different architectural patterns used recently and why we focus on one of them for architectural erosion detection.

- MVC (Model View Controller): In this pattern, we divide the application components in the model when we implement the connections with external platforms and internal data management. The controller component is used for setting the relation between the business logic and the User Interface (UI). The view component contains the UI. This pattern has been commonly used for the last 15 years due to its simplicity and popularity. However, the applications that use this pattern are very coupled, and their components depend strongly on others. It is usually to find business logic implementations and UI code fragments in the same code file or data processing in business code components. In the beginning of Android, architectural issues weren't as important as they are today. If an Android application is simple in terms of realization, it is possible to use an MVC pattern.

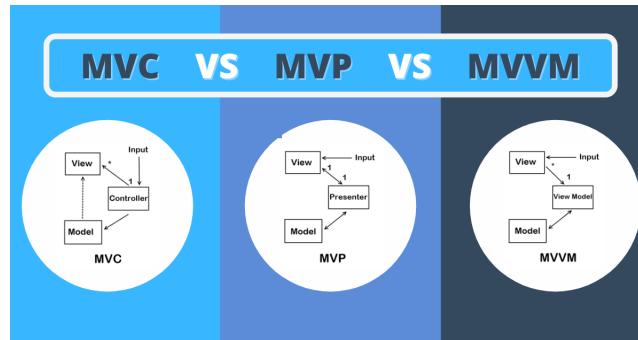


Figure 2.1: Similarities and differences between Android architectural patterns. ]

- **MVP (Model View Presenter):** This pattern is managed in a different way than the MVC pattern in the relations between business logic and UI. In this case, we use a component named presenter to manage the events and behavior of each UI view or screen. This pattern is commonly used for single Applications that do not have scalability or application overloading. This pattern divides the presenter features connected with the UI features. The disadvantages of this pattern are related to a high coupling rate inside their components and a big complexity for managing the life cycles of an application. Furthermore, it is difficult to implement new features development and maintenance for large-scale Android applications.
- **MVM (Model View View-Model):** This pattern is one of the derivatives of clean architecture, a concept widely developed in Backend and Frontend applications architectures. This pattern uses some concepts of clean architecture, like use case organization, when we implement a new feature in an independent way of others. With this pattern, we use reactive components. The application components use libraries like Dagger Hilt to implement the use of reactive data; this reactive data changes depending on a UI event. With the creation of the JetPack Compose framework. The JetPack Compose is based on reactive UI and is more declarative than the traditional form (the use of XML files and fragments structure for managing different application screens)

In an Android application, it is not mandatory to use only an architectural pattern to achieve the functional and non-functional requirements of a software project. For example, it is very



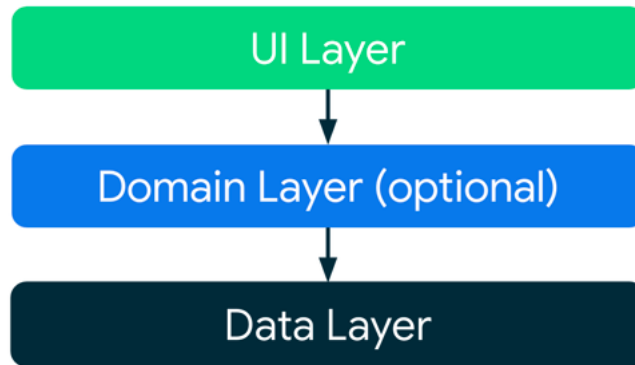


Figure 2.2: Three-layer framework for mobile development. ]

common to use the repository pattern declared in MVVM divided into two: external connections and internal data management. Today, in the actual mobile development ecosystem, the most common framework that could be implemented with one or more architectural patterns is the three layers of pattern architecture: The UI layer, an optional layer named the Domain Layer, and the Data layer. Each layer could be or could not be implemented depending on each application's requirements set.

The three-layer architecture will be used as the main architectural framework to realize this study with the MVVM architectural pattern. These patterns are the most used in modern mobile development and will give a first approach for how the developers can detect and fix architectural violations in an Android application,

### 2.2.2. Quality Attributes

The consequences of the generation of architectural erosion could impact the different quality attributes contemplated. According to Bass Bass et al. [2]. there are six attributes based on ISO-25010. Those attribute qualities are:

- Latency: This attribute measures the response time of the components according to the implemented architecture. Response times of each architectural component are very important.
- Scalability: Measures the ability of a system to grow in critical situations of system inputs.

It is important for system availability when the user's connection rate increases.

- **Security:** Nowadays, a system must give protection to its users and their data, since availability, integrity, and confidence of that data. This requirement is very important in the legal environment of a system.
- **Availability:** When the system can be available when a system failure occurs (it doesn't depend on the fail type). This quality attribute measures the time that the system uses for recovering a fail(s).
- **Integration capability:** Measures the ability of a system to integrate with other(s) system(s), measures the time and effort that the system needs to make that integration in all their layers from the data layer till the UI layer is necessary.
- **Modification capability:** Similar to the last quality attribute, it measures the time that the system needs to change one or more components inside its system. The effort measure could be many relationships (human resources effort, time costs, money costs, etc.)

## 2.3. Architectural Erosion

The Architecture Erosion concept was treated a long time ago. Since 1992, architecture erosion has had a formal definition, giving the relationship of this concept with architectural rules violations Perry [8]. It can be treated from different perspectives, from the violation of rules, structure, and quality to the evolution perspective. Furthermore, the concept has a strong relationship with other synonyms, like, for example, degradation. In another modern definition, architectural erosion is defined as the set of architectural violations that reflect the deviation of the implemented architecture from the intended architecture over time. In resume, in a more concrete definition, architectural erosion could be considered a phenomenon that reflects the deviation of an implemented architecture due to an intended architecture in a software project.

### 2.3.1. Approaches and Perspectives

Due to the original concept of architectural erosion Ruiyin et al. [9]. It could be studied and analyzed by different approaches:

- Violation perspective: Denotes how the implemented architecture violates the design principles or main constraints of the intended architecture. These violations could occur in two phases: the design phase and the maintenance and evolution phase, making different changes step by step in the short and long term.
- Structure perspective: Where the structure of a software system encompasses its components and their relationships.
- Quality perspective: it refers to the degradation of the system quality, due to architectural changes that would generate architectural smells. It could include all the quality attributes contemplated actually in the industry.
- Evolution perspective: It shows the architectural inflexibility that increases the difficulty of implementing changes in the project and, therefore, decreases the sustainability of the system.

### 2.3.2. Main Reasons and Symptoms

Different factors could provoke architectural erosion in different stages of software project development. Due to these reasons, is possible to make different solution approaches, and, therefore, the possibility for build different components based on that approaches. The main reasons founded are:

- Architecture modularization: Due to the business needs, is necessary to divide responsibilities between different components and layers in a software project. but, sometimes it could produce non-functional components and deviate from the initial intended architecture.
- Architecture complexity: if the intended architecture is very complex, is possible to deviate from it when the time goes by, producing the first symptoms of architectural erosion.

### 2.3. ARCHITECTURAL EROSION

- **Architecture size:** Due to this attribute, and with no control over the maintenance of the software project, is not possible to have good software maintenance for a long time.
- **Design Decisions:** If the design decisions during the initial stages of the project don't have enough support (like documentation, reviews, etc.), it could generate problems with the maintenance of refactoring of different components, decreasing the code quality and generating the first symptoms of architectural erosion.
- **Duplicate functionality:** As a consequence of the latest reasons too, duplicate functionality reflects the bad connection between layers of an architecture, and, could be considered as the initial symptom of architectural erosion.

There are another reasons like bad documentation, bad programming features, but, in general, the main reasons were considered accord to the architecture design stage issues.

#### 2.3.3. Consequences

Are several points of view about the real definition of consequences of architectural erosion violations inside a software project. The main problems that could generate the non correct maintenance and the deviaton of an intended architecture are:

- **Costs of software maintenance:** Due to the no implementation of architectural erosion issues, this could affect one of the non-functional defined requirements, and after that, affect the actual software infrastructure.
- **Software Performance:** When one of the quality attributes is affected by the initial architecture design planning, it incurs performance reduction of the intended architecture. In Android Apps, it could be more notorious, due to the limited resources of an mobile device, very different of a desktop device or server-type device.
- **Software quality decrease:** When architectural changes in a software project, it could affect the normal behavior of the application and, inside its source code, could imply bad code features implementation, decreasing the software quality, an important standard in a software project.

- **Software Sustainability:** The cost in terms of human resources, software infrastructure, time, money, etc, could increase if you do not attend the architectural erosion insights in your software project, affecting nonfunctional requirements, and, in a long time, affecting the user performance.

#### **2.3.4. Metrics and treatments for architectural erosion**

Today, exist some different metrics that can determine architectural erosion in different approaches. According to Baabad et al. [1]. Some metrics have been created to analyze architectural decay in open-source projects, analyzing possible reasons, indicators, and solution strategies for it. In the resume, exist around 54 metrics that could determine architectural erosion in different stages, from the design stage to the deployment stage. Those metrics have been classified by measured artifacts, level of validation, usability, applicability, comparative analysis, and support tools. Different classifications could be implemented into a tool with a specific measure strategy for analyzing architectural erosion in any system.

### 2.3. ARCHITECTURAL EROSION

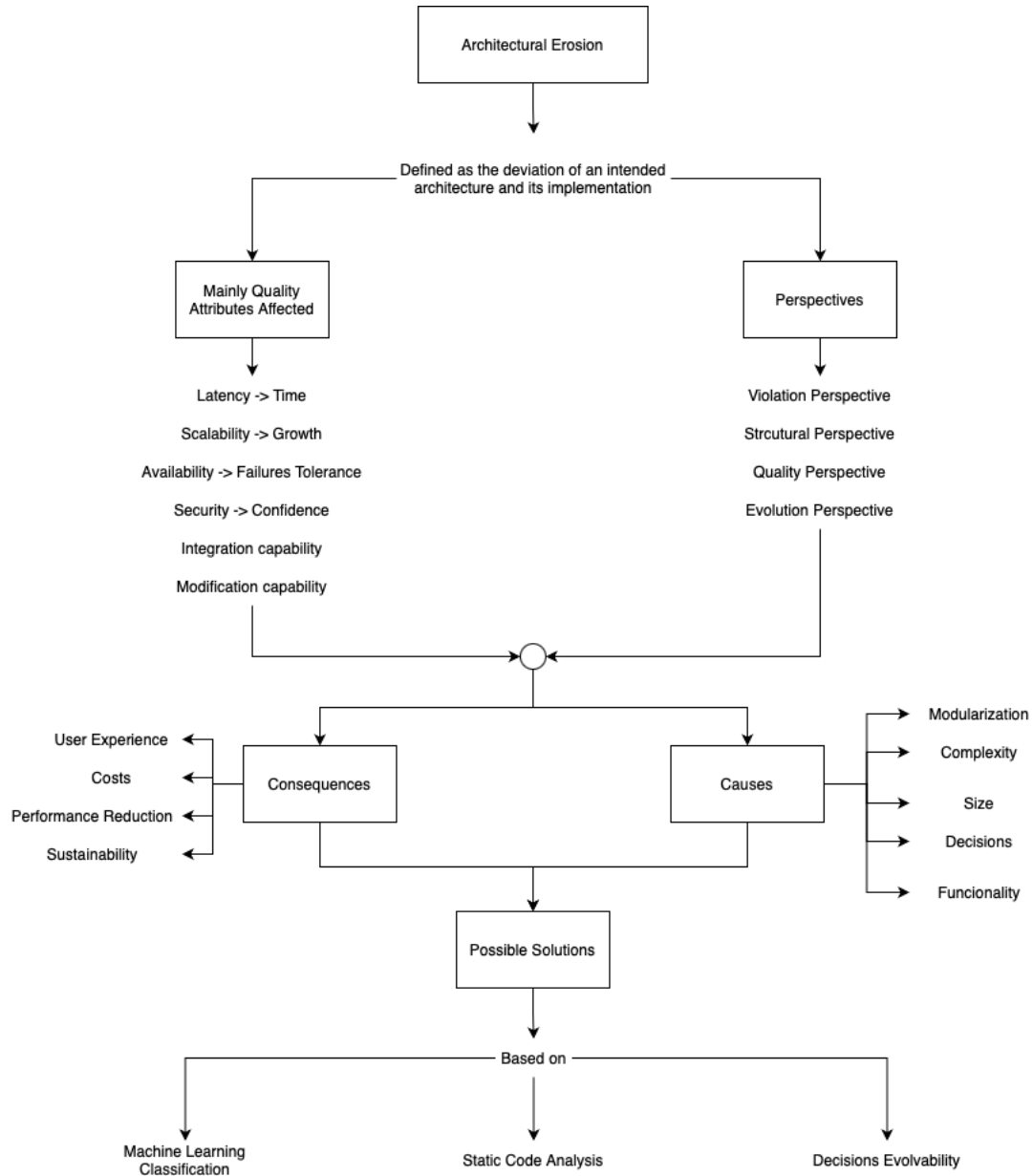


Figure 2.3: Main Concepts of Architectural Erosion in Software Engineering

## 2.4. Representative set of a sample

In statistics, several dataset sampling ways are effective in selection criteria, each alternative has a way for selecting data randomly that represents in a minor scale all the dataset. One way is to select a representative set from the original population. In this case, it is possible to make a weighted-randomly selection of some individuals from any population that represents the main features of all population *reference representative set*

## 2.5. Natural Language Processing in Software Issues Detection

Before solving architectural erosion insights into a software project, you have to detect the real violation types that could appear in your software project. There are some approaches for software issue detection using actual natural language processing methodologies that have been powered with Machine Learning Techniques' help. Natural Language Processing (NLP) gives the ability to extract relevant information for great text sets named corpus. Nowadays, NLP is very useful for many tasks like text generation or classification. Even with a different approach, the NLP actual tools have had a better performance of the same tasks in code due to its standard structure, which does not present different language variations that could present in a spoken language.

NLP has a set of different preprocessing methods for language models. Due to the complexity of text representation for computer processing, it is necessary to define a standard input structure for a model language. Before this, you have to make a series of processes for getting the standard structure of a given corpus of text. The process that enables those actions is named text normalization Jurafsky [4]. With the use of regular expressions, you could build a dictionary of words, which is useful for getting a standard structure and enable the text words with the modeling task. The main stages and process for the dictionary building are:

- Tokenization: Given a character sequence, in this case, a sequence of words of a given context, you can split that sequence into minimal processing units named tokens. These tokens are normally defined as terms of words. These tokens will create the base dictionary to begin the text processing into a language model Manning [7].

## 2.5. NATURAL LANGUAGE PROCESSING IN SOFTWARE ISSUES DETECTION

Input: Friends, Romans, Countrymen, lend me your ears;  
Output: Friends Romans Countrymen lend me your ears

Figure 2.4: Tokenization process of one sentence Manning [7]

- **Removing Stop Words:** In the basic language modeling tasks, it is necessary to remove words that do not have relevant semantic information inside a text or a text corpus; those words are named Stop Words. Stop Words are words that do not contribute to the meaning of a sentence, like prepositions, articles, etc. In actual language models (Large Language Models), it is very important to maintain Stop Words to get a better specific context for next-word prediction or sentence classification. There are different strategies for removing those words; the most common is removing by collection frequency, due to the number of appearances in the corpus, which is enormous compared with relevant words. In document retrieval, the rare words are the most important for giving an efficient model over the text corpus Manning [7].
- **Lemmatization:** For new tokens controlling and tokens derivations, is essential to create tokens since the roots of the words, to reduce inflectional forms and related forms of words with the same root. In some cases, it is difficult to implement that process because you must have a root word dictionary to get root tokens. This case, in different contexts, could generate conflicts for getting roots of specific context words [7].
- **Stemming:** This process consists of a heuristic process to cut off some characters at the end of each word, reducing the derivation of some words, with the same objective of the lemmatization process. In English, language could be an efficient technique but could have some conflicts with other languages.

### 2.5.1. Word Embeddings for Words Representation

As said in the last section, language models need to have a numerical representation of the corpus text for modeling tasks. To solve this, you must define a standard structure based on the decided model inputs. The most common structure is a word embedding representation, where you define a numerical vector for representing into a specific context (where embedding was trained) for



## CHAPTER 2. DEFINITIONS

use as input into a language model. This representation gives all model vocabulary a vector representation, where you can observe similar words, different words, and how much distance is between them. This representation is useful for similarity word management and getting the relationships between different features inside them Jurafsky [4].



Figure 2.5: Representation in tow dimensions of Similar Words give a Word Embedding Jurafsky [4]

### 2.5.2. Performance and Similarity Metrics in NLP

It is very important to maintain and define an objective in terms of performance. Different metrics represent the behavior of a classic or modern language model based on next-word probabilities (like the anagram model when you generate an n-tuple of words and calculate the occurrence probability of that word sequence). For information retrieval, when you, in the same case of word embedding representation, have a vector representation, you must use a metric based on the vector's component. In the same dimension, you could determine the similarity between two vectors and, in the case of NLP context, verify the semantic similarity between two words. One of the most used metrics in this approach is cosine similitude. This metric combines the product between the two vectors and the difference between their components. The Similitude Cosine metric is defined as:

$$\cos(v, w) = \frac{v \cdot w}{|v||w|} = \frac{\sum_{i=1}^N v_i w_i}{\sqrt{\sum_{i=1}^N w_i^2} \sqrt{\sum_{i=1}^N v_i^2}} \quad (2.1)$$

With this metric, we can conclude the similarity between two words (or two documents without another research context). If the similarity value is big, the words can be considered similar.

## 2.6. Programming Languages fundamental for Software

### Analysis

With the fundamentals of programming languages, it is possible to analyze the software building. Programming languages have a defined vocabulary, structure, and semantics. Those features make the use of AI tools and NLP techniques more effective. Furthermore, it is possible to create custom rules for checking a defined set of guidelines for a standard structure of different patterns in a software project. This is possible detecting different semantical, grammatical, and lexical patterns inside the source code of any software project. In general, a programming language is defined with the next components:

- Lexical component: In that component, we define the vocabulary and the set of words that will have a meaning for the programming language. For example, the word function in JavaScript programming language means a function declaration, or int in Java, which means the Integer primitive data type. It is necessary to define all the words that could be used in any source code file of that programming language.
- Grammatical component: With a defined set of words in the lexical component, the next step is to define the order that words could be written in a code block. It is essential to define all the possible structures that could be defined in any program and the different ways that could be written those structures. For example, most programming languages used in the industry have a defined structure of if statements and all the possible ways to write them.
- Semantical component: If we have a set of words and a defined order to write them, it is possible to build a kind of translator for a programming language. In this step, we define the type of translator with two options: for executable program building, that is a compiler, one example of it is C language programming, that a semantic visitor generate a executable program, trough of a translation process between C code fragments and machine instructions. The other option is to make an interpreter, when in execution time, we visit line by line of code and translate it into a machine instruction, one example of that way is

AST for the code : if a = b then return "equal" else return a + " not equal to " + b

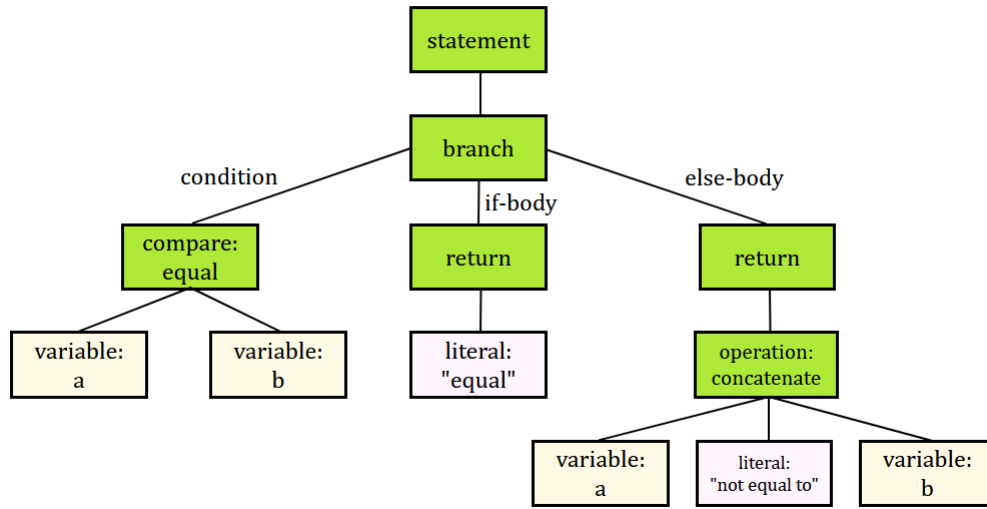


Figure 2.6: Example of an Abstract Syntax Tree (AST) statement. ]

Python programming language. In both approaches, we specify the translation strategies mainly with two components: a visitor component and a call graph component.

- Visitor component: A visitor component consists of a structure built from grammatical and lexical components of a programming language. In that structure, we can find how all the statement code blocks are defined in all source code files of a software project. We can find the name of every parameter declared in any function and the name of any class of any code statement defined in any source code file. With this component, we can detect any pattern in names and data types inside all code fragments and combine them for more customized check rules (security issues, connectivity issues, and other ones).
- Call Graph component: The call graph component is very similar to the visitor component. The main difference is that we can find all the dependency relationships between all the source code files of a software project. With this dependency structure, we can detect the high dependency between components and their high coupling rate.

## 2.6. PROGRAMMING LANGUAGES FUNDAMENTAL FOR SOFTWARE ANALYSIS

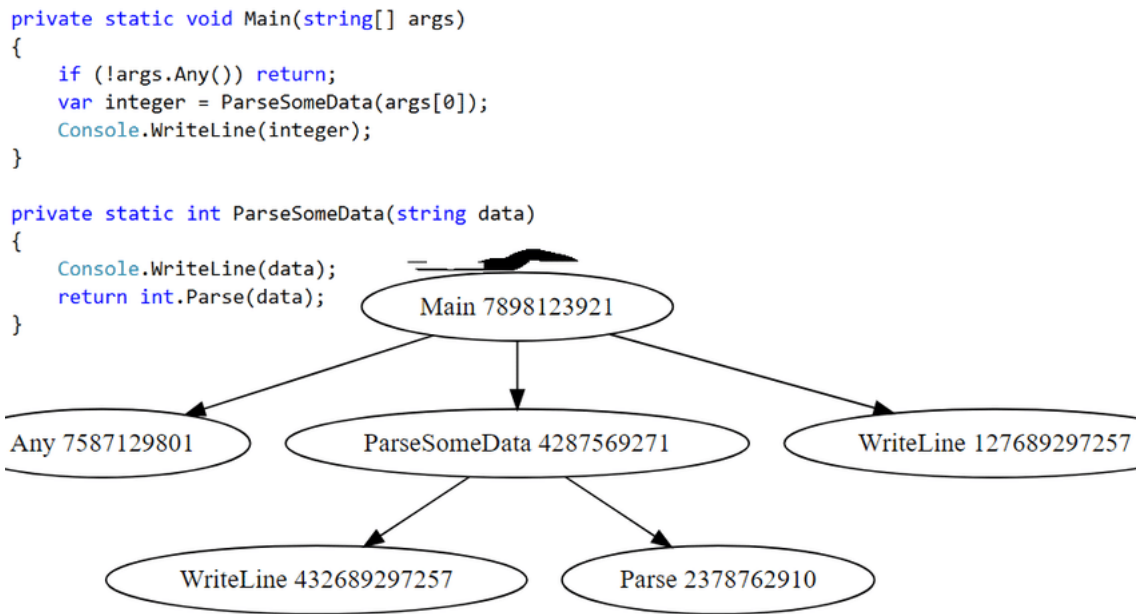


Figure 2.7: Example of an Abstract Syntax Tree (AST) statement. ]

With all the programming language components and the way of translating it to machine instructions, it is possible to make custom check rules for different violations detection to different standards. In architectural erosion detection, these concepts will be fundamental.

## *CHAPTER 2. DEFINITIONS*

# CHAPTER 3

## Related Work

---

### 3.1. Research Methodology

For architectural erosion symptoms, causes, and consequences detection, during the research, it was necessary to realize a series of steps for identification and possible recovery processes. Due to the orientation of the actual architectural erosion solvers for solving them mostly for Backend and Frontend projects, it is necessary to identify architectural erosion symptoms in Android apps. After that, with human judges, these symptoms have to be confirmed, and, finally, those symptoms will generate architectural rules for detecting them and suggest different recovering ways during the coding stage of a system.

#### 3.1.1. Architectural Erosion: An Initial Overview

As the first approach to define, explain the concept of architectural erosion in software engineering and to set the relationship between this concept and the mobile development ecosystem. In the first approach, we look for advances in static analysis solution approach in architecture quality gates in Server-Side and Frontend applications *reference paper uniandes*. In this case, we locate the cited related work in this research for search the initial motivation for solving architectural erosion issues, and find quality gates in terms of performance in Android ecosystem.

The first overview we extracted about the related in Ruiyin et al. [9]. In this paper, we find an Systematic Literature Review (SLR) where define the definition, reasons, symptoms,

### CHAPTER 3. RELATED WORK

consequences and solution approaches about architectural erosion. In resume, this approach finds 73 relevant papers about 8 research questions related with the last mentioned features. From this approach, we can execute the same query in different researches papers database for searching more actual researches, due to the publication year of this research (2021), and the time period criteria (between 2006 and 2009) for finding recent researches and advances during the last three years. The paper searches relevant papers in 7 researches databases, and make a better performed query, due to the relation of "architectural erosion" concept in civil engineering and a phenomenon presented in buildings. We present the first used search queries and its databases:

Query
("software" OR "software system" OR "software engineering") AND ("architecture" OR "architectural structure" OR "structural") AND ("erosion" OR "decay" OR "degradation" OR "deterioration" OR "degeneration")

Table 3.1: Executed Query for related work search

Since that first results, we can extract relevant papers that include new developments approaches for fixing architectural erosion issues, metrics, tools and more related work. With this, we identify in each paper three main stage for solving that issues. The detection stage, where we use different identification alternatives through developer's messages in versioning systems like for example GitHub. The Detection Stage, where we use Model Driven Development (MDD) and code patterns detection for detect the identified architectural violations rules. Finally, based on the solution approach (Design approach, Quality approach, etc), we can suggest a solution proposal for solving the detected architectural violations. In the first search iteration, we added another filter related with the publication year. We selected the papers that their publication year is between 2021 and 2024, that papers include in their bibliography the first SLR that use as reference research. Despite different troubles with the query, when different databases show papers related with civil engineering or architecture (this because the architectural erosion definition in that context is another research topic about building degradation). In this first research iteration, we found researches that synthesize different solution approaches and give

### 3.2. ARCHITECTURAL EROSION SYMPTOMS IDENTIFICATION

an overview from different perspectives, since the process of AER issues detection, metrics that could indicate a possible architectural violation in a software project like coupling metrics and relationship between classes. We observed that the use of NLP techniques could improve the AER issues detection performance based on commit messages analysis. With the use of pre-trained Word Embedding models trained in specific context, different AER issues are detected and a list of potential keywords could be generated as a type of alert of an architectural violation based on architectural model base. In the symptoms and causes approach, analysis of different base applications made in different programming languages was made. Different alternatives and methodologies were evaluated for determine the performance in architectural erosion issues detection and different solution approaches were proposed.

### 3.2. Architectural Erosion Symptoms Identification

In the stage of detection, the latest researches give feedback about identified architectural erosion symptoms and their types during different project stages. However, the detected and named symptoms are oriented to Frontend and Backend development. For mobile development oriented to Android technology, specifically made in Kotlin programming language, it doesn't exist a repository of possible architectural erosion symptoms. The most recent researches use NLP techniques based on GitHub commits. That researches analyze the main keywords written in GitHub commits that could indicate a bad architectural issue implementation, identifying a possible architectural erosion issue. With large amount of data from Git repositories is possible to make NLP analysis of that Github commits and define metrics that identify (from previously selected commits tagged by expert judges in software engineering and software architecture) similar keywords for architectural erosion. The metrics performance could be affected by the word embedding training context. The most recent research that implements that identification methodology, use a Word Embedding model trained with 10 million of Stack overflow posts, a context that use technical definitions for define features, bugs, recommendations and more of different programming languages. With this context, that word embeddings could be more powerful and efficient for detect architectural erosion issues keywords [6, 3]. Another approaches are similar to the mentioned previously. However, that approaches are only based on architectural



conformance checking. That consist on a set of different statements for a couple of judges that are professionals in software development and software architecture

### **3.3. Architectural Erosion Symptoms Detection**

### **3.4. Giving Architectural Erosion Solutions**

The latest researches give feedback about identified architectural erosion symptoms and their types during different realization project stages. However, the detected and named symptoms are oriented to Frontend and Backend development. For mobile development oriented to Android technology, specifically made in Kotlin programming language, it doesn't exist a repository of possible architectural erosion symptoms. To get this, we use an artificial intelligence approach for detecting architectural changes, and those changes and their change messages (commits in git world) are useful for identifying possible symptoms and generating rules to prevent them.

# CHAPTER 4

## Finding Architectural Erosion Issues

---

### 4.1. Architectural Erosion Keywords in Commits

#### 4.1.1. Architectural erosion symptoms Identification

Based on the last researches, architectural erosion insights identification consist on a deep analysis of information based on development judgments. This judgment can be extracted from different software versioning systems, like for example OpenStack, a software versioning platform for large-scale software projects, or GitHub, the most used software versioning platform. From those messages, we can classify and tag different code changes through code differences between code changes over time. In this process, the use of different Natural Language Processing (NLP) techniques, to find a standard architectural erosion commit definition. One innovative idea is to employ pre-trained Word Embeddings in software development contexts for potential words that could indicate an implemented architectural violation in server-side applications. With the use of embeddings and word similarity metrics like Cosine Similitude, allows us to calculate the similarity between the word embedding's numerical representation for every word since semantic function. With this study, we can standardize the main cases of architectural erosion, the different metrics that could be identified in a software project, and potential solution approaches. With those NLP metrics, we can use them to create discriminatory models for issue detection. However, in an Android context, the architectural erosion identification hasn't been enough to standardize a set of rules for detecting it in software source code due to the Backend

## CHAPTER 4. FINDING ARCHITECTURAL EROSION ISSUES

and Frontend solution approaches implemented in the study software projects (the two main projects are developed in Python). With the results of the papers extracted in the first related research overview, we can find a set of keywords extracted for the developer's code messages that indicates a potential architectural erosion issue in the implementation, that study was realized with the developer's judges and messages extracted from different version platforms like Open-Stack (previously mentioned) and Github, the most popular and used versioning platform. In this case, the project had as a reference four large open-source software projects; all applications are server-side applications. Around 50 keywords were mentioned as potential keywords that indicate an air issue. However, the words were extracted for Backend development purposes. For this reason, we use the same word extraction approach in a mobile context. During this process, we extracted from 50 open-source Android applications published around 470k GitHub commits. This is enough detail to make a preprocessing of vocabulary implemented in each GitHub commit and create new rules that could be implemented with custom lint check rules. That topic will be treated in the next chapter.

### Extracting keywords from Android Context

Based on the last mentioned research, it is possible to find potential keywords that indicate an issue or an insight inside a code implementation, with the help of NLP techniques, through similarity measurements like cosine similarity (mentioned in the definitions chapter) and pre-trained Word Embeddings, due to the numerical representation of each word of the generated vocabulary in a specific context. First, we use the PyDriller library *url PyDriller*, a useful library for repository mining, for getting code source and its attributes of different open-source Android projects made in Kotlin. The selected projects were extracted from different open-source Android project catalogs like F-Droid and other data mining repositories found with different filters like development programming language used, number of commits, and keywords in the selection criteria *url fdroidurl search*. In the first overview, we extracted 50 Android projects that have around 470K commits. With these commits, we made a text pre-processing to build a standard vocabulary and tokenize with the help of NLTK library *url nltk*, a library for making NLP operations like tokenization, lemmatization, and stemming.

#### 4.1. ARCHITECTURAL EROSION KEYWORDS IN COMMITS

Keyword
architecture, architectural, structure, structural, layer, design, violate, violation, deviate, deviation, inconsistency, inconsistent, consistent, mismatch, diverge, divergence, divergent, deviate, deviation, architecture, layering, layered, designed, violates, violating, violated, diverges, designing, diverged, diverging, deviates, deviated, deviating, inconsistencies, non-consistent, discrepancy, deviations, modular, module, modularity, encapsulation, encapsulate, encapsulating, encapsulated, intend, intends, intended, intent, intents, implemented, implement, implementation, as-planned, as-implemented, blueprint, blueprints, mis-match, mismatched, mismatches, mismatching

Table 4.1: List of initial Keywords extracted of mentioned related work

Column	Description
Name Repo	Name of the GitHub repository of Android project source code
Url Repo	GitHub URL from source code repository
Commit Message	Message of a specific commit in GitHub commits history of each Android project
Commit Hash	Hash from GitHub commit, essential for commits analysis process
File Name	List of GitHub commit modified file names
Code Changes	String with the modified source code of each GitHub commit

Table 4.2: Features of commits dataset and their description

With all the corpus from GitHub commits, we implement a text cleaning process, removing stop words and making stemming, due to the lemmatization process in a technical context is not very effective, some words do not have their respective lexical root, so the tokenizer would not consider that words. The stemming process is useful for semantic word derivation control. This

## CHAPTER 4. FINDING ARCHITECTURAL EROSION ISSUES

process is essential because a lot of words do not have semantic relevance in each GitHub commit, so consider that words could affect similarity metrics. With the processed words, we use a pre-trained Word Embedding based on millions of Stack Overflow posts. With the Gensim library *librería de gensim*, we can load the Word Embedding model get a numerical representation of each selected word, and use the cosine similitude metric for find similar words from the previous keywords. When the metric is generated, we select the 10 most similar words based on that metric. In the first overview implementation, we found around 5000 relevant commits with the updated keywords list. For efficient selection criteria, we extract a representative subset based on a weighted average made by the word frequency in the corpus text. With the first selection of a representative set, we extracted 357 GitHub commits.

Word	Cosine Similitude Average Value
<b>notion</b>	0.2674
<b>respect</b>	0.2627
<b>formal</b>	0.2482
<b>high-level</b>	0.2437
<b>tend</b>	0.2342
<b>rigid</b>	0.2315
<b>kind</b>	0.2256
<b>stronger</b>	0.2243
<b>non-linear</b>	0.2227
<b>sane</b>	0.2198

Table 4.3: Top 10 newfound words since Word Embedding cosine similitude metric

With this approach, it is possible to find potential words written in a development context that could indicate a potential issue related to different kinds of functional and nonfunctional requirements that a software project includes in its architectural design and its standards. This detection approach has many different development areas to detect different problems found in a software project. The future work related with this approach will be discussed in next chapters.

# CHAPTER 5

## Detecting Architectural Erosion Issues

---

With a set of possible causes presented in Kotlin source code files from different Android projects, it is possible to define a set of rules with the help of an Android Studio IDE tools ecosystem for Architectural Erosion issues detection. We detected architectural changes in the representative set of commits and extracted different rules and patterns that could handle an architectural violation inside the MVVM architectural pattern for Android Applications. Rules could be implemented in any IDE that supports mobile development oriented to Android, like, for example, Visual Studio Code or Sublime Text (despite its constraints).

### 5.1. UAST in Android Studio

The Android Studio Integrated Development Environment (IDE) is the most used for mobile development and Android applications building. Inside its ecosystem, there are a lot of built libraries, frameworks, and other programs that make the development process easier and get a better performance in terms of the different nonfunctional requirements and architectural standards. In addition to those programs, other tools avoid committing any bad practice implementation in terms of different standards based on a specific architectural style. This tool is named linter. A linter is an integrated tool with an IDE for code insight detection and correction (if it is possible). When we write code in Android Studio with a bad code practice, its interface shows and marks a possible insight into our code. After that, we select that marker and give a possible solution of that code insight. Every rule shows its name, its description, a possible

reason for the insight, and an optional code implementation solution.

This linter implementation integrated with the Android Studio IDE is due to the UAST structure generated by that tool. The Unified Abstract Syntax Tree (UAST) is a defined structure generated by each software project opened by the IDE that contains the AST and the Call graph structures of the source code files of that project. With this structure, we can create different semantical components (similar to the visitors components mentioned in the definitions section) that detect any pattern in different structures and code fragments inside the source code files. We can instantiate the different structures based on the Kotlin programming language grammar, since it clauses to class and another ones. Furthermore, we can access the different attributes of each structure, like name, source code file name, or package name, and their relationships with other components. In terms of rules, we can define any customized lint check rule, and with that structure's set of Kotlin programming language and their attributes, it is possible to detect any pattern and offer any solution suggestion.

Based on the standard rules implemented in the Android Studio IDE, we can similarly define custom lint check rules for aer detection based on the insights reviewed in the representative set of commits of open-source Android projects.

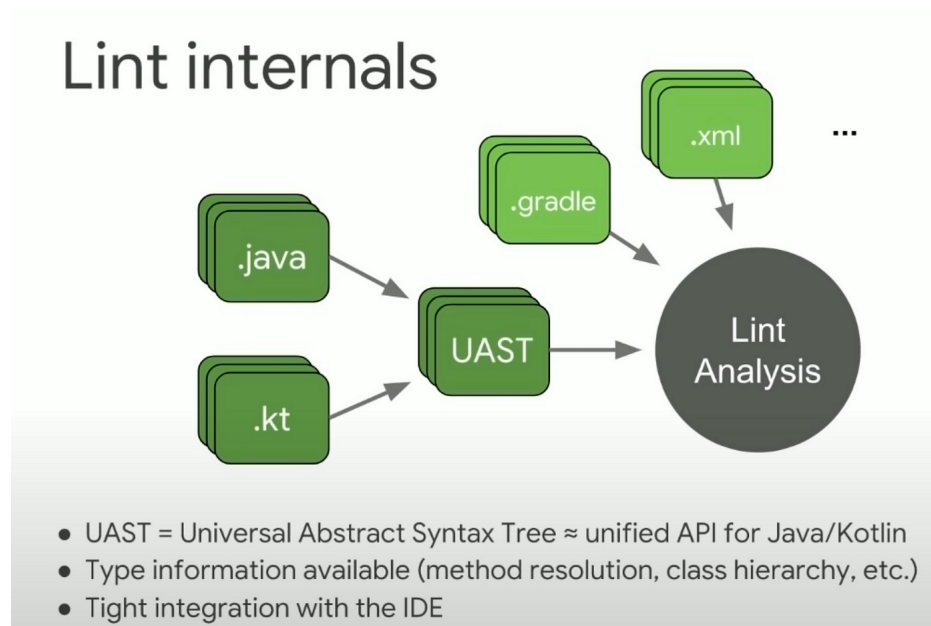


Figure 5.1: Definition of Android Studio UAST structure ]

## **5.2. Rules Definition**

## **5.3. AER Detector implementation**

## **5.4. AER linter tool testing criteria**





# CHAPTER A

TITLE

---



## Bibliography

---

The references are sorted alphabetically by first author.

- [1] A. Baabad, H. Zulzalil, S. Hassan, and S Baharom. Characterizing the architectural erosion metrics: A systematic mapping study. *IEEE*, 2022.
- [2] L. Bass, P. Clements, and R Kazman. *Software Architecture in Practice, 4th Edition*. O'reilly, 2021.
- [3] V. Efstathiou, Chatzilenas C., and D Spinellis. Word embeddings for the software engineering domain. *IEEE*, 2018.
- [4] J Jurafsky, D. Martin. *Speech ad Language Processing. An Introduction to Natural Language Processing, Computational Linguistics, and Speech Recognition with Language Models*. Stanford Web, 2024.
- [5] M Laplante, P. Kassab. *Requirements Engineering for Software and Systems. 4th Edition*. O'reilly, 2022.
- [6] P. Liang, P. Avgeriou, and L Ruiyin. Warnings: Violation symptoms indicating architecture erosion. *Cornell University*, 2023.
- [7] P. Schutze H. Manning, C. Raghavan. *An Introduction to Information Retrieval*. Cambridge University Press, 2009.
- [8] Alexander L Perry, Dewayne. Wolf. Foundations for the stufy of sotware architecture. *ACM SIGSOFT*, 1992.
- [9] L. Ruiyin, L. Peng, P. Avgeriou, and M Soliman. Understanding software architecture erosion: A systematic mapping study. *Wiley*, 2021.

## *Bibliography*

## Acronyms

---