جامعة الإمام محمد بن سعود الإسلامية
IMAM MOHAMMAD IBN SAUD ISLAMIC UNIVERSITY

كلية علوم الحاسب والمعلومات
College of Computer and Information Sciences

Workshop1

Team Members:

Al-juhara Al-sadoun - 440023448
Alya Al-shaikh - 440022910
Shamaa Elewa - 440022232
Layan Al-jabali - 440022562

Instructor:

Dr. Lamees Alhazzaa

1.0 Introduction

1.1 Purpose:

Code Quality is a term popular among the developers, where maintaining it is an essential part of software development. How code quality impacts the overall software quality? Code quality refers to the usefulness and maintainability of the code in the long term. It covers both good and bad quality code. Code quality is very important for the successful implementation of any program. maintaining the quality of code is not that easy because it requires consistency in efforts with a focused mindset of the software development team to meet the quality goals.
Writing code should be considered as an essential investment on which return will follow almost immediately. For this, the code should be readable, consistent and documented which would be easier to review, leading to lower development efforts. A well-designed software that is equally less complex would be more robust and can be tested easily. In other words, high code quality can be one of the best ways to lower the technical debt. When a developer fails to heed the code quality, it can sometimes lead to additional rework in the code. This can eventually enhance the cost of the software.
code that is considered good:
-Does what it should.
-Follows a consistent style.
-It is easy to understand.
-Has been well-documented.
-It can be tested.

1.2 Goal:

Our goal is to make the code as easy as possible to understand. We want our code to be a quick skim, And to improve code quality by Using a coding standers. is the best ways to ensure high quality code. A coding standard makes sure everyone uses the right style. It improves consistency and readability of the codebase. This is key for lower complexity and higher quality. And code standers focus on the small thing in the code and in the details. And the small thing matter.

1.3 Lessons:

The lessons learned from this workshop is Quality should be a priority from the very start of development. There isn't always the luxury of time as development progresses. That's why it's important to analyze code before code reviews begin. And it's best to analyze code as soon as it's written.
We in this workshop will analyze and evaluate this system within specific standards,
For example, we will analyze the source code quality forAlinma Bank system within the following: code quality- clean code- code structure- security issues,
Each this source code quality has specific standards we will analyze, explain and evaluate each of them.
Also we will do the same for design.
This system is expected to accurately implement these standards.

2.0 Preparation

2.1 Preparation

The first step is to read what is required in the file (workshop)and then we download a tool called Docker then open (terminal) and write "docker pull opensecurity/mobile-security framework-mobsf", then second command line"docker run -it -p 8000:8000 opensecurity/mobile- security-framework-mobsf:latest" Then the second thing, I got a link that I put in the search bar in Google, then Google will send me to MOBSF I put in the box the link to download Alanma Bank, then a code analysis page appeared for me, through which I downloaded the analysis file and the Java code After all these steps, I was able to determine the quality of the code in general and determine what is required to be done from the workshop.
We read the code, analyze it and try to understand it We divided the requirements (code documentation clean code issues security code quality) on us, then we began to search for each topic and analyzed and understood the requirements, then we began to determine the quality of the code through the criteria we found and find security issues in the code and make sure that the code supports the principles of (clean code) .

2.2 Issue:

- The number of code files was too large so It was difficult to read and understand.
- The variables were not clear by name, and some of the classes were empty and we do not know their purpose.
- There were no constant standard for evaluating the quality of the code.
- I don't know how to evaluate the quality of the code or on what basis to evaluate it.
- There were very few sources of research on the topics, and we could not find sources for some of them.
- We had a problem with the Alanma Bank download link, when it was placed in the download box, it was rejected.

3.0 Source code quality

3.1 Code Quality:

Based on the idea that the code is as close as possible to a type of literary literature. And the code over time needs maintenance, reuse, and so on It can be said that the code is read more than it is written, so I say that the quality specifications of the code are to be readable (clear).

The code (the code itself, I don't mean the comments) should be clear. If the one who reads the code cannot understand the meaning intuitively, it will make it difficult for the programmer who wants to complete the code or maintain it, and this leads to frustration and the failure of the project in the end.
Also, the code must be economical so that it does not consume a lot of system resources (RAM, memory, CPU, etc.) so that it saves costs because if the code costs a lot of resources to run, this will make the economic cost large, which leads the investor to cancel The project if the return does not cover the costs.

Why Code Quality Matters ?

It is important to pay attention to the code quality, since it impacts the overall quality of your codebase. And, quality impacts how safe, secure, and reliable your codebase is.
Good Code vs. Bad Code:

-Good code is clean code, and it stands the test of time. Bad code is low quality. It won't last long.
Code that is considered.
-good should do what it should follow a clear style
-be well-documented.
-It can be tested.

Aspects of code quality that should be measured:
Here are some of the key traits to measure for higher quality. The source code of (Alinma bank) have medium quality cause
it is apply some of four measure for higher quality .

Reliability:

Reliability measures the probability that a system will run without failure over a specific period of operation. It relates to
the number of defects and availability of the software.

Maintainability:

In my opinion our code source doesn’t support principle Maintainability
Maintainability measures how easily software can be maintained. It relates to the size, consistency, structure, and complexity of the codebase. And ensuring maintainable source code relies on a few factors, such as testability and understandability.

One of the ways to reach Maintainability is to make your functions and classes responsible for only one thing, and so you will not have to change those things for one reason only.

Some method helps to support ' Maintainability '

Builder Pattern:

Its job is instead of creating the object directly, you will call a build function (or a static factory - remember it first before the build function!-) with the required base values and you will get a Builder object and then you will call functions inside that object you can say is a setter method for optional values Finally, you call the build function inside the builder to return the object (which is Immutable).
Our code doesn't support method (Builder Pattern).

This picture is an external example of how to make (Builder Pattern) show in figure1 .

```java
// Pizza Construction using Builder Pattern

public class PizzaBuilderPattern {
    public static void main(String[] args) {
        Pizza pizza = new Pizza.Builder(10).cheese(true).peppernoi(true).build();
    }
}

class Pizza {
    // required
    private int size;

    // optionals
    private boolean cheese;
    private boolean hotdog;
    private boolean peppernoi;

    public static class Builder {
        private int size;

        private boolean cheese;
        private boolean hotdog;
        private boolean peppernoi;

        public Builder(int size) { this.size = size; }

        public Builder cheese (boolean cheese) { this.cheese = cheese; return this; }
        public Builder hotdog (boolean hotdog) { this.hotdog = hotdog; return this; }
        public Builder peppernoi (boolean pep) { this.peppernoi = pep; return this; }

        public Pizza build () { return new Pizza(this); }
    }

    private Pizza (Builder builder) {
        this.size = builder.size;
        this.cheese = builder.cheese;
        this.hotdog = builder.hotdog;
        this.peppernoi = builder.peppernoi;
    }
}
```

Figure1

This picture See how the object is configured show in figure2:

```java
Pizza pizza = new Pizza.Builder(10).cheese(true).peppernoi(true).build();
```

Figure2

Portability:

Portability measures how usable the same software is in different environments. It relates to platform independency. There isn't a specific measure of portability. We can use same App Alinma Bank in different Operating system and its communicated with different environment.

Reusability:

In my opinion our code source support principle Reusability
Reusability measures whether existing assets — such as code — can be used again. Assets are more easily reused if they have characteristics such as modularity or loose coupling.
Reusability can be measured by the amount of interdependencies.
There are many relationships between classes Extended from each other and have a lot of subclasses
Code that does not reinvent the wheel and uses the available libraries as much as possible import a lot of other libraries class from different environment Example:
Import a lot of libraries

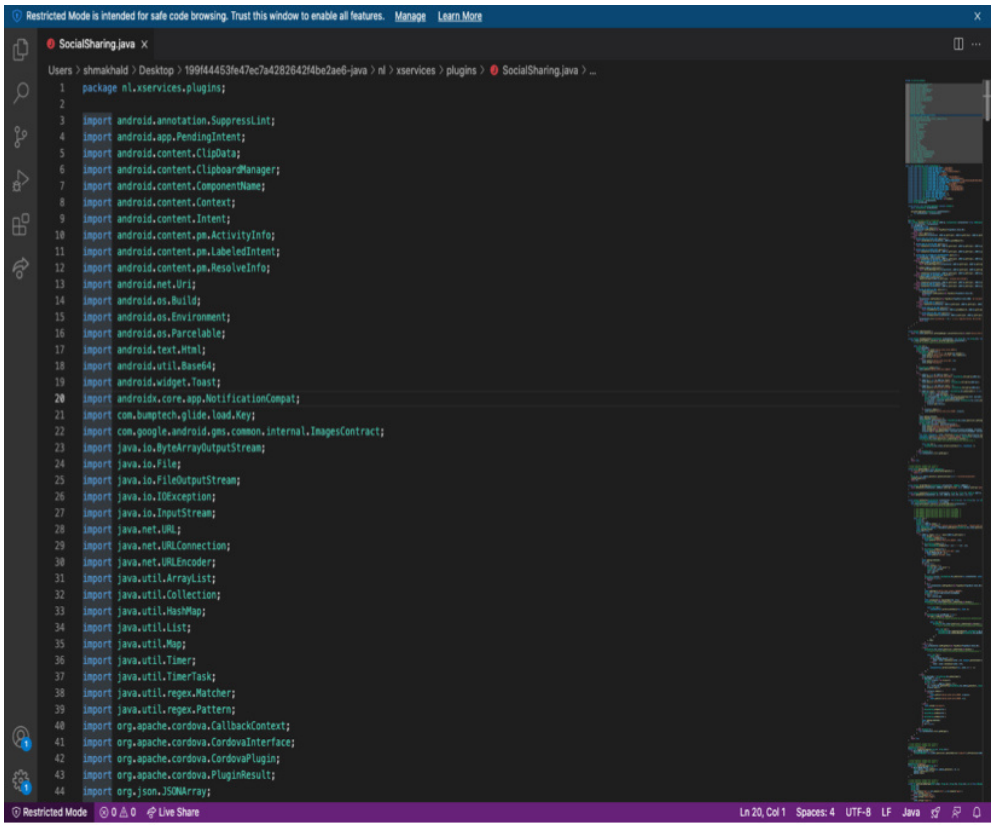

Figure3

Apply the principle of inheritance

Figure4

Some step to make quality code high :

1. Use a Coding Standard

It is crucial to ensure high quality code by using a coding standard. A coding standard ensures that everyone uses the same usage style, and it improves consistency and readability. This leads to lower complexity and higher
quality.

2. Analyze Code — Before Code Reviews

It's important to begin with quality from the early stages of development. As development progresses, time may be of the essence. That's why it's important to analyze code before code reviews begin. And it's best to analyze code the moment it's written.

3. Follow Code Review Best Practices

In order to verify the intent of the code, manual code reviews are still useful. When done properly, they improve software quality.

4. Refactor Legacy Code (When Necessary)

It is important to refactor code to improve its quality. Refactoring your code will help you reduce the complexity of your codebase. Coding Standards Coding standards are a good way to make sure your code is high quality.

By using the same style, they improve the readability and consistency of the code.

In the end, the quality of the code may vary from one system to another.

The quality of the code depends on the requirements of the system and the institution, and some standards may have to be set, but the focus of this standard varies, for example, some systems need very high protection and some need medium protection, and thus the rest of the standards are applied.

3.2 Clean Code:

Clean code is a set of rules and principles that helps to keep our code readable, maintainable, and extendable. It's one of the most important aspects of writing quality software. We (developers) spend way more time reading the code than actually writing it, which is why it's important that we write good code.
Writing code is easy, but writing good, clean code is hard.
The code we write should be simple, expressive, and free from more than a few duplicates. Expressive code means that even though we're just providing instructions to a computer, it should still be readable and clearly communicate its intent when read by humans.
Writing clean code has a number of benefits. For instance, clean code is:
- easy to understand
- more efficient
- easier to maintain, scale, debug, and refactor
It also tends to require less documentation.

Code Principles :
There are numerous coding principles you can follow to write better code, each having their own pros/cons and tradeoffs. This article covers three of the more popular principles: DRY, Boy Scot Rule, and SRP.

3.2.1 DRY:

DYR “Don’t Repeat Yourself” :
Every piece of knowledge must have a single, unambiguous, authoritative representation within a system.
This is one of the simplest coding principles. Its only rule is that code should not be duplicated. Instead of duplicating lines, find an algorithm that uses iteration. DRY code is easily maintainable. You can take this principle even further with model/data abstraction.
I will show examples of some parts of the code that got into this problem show in Figure1 Figure2 and Figure3 :

```
Object newInstance = MediaDescriptionCompatApi21.Builder.newInstance();
MediaDescriptionCompatApi21.Builder.setMediaId(newInstance, this.mMediaId);
MediaDescriptionCompatApi21.Builder.setTitle(newInstance, this.mTitle);
MediaDescriptionCompatApi21.Builder.setSubtitle(newInstance, this.mSubtitle);
MediaDescriptionCompatApi21.Builder.setDescription(newInstance, this.mDescription);
MediaDescriptionCompatApi21.Builder.setIconBitmap(newInstance, this.mIcon);
MediaDescriptionCompatApi21.Builder.setIconUri(newInstance, this.mIconUri);
Bundle bundle = this.mExtras;
```

Figure1

```
static {
    METADATA_KEYS_TYPE.put(METADATA_KEY_TITLE, 1);
    METADATA_KEYS_TYPE.put(METADATA_KEY_ARTIST, 1);
    METADATA_KEYS_TYPE.put(METADATA_KEY_DURATION, 0);
    METADATA_KEYS_TYPE.put(METADATA_KEY_ALBUM, 1);
    METADATA_KEYS_TYPE.put(METADATA_KEY_AUTHOR, 1);
    METADATA_KEYS_TYPE.put(METADATA_KEY_WRITER, 1);
    METADATA_KEYS_TYPE.put(METADATA_KEY_COMPOSER, 1);
    METADATA_KEYS_TYPE.put(METADATA_KEY_COMPILATION, 1);
    METADATA_KEYS_TYPE.put(METADATA_KEY_DATE, 1);
    METADATA_KEYS_TYPE.put(METADATA_KEY_YEAR, 0);
    METADATA_KEYS_TYPE.put(METADATA_KEY_GENRE, 1);
    METADATA_KEYS_TYPE.put(METADATA_KEY_TRACK_NUMBER, 0);
    METADATA_KEYS_TYPE.put(METADATA_KEY_NUM_TRACKS, 0);
    METADATA_KEYS_TYPE.put(METADATA_KEY_DISC_NUMBER, 0);
    METADATA_KEYS_TYPE.put(METADATA_KEY_ALBUM_ARTIST, 1);
    METADATA_KEYS_TYPE.put(METADATA_KEY_ART, 2);
    METADATA_KEYS_TYPE.put(METADATA_KEY_ART_URI, 1);
    METADATA_KEYS_TYPE.put(METADATA_KEY_ALBUM_ART, 2);
    METADATA_KEYS_TYPE.put(METADATA_KEY_ALBUM_ART_URI, 1);
    METADATA_KEYS_TYPE.put(METADATA_KEY_USER_RATING, 3);
    METADATA_KEYS_TYPE.put(METADATA_KEY_RATING, 3);
    METADATA_KEYS_TYPE.put(METADATA_KEY_DISPLAY_TITLE, 1);
    METADATA_KEYS_TYPE.put(METADATA_KEY_DISPLAY_SUBTITLE, 1);
    METADATA_KEYS_TYPE.put(METADATA_KEY_DISPLAY_DESCRIPTION, 1);
    METADATA_KEYS_TYPE.put(METADATA_KEY_DISPLAY_ICON, 2);
    METADATA_KEYS_TYPE.put(METADATA_KEY_DISPLAY_ICON_URI, 1);
    METADATA_KEYS_TYPE.put(METADATA_KEY_MEDIA_ID, 1);
    METADATA_KEYS_TYPE.put(METADATA_KEY_BT_FOLDER_TYPE, 0);
    METADATA_KEYS_TYPE.put(METADATA_KEY_MEDIA_URI, 1);
    METADATA_KEYS_TYPE.put(METADATA_KEY_ADVERTISEMENT, 0);
    METADATA_KEYS_TYPE.put(METADATA_KEY_DOWNLOAD_STATUS, 0);
}
```

Figure2

```
Bundle bundle = null;
MediaDescriptionCompat mediaDescriptionCompat = null;
MediaDescriptionCompat mediaDescriptionCompat2 = null;
MediaDescriptionCompat mediaDescriptionCompat3 = null;
Bundle bundle2 = null;
Bundle bundle3 = null;
Bundle bundle4 = null;
Bundle bundle5 = null;
RatingCompat ratingCompat = null;
Bundle bundle6 = null;
Bundle bundle7 = null;
Bundle bundle8 = null;
KeyEvent keyEvent = null;
MediaSessionCompat.ResultReceiverWrapper resultReceiverWrapper = null;
```

Figure3

In all of these photos, they did not apply this principle.
The cons of the DRY principle are that you can end up with too many abstractions, external dependency creations, and complex code. DRY can also cause complications if you try to change a bigger chunk of your codebase. This is why you should avoid DRYing your code too early. It's always better to have a few repeated code sections than wrong abstractions.

3.2.2 Boy Scout Rule:

Not everyone was a scout in his youth. Most of programmers, however, have probably heard of the Boy Scout Rule in software development. But can you name a few of the activities that are part of that rule? When I started to explore this topic, looking for what exactly needs to be done to say "I just applied the boy scout rule", I only found one thing: "Leave your code better than you found it" - everyone quotes Uncle Bob. But what does it mean? What am I supposed to do exactly?

In scouting it is easier, everyone can see if the surroundings are clean, what can be cleaned to make it cleaner, but in the code? I will try to show you a few examples from my daily work where I try to stick to this principle.

Most of us use some IDE, they are getting smarter and in many cases, they keep the code clean, suggest better solutions, "clean up" automatically with one click. However, it's worth keeping an eye on what's happening in the code and not rely entirely on the IDE.

For example, I must leave the method clean and do not modify it every time I call it !

I will show examples of some parts of the code that got into this problem show in Figure4 Figure5 and Figure6 :

```
    public void onCreate(Bundle bundle) {
        loadConfig();
        LOG.setLogLevel(this.preferences.getString("loglevel", "ERROR"));
        LOG.i(TAG, "Apache Cordova native platform version 8.1.0 is starting");
        LOG.d(TAG, "CordovaActivity.onCreate()");
        if (!this.preferences.getBoolean("ShowTitle", false)) {
            getWindow().requestFeature(1);
        }
        if (this.preferences.getBoolean("SetFullscreen", false)) {
            LOG.d(TAG, "The SetFullscreen configuration is deprecated in favor of Fullscreen,
and will be removed in a future version.");
            this.preferences.set("Fullscreen", true);
        }
        if (!this.preferences.getBoolean("Fullscreen", false)) {
            getWindow().setFlags(2048, 2048);
        } else if (!this.preferences.getBoolean("FullscreenNotImmersive", false)) {
            this.immersiveMode = true;
        } else {
            getWindow().setFlags(1024, 1024);
        }
        super.onCreate(bundle);
        this.cordovaInterface = makeCordovaInterface();
        if (bundle != null) {
            this.cordovaInterface.restoreInstanceState(bundle);
        }
    }
```

Figure4

```
    }
    imageButton.setContentDescription("Close Button");
    imageButton.setId(Integer.valueOf(i).intValue());
    imageButton.setOnClickListener(new View.OnClickListener() {
```

Figure5

```
public synchronized int registerCallback(CordovaPlugin cordovaPlugin, int i) {
    int i2;
    i2 = this.currentCallbackId;
    this.currentCallbackId = i2 + 1;
    this.callbacks.put(i2, new Pair<>(cordovaPlugin, Integer.valueOf(i)));
    return i2;
```

Figure6

In all of these photos, they did not apply this principle.

3.2.3 SRP:

emergence standard : SRP “Single Responsibility Principal” :

The SRP states that a given method/class/component should have a single reason to change. Concurrency design is complex enough to be a reason to change in it's own right and therefore deserves to be separated from the rest of the code. Unfortunately, it is all too common for concurrency implementation details to be embedded directly into another production code. Here are a few things to consider:
• Concurrency-related code has its own life cycle of development, change, and tuning.
• Concurrency-related code has its own challenges, which are different from and often more difficult than nonconcurrency-related code.
• The number of ways in which miswritten concurrency-based code can fail makes it challenging
enough without the added burden of surrounding application code. Recommendation: Keep your concurrency-related code separate from other code.

I will show examples of some parts of the code that got into this problem show in Figure7 Figure8 Figure9 and Figure10 :

```
/* access modifiers changed from: package-private */
@SuppressLint({"TrulyRandom"})
public int generateBridgeSecret() {
    this.expectedBridgeSecret = new SecureRandom().nextInt(Integer.MAX_VALUE);
    return this.expectedBridgeSecret;
}
```

Figure7

```
public synchronized int registerCallback(CordovaPlugin cordovaPlugin, int i) {
    int i2;
    i2 = this.currentCallbackId;
    this.currentCallbackId = i2 + 1;
    this.callbacks.put(i2, new Pair<>(cordovaPlugin, Integer.valueOf(i)));
    return i2;
}
```

Figure8

Figure9

Figure10

In all of these photos, they did not apply this principle.

It's not enough to write the code well. The code has to be kept clean over time. We've all seen code rot and degrade as time passes. So we must take an active role in preventing this degradation.
Clean code can be read, and enhanced by a developer other than its original author. It has unit and acceptance tests. It has meaningful names. It provides one way rather than many ways for doing one thing. It has minimal depen- dencies, which are explicitly defined, and pro- vides a clear and minimal API. Code should be literate since depending on the language, not all necessary information can be expressed clearly in code alone.

3.3 Code Structure:

In this section we discuss and assessment the code Structure and documentation in banking app.

Code Structure is to build a clean code that reduce many problem and the risk of project failures. Clean code helps development of software program especially when the System is large and contain a complex, so there are a higher chance of it begin vulnerable to error.

We well explain the code structure for the bank app system. The code structure was consistency it was meaningful the function was named according to they perform, And it was consistent indentation show in figure1 and figure2, The function of the code was separating with proper space.

```
/* access modifiers changed from: protected */
public Filter.FilterResults performFiltering(CharSequence charSequence) {
    Cursor runQueryOnBackgroundThread = this.mClient.runQueryOnBackgroundThread(charSequence);
    Filter.FilterResults filterResults = new Filter.FilterResults();
    if (runQueryOnBackgroundThread != null) {
        filterResults.count = runQueryOnBackgroundThread.getCount();
        filterResults.values = runQueryOnBackgroundThread;
    } else {
        filterResults.count = 0;
        filterResults.values = null;
    }
    return filterResults;
}
```

Figure1

```
static {
    if (Build.VERSION.SDK_INT >= 21) {
        IMPL = new CardViewApi21Impl();
    } else if (Build.VERSION.SDK_INT >= 17) {
        IMPL = new CardViewApi17Impl();
    } else {
        IMPL = new CardViewBaseImpl();
    }
    IMPL.initStatic();
}
```

Figure2

The mess build in the system was the variables style and the vertical alignment. For the variables name there was a different way for naming convention they use a CamleCase and UnderScore style show in figure3 and figure4. And they didn't care for the aligment of the code so it was a mess show in figure5 and figure6.

```
public static final int cardBackgroundColor = 2130968667;
public static final int cardCornerRadius = 2130968668;
public static final int cardElevation = 2130968669;
public static final int cardMaxElevation = 2130968670;
```

Figure3

```
public static final String APPLICATION_ID = "android.support.constraint";
public static final String BUILD_TYPE = "release";
public static final boolean DEBUG = false;
public static final String FLAVOR = "";
public static final int VERSION_CODE = -1;
public static final String VERSION_NAME = "";
```

Figure4

```
public static final int constraint_referenced_ids = 2130968727
public static final int content = 2130968728;
public static final int emptyVisibility = 2130968768;
public static final int layout_constrainedHeight = 2130968875;
public static final int layout_constrainedWidth = 2130968876;
```

Figure5

```
super(context, cursor);
this.mDropDownLayout = i;
this.mLayout = i;
this.mInflater = (LayoutInflater) context.getSystemService("layout_inflater");
```

Figure6

3.3.1Code Documentation

Code documentation is helps in understanding and correctly utilizing the software code. And having your application properly documented will make the development and maintenance efforts more efficient and will save you time and money in the long run.

In the bank app system we will explain the documentation for the classes. With many lines of code it is important to have good documentation. Well documentation content a good comment to describ and to understand the system but in this bank system there was a few comment that doesn't describes what is the function do and explain "What the code does" or "what is the code about" show in figure7 and figure8, write a good comment it can be so helpful and it needs to effectively communicate the purpose of a given implementation to the reader to understand the code espeacially in important large software system.

```java
@Override // androidx.cursoradapter.widget.CursorAdapter
public Cursor swapCursor(Cursor cursor) {
    findColumns(cursor, this.mOriginalFrom);
    return super.swapCursor(cursor);
}
```

Figure7

```java
/* access modifiers changed from: protected */
public Filter.FilterResults performFiltering(CharSequence charSequence) {
    Cursor runQueryOnBackgroundThread = this.mClient.runQueryOnBackgroundThread(charSequence);
    Filter.FilterResults filterResults = new Filter.FilterResults();
    if (runQueryOnBackgroundThread != null) {
        filterResults.count = runQueryOnBackgroundThread.getCount();
        filterResults.values = runQueryOnBackgroundThread;
    } else {
        filterResults.count = 0;
        filterResults.values = null;
    }
    return filterResults;
}
```

Figure8

The conclusin for the data structure:

the develompent should follow a coding standers to meet quality goals. To follow the standers there are a particular standers or style for the programming language, for instance:

-java- javadoc.

-python- doctest.

-ruby-docurim.

the coding standers make sure that all the developers follow a specified guidline. In this system the problem was in the varibles style they use different style they sholud follow languag standers and use one stlye but not both. Also the vertical algiment for the code was bad they didn't use it so the code was a mess, and for the documentation there was a few comments doesn't help to understand the code so we can consider the code is a bad code. Comments are necessary and helpful to translate the meaning of the code and explain it. So some developers doesn't care for the code comment or use proper documentation. A clear code helps the developers to reuse the code whenever required, coding standards play a vital role in any successful software development.

3.4 Security Issues:

Security is very important is software development the software should satisfies the unique requirements of the organization for flawless performance with minimal security risks If we want the systems to perform for years without failures or security breaches, it is important to work with a professional software development team that can design, develop and maintain the software with the latest innovations in security.

3.4.1 External Storge Permission:

Storage permissions are dangerous permissions for accessing the shared external storage. Full read and write access to any location of the volume is protected by two permissions marked as dangerous It can be harmful.

3.4.2 Data Backup:

Allowing app data backup Allowing app data to be backed up could be useful for the software developers but somehow it may lead to app hacking throw the data back up.

3.4.3 MD5 Collision Attack:

Weaknesses in the MD5 algorithm allow for collisions in output. As a result, attackers can generate cryptographic tokens or other data that illegitimately appear to be authentic.

3.4.4 MITM Attack Prevention:

An essential part of mobile man-in-the-middle attack prevention is user education. Certificate pinning is a great way to make mobile apps that handle sensitive information that much more secure and protect users too.

3.5 Design:

"The code is design" we must take care to remember that the interface is the program, and that its structures have much to say about the program structure.

Design patterns are largely about communication and expressiveness. By using the standard pattern names, the names of the classes that implement those patterns, you can succinctly describe your design to other developers.
In the system they use two type of design patterns. Adapter pattern and observe pattern. In the observe pattern is provide a loose coupling design between object that interact, loosely coupling object are flexible with changing requirements.
The adapter pattern you can separate the interface or data conversion code from the primary business logic of the program. And you can introduce new types of adapters into the program without breaking the existing clint code.

Design Patterns establishes solutions to common problems which helps to keep code maintainable, extensible and loosely coupled. And make a system seem less complex by letting you talk about it at a higher level of abstraction than that of a design notation or programming language.

3.6 Conclusions:

So, we in turn, analyzed and evaluated the code for Alinma Bank in several criteria, and we also evaluated the design for this system.
And it became clear to us in analyzing the code by dividing the analysis method for several criteria that it lacks some criteria for evaluation and analysis, and some of it is good and the concept of its analysis.
It is possible that the ambiguity in the code was in order to protect the system for the bank because this is very important for systems such as banks, but we as software engineers see that the system can become more clear and meet understandable standards.

3.7 Reference :

-ANAYA, Mariano. Clean Code in Python: Refactor your legacy code base. Packt Publishing Ltd, 2018.

- https://www.perforce.com/blog/sca/what-code-quality-and-how-improve-code-quality