Testing of NHS Contact Tracing Application

Emran Mohannad A.Altamimi
201510662
Dept. of Computer Science and Engineering
Ea1510662@qu.edu.qa

Summary:

NHS application is a contact tracing application for COVID-19 used in the UK for exposure notification, COVID-19 test booking, and venue check-in. The application was tested in this project using different techniques such as static testing techniques and dynamic testing techniques. The android version of the software was selected for analysis and testing. Static analysis tests were conducted using CodeMR and Lint tools to assess the quality of the code modules. The results of static testing showed a good quality of the software modules which was indicated by the low cyclomatic complexity in all methods in addition to other metrics such coupling, cohesion, size and depth of inheritance. Some issues related to performance and security have been found and described in this report. For dynamic testing, system testing was performed in terms of health status change of the phone carrier and notifications regarding COVID-19 test results and all test passed. In terms of unit testing, unit tests provided by the developers were assessed in terms of coverage metrics. The overall coverage of the tests was around 48% line coverage of the overall code. However, it was clear that test prioritization was performed by the developer as some packages were highly covered and some are not covered at all. To improve the coverage of the unit test suite, additional unit tests were developed in this project and executed successfully. The overall coverage was improved by around 3% and many packages with low or zero coverage were improved. In addition, mutation testing was also performed to assess the quality of the test suite. The results indicate that 4 mutants out of 17 were not killed which indicates that the test suite had some issues in detecting errors related to changing the Boolean return value and arithmetic operations and comparisons.

Keywords:

Contract tracing, Static test, JUnit, system test, Mutation test.

TABLE OF CONTENTS

1 INTRODUCTION

1.1 CONTACT TRACING APPLICATIONS

Following the outbreak of Covid-19, governments made use of technological advances to stem the spread of the virus. Contact tracing applications alert users if they have been in the proximity of a patient. Not only do these programs alert users if they are in close proximity to someone who has been infected, but they also alert users if someone they have been in contact with in the last two weeks has been afflicted with the virus.

People were concerned about their security thinking their location was exposed, however this was not the case because contact tracing applications function by communicating with devices in the immediate area of the user. More information on how contract tracing apps function may be found in the next section.

1.2 MOBILE APPLICATION TESTING

Intensive software testing is required at every stage of the software development life cycle in order to properly deploy the contact tracing application and make it accessible to the public. The fact that these systems are being implemented across a national population with tens of millions of daily users is very significant. The creation of an application for such a large user base is fraught with difficulties.

The application's high performance, reliability, and security must all be ensured at all times. Furthermore, the program must be intuitive and simple to use, with a clear user interface.

The sorts of software testing that are performed on mobile applications are the same as those that are performed on other types of software. However, in mobile applications, the application is executed on a variety of different devices (e.x. Samsung, apple, nokia, etc). Aside from that, mobile devices have less RAM and employ different network connections than desktop computers.

The types of mobile testing include but are not limited to:

Usability testing	is the app easy to use and intuitive?
Compatibility testing	does the app work well on all screen sized and OS?
Interface testing	does the flow of the application work as intended.
Services testing	do all the services of the application work online and offline?
Low-level resource	does the app still work on low-level resources devices?
Performance testing	testing the performance of the app on different device scenarios
Installation testing	does the app install properly.
Security testing	is any sensitive data compromised.

Mobile applications also require special kind of test cases. Which should cover these scenarios

1. Battery usage: The application must keep track of its battery use and ensure that it does not consume more than is reasonable.
2. Speed of the application: The application must function well on a wide range of different devices.
3. Data requirements: Is it possible for the user to install and use the program while on a data plan?
4. Memory requirement: The device should be able to be installed and run on the widest possible range of memory devices.
5. The functionality of the application: The application must not crash as a result of a lack of memory, a network failure, or any other circumstance.

Issues with software testing in mobile applications:

First and foremost, there are many various screen sizes available, which may result in some unforeseen issues, such as photos being displayed in low resolution on large screens.

The second issue is that device makers select when to roll out OS updates to their devices, making it difficult to test and maintain both versions of the operating system simultaneously.

The third point is that Android devices have some high-level constraints. For instance, for iOS versions prior to 10.0, the call kit was not available for use. For the sake of simplicity, call kit is a feature that allows a calling app to display a full-screen view to users when they receive a call from a calling app such as WhatsApp, Skype, or other similar services. Those calls, on the other hand, are displayed as a notification banner for iOS versions below 10.0.

Fourth, when developing test cases, it is necessary to take the application's permissions into consideration. In the case where an application requires permission to communicate with the user, to use the camera, and to use GPS, the application must be tested on all possible combinations of these permissions.

Last but not least, some edge circumstances that may result in unexpected crashes must be considered. For example, leaving the program running in the background for an extended period of time may result in some issues depending on how the code is built.

2 Outline of the NHS Application

A home page with a continue button is displayed when the app is initially launched after it has been installed.

The second page asks for the postal code, which is followed by a button to continue, and the third page asks for authorization.

The main action is the I feel unwell button, which provides a unique reference code that can be used to order a corona virus test as soon as possible.

The functional architecture of the application is depicted in greater depth in the diagram below.

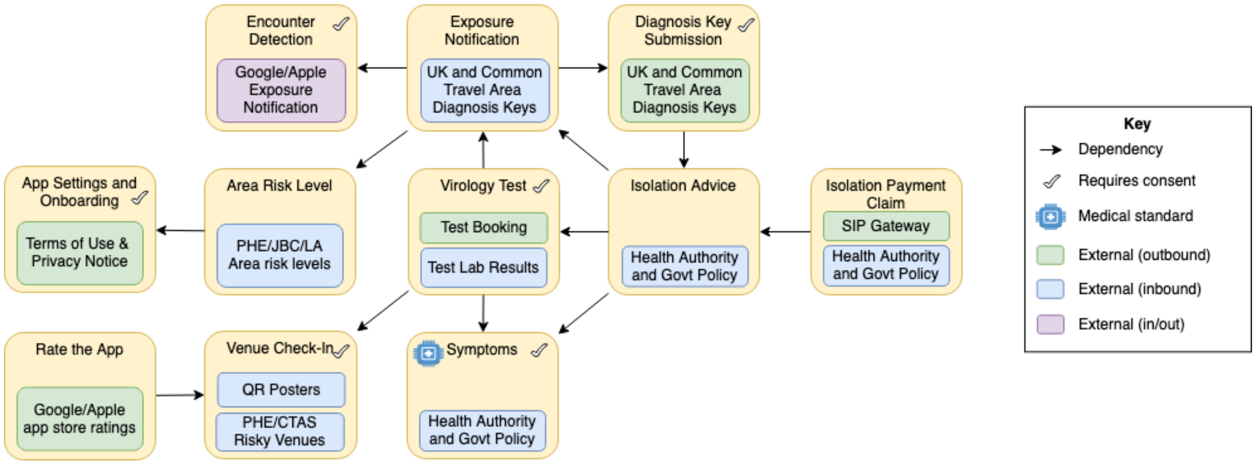


Figure 1-Functional architecture of the application

A comparison of the many implementations of the designs in use today is made in this section, taking into account factors such as information storage, partitioning of processing, capabilities, and potential security problems.

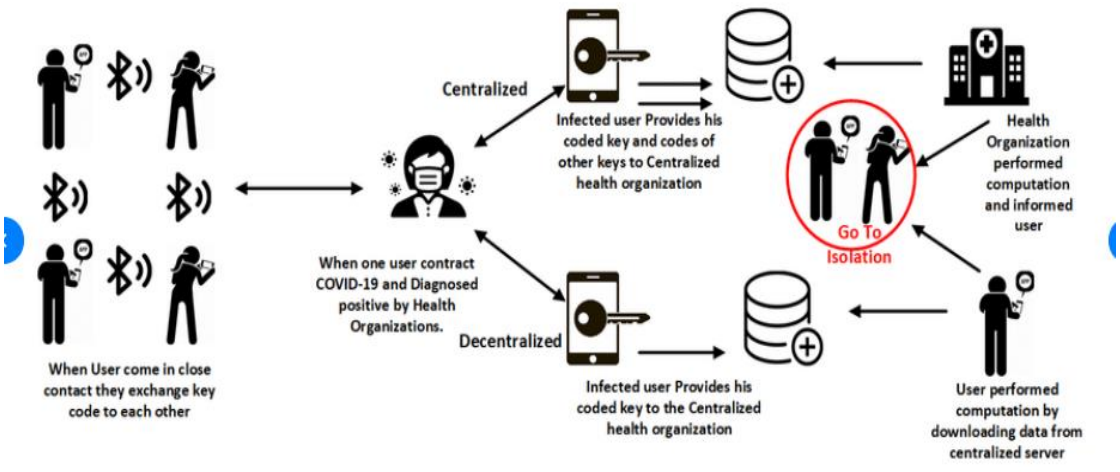
There are two different architectures of contact tracing applications:

1. Centralized: The contact tracking app's fundamental stages are as follows: user registration to assign userIDs, production of TempIDs, Bluetooth exchange of encounter messages, and upload of encounter data for server-side processing.
   The bluetrace protocol is the most widely utilized of the many protocols.
   Bluetrace records bluetooth encounters between the user's devices in order to assist contact tracing while maintaining the user's privacy. It is the users, the app server/tracing app server, and the health authorities who are involved in this process (HA).

2. De-centralized: The centralised architecture is characterized by the fact that the essential tasks are done by centralized servers. The decentralized architecture allows for the majority of calculations to be performed on consumer devices.
   Users' devices are responsible for the production of temporary identifiers for documenting encounters and the risk analysis for sending out notifications, with the server serving just as a publishing channel to facilitate the entire process.

The Private automated contact tracing protocol (PACT) is the protocol that is most commonly used in decentralized architecture.
PACT protocol does not require users to register with the app server in advance, and no personal information data is communicated to or from the server by the users.
Using a random seed generator, the software generates a privacy-preserving alias for the user device called "chirp." These chirps are broadcast in the Bluetooth beacon and exchanged between devices that come into close proximity to one another.Using a random seed generator, the software generates a privacy-preserving alias for the user device called "chirp." These chirps are broadcast in the Bluetooth beacon and exchanged between devices that come into close proximity to one another.
When a user is diagnosed with a positive result, the seed values and time information are displayed. After the user has given approval, the data is uploaded to the app server. The app server is merely a distribution route for the seed values and other important information to be published. If other users have the same seed values and time information as you, you can use this to calculate your potential exposure.

The diagram below provides a high-level overview of the two different architectures used by centralized and decentralized apps, respectively.


Centralized
Infected user Provides his coded key and codes of other keys to Centralized health organization
Health Organization performed computation and informed user
Go To Isolation
When one user contract COVID-19 and Diagnosed positive by Health Organizations.
Decentralized
When User come in close contact they exchange key code to each other
Infected user Provides his coded key to the Centralized health organization
User performed computation by downloading data from centralized server

Architectural Setup of Contact Tracing Apps B. Significance of Contact Tracing for COVID-19

Figure 2-High-level overview of the two different architectures used by centralized and decentralized apps

Possible security concerns

Reply/refection and reply to attacks:

These types of assaults occur when an impersonator attempts to launch a man-in-the-middle attack against a target. There is only one difference: the attacker is not interested in collecting personal information from the victim, but instead attempts to impersonate the sick patient. Attackers are attempting to transmit or broadcast messages at two separate places at the same time, or at two different timestamps at the same site, as a means of achieving their objective.

Device tracking:

App users' overall movement could be tracked using the information provided in their Bluetooth handshake packets.

Denial of service (DoS) attack:

When these assaults are conducted, they aim to shut down or reduce the performance of the target system, as well as making it unreachable to the intended users (or both). The attacker would inject the false message into the entire contact tracing ecosystem, resulting in the use of vital system resources (processing power, battery life, storage space, network bandwidth, and so on) both at the user device level and at the application server.

Enumeration attack:

The attacker's goal in conducting enumeration assaults is to predict the number of infected users or the number of users who have volunteered to submit tracking data to the Health authority for the purpose of risk analysis.

Construction of Social Graph:

Through the use of data mining and correlation of the data available through side channels, this form of attack can be used to reconstruct the interaction history of a user.

Linkage attack:

Links between anonymous data emitted by users and information collected through side channels are attempted in linkage attacks by the attacker. Despite the fact that the centralized design does not give the users with TempIDs for comparison, TempIDs could be associated with the mobile model if the centralized architecture is used.

Overall, being informed of the architecture and prospective attack scenarios assists testers to be more cognizant of potential flaws and security processes since we are more aware of what is going on.

2.1 NHS-COVID 19 APP ARCHITECTURE SPECIFICATION:

The system architecture diagram below depicts the whole system, including the primary system components, their communication ports, and their interconnections.

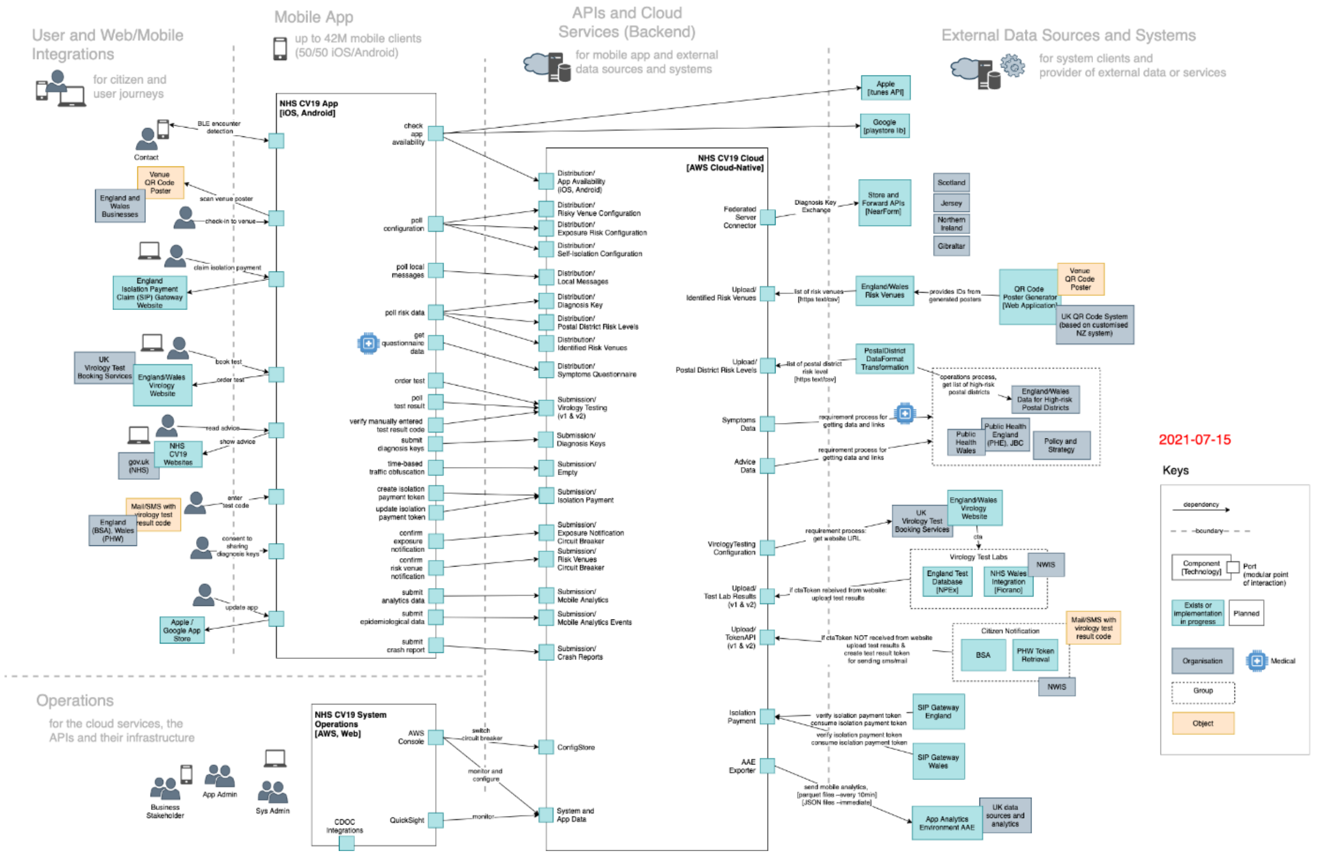
The user-centric concept of the Test and Trace application is implemented in native Android and iOS mobile applications. Encounter detection based on Bluetooth Low Energy (BLE) attenuation time is implemented using the EN Framework offered by Apple and Google.

APIs and Cloud Services (Backend) are built utilizing an AWS cloud-native serverless architecture and made available as APIs to mobile clients. The AWS Lambdas were utilized for the implementation of the services.

External system integration is handled by the backend, which follows an API-driven approach for all offered interfaces. There are connector or exporter solutions for exporting or supplying data, which use AWS Lambdas once again.

Web clients for smaller internal user groups and stakeholders are created as SPAs (single page apps), mostly React, that may be hosted on S3, as part of Operations.

AWS cloud-native components are used for security and operations.



User and Web/Mobile Integrations
for citizen and user journeys
Mobile App
up to 42M mobile clients (50/50 iOS/Android)
APIs and Cloud Services (Backend)
for mobile app and external data sources and systems
External Data Sources and Systems
for system clients and provider of external data or services
NHS CV19 App [iOS, Android]
NHS CV19 Cloud [AWS Cloud-Native]
BLE encounter detection
Contact
Venue QR Code Poster
England and Wales Businesses
scan venue poster
check-in to venue
claim isolation payment
England Isolation Payment Claim (SIP) Gateway Website
UK Virology Test Booking Services
England/Wales Virology Website
book test
order test
read advice
show advice
NHS CV19 Websites
gov.uk (NHS)
Mail/SMS with virology test result code
England (BSA), Wales (PHW)
enter test code
consent to sharing diagnosis keys
update app
Apple / Google App Store
check app availability
poll configuration
poll local messages
poll risk data
get questionnaire data
order test
poll test result
verify manually entered test result code
submit diagnosis keys
time-based traffic obfuscation
create isolation payment token
update isolation payment token
confirm exposure notification
confirm risk venue notification
submit analytics data
submit epidemiological data
submit crash report
Distribution/ App Availability (iOS, Android)
Distribution/ Risky Venue Configuration
Distribution/ Exposure Risk Configuration
Distribution/ Self-Isolation Configuration
Distribution/ Local Messages
Distribution/ Diagnosis Key
Distribution/ Postal District Risk Levels
Distribution/ Identified Risk Venues
Distribution/ Symptoms Questionnaire
Submission/ Virology Testing (v1 & v2)
Submission/ Diagnosis Keys
Submission/ Empty
Submission/ Isolation Payment
Submission/ Exposure Notification Circuit Breaker
Submission/ Risk Venues Circuit Breaker
Submission/ Mobile Analytics
Submission/ Mobile Analytics Events
Submission/ Crash Reports
ConfigStore
System and App Data
Federated Server Connector
Upload/ Identified Risk Venues
Upload/ Postal District Risk Levels
Symptoms Data
Advice Data
VirologyTesting Configuration
Upload/ Test Lab Results (v1 & v2)
Upload/ TokenAPI (v1 & v2)
Isolation Payment
AAE Exporter
Apple [itunes API]
Google [playstore lib]
Diagnosis Key Exchange
Store and Forward APIs [NearForm]
Scotland
Jersey
Northern Ireland
Gibraltar
list of risk venues [https text/csv]
England/Wales Risk Venues
provides IDs from generated posters
QR Code Poster Generator [Web Application]
Venue QR Code Poster
UK QR Code System (based on customised NZ system)
list of postal district risk level [https text/csv]
PostalDistrict DataFormat Transformation
operations process, get list of high-risk postal districts
England/Wales Data for High-risk Postal Districts
requirement process for getting data and links
Public Health Wales
Public Health England (PHE), JBC
Policy and Strategy
requirement process: get website URL
UK Virology Test Booking Services
England/Wales Virology Website
cta
Virology Test Labs
England Test Database [NPEx]
NHS Wales Integration [Fiorano]
NWIS
if ctaToken received from website: upload test results
if ctaToken NOT received from website upload test results & create test result token for sending sms/mail
Citizen Notification
BSA
PHW Token Retrieval
Mail/SMS with virology test result code
NWIS
verify isolation payment token consume isolation payment token
SIP Gateway England
SIP Gateway Wales
send mobile analytics, [parquet files –every 10min] [JSON files –immediate]
App Analytics Environment AAE
UK data sources and analytics
Operations
for the cloud services, the APIs and their infrastructure
Business Stakeholder
App Admin
Sys Admin
NHS CV19 System Operations [AWS, Web]
AWS Console
QuickSight
CDOC Integrations
switch circuit breaker
monitor and configure
monitor
2021-07-15
Keys
dependency
boundary
Component [Technology]
Port (modular point of interaction)
Exists or implementation in progress
Planned
Organisation
Medical
Group
Object

Figure 3-Detailed archeticture of NHS application

3 Applied Testing Approaches

3.1 Static Testing

Static analysis refers to the plethora of techniques used to study source code without running the application. Its primary objective is to guarantee that the code adheres to coding standards and industry standards (ex. MISRA and ISO 26262).

Static testing and analysis are often performed early in the development process; they assist developers in identifying errors early in their code, when they are easier to correct.

Static testing has various disadvantages, the most significant of which is that it lacks a knowledge of the code's functioning. This means that a piece of code may perform exactly as intended, yet static analysis tools may still flag it as an error. Additionally, certain criteria are very subjective, relying on external material, for example (use comments consistently and in a readable fashion) Static analysis methods may generate a large number of false positives and negatives.

However, static analysis is critical and provides several benefits, particularly when the code must adhere to industry standards. The primary benefits of static analysis tools include speed, early detection of errors, and pinpointing the specific location of a mistake in the code. As a result, debugging is a lot faster procedure than dynamic testing. Another advantage of static analysis tools is their depth, as static testing allows for the testing of every potential code execution route. It validates the code in real time while you work on your build. Additionally, you'll receive an in-depth analysis of where potential problems exist in the code depending on the rules followed.

3.1.1 Metrics

Static analysis may be used to examine and analyze code in a variety of ways. For example, it can be used to study classes and objects, which can be quite valuable in OOPs. Static analysis tools are used to quantify many aspects of software quality.

When there are flaws with software, the exterior quality manifests as obvious symptoms, but the core causes are unseen internal quality attributes: program structure, complexity, coupling, testability, reusability, readability, and maintainability. The fundamental internal quality aspects of software are coupling, complexity, cohesion, and size.

A simple summary of some software quality criteria is as follows:

Coupling:

A class A and B are coupled if:

• A contains an attribute that refers to (is of type) B.

• A request for the services of an item B.

• A includes a procedure that refers to B. (via return type or parameter).

• A has a local variable of type B.

• A is a subclass of class B (or implements it).

Tightly coupled systems tend to exhibit the following characteristics:

• Typically, a modification in one class results in a cascade of modifications in other classes.

• Require additional work and/or time as a result of the increasing reliance.

• It may be more difficult to reuse a class if dependent classes must be included.

Lack of Cohesion:

Calculate the degree to which a class's methods are connected to one another. High cohesion (low absence of cohesion) is often preferred since it correlates with a number of desired software characteristics, including resilience, dependability, reusability, and understandability. In comparison, low cohesiveness is connected with unfavorable characteristics such as difficulty in maintaining, testing, reusing, and even comprehending.

Complexity:
It implies that something is difficult to comprehend and describes the relationships between multiple components. Increasing the level of complexity in software raises the likelihood of unintentionally interfering with interactions and, as a result, increases the likelihood of introducing faults while making modifications.

Size

The measuring of software size is one of the most ancient and widely used types of program measurement. The number of lines or methods in the code is used to determine this. A very high count may suggest that a class or function is attempting to accomplish too much work and should be broken up into smaller parts. It could also signal that the class will be difficult to keep up to date.

Class Lines of Code

CLOC is a metric of size that is only indirectly connected to the complexity of a class.

Weighted Method Count

The McCabe complexity of a class is represented by the weighted sum of all of the class' methods and properties. If the complexity of each method is assumed to be one, then it is equal to the number of methods. The number of methods and the complexity of the system can be used to forecast the amount of time spent developing, maintaining, and testing. When a base class has a large number of methods, it has an effect on its child classes, and all of the methods in the child classes are represented in the subclass. If the number of methods in a class is large, it is possible that the class is domain specific. As a result, they have a shorter shelf life. Additionally, these classes are more prone to change and defects.

Depth of Inheritance Tree

The class's position in the inheritance tree can be determined. Has a value of 0 (zero) for the root class and non-inherited classes. The maximum length of the multiple inheritance is shown by the meter in this case. It is likely that a deeper class in the inheritance tree will inherit. As a result, it is more difficult to predict its behavior. Additionally, the development, testing, and maintenance of this class are all relatively complex.

Number of Children

The amount of direct subclasses that exist within a given class. The size of the NOC is a rough indicator of how much an application reuses its own code. It is thought that the more the number of children in a class, the greater the obligation placed on the class's keeper to ensure that the children's behavior is not disrupted. As a result, it is more difficult to modify the class and more testing is required.

CBO Coupling Between Object Classes

This refers to the number of classes with which a class is associated. In order to calculate it, other classes whose attributes or methods are utilized by a class must be added to the list, along with classes that use the attributes or methods of the supplied class. Relationships of inheritance are not permitted. The CBO metric, which serves as a measure of coupling, is associated with the reusability and testability of the class. Increased coupling indicates that the code becomes more difficult to maintain since changes in other classes can result in changes in the class under consideration. As a result, these classes are less reusable and need a greater amount of testing effort.

3.1.2 Static analysis results on the NHS-covid 19 application:

The general information after running the static analysis tool CodeMR are detailed in the table below:

Table 1-general information after running the static analysis tool CodeMR

Total lines of codes	6443		
No. of classes	313	No. of packages	27
No. of external classes	145	No. of external packages	55

After the analysis of the software, as shown in the graph below, we found that only 1 class has high coupling and most of the classes (almost 90%) have low coupling. This makes the software easier to debug and implement changes because changes in high coupled classes force a ripple effect of changes in the coupled classes. And due to the low coupling in the software its easier to reuse classes.

We also found that all methods have high cohesion which indicates that the developers have paid particular interest to encapsulation. Which indicates why only 13 classes have very high complexity.

The developers wrote 313 classes, which indicate that they focused on minimizing the size of each class and we have found that most of the classes have less than 100 LOC and only one class over 200 LOC.

We also observed that most of the classes have low inheritance with more than 70 percent of the classes having low inheritance depth and only 13 classes have high inheritance depth. This makes the code easier to develop, test, and maintain.

We also have analyzed the dependencies of the packages and found the following as shown in the below graph, which indicates relatively low dependencies.

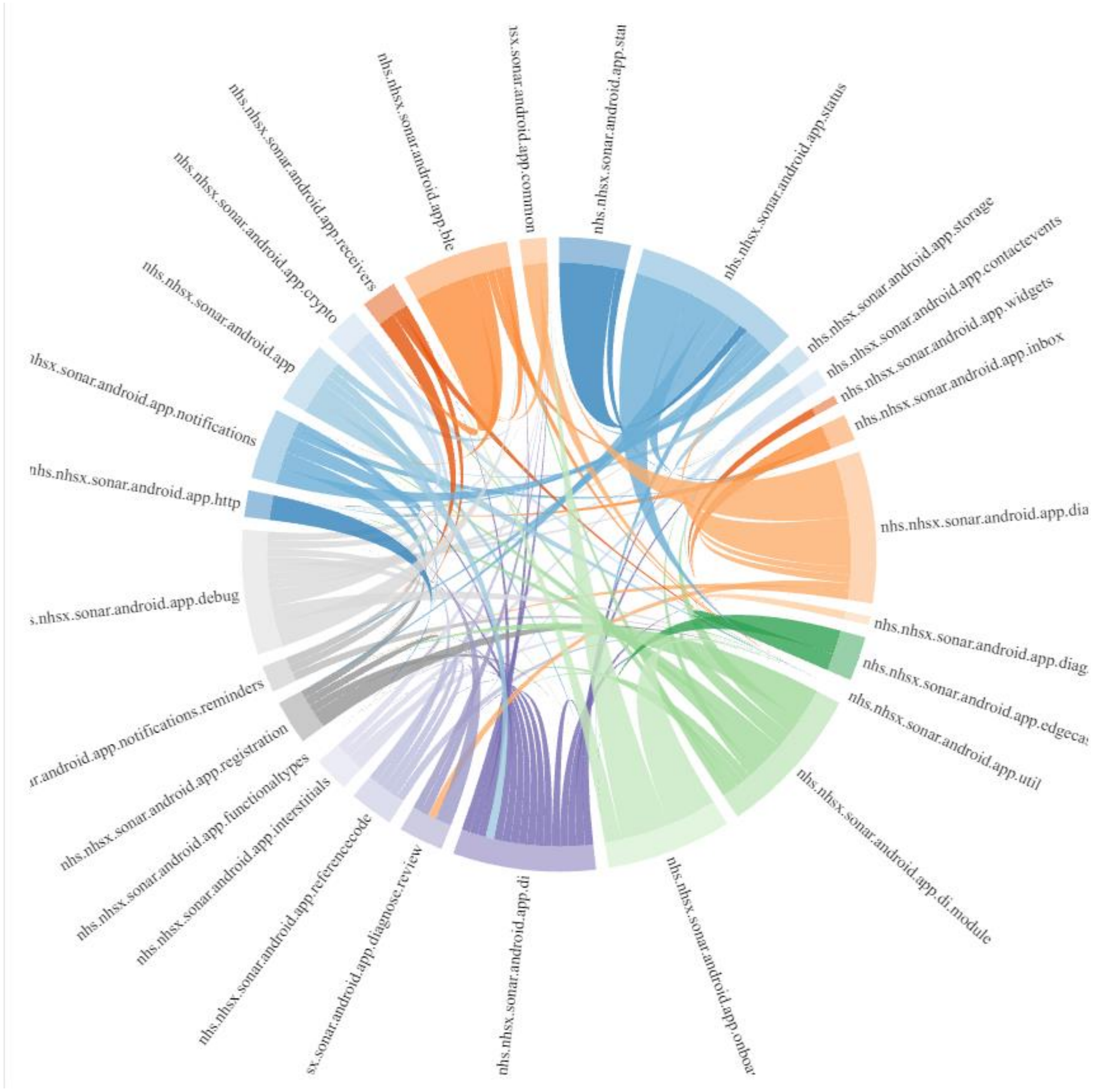


Figure 4-dependencies of the packages

Cyclomatic complexity:

Cyclomatic complexity is used to quantify the total complexity of an application or a subset of its capabilities. The software metric quantifies the logical strength of a program by examining existing decision routes in the source code. It is calculated using the control flow graph, in which each node represents an indivisible group of commands or groups inside the program.

For instance, an application with no decision points (IF, FOR, etc.) has a complexity score of 1 since the source code has just one path. If the program comprises a single IF statement, the code will contain two possible outcomes: TRUE or FALSE.

The cyclomatic complexity algorithm calculates a quantifiable value based on the number of edges and nodes in the graph, as well as the total number of linked components or exit nodes. It is illustrated below:

M = Number of Edges (E) – Number of Nodes (N) + Number of Exit Nodes (P)

Individual programs, subroutines, and methods always have a single exit node, which results in P equal to one. This number will remain constant unless several applications with numerous departure points are being evaluated concurrently.

The effect of cyclomatic complexity on the quality of your software.

This software measurement aides in the reduction of routine complexity during the development process and makes it easier to break modules down into smaller, more manageable components. Programs with a level of less than ten are regarded to be within the permissible range of cyclomatic complexity. This metric may be used to pinpoint areas of opportunity for improvement at the source-code level. Risk is calculated using the specified value:

01 to 10 - Low Risk

Between the ages of 11 and 20, there is a moderate risk.

Between the ages of 21 and 50 - High Risk

Over the Age of 50: Extremely High Risk

Using cyclomatic complexity as the key indicator of an application's or system's complexity enables enterprises to identify high-risk applications and establish improvement strategies for reducing threats, maintenance time, productivity concerns, and technical debt. Understanding the complexity of a system offers insight into areas where a produced program needs extra effort.

After analyzing the code, we identified the highest cyclomatic complexity in each package and they are summarized in the table below.

Table 2-Results of cyclomatic complexity analysis

Package Name	**Maximum Cyclomatic Complexity**	**Package Name**	**Maximum Cyclomatic Complexity**
contactevents	14	diagnose	3
diagnose.review	11	functionaltypes	3
ble	10	inbox	3
debug	9	onboarding	3
crypto	8	widgets	3
status	7	receivers	2
http	6	common	1
notifications	5	di	1
referencecode	4	edgecases	1
registration	4	interstitials	1
util	4	storage	1

Reflecting upon the results it's clear that the developers have took care to reduce the cyclomatic complexity, where only two packages have classes that are above cyclomatic complexity of ten and are considered moderate risk.

3.1.3 Security and performance testing in static analysis

We have only found minor issues with the code during the static analysis, which we expected due to the high performance and security requirements set by the UK government. Some of the issues we found are:

- Some results are set and ignored (i.e. not used)
- Duplicate IDs across layouts
- Missing backup pin
- Using BC provider
- Invalidating All RecyclerView Data
- Static Field Leaks
- Unused resources

Error explanation

Some results are set and ignored (i.e., not used):

Some methods have no side effects, and calling them without doing something without the result is suspicious.

```
                .doOnSubscribe { Timber.d("Negotiating MTU started") }
            .doOnError { e: Throwable? ->
                Timber.e("Failed to negotiate MTU: $e")
Observable.error<Throwable?>(e)
            }
            .doOnSuccess { Timber.d("Negotiated MTU: $it") }
            .ignoreElement()
```

Duplicate IDs across layouts:

It is OK for two completely separate layouts to share the same ids. However, if layouts are coupled with include tags, the id's must be unique inside any chain of included layouts, else ActivityfindViewById() may return an unexpected view as a result of the combination.

```
            app:layout_constraintStart_toStartOf="parent"
        app:layout_constraintTop_toTopOf="parent" />

    <include

        android:id="@+id/nhsLogo"
        layout="@layout/banner_icon_and_title"
        app:layout_constraintStart_toStartOf="parent"
```

../../src/main/res/layout/banner.xml:11: Defined here, included via layout\activity_edge_case.xml => layout\banner.xml defines @+id/logo

```
    android:background="@color/white">

    <LinearLayout
        android:id="@+id/logo"
        android:layout_width="match_parent"
        android:layout_height="match_parent"
        android:gravity="center_vertical">
```

../../src/main/res/layout/banner_icon_and_title.xml:9: Defined here, included via layout\activity_edge_case.xml => layout\banner_icon_and_title.xml defines @+id/logo

```
 6      android:layout_height="?actionBarSize">
 7
 8      <LinearLayout
 9          android:id="@+id/logo"

10          android:layout_width="match_parent"
11          android:layout_height="match_parent"
12          android:gravity="center_vertical">
```

Missing backup pin:

When using certificate pinning, it is critical to include a backup key so that your app's connection is not harmed if you are forced to switch to new keys or change CAs (when pinning to a CA certificate or an intermediate of that CA). Otherwise, you'll need to update the app to re-establish connectivity.

```
     <domain-config>
        <domain includeSubdomains="true">nhs.uk</domain>
        <pin-set>

            <pin digest="SHA-256">hETpgVvaLC0bvcGG3t0cuqiHvr4XyP2MTwCiqhgRWwU=</pin>
        </pin-set>
    </domain-config>
```

Using BC provider

When targetSdkVersion is P or higher, the BC provider is deprecated and will not be given.. [1]

```
    iv: ByteArray,
            data: ByteArray
        ): EncryptionResult {
            val cipher = Cipher.getInstance(AES_GCM_NoPadding,
PROVIDER_NAME)
            val gcmParameterSpec = GCMParameterSpec(GCM_TAG_LENGTH, iv)
            cipher.init(Cipher.ENCRYPT_MODE, encryptionKey, gcmParameterSpec)
            val encrypted = cipher.doFinal(data)
```

Invalidating all recyclerView Data:

The onNotifyDataSetChanged method of the RecyclerView adapter does not define what about the data set has changed, causing any observers to presume that all existing items and structure may no longer be valid. 'LayoutManagers' will be compelled to completely rebind and relay out all visible views.

```
            val event = currentList[position]
        eventHolder.itemView.setOnClickListener {
            event.expanded = !event.expanded
notifyDataSetChanged()
        }
    }
}
```

Static field leak:

Contexts will leak from a static field. Inner classes that are not static have an implicit reference to their outer class. If the outer class is a Fragment or an Activity, for example, this reference indicates that the long-running handler/loader/task will keep a reference to the activity, preventing it from being trash collected. Direct field references to activities and fragments from these longer running instances can also result in leaks. ViewModel classes should never point to non-application Contexts or Views.

```
import javax.inject.Inject

class TestViewModel @Inject constructor(
    private val context:
Context,
    private val contactEventDao: ContactEventDao,
    private val eventTracker: DebugBleEventTracker,
    private val cryptogramStorage: CryptogramStorage
```

Unused resources

Unused resources increase the size of apps and slow down build times. See appendix[3]

3.2 Dynamic testing:

In contrast to static testing, dynamic testing necessitates the execution of code and is an important step in the software development process. The graph below summarizes the various types of dynamic testing:

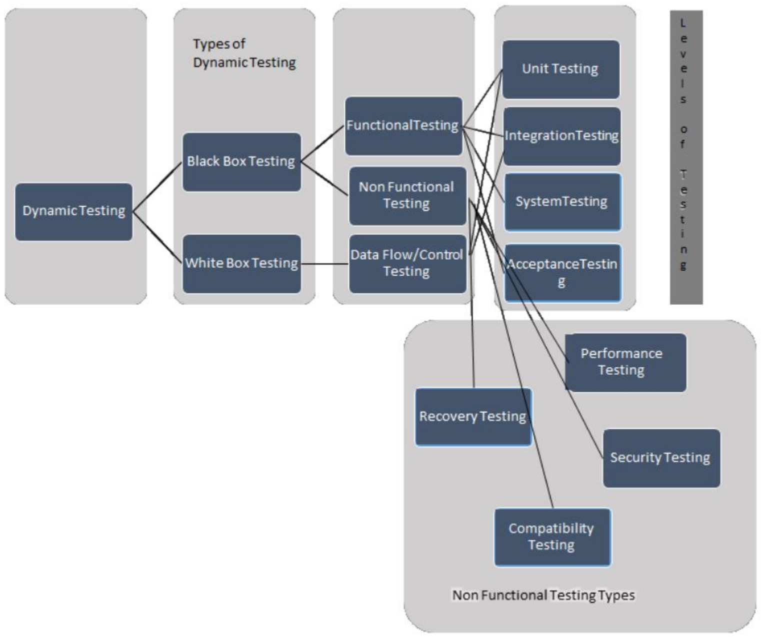


Figure 5-Dynamic testing

3.2.1 System Testing

To assess the functionality of the application an over all system testing was conducted.

Status Change and Notifications Functionalities: some screenshots are shown below

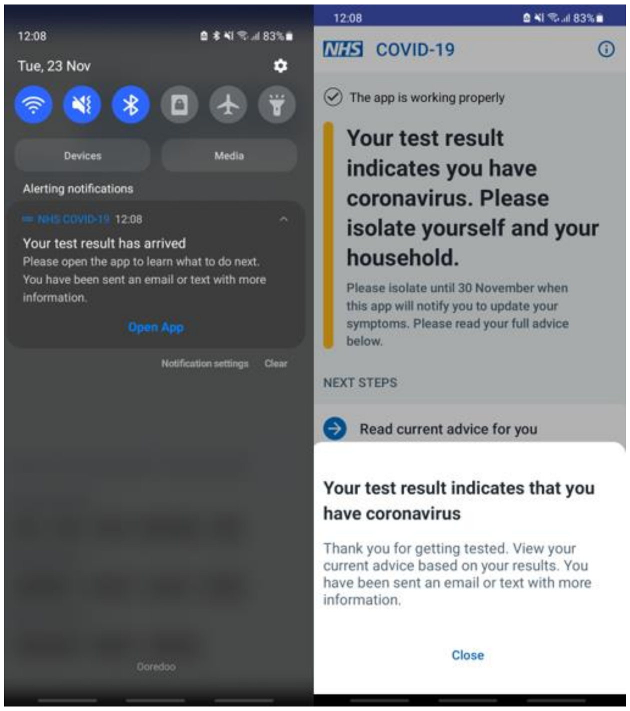
Status change and test notifications were tested on a physical device:

4 types of status were tested :

- Default: no data on infection
- Exposed: exposed as infected
- Symptoms: has symptoms
- Positive: tested positive

Notifications regarding the test results were tested:

- Positive
- Negative
- Unclear



Your test result has arrived
Please open the app to learn what to do next. You have been sent an email or text with more information.

The app is working properly

Your test result indicates you have coronavirus. Please isolate yourself and your household.

Please isolate until 30 November when this app will notify you to update your symptoms. Please read your full advice below.

NEXT STEPS

Read current advice for you

Your test result indicates that you have coronavirus

Thank you for getting tested. View your current advice based on your results. You have been sent an email or text with more information.

Close

Figure 6-Example of notifications tested

3.2.2 Unit Testing

Unit testing is a sort of testing in which individual software modules or functionalities are analyzed. Its major objective is to verify the functionality of each unit or function. A unit is the smallest component of an application that can be tested. It often contains only one or a few inputs and outputs a single value. A unit is referred to as a program in procedural programming, but object-oriented programming languages make use of Base/Superclass, abstract class, and Derived/Child class. Frameworks for unit testing, drivers, stubs, and mocks/fake objects are all used in unit testing. It is based on the White box approach. Through unit testing, businesses may:

- Enhance the quality of the code
- Create Reusable and Dependable Code
- Simplify Documentation

- Integrate Effortlessly

Without mocks, unit testing of functions is impossible. Testing is conducted using dummy objects. Mock objects act as stand-ins for missing components of a program. For instance, there may be a function that requires variables or objects that have not yet been created. Mock objects were built to test the method. In some instances, fake objects are used to replace missing components. Unit testing techniques

White-Box testing

This is referred to as transparent testing or glass box testing. The tester is aware of internal functioning throughout this sort of testing. It is uncertain what the internal structure of an item or function under test is.

Black-Box testing

It is a sort of testing in which the tester is unaware of the system's core operation. The internal structure of the to-be-tested function is unknown. Unit tests provide a fundamental grasp of API units or functions in order to comprehend the code's functionality. It is a technique for creating test cases for all functions and methods so that if a change results in a failure, the defect can be swiftly detected and corrected. A modular approach to testing is used, in which a specific functionality or section of code is tested without waiting for the remainder of the code to be completed. In conclusion, this testing is simply necessary to verify the independence of each function. This kind of testing is time and cost effective. If the same issue occurs during Acceptance Testing, the price will be higher than the cost of Unit Testing. Occasionally, when performing testing, it is necessary to build fake objects in order to satisfy or fulfill the dependence of a method or function. If one function is dependent on another and the other function has not yet been generated, the fake function object is utilized.

Code coverage:

The percentage of code that is covered by automated testing is called code coverage. Simply said, code coverage determines which statements in a body of code have been executed during a test run and which have not. In general, a code coverage system collects information about the running program and then correlates it with the source code to create a report on the code coverage of the test suite.

Code coverage is a component of the development process's feedback loop. As tests are written, code coverage identifies areas of the code that may be insufficiently tested and require extra testing. This loop will run until the coverage reaches a predefined level.

Unit testing is generally acknowledged to increase the quality and predictability of software deliveries. Are you aware, however, of the extent to which your unit tests truly test your code? How many tests are sufficient? Do you require more tests? These are the questions that code coverage analysis aims to address.

Additionally, coverage measurement aids in avoiding test entropy. As your code through numerous release cycles, unit tests may atrophy. As new code is introduced, it may fall short of the testing criteria established when the project was initially launched. Measuring code coverage enables you to ensure that your testing adheres to the required criteria. You may be certain that there will be few issues in production since you know the code not only passes its tests but is also properly tested.

In summary, code coverage is calculated for the following reasons:

1. To determine the effectiveness of our tests in testing our code.

2. To determine whether we have sufficient testing in place.

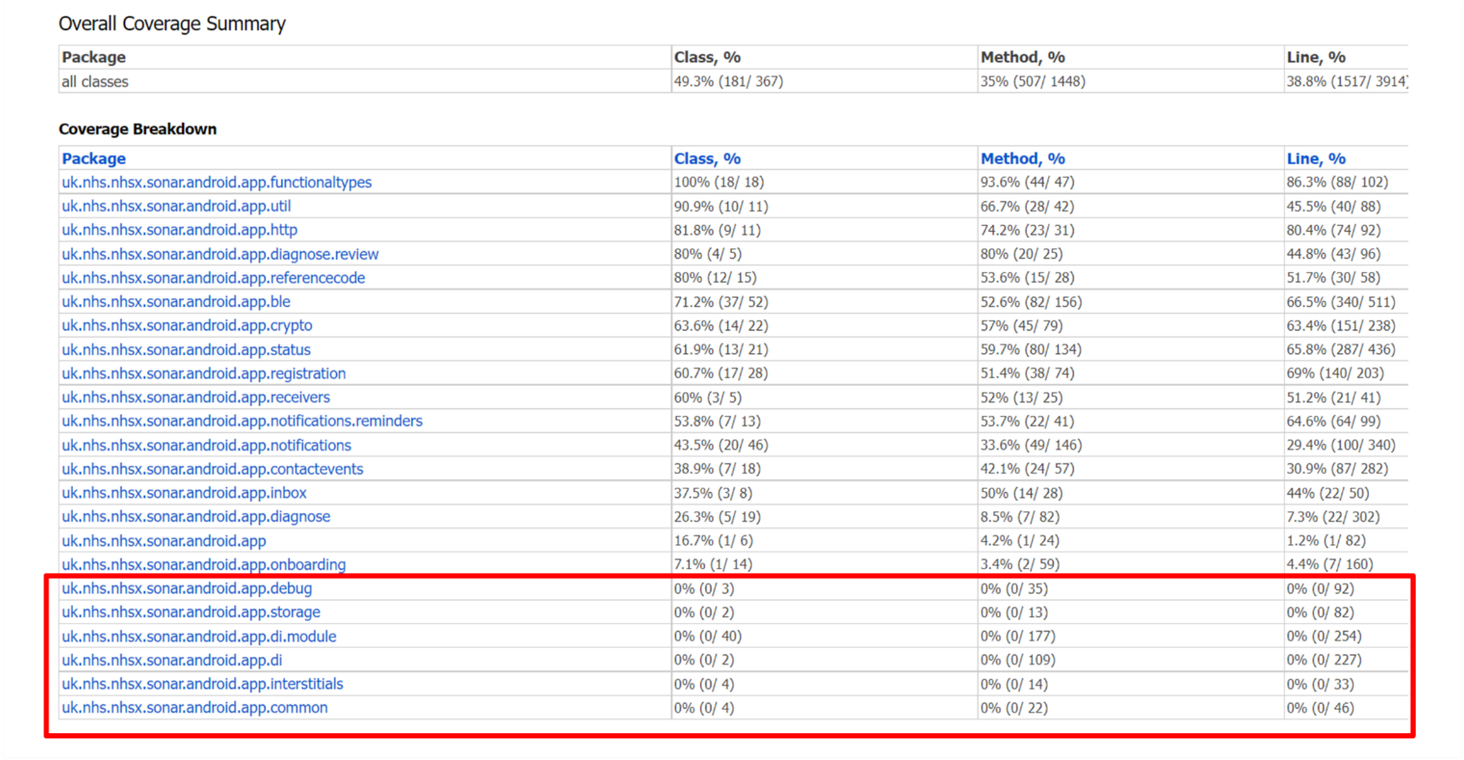
3. To ensure the test quality is maintained throughout the project's lifespan

Code coverage is not a silver bullet. Coverage is often based on an 80-20 rule. Increasing coverage values gets more challenging as new tests provide less incremental benefit. If you adhere to defensive programming approaches, in which failure circumstances are frequently tested at several levels across your project, certain code can be quite tough to reach with realistic testing levels. Coverage analysis is not a substitute for thorough code review and sound development methods.

In general, you should set a reasonable coverage objective and strive for even coverage across all of your code's modules. Reliance on a single overall coverage statistic might obscure significant coverage gaps.

Code coverage results:

The developers provided 284-unit tests for the application, and after running the developers' unit tests and calculating the code coverage using the tool provided by android studio, we discovered that the code coverage is insufficient and that certain defects may have been overlooked by the unit tests provided. In the below table



Overall Coverage Summary

Package	Class, %	Method, %	Line, %
all classes	49.3% (181/ 367)	35% (507/ 1448)	38.8% (1517/ 3914)

Coverage Breakdown

Package	Class, %	Method, %	Line, %
uk.nhs.nhsx.sonar.android.app.functionaltypes	100% (18/ 18)	93.6% (44/ 47)	86.3% (88/ 102)
uk.nhs.nhsx.sonar.android.app.util	90.9% (10/ 11)	66.7% (28/ 42)	45.5% (40/ 88)
uk.nhs.nhsx.sonar.android.app.http	81.8% (9/ 11)	74.2% (23/ 31)	80.4% (74/ 92)
uk.nhs.nhsx.sonar.android.app.diagnose.review	80% (4/ 5)	80% (20/ 25)	44.8% (43/ 96)
uk.nhs.nhsx.sonar.android.app.referencecode	80% (12/ 15)	53.6% (15/ 28)	51.7% (30/ 58)
uk.nhs.nhsx.sonar.android.app.ble	71.2% (37/ 52)	52.6% (82/ 156)	66.5% (340/ 511)
uk.nhs.nhsx.sonar.android.app.crypto	63.6% (14/ 22)	57% (45/ 79)	63.4% (151/ 238)
uk.nhs.nhsx.sonar.android.app.status	61.9% (13/ 21)	59.7% (80/ 134)	65.8% (287/ 436)
uk.nhs.nhsx.sonar.android.app.registration	60.7% (17/ 28)	51.4% (38/ 74)	69% (140/ 203)
uk.nhs.nhsx.sonar.android.app.receivers	60% (3/ 5)	52% (13/ 25)	51.2% (21/ 41)
uk.nhs.nhsx.sonar.android.app.notifications.reminders	53.8% (7/ 13)	53.7% (22/ 41)	64.6% (64/ 99)
uk.nhs.nhsx.sonar.android.app.notifications	43.5% (20/ 46)	33.6% (49/ 146)	29.4% (100/ 340)
uk.nhs.nhsx.sonar.android.app.contactevents	38.9% (7/ 18)	42.1% (24/ 57)	30.9% (87/ 282)
uk.nhs.nhsx.sonar.android.app.inbox	37.5% (3/ 8)	50% (14/ 28)	44% (22/ 50)
uk.nhs.nhsx.sonar.android.app.diagnose	26.3% (5/ 19)	8.5% (7/ 82)	7.3% (22/ 302)
uk.nhs.nhsx.sonar.android.app	16.7% (1/ 6)	4.2% (1/ 24)	1.2% (1/ 82)
uk.nhs.nhsx.sonar.android.app.onboarding	7.1% (1/ 14)	3.4% (2/ 59)	4.4% (7/ 160)
uk.nhs.nhsx.sonar.android.app.debug	0% (0/ 3)	0% (0/ 35)	0% (0/ 92)
uk.nhs.nhsx.sonar.android.app.storage	0% (0/ 2)	0% (0/ 13)	0% (0/ 82)
uk.nhs.nhsx.sonar.android.app.di.module	0% (0/ 40)	0% (0/ 177)	0% (0/ 254)
uk.nhs.nhsx.sonar.android.app.di	0% (0/ 2)	0% (0/ 109)	0% (0/ 227)
uk.nhs.nhsx.sonar.android.app.interstitials	0% (0/ 4)	0% (0/ 14)	0% (0/ 33)
uk.nhs.nhsx.sonar.android.app.common	0% (0/ 4)	0% (0/ 22)	0% (0/ 46)

Figure 7-Coverage of unit tests provided by the developers

From the results above, it is observable that the testing was prioritized based on the importance of the packages. In the functionalTypes package, 100% of the classes was covered which means extensive testing was done due to the importance of that package.

However, on packages that are less important like the onboarding package only 30 percent of the classes was covered.

And some interesting cases like the storage package, there was no coverage of the classes at all, meaning no test cases was developed for it, which leads us to believe that they were tested on hand or some other methods, or simply the unit testes were not open source.

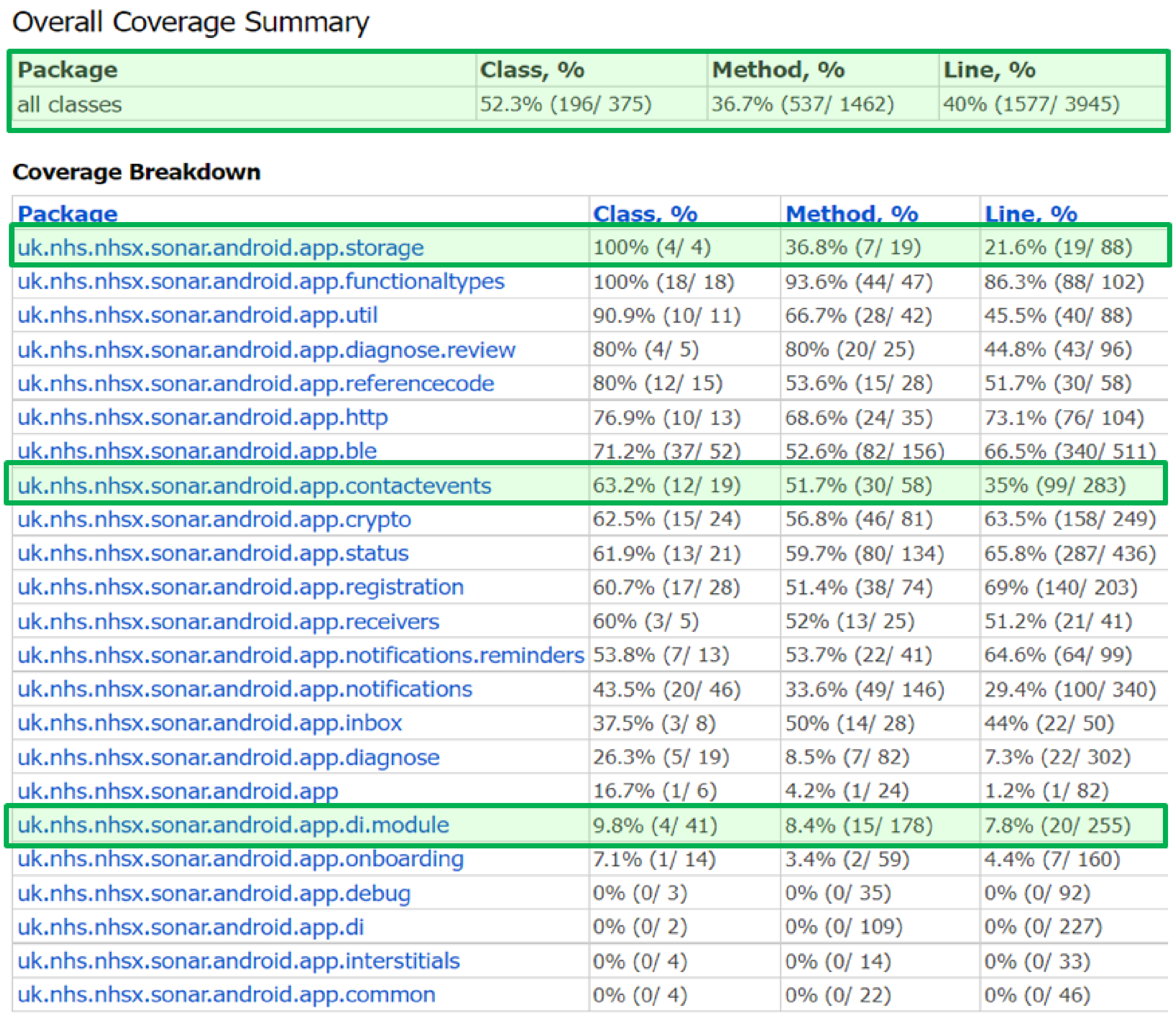
Another observation is that it is easier to cover classes, methods, line of codes and branches respectively, this is because more and more test cases need to be developed to cover all of them.

In conclusion, its clear that the testing team had to ensure the functionality of the program while also deploy the application as fast as possible due to the pandemic, to solve this, they prioritized the testing to the core and most important packages of the software.

Additional unit tests

To improve on the coverage, we wrote 10 additional unit tests, mentioned in appendix 1. And then the coverage tests were run again, to compare the coverage between the test suite before and after.

The overall coverage after the unit tests improved to 52.3%. in the new unit tests we addressed the issue mentioned earlier of some packages having zero coverage, like the storage package. After developing the new unit tests, 100% class coverage was achieved with almost 40% method coverage.


Overall Coverage Summary

Package	Class, %	Method, %	Line, %
all classes	52.3% (196/ 375)	36.7% (537/ 1462)	40% (1577/ 3945)

Coverage Breakdown

Package	Class, %	Method, %	Line, %
uk.nhs.nhsx.sonar.android.app.storage	100% (4/ 4)	36.8% (7/ 19)	21.6% (19/ 88)
uk.nhs.nhsx.sonar.android.app.functionaltypes	100% (18/ 18)	93.6% (44/ 47)	86.3% (88/ 102)
uk.nhs.nhsx.sonar.android.app.util	90.9% (10/ 11)	66.7% (28/ 42)	45.5% (40/ 88)
uk.nhs.nhsx.sonar.android.app.diagnose.review	80% (4/ 5)	80% (20/ 25)	44.8% (43/ 96)
uk.nhs.nhsx.sonar.android.app.referencecode	80% (12/ 15)	53.6% (15/ 28)	51.7% (30/ 58)
uk.nhs.nhsx.sonar.android.app.http	76.9% (10/ 13)	68.6% (24/ 35)	73.1% (76/ 104)
uk.nhs.nhsx.sonar.android.app.ble	71.2% (37/ 52)	52.6% (82/ 156)	66.5% (340/ 511)
uk.nhs.nhsx.sonar.android.app.contactevents	63.2% (12/ 19)	51.7% (30/ 58)	35% (99/ 283)
uk.nhs.nhsx.sonar.android.app.crypto	62.5% (15/ 24)	56.8% (46/ 81)	63.5% (158/ 249)
uk.nhs.nhsx.sonar.android.app.status	61.9% (13/ 21)	59.7% (80/ 134)	65.8% (287/ 436)
uk.nhs.nhsx.sonar.android.app.registration	60.7% (17/ 28)	51.4% (38/ 74)	69% (140/ 203)
uk.nhs.nhsx.sonar.android.app.receivers	60% (3/ 5)	52% (13/ 25)	51.2% (21/ 41)
uk.nhs.nhsx.sonar.android.app.notifications.reminders	53.8% (7/ 13)	53.7% (22/ 41)	64.6% (64/ 99)
uk.nhs.nhsx.sonar.android.app.notifications	43.5% (20/ 46)	33.6% (49/ 146)	29.4% (100/ 340)
uk.nhs.nhsx.sonar.android.app.inbox	37.5% (3/ 8)	50% (14/ 28)	44% (22/ 50)
uk.nhs.nhsx.sonar.android.app.diagnose	26.3% (5/ 19)	8.5% (7/ 82)	7.3% (22/ 302)
uk.nhs.nhsx.sonar.android.app	16.7% (1/ 6)	4.2% (1/ 24)	1.2% (1/ 82)
uk.nhs.nhsx.sonar.android.app.di.module	9.8% (4/ 41)	8.4% (15/ 178)	7.8% (20/ 255)
uk.nhs.nhsx.sonar.android.app.onboarding	7.1% (1/ 14)	3.4% (2/ 59)	4.4% (7/ 160)
uk.nhs.nhsx.sonar.android.app.debug	0% (0/ 3)	0% (0/ 35)	0% (0/ 92)
uk.nhs.nhsx.sonar.android.app.di	0% (0/ 2)	0% (0/ 109)	0% (0/ 227)
uk.nhs.nhsx.sonar.android.app.interstitials	0% (0/ 4)	0% (0/ 14)	0% (0/ 33)
uk.nhs.nhsx.sonar.android.app.common	0% (0/ 4)	0% (0/ 22)	0% (0/ 46)

Figure 8-Coverage results after adding unit tests

3.2.3 Mutation Testing

Mutation testing, sometimes called code mutation testing, is a type of white box testing in which testers modify certain components of an application's source code to confirm that a software test suite can identify the modifications. Changes made to the software are meant to induce mistakes. Mutation testing is used to guarantee the quality of a software testing suite, not the applications that will be tested by the suite.

Typically, mutation testing is used to do unit tests. The objective is for the software test to be capable of detecting all instances of altered code. Changes (referred to as mutations) are implemented by altering an existing line of code to a different value. For instance, a statement may be erased or duplicated, true or false expressions may be updated, and other variables may be modified.

Typically, mutations introduced into a program's code are modest and comprise only one variable that produces a defect or error. Multiple variants of the original program are then created, each with its unique mutation. These variants are referred to as mutants. Following that, the mutants are examined in conjunction with the original application. After doing the tests, testers should compare the results to the original program's test.

If the mutant tests uncover the same number of faults as the original tests, testers will know that either the code failed to run, or the software testing suite employed failed to identify the changes. After that, the software testing suite should be improved to make it more effective. Once the testing program is improved, the mutants can be retained and employed in another code mutation test. If the test results from the mutants and the original programs disagree, and the software test detects the mutants' flaws, the mutants can be destroyed or killed.

Following that, the software test suite may be assessed using the mutation score. The mutation score is calculated by multiplying the proportion of died mutants by the total number of mutants.

Benefits and drawbacks

Among the advantages of code modification are the following:

1. The capability of identifying deficient tests or code.

2. Is capable of detecting errors at a high rate.

3. With the rising usage of object-oriented frameworks and unit tests, additional mutation testing tools have become available.

Mutation scores can help businesses determine the utility of their testing suite.

Among the disadvantages of code modification are the following:

1. The enormous number of mutants employed to test a test suite may result in a potentially confusing experience while testing each version, reducing the feasibility of mutation testing in the absence of automation.

2. Mutation testing may be time consuming and costly due to the enormous number of mutants examined.

Results of mutation testing:

In this project, we targeted 6 packages in our mutation testing, which are encrypter, contactEvent, Scanner, Ble, BluetoothService, and DeviceDetection. The following mutations were implemented as:

1. Return Boolean always false
2. Return Boolean always true
3. Return an empty object
4. Arithmetic operation change
5. If part of the condition removed
6. Equals to not equals
7. Remove not operator
8. OR to AND
9. AND to OR
10. >= to ==
11. >= to >
12. >= to <
13. >= to <=
14. >= to ==

The mutation tests are listed in appendix 2. After running the tests we have identified 4 mutants that were not killed by the test suite. Table 3 shows a summary of the results of the mutation testing. The results indicate that 4 mutants out of 17 were not killed which indicates that the test suite had some issues in detecting errors related to changing the Boolean return value and arithmetic operations and comparisons.

Table 3-Summary of results of mutation testing

Mutation type	Package/class	Mutation	Killed?
Always return True/False	crypto/Encrypter	Always false	Yes
	crypto/Encrypter	Always true	Yes
	contactevents/ContactEvent	Always false	Yes
	contactevents/ContactEvent	Always true	No
Return an empty object	crypto/Encrypter	Empty object	Yes
Arithmetic operation change	contactevents/ContactEvent	+ to -	No

If statement removed	ble/Scanner	IF removed	Yes
Equality comparison	ble/Scanner	== to !=	Yes
	ble/Scanner	== to !=	Yes
	ble/Scanner	== to !=	Yes
	ble/BluetoothService	Remove ! operator	No
Logical operators	util/DeviceDetection	OR to AND	Yes
	util/DeviceDetection	AND to OR	Yes
Arithmetic comparison	util/DeviceDetection	>= to >	Yes
	util/DeviceDetection	>= to <	Yes
	util/DeviceDetection	>= to <=	Yes
	util/DeviceDetection	>= to ==	No

4 Lessons Learned

To further improve the testing process of the application, it is suggested to:

- Develop more unit tests covering more packages and in terms of lines methods and branches.
- Develop more unit tests covering different cases and boundary conditions.
- Perform full security testing.
- Perform automated mutation testing with more mutation types and test cases.

5 Conclusion

In the past year many countries opted for the contact trace application to deal with outbreak of the corona virus. However, due to the high risk of exposure to the public's information, extensive testing is required.

The NHS covid 19 application located in the UK was selected to be analyzed and tested, static testing techniques and dynamic testing techniques were conducted on the android version of the software.

Static analysis techniques were conducted to assess the quality of the code. And it was observed that the complexity of the code is low, as reflected from the cyclomatic complexity results, where only two packages contained methods with a cyclomatic complexity of more than 10, and most of the rest were below 5.

Other static analysis metrics were also conducted like coupling, cohesion, size and depth of inheritance and it was observed that the code is structured well, with low complexity and is easy to maintain, test and develop.

The unit tests provided by the developer were assessed in terms of coverage metrics. The overall coverage of the tests was around 48 percent of the overall code. However, test prioritizing was done by the developer due to time and cost constraints, because some packages had high coverage while others had near zero coverage.

To improve the coverage of the test suite, additional unit tests were developed and executed successfully. The overall coverage was improved by around 3 percent. However, it's notable to mention that 100 percent coverage were achieved in some packages that had near zero coverage before, and improved the coverage of the tests in other packages.

Mutation tests were developed to also assess the quality of the test suite, four mutants out of seventeen were not killed which indicates that the test suite failed to detect them, it's worth mentioning that all the live mutants were changing the Boolean return value and arithmetic operations and comparisons.

6 Appendices

6.1 Appendix 1: Code of Additional Unit Tests

```
package uk.nhs.nhsx.sonar.android.app.additionalTests
import android.bluetooth.BluetoothAdapter
import android.content.Context
import android.content.pm.PackageManager
import android.content.pm.PackageManager.FEATURE_BLUETOOTH_LE
import io.mockk.every
import io.mockk.mockk
import io.mockk.mockkStatic
import org.assertj.core.api.Assertions.*
import org.hamcrest.core.*
import org.hamcrest.core.IsInstanceOf.instanceOf
import org.junit.Before
import org.junit.Test
import uk.nhs.nhsx.sonar.android.app.di.module.AppModule
import uk.nhs.nhsx.sonar.android.app.di.module.AppModule_ProvideContextFactory
import uk.nhs.nhsx.sonar.android.app.di.module.CryptoModule
import uk.nhs.nhsx.sonar.android.app.di.module.PersistenceModule
import uk.nhs.nhsx.sonar.android.app.storage.AppDatabase
import uk.nhs.nhsx.sonar.android.app.util.LocationHelper
import uk.nhs.nhsx.sonar.android.app.util.NotificationManagerHelper
import java.security.KeyStore
import kotlin.test.assertEquals
import kotlin.test.assertNotNull

class ModuleTests {

    private val adapter = mockk<BluetoothAdapter>()
    private val context = mockk<Context>()
    private val manager = mockk<PackageManager>()
    private val locationHelper = mockk<LocationHelper>()
    private val notifHelper = mockk<NotificationManagerHelper>()
    private val keystore = mockk<KeyStore>()
    private val database = mockk<AppDatabase>()


    @Before
```

```
fun setUp() {
    every { context.packageManager } returns manager
    mockkStatic("uk.nhs.nhsx.sonar.android.app.util.AccessibilityHelperKt")
}
@Test
fun `app module provide context()`() {
    val module = AppModule(context,locationHelper,notifHelper)
    assertEquals(module.provideContext(), context)
}
@Test
fun `app module provide location helper()`() {
    val module = AppModule(context,locationHelper,notifHelper)
    assertEquals(module.provideLocationHelper(),locationHelper)
}
@Test
fun `app module provide notification manager helper()`() {
    val module = AppModule(context,locationHelper,notifHelper)
    assertEquals(module.provideNotificaitonManagerHelper(),notifHelper)
}
@Test
fun `app module device model()`() {
    val module = AppModule(context,locationHelper,notifHelper)
    assertNotNull(module.deviceModel())
}
@Test
fun `app module device os version()`() {
    val module = AppModule(context,locationHelper,notifHelper)
    assertNotNull(module.deviceOsVersion())
}
@Test
fun `crypto module cryptogram storage()`() {
    val cryptoModule = CryptoModule(context,keystore)
    assertNotNull(cryptoModule.provideCryptogramStorage(context))
}
@Test
fun `crypto module constructor and providers()`() {
    val cryptoModule = CryptoModule(context,keystore)
    assertNotNull(cryptoModule.providePublicKeyStorage())
    }
    @Test
    fun `persistence module provide database()`() {
        val persistenceModule = PersistenceModule(context)
        assertNotNull(persistenceModule.provideDatabase())
    }
    @Test
    fun `persistence module provide contact Dao()`() {
        val persistenceModule = PersistenceModule(context)
        val database2 = persistenceModule.provideDatabase()
        assertNotNull(persistenceModule.provideContactEventDao(database2))
    }
    @Test
    fun `persistence module provide colocation data provider()`() {
        val persistenceModule = PersistenceModule(context)
        val database2 = persistenceModule.provideDatabase()
        val dao = persistenceModule.provideContactEventDao(database2)
        assertNotNull(persistenceModule.provideCoLocationDataProvider(dao))
    }
}
```

6.2 Appendix 2: Mutation Test Cases

Package/class: Crypto/Encrypter

1.1- Return Boolean always false

```
fun                    canEncrypt():                    Boolean                    =
    keyStorage.providePublicKey() != null
```

mutant: killed

```
fun canEncrypt(): Boolean =
    false
```

1.2- Return Boolean always true

mutant: killed

```
fun canEncrypt(): Boolean =
    true
```

2.1- Return an empty object

```
private fun generateSharedSecret(
    privateKey: PrivateKey,
    publicKey: PublicKey
): ByteArray {
    val keyAgreement = KeyAgreement.getInstance(ECDH,
PROVIDER_NAME)
    keyAgreement.init(privateKey)
    keyAgreement.doPhase(publicKey, true)
    val secret = keyAgreement.generateSecret()
    if (secret.size != 32)
        throw IllegalArgumentException("Shared secret should be
256 bits (32 bytes)")
    return secret
}
```

mutant: killed

```
private fun generateSharedSecret(
    privateKey: PrivateKey,
    publicKey: PublicKey
): ByteArray {
    val keyAgreement = KeyAgreement.getInstance(ECDH,
PROVIDER_NAME)
    keyAgreement.init(privateKey)
    keyAgreement.doPhase(publicKey, true)
    val secret = keyAgreement.generateSecret()
    if (secret.size != 32)
        throw IllegalArgumentException("Shared secret should be
256 bits (32 bytes)")
    return ByteArray(5)
}
```

Package/class: contactevents/ContactEvent

3.1- Return Boolean always false

```
override fun equals(other: Any?): Boolean {
    if (this === other) return true
    if (javaClass != other?.javaClass) return false

    other as ContactEvent

    if (id != other.id) return false
    if (!sonarId.contentEquals(other.sonarId)) return false
    if (rssiValues != other.rssiValues) return false
    if (rssiTimestamps != other.rssiTimestamps) return false
    if (txPowerInProtocol != other.txPowerInProtocol) return false
    if (txPowerAdvertised != other.txPowerAdvertised) return false
    if (timestamp != other.timestamp) return false
    if (transmissionTime != other.transmissionTime) return false
    if (!hmacSignature.contentEquals(other.hmacSignature)) return
false
    if (!countryCode.contentEquals(other.countryCode)) return
false
    if (duration != other.duration) return false

    return true
}
```

mutant: killed

```
override fun equals(other: Any?): Boolean {
    return false
}
```

3.2- Return Boolean always true

Mutant: Not killed

```
override fun equals(other: Any?): Boolean {
    return true
}
```

4.1- Arithmatic operation change

```
override fun hashCode(): Int {
    var result = id.hashCode()
    result = 31 * result + sonarId.contentHashCode()
    result = 31 * result + rssiValues.hashCode()
    result = 31 * result + rssiTimestamps.hashCode()
    result = 31 * result + txPowerInProtocol
    result = 31 * result + txPowerAdvertised
    result = 31 * result + timestamp.hashCode()
    result = 31 * result + transmissionTime
    result = 31 * result + hmacSignature.contentHashCode()
    result = 31 * result + countryCode.contentHashCode()
    result = 31 * result + duration
    return result
}
```

mutant: Not killed

```
override fun hashCode(): Int {
    var result = id.hashCode()
    result = 31 * result - sonarId.contentHashCode()
    result = 31 * result - rssiValues.hashCode()
    result = 31 * result - rssiTimestamps.hashCode()
    result = 31 * result - txPowerInProtocol
    result = 31 * result - txPowerAdvertised
    result = 31 * result - timestamp.hashCode()
    result = 31 * result - transmissionTime
    result = 31 * result - hmacSignature.contentHashCode()
    result = 31 * result - countryCode.contentHashCode()
    result = 31 * result - duration
    return result
}
```

Package/class: ble/Scanner

5.1- if part of the condition removed

```
val operation = if (identifier != null) {
    readOnlyRssi(identifier)
} else {
    readIdAndRssi()
}
```

mutant: killed

```
val operation = readIdAndRssi()
```

Package/class: ble/Ble

6.1- equals to not equals

```
fun BluetoothGattCharacteristic.isDeviceIdentifier(): Boolean =
    this.uuid == SONAR_IDENTITY_CHARACTERISTIC_UUID
```

mutant: killed

```
fun BluetoothGattCharacteristic.isDeviceIdentifier(): Boolean =
    this.uuid != SONAR_IDENTITY_CHARACTERISTIC_UUID
```

7.1- equals to not equals

```
fun BluetoothGattCharacteristic.isKeepAlive(): Boolean =
    this.uuid == SONAR_KEEPALIVE_CHARACTERISTIC_UUID
```

mutant: killed

```
fun BluetoothGattCharacteristic.isKeepAlive(): Boolean =
    this.uuid != SONAR_KEEPALIVE_CHARACTERISTIC_UUID
```

8.1- equals to not equals

```
fun BluetoothGattDescriptor.isNotifyDescriptor(): Boolean =
    this.uuid == NOTIFY_DESCRIPTOR_UUID
```

mutant: killed

```
fun BluetoothGattDescriptor.isNotifyDescriptor(): Boolean =
    this.uuid != NOTIFY_DESCRIPTOR_UUID
```

Package/class: ble/ BluetoothService

9.1- remove not operator

BluetoothService

```
private fun injectIfNecessary() {
    if (!isInjected) {
        appComponent.inject(this)
        isInjected = true
    }
}
```

mutant: Not killed

```
private fun injectIfNecessary() {
    if (isInjected) {
        appComponent.inject(this)
        isInjected = true
    }
}
```

Package/class: util/DeviceDetection

10.1- OR to AND

```
fun isUnsupported(): Boolean =
    bluetoothAdapter == null ||
!context.packageManager.hasSystemFeature(FEATURE_BLUETOOTH_LE) ||
        (bluetoothAdapter.isEnabled &&
bluetoothAdapter.bluetoothLeAdvertiser == null)
```

mutant: killed

```
fun isUnsupported(): Boolean =
    bluetoothAdapter == null ||
!context.packageManager.hasSystemFeature(FEATURE_BLUETOOTH_LE) &&
        (bluetoothAdapter.isEnabled &&
bluetoothAdapter.bluetoothLeAdvertiser == null)
```

10.2- AND to OR

Mutant: killed

```
fun isUnsupported(): Boolean =
    bluetoothAdapter == null ||
!context.packageManager.hasSystemFeature(FEATURE_BLUETOOTH_LE) ||
        (bluetoothAdapter.isEnabled ||
bluetoothAdapter.bluetoothLeAdvertiser == null)
```

11.1- >= to >

```
fun isTablet(): Boolean =
    simulateTablet || context.smallestScreenWidth() >=
TABLET_MINIMUM_SCREEN_WIDTH
```

mutant: killed

```
fun isTablet(): Boolean =
    simulateTablet || context.smallestScreenWidth() >
TABLET_MINIMUM_SCREEN_WIDTH
```

11.2- >= to <

mutant: killed

```
fun isTablet(): Boolean =
    simulateTablet || context.smallestScreenWidth() <
TABLET_MINIMUM_SCREEN_WIDTH
```

11.3- >= to <=

mutant: killed

```
fun isTablet(): Boolean =
    simulateTablet || context.smallestScreenWidth() <=
TABLET_MINIMUM_SCREEN_WIDTH
```

11.4- >= to ==

mutant: Not killed

```
fun isTablet(): Boolean =
    simulateTablet || context.smallestScreenWidth() ==
TABLET_MINIMUM_SCREEN_WIDTH
```

6.3 Appendix 3

```
17
18      <color name="color_nhsuk_grey_5">#F0F4F5</color>
19      <color name="color_nhsuk_grey_4">#AEB7BD</color>
20      <color name="color_nhsuk_grey_3">#E5E5E5</color>

21
22      <color name="card_background_grey">#B3D8DDE0</color>
23
```

../../src/main/res/values/colors.xml:22: The resource `R.color.card_background_grey` appears to be unused

```
19      <color name="color_nhsuk_grey_4">#AEB7BD</color>
20      <color name="color_nhsuk_grey_3">#E5E5E5</color>
21
22      <color name="card_background_grey">#B3D8DDE0</color>

23
24      <color name="text_sub_title">#4C6272</color>
25      <color name="amber">#FFB81C</color>
```

../../src/main/res/drawable/ic_arrow_back.xml:5: The resource `R.drawable.ic_arrow_back` appears to be unused

```
  ~ Copyright © 2020 NHSX. All rights reserved.
  -->

<vector android:height="24dp" android:tint="#FFFFFF"

        android:viewportHeight="24.0" android:viewportWidth="24.0"
        android:width="24dp"
xmlns:android="http://schemas.android.com/apk/res/android">
    <path android:fillColor="#FF000000"
```

../../src/main/res/drawable/ic_baseline_close_24.xml:5: The resource `R.drawable.ic_baseline_close_24` appears to be unused

```
  ~ Copyright © 2020 NHSX. All rights reserved.
  -->

<vector xmlns:android="http://schemas.android.com/apk/res/android"

    android:width="24dp"
    android:height="24dp"
    android:viewportWidth="24"
```

../../src/main/res/drawable/ic_close.xml:5: The resource `R.drawable.ic_close` appears to be unused

```
  ~ Copyright © 2020 NHSX. All rights reserved.
  -->

<vector android:height="24dp" android:tint="#FFFFFF"

        android:viewportHeight="24.0" android:viewportWidth="24.0"
        android:width="24dp"
xmlns:android="http://schemas.android.com/apk/res/android">
    <path android:fillColor="#FF000000"
```

../../src/main/res/drawable/ic_info.xml:5: The resource `R.drawable.ic_info` appears to be unused

```
  ~ Copyright © 2020 NHSX. All rights reserved.
  -->

<vector xmlns:android="http://schemas.android.com/apk/res/android"

        android:width="24dp"
        android:height="24dp"
        android:viewportWidth="24.0"
```

../../src/main/res/drawable/nhs.png: The resource `R.drawable.nhs` appears to be unused

../../src/main/res/values/strings.xml:107: The resource `R.string.status_latest_advice` appears to be unused

```
  <string name="progress_three_fifth">3/5</string>
  <string name="progress_four_fifth">4/5</string>
  <string name="progress_five_fifth">5/5</string>
  <string name="status_latest_advice"><u>Read current advice</u></string>

  <string name="status_what_to_do"><u>Read what to do next</u></string>
  <string name="status_not_feeling_well">I feel unwell</string>
  <string name="status_not_feeling_well_desc">I might have symptoms of coronavirus
and want to know what to do next.</string>
```

../../src/main/res/values/strings.xml:108: The resource `R.string.status_what_to_do` appears to be unused

```
105    <string name="progress_four_fifth">4/5</string>
106    <string name="progress_five_fifth">5/5</string>
107    <string name="status_latest_advice"><u>Read current advice</u></string>
108    <string name="status_what_to_do"><u>Read what to do next</u></string>

109    <string name="status_not_feeling_well">I feel unwell</string>
110    <string name="status_not_feeling_well_desc">I might have symptoms of coronavirus and want to know what to do next.</string>
111    <string name="status_default_title">Follow the current advice to stop the spread of coronavirus</string>
```

../../src/main/res/values/strings.xml:125: The resource `R.string.app_rationale` appears to be unused

```
122    <string name="post_code_title">Enter the first part of your home postcode</string>
123    <string name="post_code_rationale"><![CDATA[This information will be used to help your local NHS respond to the coronavirus in your area and to understand the spread of the disease across the UK. Your postcode will not be used to track your location.]]></string>
124    <string name="valid_post_code_is_required">Please enter the first part of a valid postcode. For example: PO30, E2, M1, EH1, L36</string>
125    <string name="app_rationale">This app can let you know if you may have been near someone with coronavirus symptoms (COVID-19).</string>

126    <string name="symptoms_date_prompt_all">When did you first start showing any of these symptoms?</string>
127    <string name="today">Today</string>
128    <string name="yesterday">Yesterday</string>
```

../../src/main/res/values/strings.xml:148: The resource `R.string.update_symptoms_prompt_with_negative_test` appears to be unused

```
145    <string name="i_do_not_stomach">I do not have any of these symptoms: diarrhoea, nausea, vomiting or loss of appetite</string>
146    <string name="date_selection_error">Select the date you first started showing any of these symptoms</string>
147    <string name="update_symptoms_prompt">Please update your symptoms to view your latest advice.</string>
148    <string name="update_symptoms_prompt_with_negative_test">Thank you for getting tested. You have been sent an email or text with more information.\n\nIf you are having symptoms, please update your symptoms to see the latest advice for you. If you do not have any symptoms, you can now follow the advice for the general public.</string>

149    <string name="update_my_symptoms">Update my symptoms</string>
150    <string name="no_symptoms">I no longer have symptoms</string>
151    <string name="advice_dialog_title">Your advice has changed</string>
```

../../src/main/res/values/strings.xml:179: The resource `R.string.status_description_03` appears to be unused

```
176    <string name="follow_until_exposed">Please isolate until %1$s. Please read your full advice below.</string>
177    <string name="follow_until_symptomatic">Please isolate until %1$s when this app will notify you to update your symptoms. Please read your full advice below.</string>
178    <string name="status_description_01">If you do not have any symptoms, there is no need to do anything right now. If you develop symptoms, please come back to this app.</string>
179    <string name="status_description_03">If you develop symptoms, please come back to this app.</string>
```

```
180    <string name="re_enable_bluetooth_title">Please turn your Bluetooth back
on</string>
181    <string name="re_enable_bluetooth_rationale_p1">This app works only if your
Bluetooth is on at all times.</string>
182    <string name="re_enable_bluetooth_rationale_p2">%1$s works using Bluetooth
signals, to determine when you have been near someone else with this app who has tested
positive for coronavirus. %1$s can alert you or others that may be at risk.</string>
```

../../src/main/res/values/strings.xml:191: The resource `R.string.testing_information` appears to be unused

```
188    <string name="re_allow_location_permission_rationale_p1">To make Bluetooth work,
we also need you to turn on location services and give the app permission to use it. Your
location will not be tracked. Your location and personal identity will remain
protected.</string>
189    <string name="re_allow_location_permission_rationale_p2">To enable your location
permissions you need to go to your Settings and allow this app to access location
services.</string>
190    <string name="re_allow_location_permission_rationale_p3">%1$s works using
Bluetooth signals, to determine when you have been near someone else with this app who
has tested positive for coronavirus.</string>
191    <string name="testing_information">Testing information</string>

192    <string name="reference_code">Reference code</string>
193    <string name="loading">Loading…</string>
194    <string name="error">Error</string>
```

../../src/main/res/values/strings.xml:192: The resource `R.string.reference_code` appears to be unused

```
189    <string name="re_allow_location_permission_rationale_p2">To enable your location
permissions you need to go to your Settings and allow this app to access location
services.</string>
190    <string name="re_allow_location_permission_rationale_p3">%1$s works using
Bluetooth signals, to determine when you have been near someone else with this app who
has tested positive for coronavirus.</string>
191    <string name="testing_information">Testing information</string>
192    <string name="reference_code">Reference code</string>

193    <string name="loading">Loading…</string>
194    <string name="error">Error</string>
195    <string name="post_code_example">Example: CE1B</string>
```

../../src/main/res/values/strings.xml:194: The resource `R.string.error` appears to be unused

```
191      <string name="testing_information">Testing information</string>
192      <string name="reference_code">Reference code</string>
193      <string name="loading">Loading…</string>
194      <string name="error">Error</string>

195      <string name="post_code_example">Example: CE1B</string>
196      <string name="post_code_rationale_title">Why do we need this?</string>
197      <string name="enable_permissions">Enable permissions</string>
```

../../src/main/res/values/strings.xml:208: The resource `R.string.high_temperature` appears to be unused

```
205    <string name="confirm_diagnosis_text_1">The information you have provided will be
used to help protect your community and the NHS. The NHS needs accurate data on the
number of people with symptoms.</string>
206    <string name="confirm_diagnosis_text_2">Together we can help save lives, protect
the NHS, and stop the spread of coronavirus in the UK.</string>
207    <string name="guidance_for_workplace">Guidance for your workplace</string>
```

```
208    <string name="high_temperature">High temperature</string>
209    <string name="continuous_cough">Continuous cough</string>
210    <string name="info">Link to more information about NHS COVID-19 app</string>
211    <string name="please_book_test">Please book a coronavirus test immediately. Write
down your reference code and phone</string>
```

../../src/main/res/values/strings.xml:209: The resource `R.string.continuous_cough` appears to be unused

```
206    <string name="confirm_diagnosis_text_2">Together we can help save lives, protect
the NHS, and stop the spread of coronavirus in the UK.</string>
207    <string name="guidance_for_workplace">Guidance for your workplace</string>
208    <string name="high_temperature">High temperature</string>
209    <string name="continuous_cough">Continuous cough</string>
210    <string name="info">Link to more information about NHS COVID-19 app</string>
211    <string name="please_book_test">Please book a coronavirus test immediately. Write
down your reference code and phone</string>
212    <string name="test_booking_number"><u>0800 540 4900</u></string>
```

../../src/main/res/values/strings.xml:211: The resource `R.string.please_book_test` appears to be unused

```
208    <string name="high_temperature">High temperature</string>
209    <string name="continuous_cough">Continuous cough</string>
210    <string name="info">Link to more information about NHS COVID-19 app</string>
211    <string name="please_book_test">Please book a coronavirus test immediately. Write
down your reference code and phone</string>
212    <string name="test_booking_number"><u>0800 540 4900</u></string>
213    <string name="page_1_of_6">Page 1 of 6</string>
214    <string name="page_2_of_6">Page 2 of 6</string>
```

../../src/main/res/values/strings.xml:212: The resource `R.string.test_booking_number` appears to be unused

```
209    <string name="continuous_cough">Continuous cough</string>
210    <string name="info">Link to more information about NHS COVID-19 app</string>
211    <string name="please_book_test">Please book a coronavirus test immediately. Write
down your reference code and phone</string>
212    <string name="test_booking_number"><u>0800 540 4900</u></string>
213    <string name="page_1_of_6">Page 1 of 6</string>
214    <string name="page_2_of_6">Page 2 of 6</string>
215    <string name="page_3_of_6">Page 3 of 6</string>
```

../../src/main/res/values/strings.xml:237: The resource `R.string.get_tested` appears to be unused

```
234    <string name="apply_for_test_now">Book a coronavirus test now</string>
235    <string name="apply_for_test_desc">You can now book a coronavirus test for you and
other people in your household. This test will tell you whether you currently have
coronavirus or not.</string>
236    <string name="apply_for_test_need_ref">In order for this app to receive your test
result, you will be asked for this 8-digit app reference code later.</string>
237    <string name="get_tested">Get tested</string>
238    <string name="reference_code_title">Information about testing</string>
239    <string name="reference_code_when_1">When you need a test</string>
240    <string name="reference_code_description_1">If you feel unwell you should let this
app know and it can help you book a coronavirus test.</string>
```

../../src/main/res/values/styles.xml:62: The resource `R.style.TextViewSubStatus` appears to be unused

```xml
        <item name="android:textColor">@color/text_primary_color</item>
    </style>

    <style name="TextViewSubStatus" parent="Widget.AppCompat.TextView">

        <item name="android:lineSpacingMultiplier">1.18</item>
        <item name="android:textSize">18sp</item>
        <item name="android:textStyle">bold</item>
```