White Box Testing

Group 6

Made by – Archit Jugran , 160101087

Shubham Goel , 160101083

Yagyansh Bhatia , 160101079

Index

Page #

1. Introduction

White-box testing (also known as clear box testing, glass box testing, transparent box testing, and structural testing) is a method of testing software that tests internal structures or workings of an application, as opposed to its functionality (i.e. black-box testing). In white-box testing an internal perspective of the system, as well as programming skills, are used to design test cases. The tester chooses inputs to exercise paths through the code and determine the expected outputs. This is analogous to testing nodes in a circuit, e.g. in-circuit testing (ICT). White-box testing can be applied at the unit, integration and system levels of the software testing process. Although traditional testers tended to think of white-box testing as being done at the unit level, it is used for integration and system testing more frequently today. It can test paths within a unit, paths between units during integration, and between subsystems during a system–level test. Though this method of test design can uncover many errors or problems, it has the potential to miss unimplemented parts of the specification or missing requirements.

White-box test design techniques include the following code coverage criteria:

- Control flow testing
- Data flow testing
- Branch testing
- Statement coverage
- Decision coverage
- Modified condition/decision coverage
- Prime path testing
- Path testing

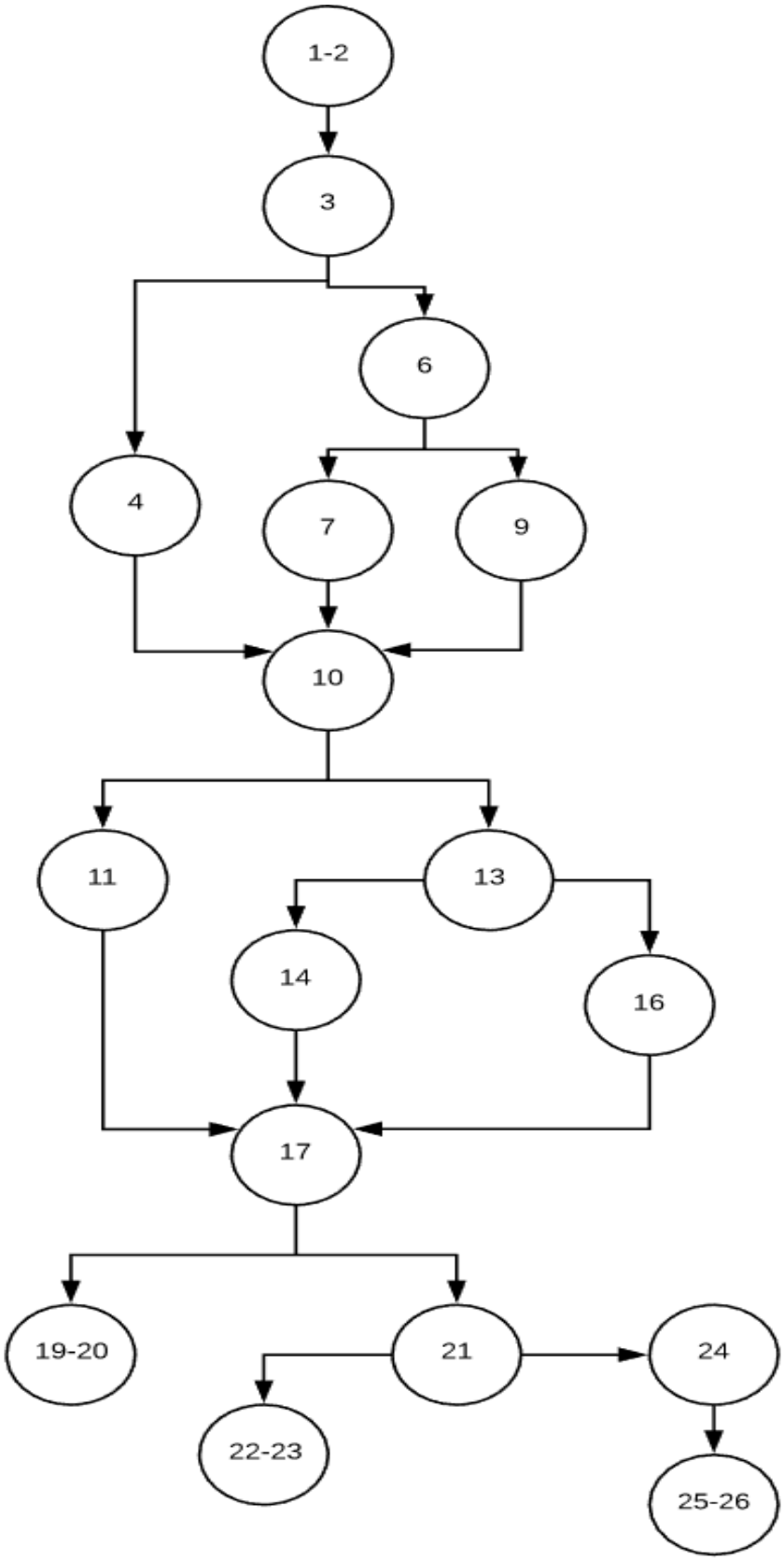
NOTE – In our document, functions – onResume(), onPause() and AuthStateListener() do not need white box testing because they have already been tested by Google and are being used as a black box in our code.

2. Module Message Generator

2.1 Code

```
public void message_generator (int newstate,int prevstate) {
    int newrange,prevrange;
    if(newstate<5){
        newrange=0;
    }
    else if(newstate<8) {
        newrange=1;
    }
    else newrange=2;
    if(prevstate<5){
        prevrange=0;
    }
    else if(prevstate<8) {
        prevrange=1;
    }
    else prevrange=2;
    if (newrange > prevrange)
    {
        strtitle = "Increase in attention";
        strtext = "Keep it up " + newrange + " " + prevrange;
    } else if (newrange < prevrange) {
        strtitle = "Decrease in attention";
        strtext = "Warning: Please pay more attention " + newrange + " " + prevrange;
    } else {
        strtitle="";
        strtext="";
    }
}
```

2.2 Control Flow Graph



2.3 Linearly Independent Paths and Test Cases

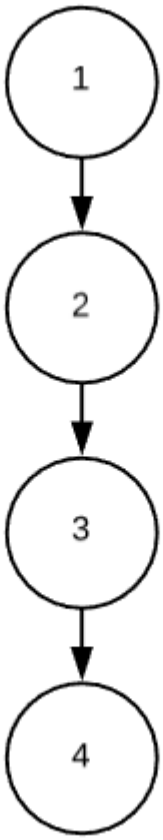
1. 1-2 -> 3 -> 6 -> 7 -> 10 ->11 -> 17 -> 19-20
    a. Test case : prevstate=1 , newstate = 5
    b. Expected Output : strtitle = “Increase in attention” , strtext= “Keep it up 1 0” , prevrange =0 ,newrange = 1
    c. Observed Output : strtitle = “Increase in attention” , strtext= “Keep it up 1 0” , prevrange =0 ,newrange = 1
2. 1-2 -> 3 -> 4 -> 10 ->13 -> 14 -> 17 -> 21 -> 22-23
    a. Test case : prevstate=5 , newstate = 1
    b. Expected Output : strtitle = “Decrease in attention” , strtext= “Warning : Please pay more attention 0 1” , prevrange =1 ,newrange = 0
    c. Observed Output : strtitle = “Decrease in attention” , strtext= “Warning : Please pay more attention 0 1” , prevrange =1 ,newrange = 0
3. 1-2 -> 3 -> 6 -> 9 -> 10 ->13 -> 16 -> 17 -> 21 -> 24 -> 25-26
    a. Test case : prevstate=9 , newstate = 10
    b. Expected Output : strtitle = “” , strtext= “” , prevrange =2 ,newrange = 2
    c. Observed Output : strtitle = “” , strtext= “” , prevrange =2 ,newrange = 2

3. Module Random number generator

3.1 Code

```
1  public int random_number() {
2      Random rand = new Random();
3      int newstate = rand.nextInt(10);
4      return newstate;
5  }
```

3.2 Control Flow Graph



3.3 Linearly Independent Paths and Test Cases

Linearly Independent Paths

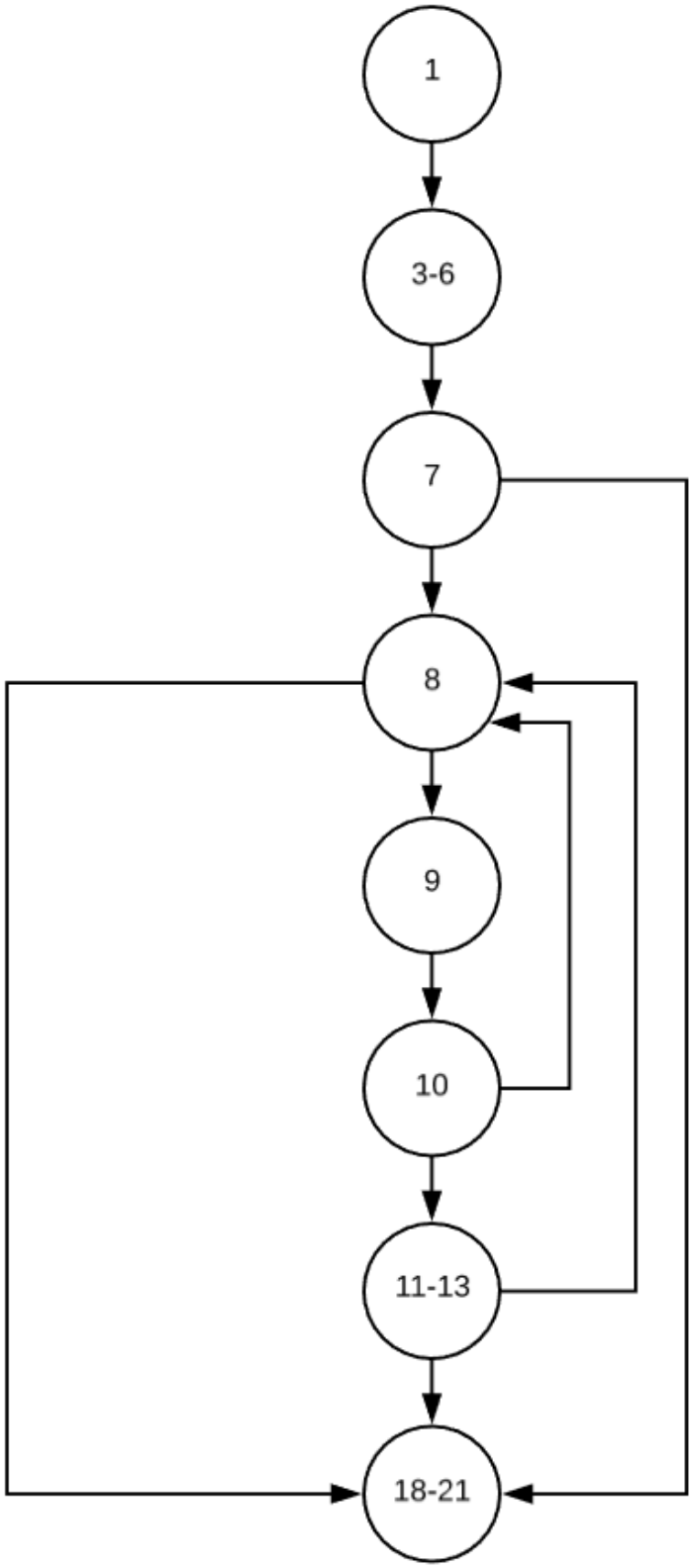
1. **1 -> 2 -> 3 -> 4**
    a. **Expected output : Random number between 1 to 10**
    b. **Observed output : Random number between 1 to 10**

4. Module Confirm Notification

4.1 Code

```
public void confirmnotification(final String notification3)
{
    MyRef.push().setValue("junk");
    ValueEventListener evventListener = new ValueEventListener() {
        @Override
        public void onDataChange(DataSnapshot datasnapshot) {
            if(Flag==0){
                for (DataSnapshot ds : datasnapshot.getChildren()) {
                    String notificationn = ds.getValue(String.class);
                    if (notification3.equalsIgnoreCase(notificationn)) {
                        DatabaseReference ref = ds.getRef();
                        ref.setValue("Confirmed : " + notificationn);
                        Flag = 1;
                    }
                }
            }
        }
        @Override
        public void onCancelled(DatabaseError databaseerror) {}
    };
    MyRef.addValueEventListener(evventListener);
}
```

4.2 Control Flow Graph



4.3 Linearly Independent Paths and Test Cases

1. **1 -> 3-6 -> 7 -> 8 -> 9 ->10 -> 11-13 -> 18-21**
   Test case: notification3 : 0 Yagyansh Bhatia Warning: Please pay more attention 0 1

   Snapshot ds : 0 Yagyansh Bhatia Warning: Please pay more attention 0 1

2. **1 -> 3-6 -> 7 -> 8 -> 18-21**
   Test case : notification3 : 0 Yagyansh Bhatia Warning: Please pay more attention 0 1

   Snapshot ds: (empty)

3. **1 -> 3-6 -> 7 -> 8 -> 9 ->10 -> 8 -> 9 -> 10 -> 11-13 -> 8 ->18-21**
   Test case : notification3 : 1 Yagyansh Bhatia Warning: Please pay more attention 0 1

   Snapshot ds : 0 Yagyansh Bhatia Keep it up 1 0

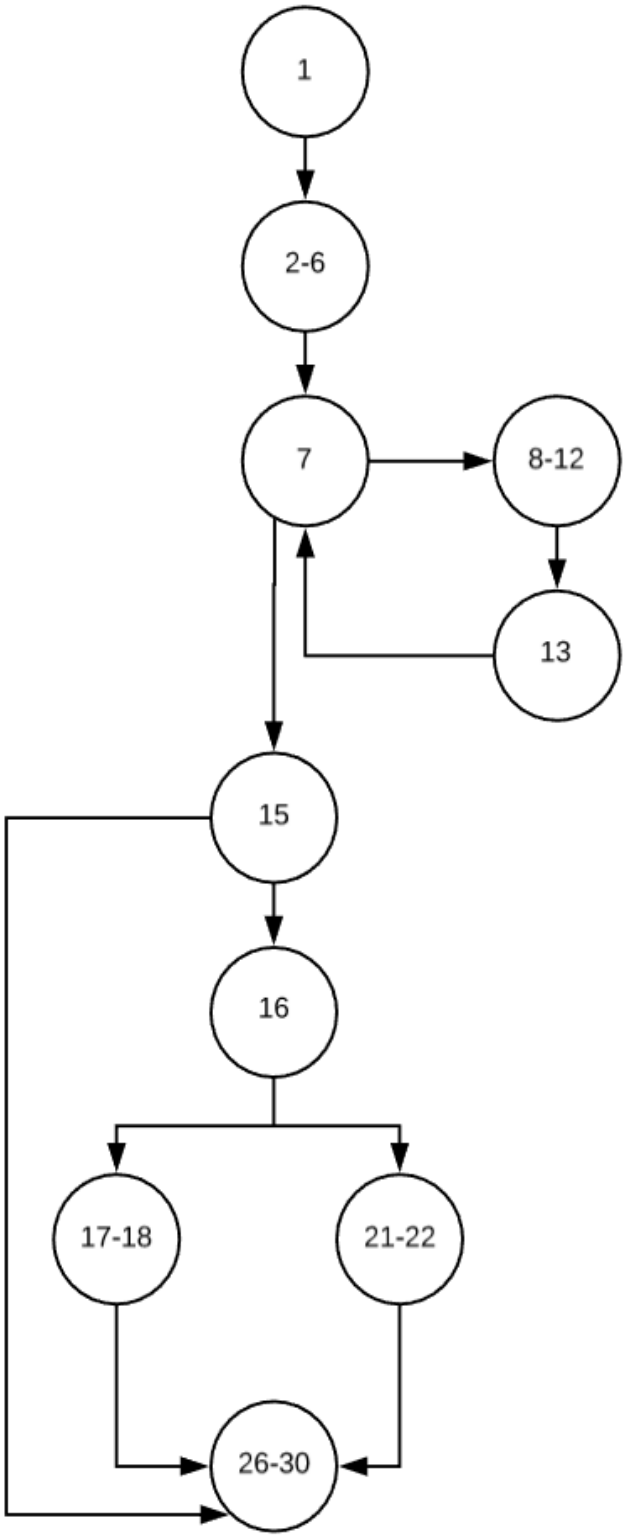
   1 Yagyansh Bhatia Warning: Please pay more attention 0 1

5. Module Alert Decider

5.1 Code

```
public void alert_decider(){
    ValueEventListener eventListener = new ValueEventListener() {
        @Override
        public void onDataChange(DataSnapshot datasnapshot) {
            int cntlow=0,totalcnt=0;
            Users_List.clear();
            for(DataSnapshot ds : datasnapshot.getChildren()) {
                totalcnt++;
                String latestnoti = ds.child("latestnotification").getValue(String.class);
                int len=latestnoti.length();
                int currentstate=(int)(latestnoti.charAt(len-3)-'0');
                if(currentstate==0) cntlow++;
                Users_List.add("Name : " + ds.child("name").getValue(String.class) + " || Current State : " + currentstate);
            }
            if(Position.equalsIgnoreCase("teacher")) {
                if (cntlow * 1.0 / totalcnt >= 0.5) {
                        FirebaseMessaging.getInstance().subscribeToTopic("lowlow"+Room);
                        Database.getReference("cloudtriggerlow").push().setValue(Room);
                }
                else {
                    FirebaseMessaging.getInstance().subscribeToTopic("highhigh"+Room);
                    Database.getReference("cloudtriggerhigh").push().setValue(Room);
                }
            }
        }
        @Override
        public void onCancelled(DatabaseError databaseError) {}
    };
    CountRef.addValueEventListener(eventListener);
}
```

5.2 Control Flow Graph



5.3 Linearly Independent Paths and Test Cases

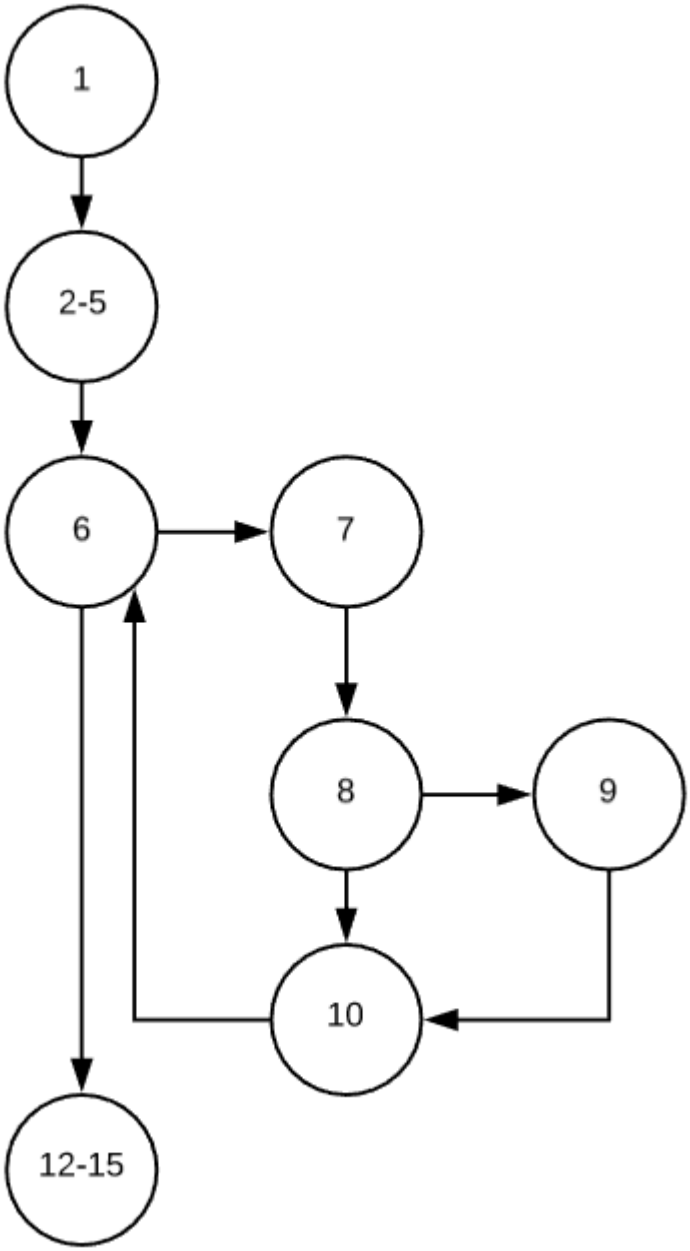
1. **1 -> 2-6 -> 7 -> 8-12 -> 13 -> 7 -> 15 -> 26-30**
   A.Test case: Snapshot ds : 0 Yagyansh Bhatia Warning: Please pay more attention 0 1
   Position : student
   Totalcnt: 1
   cntlow:1
2. **1 -> 2-6 -> 7 -> 8-12 -> 13 -> 7 -> 15 -> 16 -> 17-18 -> 26-30**
   A.Test case: Snapshot ds : 0 Yagyansh Bhatia Warning: Please pay more attention 0 1
   Position : teacher
   Totalcnt: 1
   cntlow:1
3. **1 -> 2-6 -> 7 -> 8-12 -> 13 -> 7 -> 15 -> 16 -> 21-22 -> 26-30**
   A. Test case: Snapshot ds : 0 Yagyansh Bhatia Keep it up 1 0
   Position : teacher
   Totalcnt: 1
   Cntlow:0

6. Module log_store

6.1 Code

```
public void log_store(){
    ValueEventListener evventListener = new ValueEventListener() {
    @Override
    public void onDataChange(DataSnapshot datasnapshot) {
    Log_List.clear();
    for(DataSnapshot ds : datasnapshot.getChildren()) {
        String latestnoti = ds.getValue(String.class);
        if(!latestnoti.equalsIgnoreCase("junk"))
            Log_List.add(latestnoti);
        }
    }
    @Override
    public void onCancelled(DatabaseError databaseerror) {}
    };
    MyRef.addValueEventListener(evventListener);
}
```

6.2 Control Flow Graph



6.3 Linearly Independent Paths and Test Cases

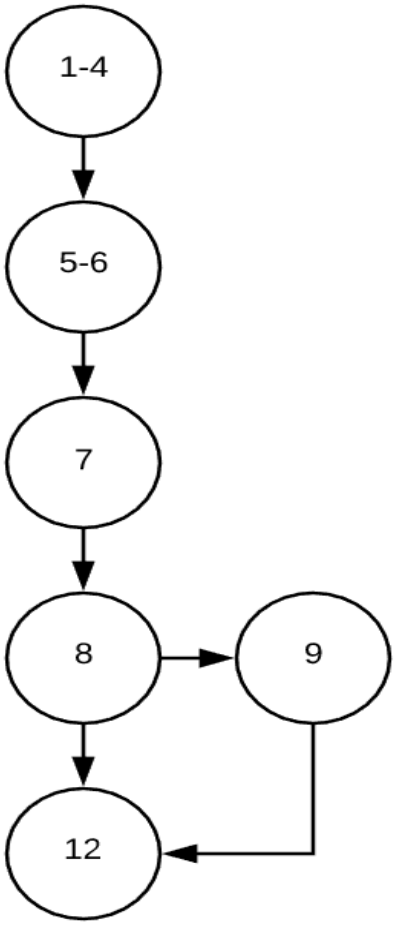
1. **1 -> 2-5 -> 6 -> 7 -> 8 -> 9 -> 10 -> 6 -> 12-15**
   A. Test case: Snapshot ds : 0 Yagyansh Bhatia Warning: Please pay more attention 0 1
   Latestnoti: 0 Yagyansh Bhatia Warning: Please pay more attention 0 1
   Log_list: 0 Yagyansh Bhatia Warning: Please pay more attention 0 1
2. **1 -> 2-5 -> 6 -> 7 -> 8 -> 10 -> 6 -> 12-15**
   A. Test case: Snapshot ds : junk
   Latestnoti:junk
   Log_list: (Empty)

7. Module PeriodicClockSignalGenerator

7.1 Code

```
1  public void periodic_clock_generator(){
2  final Handler handler = new Handler();
3  final Button btn = (Button) findViewById(R.id.button);
4  Runnable run = new Runnable() {
5      @Override
6      public void run() {
7          handler.postDelayed(this,10000);
8          if(! Username.equalsIgnoreCase("NULL"))
9              btn.performClick();
10         }
11     };
12     handler.post(run);
13 }
```

7.2 Control Flow Graph



7.3 Linearly Independent Paths and Test Cases

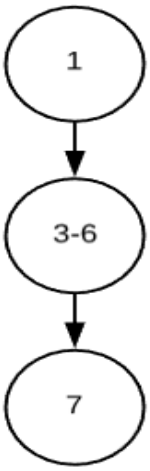
1. **1-4 -> 5-6 -> 7 -> 8 -> 9 -> 12**
   Test Case: Username - Shubham Goel
2. **1-4 -> 5-6 -> 7 -> 8 -> 12**
   Test Case: Username – NULL

8. Module signout

8.1 Code

```
1  protected void signout (View view)
2  {
3      AuthUI.getInstance().signOut(this);
4      Intent i =new Intent(this,MainActivity.class);
5      startActivity(i);
6      Username="NULL";
7  }
```

8.2 Control Flow Graph



8.3 Linearly Independent Paths and Test Cases

1. **1 -> 3-6 -> 7**
   Expected behaviour : User Signed out

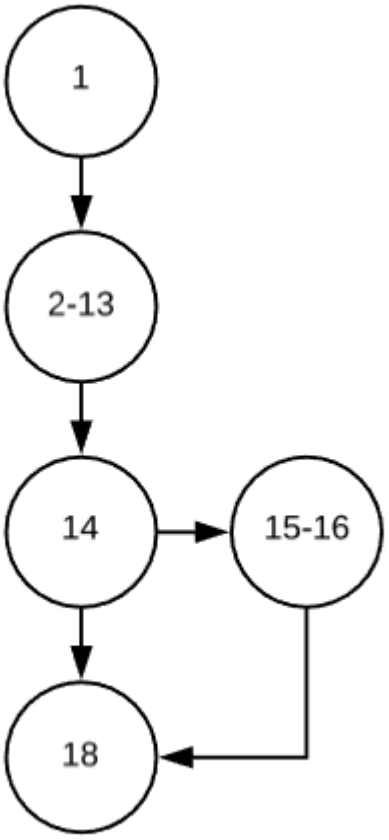
   Observed behaviour : User signed out

9. Module Student Notifier

9.1 Code

```
public void notificationsender() {
    Notification.setSmallIcon(R.drawable.ic_launcher_background);
    Notification.setTicker("ticker");
    Notification.setWhen(System.currentTimeMillis());
    Notification.setContentTitle(Username+" "+strtitle);
    Notification.setContentText(Username+" "+strtext);
    Intent intent=new Intent(this,Confirm.class);
    String confirmednotification=counter + " "+Username + " " + strtext;
    intent.putExtra("notification",confirmednotification.toString());
    intent.putExtra("room",Room.toString());
    PendingIntent pendingIntent=PendingIntent.getActivity(this,0,intent,PendingIntent.FLAG_UPDATE_CURRENT);
    Notification.setContentIntent(pendingIntent);
    NotificationManager nf=(NotificationManager) getSystemService(NOTIFICATION_SERVICE);
    if(Position.equalsIgnoreCase("student")) {
        nf.notify(Id, Notification.build());
        stateupdate();
    }
}
```

9.2 Control Flow Graph



9.3 Linearly Independent Paths and Test Cases

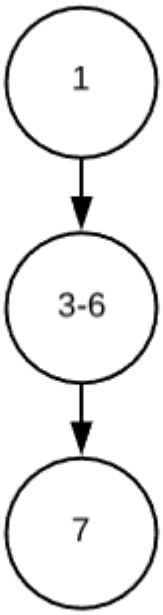
1. 1-> 2-13 -> 14 -> 15-16 -> 18
    a. Test case : Position : student
    b. Expected behaviour : Notification sent to student
    c. Observed behaviour : Notification sent to student
2. 1-> 2-13 -> 14 -> 18
    a. Test case : Position : teacher
    b. Expected behaviour : Notification not sent
    c. Observed behaviour : Notification not sent

10. Module Student state update

10.1 Code

```
1  public void stateupdate()
2  {
3      FirebaseMessaging.getInstance().subscribeToTopic("student");
4      CountRef.child(Username).child("latestnotification").setValue(strtext);
5      CountRef.child(Username).child("name").setValue(Username);
6      MyRef.push().setValue(counter+" "+Username + " " + strtext);//pushing notification to database
7      counter++;
8  }
```

10.2 Control Flow Graph



10.3 Linearly Independent Paths and Test Cases

1. **1 -> 3-6 -> 7**

   Test case: counter = 5 , Username = "Yagyansh" , strtext = "Warning : Please pay more attention 0 1"

   Expected database at countRef : Yagyansh →

   Latestnotification → Warning : Please pay more attention 0 1

   Name → Yagyansh

Expected database at myRef :

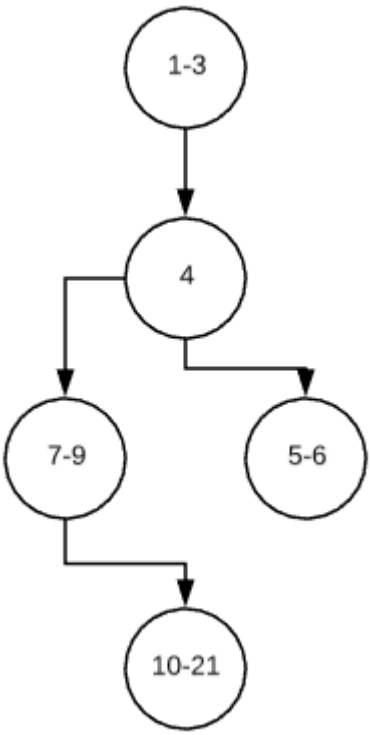
pushid --> 2 Yagyansh Warning : Please pay more attention 0 1

11. Module Vibrate Notifier

11.1 Code

```
functions.database.ref('cloudtriggerlow').onWrite((event) => {
    const data = event.data;
    console.log('Message received');
    if(!data.changed()){
        console.log('Nothing changed');
        return;
    }else{
        console.log(data.val());
    }
    const payLoad = {
        notification:{
            title: 'ATTENTION',
            body: '50% class down',
            sound: "default"
        }
    };
    const options = {
        priority: "high",
        timeToLive: 60*60*2
    };
    return admin.messaging().sendToTopic("lowlow", payLoad, options);
});
```

11.2 Control Flow Graph



11.3 Linearly Independent Paths and Test Cases

1. **1-3 -> 4 -> 5-6**
   Test case: Data not changed
   Expected behaviour: nothing
   Observed Behaviour : nothing
2. **1-3 -> 4 -> 7-9 -> 10-21**
   Test case: junk pushed to database at "cloudtriggerlow"
   Expected behaviour: Notification sent to teacher
   Observed Behaviour : Notification sent to teacher