# INTRODUCTION

A touchscreen is an electronic visual display that the user can control through simple or multi-touch gestures by touching the screen with a special stylus/pen and-or one or more fingers. Some touchscreens use an ordinary or specially coated gloves to work while others use a special stylus/pen only. The user can use the touchscreen to react to what is displayed and to control how it is displayed (for example by zooming the text size).

The touchscreen enables the user to interact directly with what is displayed, rather than using a mouse, touchpad, or any other intermediate device (other than a stylus, which is optional for most modern touchscreens).

Touchscreens are common in devices such as game consoles, personal computers, tablet computers, and smartphones. They can also be attached to computers or, as terminals, to networks. They also play a prominent role in the design of digital appliances such as personal digital assistants (PDAs), satellite navigation devices, mobile phones, and video games and some books (Electronic books).



Fig 1.1 picture depicting a touch screen

The popularity of smartphones, tablets, and many types of information appliances is driving the demand and acceptance of common touchscreens for portable and functional electronics. Touchscreens are found in the medical field and in heavy industry, as well as for automated teller machines (ATMs), and kiosks such as museum displays or room automation, where keyboard and mouse systems do not allow a suitably intuitive, rapid, or accurate interaction by the user with the display's content.

Historically, the touchscreen sensor and its accompanying controller-based firmware have been made available by a wide array of after-market system integrators, and not by display, chip, or motherboard manufacturers. Display manufacturers and chip manufacturers worldwide have acknowledged the trend toward acceptance of touchscreens as a highly desirable user interface component and have begun to integrate touchscreens into the fundamental design of their products.

## HISTORY

E.A. Johnson described his work on capacitive touch screens in a short article which is published in 1965 and then more fully—along with photographs and diagrams—in an article published in 1967. A description of the applicability of the touch technology for air traffic control was described in an article published in 1968. Frank Beck and Bent Stumped, engineers from CERN, developed a transparent touch screen in the early 1970s and it was manufactured by CERN and put to use in 1973. This touchscreen was based on Bent Stumpe's work at a television factory in the early 1960s. A resistive touch screen was developed by American inventor G. Samuel Hurst who received US patent #3,911,215 on Oct. 7, 1975. The first version was produced in 1982.

In 1972, a group at the University of Illinois filed for a patent on an optical touch screen.[ These touch screens became a standard part of the Magnavox Plato IV Student Terminal. Thousands of these were built for the PLATO IV system. These touch screens had a crossed array of 16 by 16 infrared position sensors, each composed of an LED on one edge of the screen and a matched phototransistor on the other edge, all mounted in front of a monochrome plasma display panel. This arrangement can sense any fingertip-sized opaque object in close proximity to the screen. A similar touch screen was used on the HP-150 starting in 1983; this was one of the world's earliest commercial touchscreen computers. HP mounted their infrared transmitters and receivers around the bezel of a 9" Sony Cathode Ray Tube (CRT).

In the early 1980s General Motors tasked its Delco Electronics division with a project aimed at replacing an automobile's nonessential functions (i.e. other than throttle, transmission, braking and steering) from mechanical or electro-mechanical systems with solid state alternatives wherever possible. The finished device was dubbed the ECC for "Electronic Control Centre”, a digital computer and software control system hardwired to various peripheral sensors, servos, solenoids, antenna and a monochrome CRT touchscreen that functioned both as display and sole method of input. The EEC replaced the traditional mechanical stereo, fan, heater and air conditioner controls and displays, and was capable of providing very detailed and specific information about the vehicle's cumulative and current operating status in real time. The ECC was standard equipment on the 1985-1989 Buick Riviera and later the 1988-89 Buick Reatta, but was unpopular with consumers partly due to technophobia on behalf of some traditional Buick customers, but mostly because of costly to repair technical problems suffered by the ECC's touchscreen which being the sole access method, would render climate control or stereo operation impossible.

### RESISTIVE TOUCH SCREEN

In electrical engineering, resistive touchscreens are touch-sensitive computer displays composed of two flexible sheets coated with a resistive material and separated by an air gap or microdots. There are two different types of metallic layers. The first type is called Matrix, in which striped electrodes on substrates such as glass or plastic face each other. The second type is called Analogue which consists of transparent electrodes without any patterning facing each other. As of 2011 analogue offered lowered production costs.[citation needed] When contact is made to the surface of the touchscreen, the two sheets are pressed together. On these two sheets there are horizontal and vertical lines that, when pushed together, register the precise location of the touch. Because the touchscreen senses input from contact with nearly any object (finger, stylus/pen, palm) resistive touchscreens are a type of "passive" technology.

For example, during operation of a four-wire touchscreen, a uniform, unidirectional voltage gradient is applied to the first sheet. When the two sheets are pressed together, the second sheet measures the voltage as distance along the first sheet, providing the X coordinate. When this contact coordinate has been acquired, the voltage gradient is applied to the second sheet to ascertain the Y coordinate. These operations occur within a few milliseconds, registering the exact touch location as contact is made, provided the screen has been properly calibrated for variations in resistivity. Resistive touchscreens typically have high resolution (4096 x 4096 DPI or higher), providing accurate touch control. Because the touchscreen responds to pressure on its surface, contact can be made with a finger or any other pointing device.

The resistive touch screen is good in the point of user interface but the way keyboard like pressing keys seemed not good for many people and thus resulted in the deeper study which resulted in the capacitance touch screen which recognise even a very very low pressure tap. Thus the use of the resistive touch has slowly reduced and now today we don’t find any of those type.

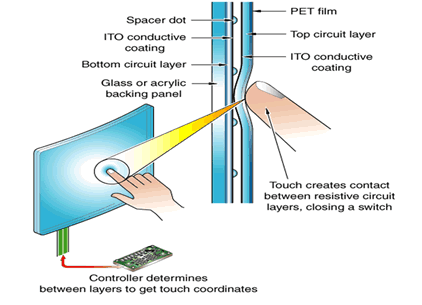


Fig-1.2 resistive touch screen

### CAPACITIVE TOUCH SCREEN

In electrical engineering, capacitive sensing is a technology, based on capacitive coupling that takes human body capacitance as input. Capacitive sensors detect anything that is conductive or has a dielectric different from that of air.

Many types of sensors use capacitive sensing, including sensors to detect and measure proximity, position or displacement, humidity, fluid level, and acceleration. Human interface devices based on capacitive sensing, such as trackpads, can replace the computer mouse. Digital audio players, mobile phones, and tablet computers use capacitive sensing touchscreens as input devices. Capacitive sensors can also replace mechanical buttons. There is also a musical instrument, the theremin, that uses capacitive sensing to allow a human player to control volume and pitch without physically touching the instrument.

Capacitive sensors are constructed from many different media, such as copper, Indium tin oxide (ITO) and printed ink. Copper capacitive sensors can be implemented on standard FR4 PCBs as well as on flexible material. ITO allows the capacitive sensor to be up to 90% transparent (for one layer solutions, such as touch phone screens). Size and spacing of the capacitive sensor are both very important to the sensor's performance. In addition to the size of the sensor, and its spacing relative to the ground plane, the type of ground plane used is very important. Since the parasitic capacitance of the sensor is related to the electric field's (e-field) path to ground, it is important to choose a ground plane that limits the concentration of e-field lines with no conductive object present.

Designing a capacitance sensing system requires first picking the type of sensing material (FR4, Flex, ITO, etc.). One also needs to understand the environment the device will operate in, such as the full operating temperature range, what radio frequencies are present and how the user will interact with the interface.

There are two types of capacitive sensing system: mutual capacitance, where the object (finger, conductive stylus) alters the mutual coupling between row and column electrodes, which are scanned sequentially; and self- or absolute capacitance where the object (such as a finger) loads the sensor or increases the parasitic capacitance to ground. In both cases, the difference of a preceding absolute position from the present absolute position yields the relative motion of the object or finger during that time. The technologies are elaborated in the following section.

* **Surface capacitance**

In this basic technology, only one side of the insulator is coated with conductive material. A small voltage is applied to this layer, resulting in a uniform electrostatic field. When a conductor, such as a human finger, touches the uncoated surface, a capacitor is dynamically formed. Because of the sheet resistance of the surface, each corner is measured to have a different effective capacitance. The sensor's controller can determine the location of the touch indirectly from the change in the capacitance as measured from the four corners of the panel: the larger the change in capacitance, the closer the touch is to that corner. With no moving parts, it is moderately durable, but has low resolution, is prone to false signals from parasitic capacitive coupling, and needs calibration during manufacture. Therefore, it is most often used in simple applications such as industrial controls and interactive kiosks.

* **Projected capacitance**

Projected capacitive touch (PCT) technology is a capacitive technology which allows more accurate and flexible operation, by etching the conductive layer. An X-Y grid is formed either by etching one layer to form a grid pattern of electrodes, or by etching two separate, perpendicular layers of conductive material with parallel lines or tracks to form the grid; comparable to the pixel grid found in many liquid crystal displays (LCD).

The greater resolution of PCT allows operation with no direct contact, such that the conducting layers can be coated with further protective insulating layers, and operate even under screen protectors, or behind weather and vandal-proof glass. Because the top layer of a PCT is glass, PCT is a more robust solution versus resistive touch technology. Depending on the implementation, an active or passive stylus can be used instead of or in addition to a finger. This is common with point of sale devices that require signature capture. Gloved fingers may or may not be sensed, depending on the implementation and gain settings. Conductive smudges and similar interference on the panel surface can interfere with the performance. Such conductive smudges come mostly from sticky or sweaty finger tips, especially in high humidity environments. Collected dust, which adheres to the screen because of moisture from fingertips can also be a problem. There are two types of PCT: self capacitance, and mutual capacitance.

Mutual capacitive sensors have a capacitor at each intersection of each row and each column. A 12-by-16 array, for example, would have 192 independent capacitors. A voltage is applied to the rows or columns. Bringing a finger or conductive stylus near the surface of the sensor changes the local electric field which reduces the mutual capacitance. The capacitance change at every individual point on the grid can be measured to accurately determine the touch location by measuring the voltage in the other axis. Mutual capacitance allows multi-touch operation where multiple fingers, palms or styli can be accurately tracked at the same time.

Self-capacitance sensors can have the same X-Y grid as mutual capacitance sensors, but the columns and rows operate independently. With self-capacitance, current senses the capacitive load of a finger on each column or row. This produces a stronger signal than mutual capacitance sensing, but it is unable to resolve accurately more than one finger, which results in "ghosting", or misplaced location sensing.

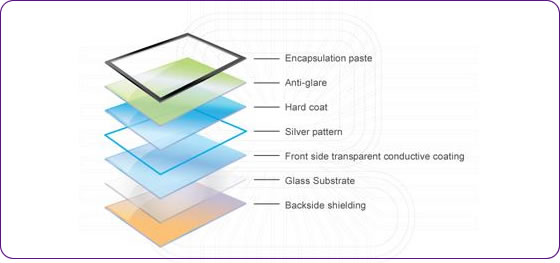


Fig-1.3 capacitive touch screen layers

### MULTI TOUCH

In computing, multi-touch refers to the ability of a surface (a trackpad or touchscreen) to recognize the presence of more than one or more than two points of contact with the surface. This plural-point awareness is often used to implement advanced functionality such as pinch to zoom or to activate certain subroutines attached to predefined gestures.

The two different uses of the term are the result of the quick developments in this field, which resulted in many companies using the term "multi-touch" for marketing purposes for older technology that is called gesture-enhanced single-touch or several other terms by other companies and researchers. There are several other similar or related terms that attempt to differentiate between whether a device can exactly determine or only approximate the location of different points of contact and that attempt to further differentiate between the various technological capabilities,[5] but they are often used as synonyms in marketing.

Multi-touch technology began in 1982, when the University of Toronto's Input Research Group developed the first human-input multi-touch system, using a frosted-glass panel with a camera placed behind the glass. In 1985, the University of Toronto group including Bill Buxton developed a multi-touch tablet that used capacitance rather than bulky camera-based optical sensing systems. Touchscreens would not be popularly used for video games until the release of the Nintendo DS in 2004. Until recently, most consumer touchscreens could only sense one point of contact at a time, and few have had the capability to sense how hard one is touching. This has changed with the commercialization of multi-touch technology.



Fig-1.4 multi touch

## PROBLEM STATEMENT

The main part of a computer or a mobile device is the display. It is the medium of interaction between the user and the system. That is we can see the output and interact with the applications through GUI presented on the screen. Today the smartphones usage is growing tremendously day by day. In a smartphone also the screen is the medium of interaction. In the case of smartphone we control the system using touch screen (capacitive multi touch) unlike the traditional computers that use keyboard and mouse.

The touch screen detects the input whenever we lay our finger on the screen and it responds to it accordingly. Most of the capacitance touch screens detect only the input given through the skin part of the body. Today’s high end smartphones have the ability to detect the input through nails, pencils, stylus and also with gloves. As touchscreen is the most important and only input for the smartphones, the problems in the touchscreen will affect the working of the device badly. Sometimes while using different applications we may feel that the device is not responding to our touch input. Then we will feel that there might be some problem with our touch screen. Actually the problem may also occur due to the bugs in the application in which we have faced the problem. So we can’t directly come to the conclusion that our touch screen has some issues. We will reach the service centres when we get some problems with the device. But going to the service centre every time we face a similar situation, which is generally often in smartphones is not good and advisory. So there is a need for checking whether the problem exists in the application or our device’s touch screen. The Touch screen tester is the software that allows us to check whether our touch screen is working perfectly or not. This application will check touch response for each and every pixel on the screen even for high end smartphones that sport a 2560\*1440 (Quad HD).

The application when started by the user will show him a complete black screen and whenever the user swipe around the screen, the screen will turn to white wherever it recognise the input. After swiping all over the screen we can easily find the defective part of the touch screen as that part will still remain in the black colour. Thus it provides an easy and efficient way of testing of our touch screen with in a minute.

## SCOPE OF THE PROJECT

Nowadays, Android phone has emerged as the world’s most popular mobile platform. Android is the world’s most popular mobile platform. It’s the largest installed base of any mobile platform and growing fast. Millions of users are using android phones and android application is becoming more and more popular.

So the problems that users are facing with the phones will also be there. To check one of those main problems that is touch screen which is the medium of interaction of the device and the user, we want to develop an application to check whether their touch screen is properly working or not. This will help the user not to reach the service centre each time when he faces small problems with his touch screen. Our application will help him know whether there is problem in the hardware or just the problem he faced is due to a bug in the system.

## EXISTING SYSTEM VS TOUCH SCREEN TESTER

There are so many applications available at present to check whether the touch screen is working properly or not. But there are no true and effective applications that can test each and every pixel of the touch screen. Our application will truly check each and every pixel of the screen and there by provides the efficient way to check the working of the touch screen.

# LITERATURE SURVEY

There are many people that feel that a touchscreen is not an improvement to a phone. These people are that that have gotten used to the use of a keyboard and have made it a habit to take these and use them as opposed to the touchscreen. Then, there are those that swear by the use of a touchscreen as they see it is easier to use than a keyboard. The problem that a lot of these people have is that of their fingers being way too big to hit a single key and they end up hitting multiple keys at once.

The implementation of a touchscreen makes the process of using a smartphone a lot easier and affords a person the chance to make sure that they are hitting the items that they want instead of having to take a chance that they hit the right key on a keyboard. In using icons as well as an onscreen keyboard, the user is able to make use of many of the advancements that have come from taking the smartphone and placing it in their everyday life. There is also the use of a stylus that helps making the use of a touchscreen to be a lot easier and as a result a lot more productive for a user.

For those that argue that the touchscreen is not a good advancement, their big argument is that the screen tends to be damaged way too easy and therefore makes the phone useless. This is a risk that is associated with the use of these phones, but one that is well worth it in getting the ease of use from not hitting the wrong keys all the time.

The use of a keyboard can often slow a person down in getting their work done. This can also cause mistakes to be made. The reason that mistakes can be made, is that many people are very visual when they are using their phone; this means that they see with their eyes and are not really thinking about what a key that they are hitting really does. This is why the touchscreen is more preferred it also allows for the phone to be smaller is size as opposed to using a keyboard design.

For those that question is a touchscreen really an improvement? After reading this article, the user is able to see that there are many advancements that have been made to the smartphone and one of these is the use of an onscreen keyboard as well as a touch screen that makes their life a lot easier in the long run. In the years to come the use of a touchscreen will become a lot more popular and will as a result allow many of the users that take to using these to know that they are getting the best deal for their phones and see a lot of the advantages that come from a touchscreen. There will always be those that will fight the touchscreen these people will eventually see the many advantages that come from a touchscreen.

Smart phones have introduced great easiness in our daily life by mobile applications. Nowadays, it is possible to complete tasks on-the-go, without the need of computer. One of the facilities provided is the mechanism to buy flight tickets via mobile phones. However, in the development phase, often poor consideration of end-users' usability requirements leads to under utilisation of such facility, thus decreasing potential profit of companies. The purpose of this work is to investigate usability problems for mobile flight booking applications on touch-screen phones and suggest solutions. Main expectations of users are presented from HCI (Human Computer Interaction) perspective and discussed through a case study. Questionnaire and interview methods were used for collecting data. Paper prototype has also been utilized to verify users' expectations of mobile flight booking more accurately. Results reveal that the users are very much concerned with the easiness and the lucidness of functions. Usability is a highly considerable subject for users to prefer a mobile flight booking application over booking tickets via online/ or agencies.

## USAGE IN COMPUTING

A touchscreen is an electronic visual display that the user can control through simple or multi-touch gestures by touching the screen with a special stylus/pen and-or one or more fingers. Some touchscreens use an ordinary or specially coated gloves to work while others use a special stylus/pen only. The user can use the touchscreen to react to what is displayed and to control how it is displayed .

The touchscreen enables the user to interact directly with what is displayed, rather than using a mouse, touchpad, or any other intermediate device (other than a stylus, which is optional for most modern touchscreens).

Touchscreens are common in devices such as game consoles, personal computers, tablet computers, and smartphones. They can also be attached to computers or, as terminals, to networks. They also play a prominent role in the design of digital appliances such as personal digital assistants (PDAs), satellite navigation devices, mobile phones, and video games and some books (Electronic books).

The popularity of smartphones, tablets, and many types of information appliances is driving the demand and acceptance of common touchscreens for portable and functional electronics. Touchscreens are found in the medical field and in heavy industry, as well as for automated teller machines (ATMs), and kiosks such as museum displays or room automation, where keyboard and mouse systems do not allow a suitably intuitive, rapid, or accurate interaction by the user with the display's content.

Historically, the touchscreen sensor and its accompanying controller-based firmware have been made available by a wide array of after-market system integrators, and not by display, chip, or motherboard manufacturers. Display manufacturers and chip manufacturers worldwide have acknowledged the trend toward acceptance of touchscreens as a highly desirable user interface component and have begun to integrate touchscreens into the fundamental design of their products.



Fig 2.1 touch screen in mobile



Fig 2.2 touch screen in laptop

# SYSTEM REQUIREMENTS ANALYSIS

## FUNCTIONAL REQUIREMENTS

Android APIs will be available on every Android phone, but there are a few APIs which have special concerns: the "optional" APIs.

These are optional in the sense that a given handset may not support them fully or even at all. For instance, a given handset may not have GPS or Wi-Fi hardware. In this case, the APIs for accessing these features will still be present, but they may not work in the same way. For instance, the location API will still exist on devices without GPS, but there may simply be no installed provider, meaning that the API can't be usefully used.

## NON FUNCTIONAL REQUIREMENTS

The correct specification and adherence of non-functional requirements similarly plays at least an equal – if not a greater – role in the success of mobile applications also. This is due to the following reasons:

Mobile devices are uniquely constrained in several dimensions such as the processor speed, memory, multi-tasking support, available network bandwidth, screen real estate etc. These constra ints translate into strict bounds being imposed on the operating characteristics of an application running on a mobile device.

Mobile applications need to operate successfully (or degrade gracefully) within a wide spectrum of operating conditions, such as a range of supported screen resolutions and form factors, network bandwidth situations and network types (2G, 3G, WiFi) etc.

Mobile applications sometimes need to interact with the device’s sensors such as GPS, accelerometer, the ambient light sensor, camera etc. The application must respect the sensor’s operating characteristics such as its operating range, sensitivity, accuracy, minimum polling interval etc.

Finally, users expect a different quality of user experience from an application running on the mobile device than they do from their desktop computer. For example, it is much less acceptable to have to reboot the phone when a mobile application hangs. Thus the latitude for error tends to be less for a mobile application than for the desktop version of the application in the same situation.

Thus, it is critical to:

* Capture these requirements as accurately and exhaustively as possible,
* Ensure that the application’s design addresses each requirement, and
* Develop the application to meet these requirements, and
* Test the application against each requirement so as to verify compliance.

## SOFTWARE REQUIREMENTS

The software required for Android development is free and readily available on the Web:

* The Java Development Kit (JDK).
* The Eclipse IDE (Java Developers version) with the Android plug-in is not technically essential since everything it does can be done using the command line, but it simplifies so many things that we will consider it to be essential.
* The Android SDK and add-ons such as the Google Maps SDK.

## SUPPORTED OPERATING SYSTEMS

* + Windows XP (32-bit), Vista (32- or 64-bit), or Windows 7 (32- or 64-bit)
  + Mac OS X 10.5.8 or later (x86 only)
  + Linux (tested on Ubuntu Linux, Lucid Lynx)
  + GNU C Library (glibc) 2.7 or later is required.
  + On Ubuntu Linux, version 8.04 or later is required.
  + 64-bit distributions must be capable of running 32-bit applications. For information about how to add support for 32-bit applications, see the Ubuntu Linux installation notes.

## HARDWARE REQUIREMENTS

* Memory of 4GB RAM or more
* Monitor resolution of 1024 x 768 or higher
* Intel Pentium 4 or AMD Athlon 2GHz or higher
* 1GB(or more) available hard disk space

## MODULES AND THEIR FUNCTIONALITIES

Our project has two modules. The first one is for the front end programming that is XML. The second one is JAVA for back end programming.

**XML**:

When you’re getting started with developing Android apps using Eclipse and the ADT plugin, Eclipse’s powerful graphical layout editor is a great place to start visually designing your user interface. However, this "what you see is what you get" approach has its limitations, and at some point you'll need to switch to XML. One of the major benefits of declaring your UI in XML is the ability to keep the UI and the behaviour of your app separate, giving you the freedom to tweak your app’s presentation without disrupting its underlying functionality.

We used XML to design the home page for the touch screen tester. Using label we have given the description in the home page. We have also used a button named proceed to jump the screen to next page when the user clicks that button. Here the user interface part is designed using XML as the front end programming language.

**Java:**

Android applications are developed using the Java language. As of now, that’s really your only option for native applications. Java is a very popular programming language developed by Sun Microsystems (now owned by Oracle). Developed long after C and C++, Java incorporates many of the powerful features of those powerful languages while addressing some of their drawbacks. Still, programming languages are only as powerful as their libraries. These libraries exist to help developers build applications.

Some of the Java’s important core features are:

* It’s easy to learn and understand
* It’s designed to be platform-independent and secure, using virtual machines
* It’s object-oriented

Android relies heavily on these Java fundamentals. The Android SDK includes many standard Java libraries (data structure libraries, math libraries, graphics libraries, networking libraries and everything else you could want) as well as special Android libraries that will help you develop awesome Android applications.

As java is the only viable option to code in the Eclipse, we selected it to code the backend programming. All the tasks like converting the black pixels to white, recognising the touch input and sliding and then responding to the respective action is done by running the corresponding java code.

# UML DIAGRAMS

Under the technical leadership of those three (Rumbaugh, Jacobson and Booch), a consortium called the UML Partners was organized in 1996 to complete the Unified Modeling Language (UML) specification, and propose it to the Object Management Group (OMG) for standardisation. The partnership also contained additional interested parties (for example HP, DEC, IBM and Microsoft). The UML Partners' UML 1.0 draft was proposed to the OMG in January 1997 by the consortium. During the same month the UML Partners formed a group, designed to define the exact meaning of language constructs, chaired by Cris Kobryn and administered by Ed Eykholt, to finalize the specification and integrate it with other standardization efforts. The result of this work, UML 1.1, was submitted to the OMG in August 1997 and adopted by the OMG in November 1997

It is important to distinguish between the UML model and the set of diagrams of a system. A diagram is a partial graphic representation of a system's model. The set of diagrams need not completely cover the model and deleting a diagram does not change the model. The model may also contain documentation that drives the model elements and diagrams (such as written use cases).

UML diagrams represent two different views of a system model:

Static (or structural) view: emphasizes the static structure of the system using objects, attributes, operations and relationships. The structural view includes class diagrams and composite structure diagrams.

Dynamic (or behavioral) view: emphasizes the dynamic behavior of the system by showing collaborations among objects and changes to the internal states of objects. This view includes sequence diagrams, activity diagrams and state machine diagrams.

**Structure diagrams**:

Structure diagrams emphasize the things that must be present in the system being modeled. Since structure diagrams represent the structure, they are used extensively in documenting the software architecture of software systems. For example, the component diagram which describes how a software system is split up into components and shows the dependencies among these components.

**Behavior diagrams:**

Behavior diagrams emphasize what must happen in the system being modeled. Since behavior diagrams illustrate the behavior of a system, they are used extensively to describe the functionality of software systems. As an example, the activity diagram describes the business and operational step-by-step activities of the components in a system.

## USECASE DIAGRAM

Use case diagrams are the diagrammatic representation depicting user’s interactions with the system. This diagram shows different types of users and various ways in which these users interact with the system. Figure 4.1 shows the use case diagram for a User and admin. It shows all the different possible ways in which a user can use the VR-CSE application. Every User can use the system through their mobile phones to know about the department.. The user can view the news, about us, attendance, checks exams, views gallery and he can download syllabus and time tables’ .Here all the details are coming from server.

 Fig-4.1 use case diagram

## SEQUENCE DIAGRAM

Sequence diagram is an interaction diagram which shows how the processes interact with one another and in what order. It shows the object interactions arranged in time sequence. It represents the objects and classes involved in the scenario. It also shows the sequence of messages exchanged between those objects which are needed to perform different functionality of the scenario. Sequence diagrams are associated with use case realizations of the Logical View of the system. The given diagram (fig 4.2) shows the sequence structure for the Display.

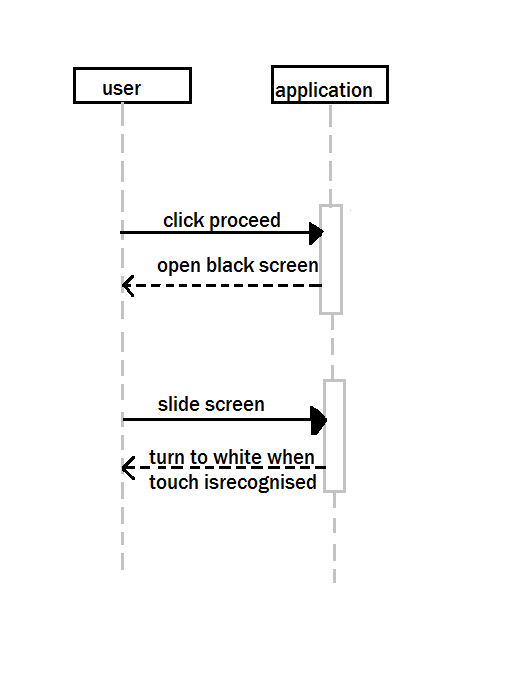
****

Fig 4.2 sequence diagram

## CLASS DIAGRAM

The class diagram is the main building block of object oriented modelling. It is used both for general conceptual modelling of the systematics of the application, and for detailed modelling translating the models into programming code. Class diagrams can also be used for data modelling. The classes in a class diagram represent both the main objects, interactions in the application and the classes to be programmed.



Fig - 4.3 class diagram

## ACTIVITY DIAGRAM

Activity diagram is a graphical representation of the workflow and the sequence of activities used to describe the functioning of the system. This diagram shows the overall control flow of the system.

The figure 4.3 shows the activity diagram of the user. The user login leads to all the options that can be performed by the user. Its basic function is to view the details that the user wants. Then the user can perform different functions. He can see the attendance, view the gallery, views updated news, downloads syllabus and time tables

Fig 4.4 activity diagram

# CODING

## XML

<?xml version=*"1.0"* encoding=*"utf-8"*?>

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

package=*"screentester.android.app"*

android:versionCode=*"1"*

android:versionName=*"1.0"* >

<uses-sdk android:minSdkVersion=*"7"* />

<application

android:icon=*"@drawable/ic\_launcher"*

android:label=*"@string/app\_name"* >

<activity

android:name=*".proceed"*

android:label=*"@string/app\_name"* >

<intent-filter>

<action android:name=*"android.intent.action.MAIN"* />

<category android:name=*"android.intent.category.LAUNCHER"* />

</intent-filter>

</activity>

<activity

android:name=*".TouchActivity"*

android:label=*"@string/app\_name"* >

<intent-filter>

<action android:name=*"android.intent.action.MAIN"* />

<category android:name=*"android.intent.category.DEFAULT"* />

</intent-filter>

</activity>

</application>

</manifest>

### XLM FUNCTION AND CLASSES:

#### PACKAGE

A full Java-language-style package name for the application. The name should be unique. The name may contain uppercase or lowercase letters ('A' through 'Z'), numbers, and underscores ('\_'). However, individual package name parts may only start with letters. To avoid conflicts with other developers, you should use Internet domain ownership as the basis for your package names (in reverse). For example, applications published by Google start with com.google. You should also never use the com.example namespace when publishing your applications. The package name serves as a unique identifier for the application. It's also the default name for the application process (see the <application> element's process process attribute) and the default task affinity of an activity (see the <activity> element's task Affinity attribute).

#### ANDRIOD VERSION: CODE

An internal version number. This number is used only to determine whether one version is more recent than another, with higher numbers indicating more recent versions. This is not the version number shown to users; that number is set by the version Name attribute. The value must be set as an integer, such as "100". You can define it however you want, as long as each successive version has a higher number. For example, it could be a build number. Or you could translate a version number in "x.y" format to an integer by encoding the "x" and "y" separately in the lower and upper 16 bits. Or you could simply increase the number by one each time a new version is released.

#### ANDROID VERSION NAME

The version number shown to users. This attribute can be set as a raw string or as a reference to a string resource. The string has no other purpose than to be displayed to users. The version Code attribute holds the significant version number used internally.

#### ANDROID ICON

An icon representing the activity. The icon is displayed to users when a representation of the activity is required on-screen. For example, icons for activities that initiate tasks are displayed in the launcher window. The icon is often accompanied by a label (see the android: label attribute).This attribute must be set as a reference to a drawable resource containing the image definition. If it is not set, the icon specified for the application as a whole is used instead (see the <application> element's icon attribute).The activity's icon — whether set here or by the <application> element — is also the default icon for all the activity's intent filters (see the <intent-filter> element's icon attribute).

#### ANDROID LABEL

A user-readable label for the activity. The label is displayed on-screen when the activity must be represented to the user. It's often displayed along with the activity icon. If this attribute is not set, the label set for the application as a whole is used instead (see the <application> element's label attribute).The activity's label — whether set here or by the <application> element — is also the default label for all the activity's intent filters (see the <intent-filter> element's label attribute).The label should be set as a reference to a string resource, so that it can be localized like other strings in the user interface. However, as a convenience while you're developing the application, it can also be set as a raw string.

#### ANDROID MAIDSDK VERSION

An integer designating the minimum API Level required for the application to run. The Android system will prevent the user from installing the application if the system's API Level is lower than the value specified in this attribute. You should always declare this attribute.

#### ANDRIOD MAIN SDK VERSION-2

An integer designating the maximum API Level on which the application is designed to run.In Android 1.5, 1.6, 2.0, and 2.0.1, the system checks the value of this attribute when installing an application and when re-validating the application after a system update. In either case, if the application's maxSdkVersion attribute is lower than the API Level used by the system itself, then the system will not allow the application to be installed. In the case of re-validation after system update, this effectively removes your application from the device.

## JAVA CODE

**package** screentester.android.app;

**Import** android.app.Activity;

**Import** android.content.Intent;

**Import** android.os.Bundle;

**Import** android.view.View;

**Import** android.view.View.OnClickListener;

**import** android.widget.Button;

**public** **class** proceed **extends** Activity {

/\*\* Called when the activity is first created. \*/

Button b;

@Override

**public** **void** onCreate(Bundle savedInstanceState)

{

**super**.onCreate(savedInstanceState);

setContentView(R.layout.*main*);

b = (Button) findViewById(R.id.*one*);

b.setOnClickListener (**new** OnClickListener()

{

public **void** onClick(View arg0)

{

// **TODO** Auto-generated method stub

Intent i = **new** Intent(proceed.**this**,TouchActivity.**class**);

startActivity(i);

}

}

}

}

package screentester.android.app;

**import android.app.Activity;**

**import android.content.Context;**

**import android.graphics.Canvas;**

**import android.graphics.Color;**

**import android.graphics.Paint;**

**import android.graphics.Paint.Join;**

**import android.graphics.Path;**

**import android.os.Bundle;**

**import android.view.MotionEvent;**

**import android.view.View;**

public class TouchActivity extends Activity

{

/\*\* Called when the activity is first created. \*/

float x,y;

@Override

public void onCreate(Bundle savedInstanceState)

{

super.onCreate(savedInstanceState);

setContentView(new myView(this));

}

public class myView extends View

{

Paint myPaint=new Paint();

Path myPath = new Path();

public myView(Context context){

super(context);

myPaint.setColor(Color.WHITE);

myPaint.setAntiAlias(true);

myPaint.setStrokeJoin(Join.ROUND);

myPaint.setStyle(Paint.Style.STROKE);

//myPaint.setStrokeJoin(Paint.Style.STROKE);

myPaint.setStrokeWidth(50);

}

public void onDraw(Canvas can){

can.drawPath(myPath, myPaint);

//can.drawCircle(x, y, 1, myPaint);

}

public boolean onTouchEvent(MotionEvent e)

{

int ac = e.getAction();

x = e.getX();

y = e.getY();

if(ac==MotionEvent.ACTION\_DOWN)

{

myPath.moveTo(x, y);

}

if(ac==MotionEvent.ACTION\_MOVE)

{

myPath.lineTo(x, y);

}

invalidate();

return true;

}

}

}

### JAVA FUNCTIONS AND CLASSES

#### ANDROID.APP WIDGET

Contains the components necessary to create "app widgets", which users can embed in other applications (such as the home screen) to quickly access application data and services without launching a new activity.

#### ANDROID.APP GRAPHICS

Provides low level graphics tools such as canvases, color filters, points, and rectangles that let you handle drawing to the screen directly.

#### ANDROID .OS

Provides basic operating system services, message passing, and inter-process communication on the device.

#### ANDROID CONTENT

Contains classes for accessing and publishing data on a device. It includes three main categories of APIs: Content sharing (android. Content) for sharing content between application components. The most important classes are: Content Provider and Content Resolver for managing and publishing persistent data associated with an application. Intent and Intent Filter, for delivering structured messages between different application components—allowing components to initiate other components and return results.Package management (android.content.pm)For accessing information about an Android package (an .apk), including information about its activities, permissions, services, signatures, and providers. The most important class for accessing this information is PackageManager.Resource management (android.content.res) for retrieving resource data associated with an application, such as strings, drawables, media, and device configuration details. The most important class for accessing this data is Resources.

#### ANDROID.VIEW

Contains classes for accessing and publishing data on a device. It includes three main categories of APIs: Content sharing (android. Content) for sharing content between application components. The most important classes are: Content Provider and Content Resolver for managing and publishing persistent data associated with an application. Intent and Intent Filter, for delivering structured messages between different application components—allowing components to initiate other components and return results.Package management (android.content.pm)For accessing information about an Android package (an .apk), including information about its activities, permissions, services, signatures, and providers. The most important class for accessing this information is PackageManager.Resource management (android.content.res) for retrieving resource data associated with an application, such as strings, drawables, media, and device configuration details. The most important class for accessing this data is Resources.

#### SETSTROKE WIDTH

This enables the application to change the width of the pixel count that reacts to the touch on the screen. It is a predefined function in android application. Its function is android.set.StrokeWidth.

#### ACTION DOWN

Constant for getActionMasked(): A pressed gesture has started, the motion contains the initial starting location.This is also a good time to check the button state to distinguish secondary and tertiary button clicks and handle them appropriately. Use getButtonState () to retrieve the button state.

#### ACTION MOVE

Constant for getActionMasked (): A change has happened during a press gesture (between ACTION\_DOWN and ACTION\_UP). The motion contains the most recent point, as well as any intermediate points since the last down or move event.

# TESTING

**What is testing?**

A process of executing a program with the explicit intention of finding errors, that is making the program fail. Testing is the process of detecting errors. Testing performs a very critical role for quality assurance and for ensuring the reliability of software. The results of testing are used later on during maintenance also.

## ANDROID AUTOMATED TESTING

The Android framework includes an integrated testing framework that helps you test all aspects of your application and the SDK tools include tools for setting up and running test applications. Whether you are working in Eclipse with ADT or working from the command line, the SDK tools help you set up and run your tests within an emulator or the device you are targeting.

## ANDROID TEST STRATEGY

Automated testing of Android applications is especially important because of the huge variety of available devices. As it is not possible to test Android application on all possible device configurations, it is common practice to run Android test on typical device configurations.

Having reasonable test coverage for your Android application helps you to enhance and maintain the Android application.

## UNIT TESTS VS. INTEGRATION TESTS ON ANDROID

A unit test tests only the functionality of a certain component. Let's, for example, assume a button in an Android activity is used to start another activity. A unit test would determine if the corresponding intent was issued, not if the second activity was started. An integration test would also check if the activity was correctly started. Android provides different classes for unit and integration tests.

## WHAT TO TEST ON ANDROID APPLICATIONS

**Test area Description:**

Activity life cycle events: You should test if you activity handles the Android life cycle events correctly. You should also test if the configuration change events are handled well and if instance state of your user interface components is restored.

**Different device configurations**:

You should also test if your application behaves well on different device configurations.

## UNIT TESTING

It concentrates on each unit of the software as implemented in source code and is a white box oriented. Using the component level design description as a guide, important control paths are tested to uncover errors with in the boundary of the module. In the unit testing, the steps can be conducted in parallel for multiple components. In my project we tested all the modules individually related to main function codes and attacks also.

## INTEGRATION TESTING

Here focus is on design and construction of the software architecture. Integration testing is a systematic technique for constructing the program structure while at the same time conducting tests to uncover errors associated with interfacing. The objective is to take unit tested components and build a program structure that has been dictated by design. The goal here is to see if modules can be integrated properly, the emphasis being on testing interfaces between modules.

This testing activity can be considered as testing the design and hence the emphasis on testing module interactions. In this project the main system is formed by integrating ll modules. When integrating all the modules we have checked whether the integration effects working of any of the services by giving different combinations of inputs with which the two services run perfectly before integration.

## VALIDATION TESTING

In this, requirements embedded as part of software requirement analysis are validated against the software that has been constructed i.e., validation succeeds when software functions in a manner that can reasonably expected by the customer.

## SYSTEM TESTING

System testing of software of hardware is testing conducted on a complete, integrated system to evaluate the system's compliance with is specified requirements. System testing falls within the scope of black box testing and as such should require no knowledge of the inner design of the code or logic.

Here the entire software system is tested. The reference document for this process is the requirements document and the goal is to see if software meets its requirements.

## TEST OBJECTIVE

• Navigation of xml screens must work perfectly.

• The entry screen, messages and responses must not be delayed.

## TEST RESULTS

All the test cases mentioned below passed successfully. No defects encountered

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Test No** | **Test Case** | **Desired Input** | **Desired Output** | **Results** |
|  |  |  |  |  |
|  |  |  |  |  |
| 1 | Launching of application | Tapping application | Opening of app with homepage as default page | Passed |
|  |  |  |  |  |
| 2 | On clicking Proceed  icon | Tap on proceed | If yes opens the testing screen | Passed |
|  |  |  |  |  |
|  |  |  |  |  |
| 3 | Touch response | slide | Black pixels convert to white | Passed |
|  |  |  |  |  |
| 4 | Device working | Slide all over screen | Every pixel coverts to white | Passed |
|  | completely |  |  |  |
|  |  |  |  |  |
| 5 | Device not working completely | slide | No change in screen colour | Passed |
|  |  |  |  |  |
| 6 | Device has some issues | Slide all over screen | Only specific areas of screen still remains in black | Passed |
|  |  |  |  |  |

# OUTPUT SCREENS/RESULT

The output screens are as follows:



Fig 7.1: MENU SCREEN

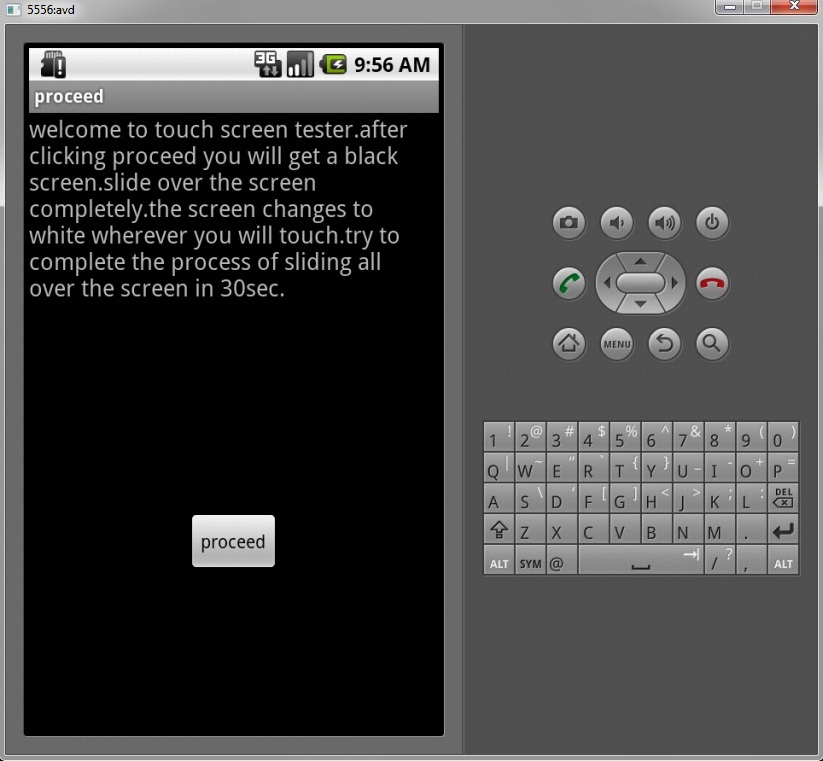


Fig 7.2: Application Main Screen



Fig 7.3: Application Working

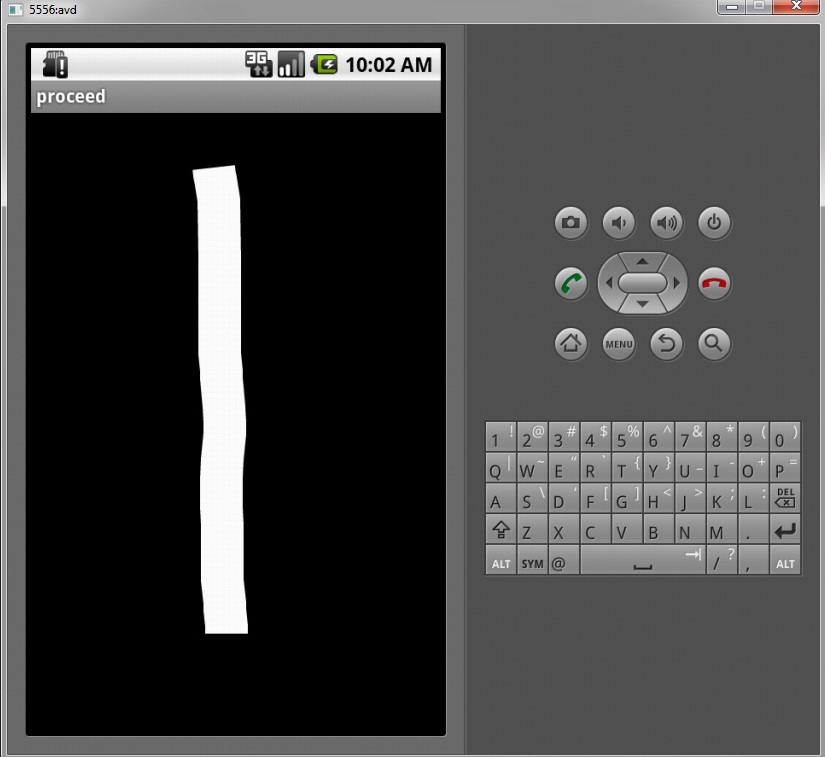


Fig 7.4: 50 PPT Screen

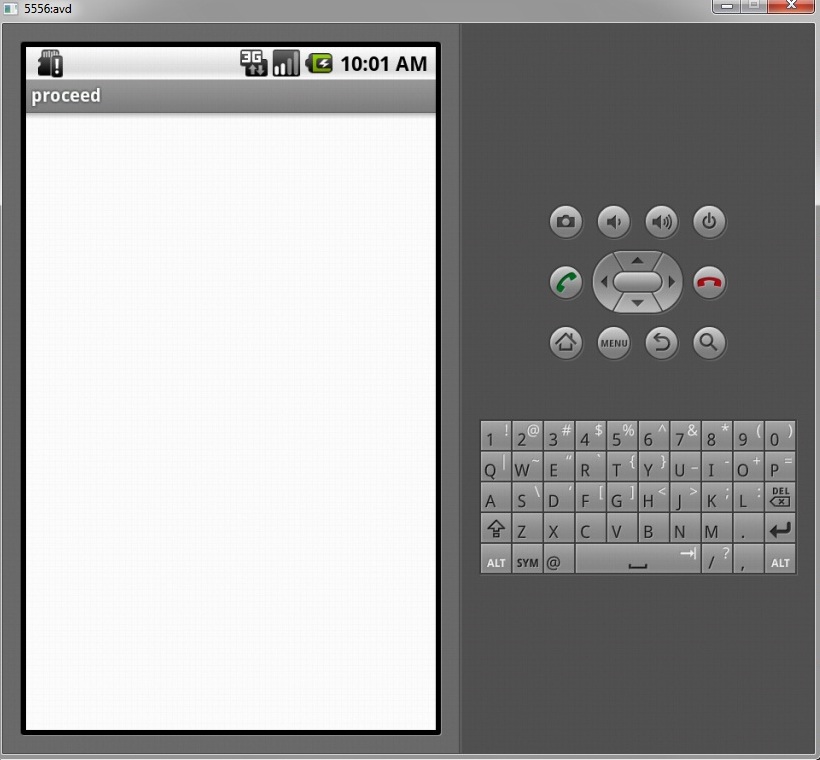


Fig 7.5: Screen Completely Working

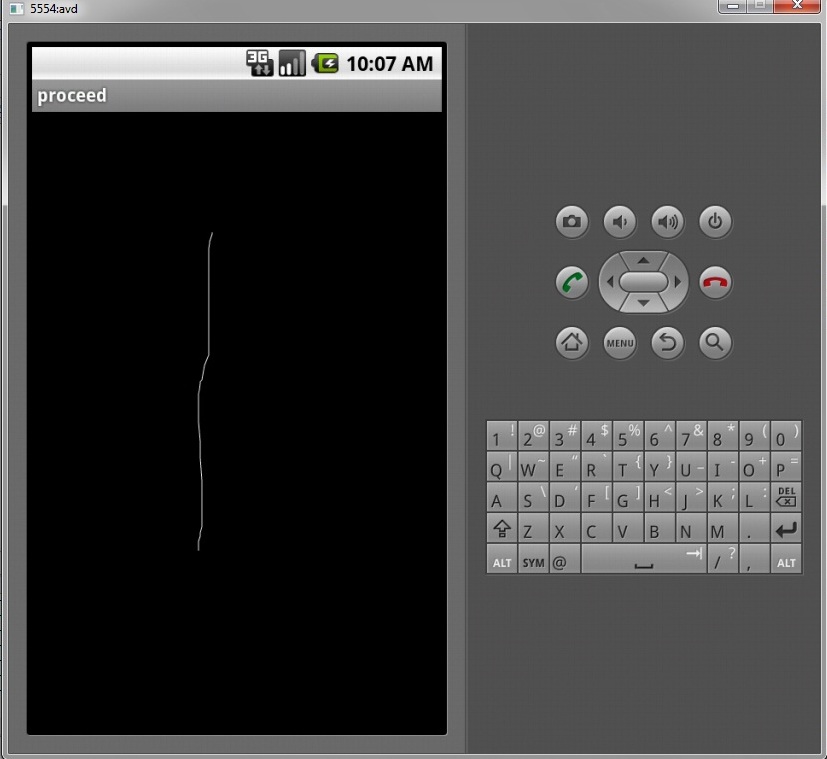


Fig 7.6 : 1 PPT Screen

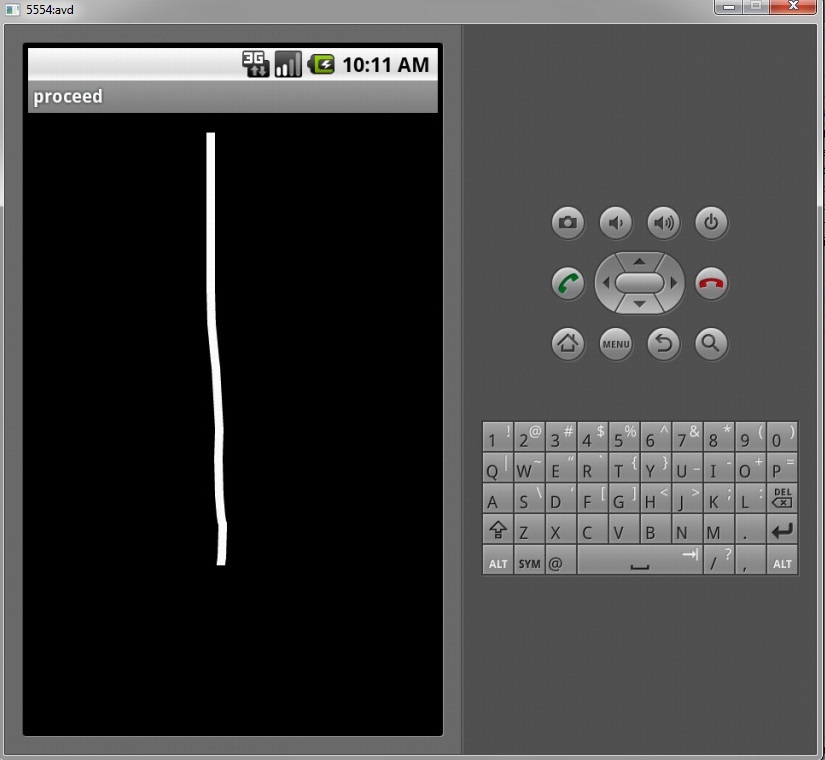


Fig 7.7: 10 PPT Screen

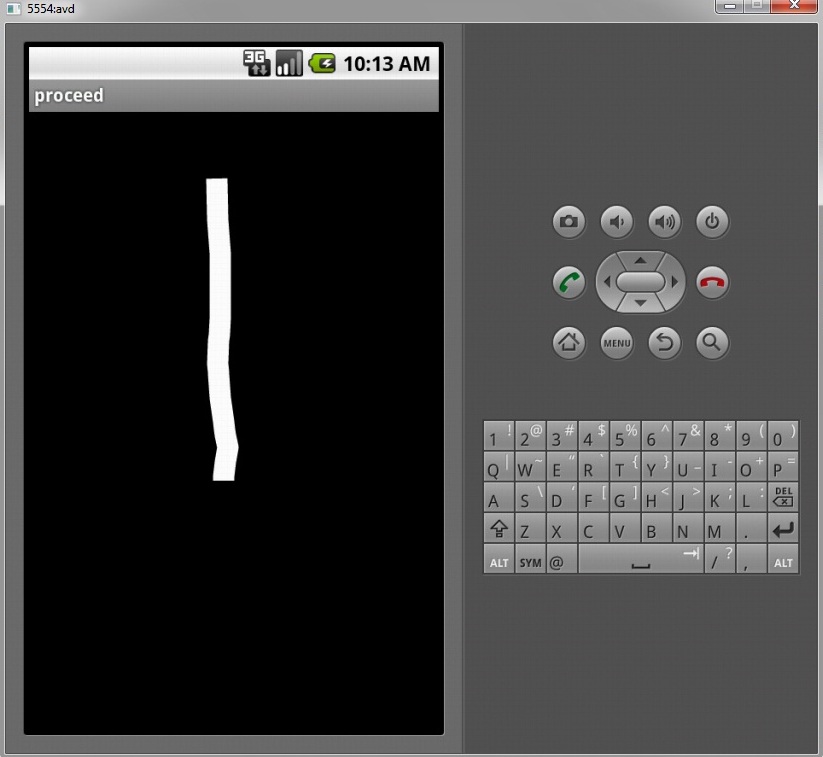


Fig 7.8 25 PPT Screen

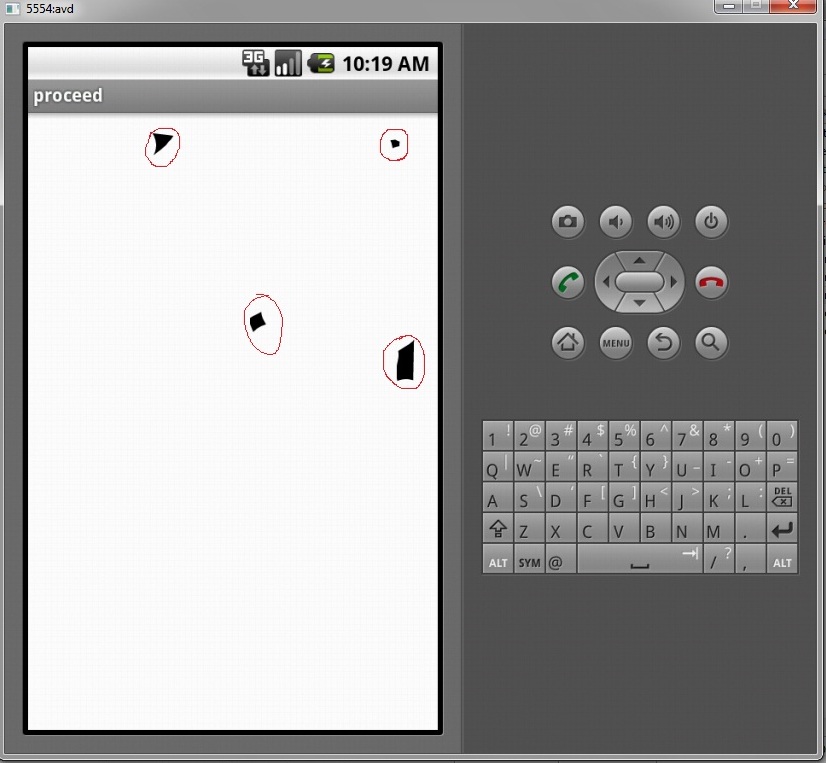


Fig 7.9: Screen If Not Working Properly

# FUTURE WORK AND EXTENSION

The touch screen also have the capability of multi touch. So we can extent the project to check whether the multi touch is working properly or not. There is also a factor of touch screen that is the number of points of multi touch it supports. Some devices support multi touch of 2 fingers only. Some devices support 5 points of multi touch and some high end devices even support 10 points of multi touch. We can extend our project to test the capability of multi touch that is the number of points of multi touch it supports and also the working of the multi touch.

Hardware is not ended with touch screen. There are even more hardware components like the sensors and display. We will extend the capability of our hardware tester to calibarate the accuracy of sensor and also the accuracy of the display.

# REFERENCES

[www.developer.android.com](http://www.developer.android.com)

[www.xda-developers.com](http://www.xda-developers.com)

[www.stackoverflow.com](http://www.stackoverflow.com)

[www.apptopia.com](http://www.apptopia.com)

[www.andevcon.com](http://www.andevcon.com)