**Android applikations (app) utveckling**

**Fredrik Lindell**

**Referat**

1. **vad är android**
   1. **Hur utvecklingen fungerar**
2. **Huvuddelarna av plattformen**
   1. **Linux kernel lagret**
   2. **Native lagret**
   3. **Dalvik**
   4. **Android och Java**
   5. **Application Framework**
   6. **Application**
   7. **Publicering**
3. **Main Building Blocks**
   1. **Aktiviteter**
   2. **Intents**
   3. **Services**
   4. **Content Providers**
   5. **Broadcast Receivers**
   6. **Applicaton Context**
4. **verktyg**
   1. **Eclips**
   2. **Android studio**
   3. **Android SDK**
      1. **Android SDK Tools**
5. **Användargränssnitt**

**Referat**

Handlar mera allmänt om Android och hur Android operativsystem används av användare, utvecklare och marknadsförare. Hur Android är uppbyggd och Hur det skiljer sig från andra operativ system.

Android översikt: Introduktion till Android och des historik. Stack: En översikt på Androids operativsystemets delar. Primära byggdelar: Androids komponenter som används för att utveckla en app. Intents: funktioner som gör utvecklingen för applikationer lättare. Gränssnitt: utveckling av gränssnitt för användare. Livscykel: hur länge en applikation ”lever”.

Går igenom alla Android lagren, livscykeln på aktiviteter, verktygen som används med Android och användare gränssnittet.

**1.0 Vad är Android**

[1]Android är en öppen källplattform för mobila enheter. Det betyder att man får en bas som man kan utveckla hur som helst utan att behöva tänka på licenser. Allt inom Android förutom Linux-kernelen är licenserat under ”business-frendly licenses”. Tack vare denna licens får utvecklare tillgång till hela plattformens källkod och tillverkare kan lätt överföra Android operativsystem till sina enheter.

Allt vad en utvecklare behöver för att börja programmera i Android är Android SDK (Software Development Kit) som ger utvecklare tillgång till verktyg och ramar för att utveckla applikationer snabbt och lätt. Då användarna får Android så fungerar det direkt och de har tillgång till att finjustera sina mobiler. Tillverkare får ett program de kan anpassa till flera olika enheter.

När Android utvecklades tänkte man på dåvarande mobila enheters begränsningar som troligtvis inte kommer att ändras i framtiden. Man kom fram till två saker i batteriets storlek samt kraft och storleken på mobilen kommer att påverka hastigheten och minnets storlek. Dessa två begränsningar togs i beaktande redan från början då man utvecklade Androidplattformen och detta ledd till att Android blev ett program som kan köras på många olika enheter.

Det började år 2005 med att Google köpte Android. Googles intresse i företaget var att sprida sina medie tjänster vilket de gjorde genom att få Android på så många enheter som möjligt. År 2008 gjordes första versionen av Android SDK, 2009 var Android på över 20 olika enheter och de nya versionerna av Android kom ut som Cupcake och Donut, 2010 var Android det andra bäst säljande företaget inom smarttelefoner. Efteråt har Android blivit först inom mobil säljning och finns från smarttelefoner till TV-apparater.

Från början då Google köpte Android utvecklade ett program som skulle se till att Android var kompatibelt. Så användarna inte måste tänka på vad för hårdvara och mjukvara de behöver för att köra en app. En applikation skall kunna köras på vilken Androidversion som helst och programmet som skapades var Compatibility Test Suit (CTS).

CTS hjälper till i utvecklingen av applikationer vad som behövs, vad som borde vara med och vad som är frivilligt för att få applikationen att vara kompatibel. CTS är inte något som är nödvändigt i Androidprogrammering; om man vill ha ett program som inte är tänkt för alla enheter kan man lämna bort CTS.

**1.1 Hur utvecklingen fungerar**

[2]Android är designat som en räcka där applikationen är högst upp och Linuxkernelen är längst ner. Oftast arbetar man på applikationsnivån. Kernelen är för att kommunicera mellan mjukvaran och hårdvaran. Då man programmerar i Android ingår en hel del standardapplikationer som t.ex. e-post, SMS, kalender, kartor, webbläsare och kontakter. Programmeringsspråket man använder i Android är java men inkluderar även C och C++ bibliotek. Utvecklare har tillgång till allt som finns i standardramverket. Applikationernas arkitektur är sådan att man skal kunna använda dem med andra applikationer om man t.ex. du har applikationerna anteckningsblock och kartor, antecknar en adress i anteckningsblocket och sedan vill använda kartan för att se var adressen är istället för att hoppa mellan applikationerna kan man kombinera dem.

Androidapplikationer körs med filformatet Dalvik vilket är utvecklat för att använda så lite utrymme som möjligt och har ändelserna .dex (Dalvik EXecutable) eller .odex(Optimized Dalvik EXecutable)

Eftersom Android är javabaserat kan man installera en plug-in för att få Eclipse om man vill ha det

**2.0 Huvuddelarna av plattformen**

**2.1 Linux kernel lagret**

Android operativsystemet är gjort som en stack det vill säga många lager ovanpå varandra. Bottenlagret/grunden till Android är Linux kernelen. en del av de största skälen till att Linux kernelen valdes är: den är lätt att få Linux att fungera på olika hårdvaruarkitekturer. Att använda Linux kernelen som grund gör att man inte behöver tänka på hårdvaran. Linux kernel ökar säkerheten genom att köra alla Android applikationer som Linux processer som är godkända av Linux. I Android är kernelen det enda som kan ge rättigheter till applikationer. Detta leder till ett enkelt och kraftfullt säkerhetssystem. Linux kernelen erbjuder många funktioner som Android tar nytta av som tillexempel mine, strömegenskaper, nätverk och radio funktioner. Bara för att grunden är Linux baserad betyder det inte att Android är en Linux produkt. Många vanliga funktioner i en Linux produkt fatas i Android. Android har däremot utvecklat Linux kernelen med funktioner som är mer passande för mobila enheter.

**2.2 Native lagret**

Efter Linux kernel lagret kommer Native lagret även kallat user space. Native lagret innehåller HAL, native libraries, nativ daemons och native tools.

HAL står för Hardware Abstraction Layer. Linux valdes för att det kan köras på så många enheter. Problemet är att alla enheters tillgång till drivrutinerna är olika. Android har löst problemet genom att ta de största enheterna och sammanfatta deras drivrutiner till native library. Huvud funktionen med HAL är att ge enheter de sammanfattade drivrutinerna som saknas. HALs andra funktion är att se till att ingen GPL kod kommer med till användarrummet. Orsaken är för att om programmet har GPL kod i sig kommer programmet att falla under GPL lisens vilket Android inte vill. HAL är som en bufert mellan kernel lagret och resten av staken.

Native Libraries är ett bibliotek med C/C++ dess uppgift är att hjälpa Applications Framework lagret vilket är nästa lager efter Native lagret. Biblioteket kan vara gjort specifikt för det Android operativ systemet men oftast är biblioteket gjort med open source resurser. En del av de specifika biblioteken som är gjorda är: Bionic: Hjälpertill att blockera LGPL kod från att nå användar rymden vilket skulle göra att utvecklare måste publicera allt de gör. Bionic använder sin egen lisens Apache/MIT villket gör att det utvecklarna gör inte måste publiseras. Binder:Gör att Android appar kan kommunicera med varandra. Framework libraries: Är ett antal bibliotek som hjälper systemet som till exempel: position, media, installering, telefoni, WiFi. En del av de open source biblioteken är: Webkit: Web läsare. SQLite: En SQL databas som Framework kan använda. Apache Harmony: Gör att man kan få java bibliotek. OpenGL: 3D grafik bibliotek. Open SSL: ser till att kopplingen är säker.

Native Daemons är vanligtvis kod som körs för att hjälpa systemet. En del exempel: Service Manager: kör alla andra framework tjänster. Radio interfacelayer daemon: hjälper med telefonin. Installation daemon: hjälper till att hantera appar, installation, uppdateringar och rätighets tillstånd. Media server: hjälper till med kameran, radio och andra medier tjänster. Android Debug Bridge: hjälper till med utvecklingen specifikt med kontakten mellan bords datorn och Android enheten.

Native Tools har många standrad Linux command-lines verktyg och så kör den alla Native demons. Liksom andra operativ system har Android command-lines för utvecklare. Android har ett mindre antal Linux commands för de kör med Android tool box; därför byter de flesta ut tool boxen mot Linux busybox vilket innehåller flera Linux commands.

**2.3Dalvik**

Dalvik är en virtuell maskin specifikt gjord för Android. Java Virtual Machine (VM) var designad för att passa in i alla system. Dalvik är däremot spesifierad på mobila enheter. Dalvik skulle specifikt tackla de två problemen med mobila enheter som troligtvis inte kommer att ändra i framtiden storlek och batteriförbrukning. Java VM är designat som stack-baserat medan Dalvik VM är designat som register-baserat. Detta gör att Dalvik sparar på ström som skulle fara till värme. Då Java VM skall initiera ett objekt söker den efter filen på hårddisken och sedan flyta den till RAMet. Detta går bra i en bordsdator för en mobil enhet är hårddisken och RAMet samma vilket betyder att Dalvik kan peka direkt till de filer som skall användas och endast flyta de objekt som skall ändras till RAMet. Dalvik ser till att alla applikationer körs i separata processer för att öka säkerheten. Detta leder till att man har flera Dalvik VM igång i bakgrunden men de är designade att inte ta så mycket utrymme och de delar på ett systembibliotek.

Java har många produkter som är gratis licensierade Java VM är inte en av dem. Dalvik gjordes för att få bort Java VM licensen och ersätta den med en mer open sorce licens. Att det inte skulle finnas en VM som är open sorce var ett problem som fanns under Dalviks utveckling år 2005. Nuförtiden finns det open sorce alternativ för VM.

**2.4 Android och Java**

I java compiler skriver du källkoden som sedan kompileras till byte kod som sedan körs med java VM. I Android används fortfarande java compiler som skriver källkod som kompileras till byte kod sedan med Dalvik compiler kompileras det till Dalvik byte kod som sedan exekveras i Dalvik VM. När man utvecklade Dalvik var Java källkod väldigt ostabil den ändrades och uppdaterades ofta. Därför utvecklades Dalvik att kompilera från Java byte kod som var mycket mer stabil. Detta lede till att Android programmering kunde skrivas i andra språk än Java; så länge källkoden av språket kan kompileras till Java byte kod.

**2.5 Application Framework**

Application Framework är nästa lager efter Native lagret. Detta lager innehåller Java bibliotek och tjänster som hjälper utvecklare att utveckla appar. Biblioteken är specifikt gjorda för Android.

**2.6 Application**

Application är sista lagret vilket är själva appen det vill säga ett verktyg åt användaren att kommunicera med appen. En applikation är en fil Zipp som kan öppnas som kallas för Android application package (APK) en APK innehåller de följande componenter

Android Manifest file:

Är huvud filen som ger en överblick av appen och innehåller de huvudsakliga delarna som appen komponenter, rättigheter och håller reda på minsta möjliga API nivån för att appen skall köra.

Dalvik executable:

All Java kälkod som är kompilerad till Dalvik exekutable det som kör själva applikationen

Resorces:

Allt som inte är kod som bilder, ljud/video clip och XML filer som beskriver layouten, språket

Native Libraries:

Detta är valfri. Applikationen kan innehålla native kod som C/C++ bibliotek dessa bibliotek kan packas ihop med APK filerna.

Signatures:

APK filen innehåller en digital signatur som certifierar att du är skaparen av applikationen

**2.7 Publicering**

En Android applikation måste vara signerad fören den kan installeras. Fören utvecklaren kan publicera en applikation måste utvecklaren signera den. Då det kommer till distribution av applikationer har de flesta plattformar en butik som styr och bestämmer om distributionen. Android tillåter flera olika butiker att distribuera applikationerna med sina egna regler. Detta leder till butikerna konkurrerar med varandra. Största Android distribueraren förtillfälle är Google Play. Eftersom Google Play är så stort brukar tillverkare förhands installera Google Play på enheterna för att kunna erbjuda de flesta av apparna åt användaren. Android vet detta och använder det till sin fördel genom att få utveklare att följa Android Compatibility Test Suite (CTS). Applikationer kan även distribueras via internet och USB kabel ||dessa metoder har hög risk för virus-, spion- och andra skadliga program. Google Play är mer säkrare tack vare CTS.

**3.0 Main Building Blocks**

”main bilding blocks” är de enheter som Android erbjuder utvecklare att bygga en app utav. Det vill säga huvuddelarna som används för att skapa en applikation. En applikation är oftast uppbyggd av flera enheter. Med Android building blocks kan man lätt bryta ut enheterna, jobba med dem individuellt och sedan sätta ihop dem igen. En app är så att säga en samling av aktiviteter, tjänster, leverantörer och mottagare.

**3.1 Aktiviteter**

En aktivitet är vanligtvis ett fönster som användaren ser. En applikation har vanligtvis många aktiviteter som användaren hoppar mellan. Detta betyder att aktiviteter är de mest visuella delarna av applikationen. Man kan jämföra aktiviteter med en webbsida. En webbsida har en huvudsida. En applikation har sin huvudaktivitet som är vanligtvis det första man ser när man kör appen. En hemsida har navigerings medel mellan alla sidor. En applikation har liknande medel för att komma åt alla aktiviteter. Tillexempel användaren ser på en aktivitet i en applikation sedan hoppar användaren till en annan aktivitet i en annan applikation som tillexempel e-post applikation för att skriva ett meddelande till någon.

När en aktivitet körs används många resurser som tillexempel skapa en ny Linux process, fördelar minnet åt alla UI objekt, hämta alla objekt från XML layouten och säter upp hela skärmen. Eftersom operativ systemet gör så mycket jobb för att köra en aktivitet tar Activity Manager hand om aktivitetens livs cykel. Activity Manager finns i Framework lagret och dess uppgift är att skapa, förstöra och tarhandom aktiviteter. När en användare hoppar till en ny aktivitet flyttar Activity Managern den gamla aktiviteten till en plats där den kan vänta. Tackvare detta om användaren vill gå tillbaka till en föregående aktivitet går det mycket fortare. Om en aktivitet inte har använts på länge kommer den att förstöras för att skapa mera rum för de aktiviteter som används. Allt detta är för att öka hastigheten i gränssnittet. Utvecklare kan inte påverka aktivitetens livscykel vilket tillstånd aktiviteten är i.

[bild]

Aktivitets livs sykel

Starting tillstånd

Aktiviteten är i Starttillståndet då aktiviteten inte är i minnet. Eventuellt kommer aktiviteten att flytas från Start till Runing tillstånd denna flyt är den mest krävande operationen för enheten. Därför finns livscykeln för att minska antalet gånger denna operationen görs.

Running tillstånd

Då aktiviteten är i Runing tillstånd betyder det att aktiviteten visas åt användaren och väntar på input. Endast en aktivitet kan vara i Runing tillståndet åt gången. Då aktiviteten är i Runing tillstånd får aktiviteten prioritering den får minne och resurser för att köra så fort som möjligt.

Paused tillstånd

En aktivitet far till Paused tillståndet då användaren inte kan påverka den men den är fortfarande på skärmen eller i bakgrunden av den nuvarande aktiva aktiviteten. En pausad aktivitet har fortfarande hög prioritet av mine och |\_| för det är högst troligt att användaren kommer tillbaka till aktiviteten.

Stopped tillstånd

Aktiviteten flytas till Stopped tillstånd då aktiviteten inte används och den är inte i bakgrunden. Aktiviteten sparas ifall att aktiviteten behövs snart igen. I det här skedet finns det två alternativ: aktiviteten begärs och skickas till Runing tillståndet. Aktiviteten skickas till Destroy tillståndet för den används inte och den kommer troligtvis inte att användas på länge. Beslutet om aktiviteten sparas eller skickas till Destroy sköts av Activity Manager vilket utvecklaren inte har tillgång till.

Destroy tillstånd

Aktiviteter skickas hit då Activity Manager bestemer att aktiviteten inte längre behövs. Aktiviteten kan köra endel funktioner som spara information som inte sparats sedan tas aktiviteten bort från minnet för att göra rum åt de aktiviteterna som behövs.

**3.2 Intents**

Intents fungerar ungefär på samma sätt som länkar på en hemsida. Intents är medelanden som skickas mellan huvud byggnads blocken. Dessa meddelanden startar aktiviteter, startar/stoppar tjänster. Intents kan delas upp i två olika sorters meddelanden Explicit och Implicit. Explicit är då sändaren specificerar mottagaren. Implicit är ett meddelande åt första möjliga enhet som kan göra uppgiften.

**3.3 Services**

Services är som aktiviteter men Services kan inte påverkas av användaren. Användaren måste fara via en aktivitet för att påverka Servicen. Livscykeln för en Service är enkel den har endast start och stop. Exempel en Service kan vara musikclip du kan endast starta eller stoppas och den kan endast påverkas av musik app aktiviteten.

**3.4 Content Providers**

Content Providers är gränssnitt för att förmedla data mellan applikationer. Skillnaden mellan Content Providers och Intents är att Intents är för mindre data Content Providers är för stora mängder data. Exempel på Content Provider data som delas mellan applikationer. Användarinformation, system inställningar, foton och musik.

**3.5 Broadcast Receivers**

Receivers är kod som är inaktiv och väntar på en event; en event är en Intent. I systemet händer många event som exempel: Sms-meddelande, ett telefon samtal, batteriet börjar ta slut eller när telefonen startar. Alla dessa event meddelar Receivers och dessa kan aktiveras beroende på eventen. När de aktiveras körs kod som exempelvis: starta en aktivitet, starta tjänster.

**3.6 Applicaton Context**

Application context innehåller alla byggnads block alltså Aktiviteter, Services, Content provider och Brodcast receiver alla dessa block är en application. Application context gör att byggnads blocken kan dela sin data och resurser med andra byggnads block. Application context innehåller även Dalvik Virtual maskinen och ett dedikerat filsystem. Application context har sin egen Linux ID och Linux process. Application context skapas då den första komponenten av applikationen körs. Application context lever så länge applikationen lever detta betyder att Application context livscykel är separat från aktivitets livscykeln. Då Activity Managern har terminerat alla byggnads block terminerar den även Application context.

**4.0 verktyg**

Android är inte enbart kompatibelt med flera olika enheter, utan även med olika kompileringsprogram. För att ett kompileringsprogram skall kunna kompilera koden till Android måste programmet klara av att kompilera källkoden till byte kod. Android kan kompilera byte koden till Android programmeringsspråk det vill säga Java kod. De mest vanliga verktygen som används är eclips och Android studio med Android SDK(Software Development Kit). Eclips är baserat på Java och är det vanligaste programmet att skriva Java kod. Android studio är ett verktyg specifikt gjort för Android programmering. Android SDK innehåller utvecklar verktyg.

**4.1 Eclips**

Eclips är en plattform för att skriva programkod. Eclips är licenserat under Eclipse Public License vilket gör att programmet är gratis. Detta gör att programmet passar bra med Androids policy. Eclips är huvudsakligen för programmeringsspråket Java men det går att programmera med andra språk tack vare plug-ins. Android språket är baserat på Java vilket gör att Eclips går bra att använda för att skapa Android applikationer. Som plug-in kan man ladarner Android SDK för att få in Android verktyg vilket hjälper en programmera för Android enheter.

**4.2 Android studio**

Android studio är specifikt gjord för Android programmering. Programmet erbjuder användaren kodeditering, debug, testning av kod och profilverktyg för att användaren skall skapa hög kvalitets Android applikationer. Android studio klarar av att uppdatera applikationer utan att behöva starta om applikationen. Programmet kan köra applikationen på olika simulationer av mobila enheter vilket betyder att användaren kan kontrollera på sin dator om applikationen fungerar på en vis enhet. Då användaren programmerar granskas studion koden och erbjuder förslag på kod för att minska på programmerings tiden. Användaren behöver inte jobba med standard konfigurationerna. Android Studio tillåter användaren att ändra på konfigurationerna och välja vilka bibliotek som skall användas. Programmeraren kommer troligtvis att jobba i ett team därför erbjuder Android Studio verktyg som stöder sådan arbetsprocess. Studion synkroniserar alla i teamet så alla kan jobba med programmet samtidigt. Android studio har verktyg som gör det lättare att optimera applikationen till Android enheten. Eftersom Android enheter har så många olika skärm storlekar kan man få applikationen i olika versioner och jobba med dem individuellt. Android Studio erbjuder mallar och färdiga applikationer åt användaren. Detta betyder att användaren kan ta färdig kod som är väll gjord och använda den i sin applikation. Android Studio hjälper användaren med applikationens prestationer, säkerhet och korrekthet. Android Studio erbjuder program som kan köra tester och samla information på användarens program. Om användaren använder programmeringsspråket C/C++ så har Android Studio inga problem att läsa det. Programmering kan vara svårt därför erbjuder Android Studio ”drag and drop” funktioner. Specifikt då det kommer till layouten som är XML filer. ”drag and drop” layouten kommer att anpassas till alla enheter. Eftersom mobila enheter har så lite minne kommer Android Studio utrustad med APK analysator. APK analysatorn visar storleken på alla enheter i applikationen så programmeraren kan om det behövs göras justeringar i applikationen. Ett av verktygen som ärbjuds är Vector Asset Studie. Vector Asset studio gör att programmeraren får tillgång till material från Google, skapa bitmap filer för att kunna stöda äldre Android versioner och kan importera SVG och PSD filer. Android Studio erbjuder ännu Translation Editor vilket erbjuder översättningar och visar vad som är översätt och vad som är kvar.

**4.3 Android SDK**

Android SDK är ett paket med viktiga verktyg för att programmera Android appar. De följande är de viktigaste verktygen i SDK.

**4.3.1 Android SDK Tools**

Dessa verktyg måste finnas med i en Android applikation. *APKanalyzer* visar storleken på alla enheter i applikationen. Analysen körs efter programmet är klar. Efter att analysen har körts får programmeraren en rapport på enheterna i programmet som kan jämföras. Tack vare APK analysen kan man reducera tiden som spenderas på att laga bugar i programmet och så kan den reducera storleken av APK filerna. *AVDmanager* ger programmeraren möjligheten att skapa och hantera Android Virtual Devices (AVD) från en kommandopanel. En AVD gör det möjligt för programmeraren att simulera enheter som är totalt annorlunda från en vanlig bordsdator som tillexempel smart TV eller smart armbands kloka. *Jobb* gör att programmeraren kan kryptera och dekryptera APK expansions filer i Opaque Binary Blob (OBB) format. Google play tillåter endast 100MB stora APK filer men tillåter två stora expansions filer som tillägg till APK programmet. Expansions filer är oftast för grafik, ljud och video. *Lint* är en scannar som hjälper programmeraren att strukturera kod. Scannaren söker efter onödigt som tar upp minne och kod som inte stöds. Scannern körs utan att man behöver köra appen eller ett test. *Monkeyruner* ger kommandoradsgränssnittet åt program som kan påverka Android enheter. Tack vare Monkeyruner kan en programmerare skriva ett Phyton program som kan installera Android appar, testa paket, köra appen, skicka tangent tryck till appen och ta skärmdumpar av appen. Monkeyrunner är huvudsakligen gjord för att testa applikationer och enheter. Den är inte begränsad till en applikation eller enhet den kan köra flera tester åt gången. Mankyrunner fungerar så att användaren ger input via tangentbordet som svar får man skärm dumpar dessa skärm dumpar testas mot liknande skärm dumpar som man vet är korrekt. Programmeringsspråkets test funktioner fungerar tillsammans med Monkeyrunner. *SDKmanager* gör det möjligt att visa, installera, uppdatera och avinstallera Android SDK paket.

**4.3.2 Android SDK Build Tools**

Dessa verktyg behövs för att göra en Android applikation men dessa är väldigt ovanligt att en programmerare behöver. *APKsigner* signerar APK filer och kontrollerar att APK filer har signatur. Den ger programmeraren möjligheten att signera APK och att kontrollera att alla versioner av Android kommer att bekräfta signaturen. *Zipalign* optimerar APK filer genom att se till att all ooptimerad data börjar med en specifik inriktning relevant till början av filen.

**4.3.3 Android SDK Platform Tools**

Dessa verktyg uppdateras varje gång en ny version av Android kommer ut för att få dem att stöda de nya versionerna. *Adb* Android Debug Bridge(adb) kommunicerar med enheter. Kommandon underlättas väldigt mycket av enhetens aktiviteter som instalation, felsökning och så ger den tillgång till en Unix shell. Adb är ett klientserverprogram med tre komponenter. En klient som skickar kommandon, en daemon vilket kör kommandon och servern som sköter om kommunikationen mellan klienten och daemon. adb börjar med att se om det redan finns en server om det inte finns så sammankopplas alla enheter. *etc1too1* kodar och komprimerar PNG filer till ETC1 format och avkodar ETC1 filer till PNG filer. *Fast boot* blinkar enheter med plattform och andra system bilder. Detta återställer enheten till fabriksinställningar. *Logcat* genom adb visas app och system loggar.

**4.3.4 Android Emulator**

Dessa verktyg behövs för att köra emulatorer. *Emulator* ett verktyg som låter programmeraren köra applikationer i en Android miljö. Detta är ett bra sätt att testa applikationen och då man korrigerar fel. *Mksdcard* skapar en Fat 32 diskbild som kan användas med emulatorn för att simulera externa lagringskort.

**5.0 användargränssnittet**

Androids användargränssnitt är annorlunda från andra operativsystem eftersom Android fokuserar på mobila enheter. Androids gränssnitt kan göras på två sätt Declaratively och Programmatically. Oftast används en kombination av båda.

**5.1 Declaratively**

Declarative använder sig av XML för att göra utseendet på gränssnittet. XML kodning fungerar på samma sätt som HTML kodning. HTML och XML fungerar så att man specifikt beskriver allt som exempel var enheter skall liga, färger, bilder, storlekar och stilar. Specifikationerna kan vara enda ner till pixelnivån detaljerade. Man brukar jobba inom rutfält varje ruta designeras till ett vist fält i gränssnittet som header, body och footer. Det som är bra med Declarativ design är att programmerare kan använda WYSIWYG verktyg. De flesta av dessa verktyg finns i Android Development Tools (ADT). En viss del av verktygen finns i utomstående program. Dessutom är XML enkelt att läsa även för de som inte är bekanta med Android plattformen. Nackdelen med Declarativ design är att XML är begränsat till vad programmerare kan göra, specifikt input från användaren.

**5.2 Programmatically**

I Programmatic design används Java istället för XML. I Java går det lättare att skapa knapar till programmet som gör att man får användare input. Att använda java är inte lika lät som XML och så kan java kod bli längre. Allt det man kan göra i Declarativ design kan man göra i Programmatic design. Detta betyder att Programmatic är oftast bättre än Declarativ på grund av att i java kan man specifiera vad en knap skall göra.

**5.3 Kombination**

En kombination av Declarativ och Programmatic är den bästa designen. Man använder Deklarativt(XML) för allt statiskt om gränssnittet. Byter till Programmatic(Java) för alla enheter som skall vara i gränssnittet. För att förtydliga man använder XML för hur knaparna skall se ut och Java för vad de skall göra.

**5.4 Layouts**

Gränssnittets element organiseras upp i *Views* och *Layouts*. Allt som syns som knapar, märken och textrutor far till Views. Layouts innehåller Views det vill säga Layout grupperar Views element tillsammans. En Layout kan inehåla flera Layouts för de mer komplicerade gränssnitten.

*Linear Layout* är en av de simplaste layouterna och det mest vanligaste. Layouten är en rutrad antingen vertikalt eller horisontell. *Table Layout* har ett antal rader med ett antal kolumner i sig. Helt enkelt det är ett rutfält man kan korrigera hur man vill. *Frame Layout* placerar rutorna på varandra. denna layout är mest använd för flikar. *Relative layout* fungera på liknande sätt som Table Layout skillnaden är den att i Relative har rutorna ID värden som säger var de skall vara. Att jobba med Table är att lägga till och ta bort rutor för att få den design man vill ha. I Relative är det bara att designa rutan och ange ID var den skall vara.

**Referenser:**

[1]Book: Learning Android Develop mobile apps using java and eclipse

Författare: Marko Gargenta & Masumi Nakamura

[2]Artikel: A Developer´s First Look At Android

<https://s3.amazonaws.com/academia.edu.documents/30551848/andoid--tech.pdf?AWSAccessKeyId=AKIAIWOWYYGZ2Y53UL3A&Expires=1518774229&Signature=2okGnBA2z1vDqRO2BxicqLIqaZk%3D&response-content-disposition=inline%3B%20filename%3DWhat_is_Android.pdf>

[3]Artikel: The Architecture of Open source Applicatons.

Skriven av Kim Moir

<http://aosabook.org/en/eclipse.html>

[4] företag hemsida: Android Studio

<https://developer.android.com/studio/features.html>