Högskoleingenjörsutbildning i datateknik

Grundnivå, 15 hp

Stockholm, Sverige Klicka här för att ange aktuellt årtal

Klicka här för att ange underrubrik

Emre DEmirel

SHifat chowdhury

**Återanvända modellkod skriven i C på Android, iOS och HTML5**

Examensarbete inom datateknik

Grundnivå, 15 hp

Handledare på KTH: Jonas Willén

Examinator: Ibrahim Orhan

TRITA-CBH-GRU-20XX:XX

KTH

Skolan för kemi, bioteknologi och hälsa

141 52 Huddinge, Sverige

Reuse model code written in C across Android, iOS and HTML5

EMRE DEMIREL  
SHIFAT CHOWDHURY

Sammanfattning

Sammanfattningen ska skrivas på svenska och kortfattat summera rapportens innehåll på tio till femton rader samt ett antal nyckelord (högst tio stycken). Sammanfattningen ska kortfattat redogöra för problemställningen och dess relevans, lösningen och resultatet så precist som möjligt. Sammanfattningen ska kunna läsas fristående och inte innehålla referenser och specifika akronymer som inte förklaras.

Nyckelord

Återanvändning av modellkod, C, Android, iOS, HTML5, JNI, WebAssembly, Swift

Abstract

Innehåller en engelsk översättning av den svenska sammanfattningen. Skrivs rapporten på engelska ska sammanfattningen skrivas på svenska.

Keywords

Reuse model code, C, Android, iOS, HTML5, JNI, WebAssembly, Swift

Innehållsförteckning

[1 Inledning 1](#_Toc135203722)

[1.1 Problemformulering 1](#_Toc135203723)

[1.2 Målsättning 1](#_Toc135203724)

[1.3 Avgränsningar 1](#_Toc135203725)

[1.4 Författarnas bidrag till examensarbetet 1](#_Toc135203726)

[2 Teori och bakgrund 3](#_Toc135203727)

[2.1 Bakgrund 3](#_Toc135203728)

[2.2 Tidigare arbeten 3](#_Toc135203729)

[2.3 Teknisk bakgrund 5](#_Toc135203730)

[**2.3.1** **C Programmering** 5](#_Toc135203731)

[**2.3.2** **Android** 5](#_Toc135203732)

[**2.3.3** **Jetpack Compose** 5](#_Toc135203733)

[**2.3.4** **Kotlin** 6](#_Toc135203734)

[**2.3.5** **Android Studio** 6](#_Toc135203735)

[**2.3.6** **iOS** 6](#_Toc135203736)

[**2.3.7** **Swift** 7](#_Toc135203737)

[**2.3.8** **SwiftUI** 7](#_Toc135203738)

[**2.3.9** **Xcode** 8](#_Toc135203739)

[**2.3.10** **HTML5** 8](#_Toc135203740)

[**2.3.11** **Javascript** 9](#_Toc135203741)

[**2.3.12** **Visual Studio Code** 9](#_Toc135203742)

[2.4 Återanvändning av modellkod 9](#_Toc135203743)

[2.5 Verktyg för att binda återanvändbar modellkod skriven i C 10](#_Toc135203744)

[**2.5.1** Java Native Interface (JNI) 10](#_Toc135203745)

[**2.5.2** **C interoperabilitet med Swift** 13](#_Toc135203746)

[**2.5.3** Webassembly 14](#_Toc135203747)

[3 Metod 18](#_Toc135203748)

[3.1 Vetenskaplig metod 18](#_Toc135203749)

[3.2 Prototyp 19](#_Toc135203750)

[3.3 Utvärdering av prototyp 19](#_Toc135203751)

[**3.3.1** **Kodstorlek** 19](#_Toc135203752)

[**3.3.2** **Prestanda** 19](#_Toc135203753)

[4 Resultat 20](#_Toc135203754)

[4.1 Implementering av applikationer 20](#_Toc135203755)

[**4.1.1** **Användargränsnittet** 20](#_Toc135203756)

[**4.1.2** **Modellkod** 21](#_Toc135203757)

[**4.1.3** **Bindning av modellkod skriven i Swift** 24](#_Toc135203758)

[**4.1.4** **Bindning av modellkod skriven i Android** 25](#_Toc135203759)

[**4.1.5** **Bindning av modellkod på HTML5** 26](#_Toc135203760)

[4.2 Kodstorlek 27](#_Toc135203761)

[4.3 Prestanda 29](#_Toc135203762)

[5 Analys och diskussion 30](#_Toc135203763)

[5.1 Utveckling av applikationer 30](#_Toc135203764)

[5.2 Diskussion om resultaten från undersökningen av kodstorlek och prestandatester 31](#_Toc135203765)

[5.3 Hållbar utveckling och kostnader 32](#_Toc135203766)

[6 Slutsatser 33](#_Toc135203767)

[Källförteckning 35](#_Toc135203768)

[Bilagor 37](#_Toc135203769)

[Bilaga A – Rubrik för bilaga A 37](#_Toc135203770)

[Bilaga B – Rubrik för bilaga B 38](#_Toc135203771)

# Inledning

Denna inledning kommer att behandla problemformulering, målet med arbete, avgränsning och författarens bidrag till arbetet.

## Problemformulering

Människor i dagens samhälle spenderar alltmer av sin tid på mobila enheter. Ökningen innebär att flera företag satsar på den mobila marknaden för att nå ut till sina potentiella kunder. Företag som satsar på de populära mobila plattformarna behöver då plattformsspecifika kompetens. Dessa mobila plattformar har då specifika språk, utvecklingsmiljöer, ramverk för att konstruera dessa applikationer, vilket innebär för företag att det blir dyrare då man behöver utveckla applikationer på flera plattformar för att nå ut till alla sina kunder på marknaden. Samtidigt bidrar det tidsförlust då ramverket/miljön som används för att utveckla appar på dessa mobila plattformar har förändrats markant under de senaste åren. Detta innebär att man behöver lära sig de nya teknologierna för att fortsätta utveckla appar för dessa mobila plattformar. Förutom det har många företag legacy-kod skriven i C som ska återanvändas. För KTH:s beräkning så vill man undersöka om det är möjligt att överföra modellkod skriven i C på iOS, Android, HTML5.

## Målsättning

Målet är att undersöka möjligheter att återanvända C-modellkod på HTML5, iOS och Android. För detta arbete har vi formulerat följande frågor:

* Går det att återanvända kod på dessa plattformar?
* Finns det verktyg (ramverk/bibliotek/kompilator/plugins) som kan hjälpa och köra den överförda koden på plattformen?
* Analysera hur mycket prestanda dras det med den applikationen som har originalkoden jämfört med den applikationen som har den överförda koden.
* Vilka faktorer påverkar kodens komplexitet vid implementering av återanvändning av C modellkod

## Avgränsningar

Eftersom detta ämne är brett, analyseras endast överföring av modellkoden mellan plattformarna Android, iOS, HTML5. Modelldelen hanterar logiken på applikationen.

## Författarnas bidrag till examensarbetet

Beskriv vilken del av arbetet som du själv har gjort, och vad du har fått hjälp med t.ex. av kollegor. Ange om du har redovisat någon del av arbetet under tidigare kurser eller examensarbeten. Har arbetet och rapporten utförts av fler studenter ska de enskilda författarnas bidrag redogöras för.

# Teori och bakgrund

Detta kapitel går igenom bakgrund, tidigare arbeten, plattformarna, ramverk samt möjligheterna till återanvändning av kod på dessa plattformar. Detta är då till syfte att läsaren ska få en förståelse/inblick av dessa plattformar och återanvändning av kod innan man beger sig mot metod och resultatavsnittet.

## Bakgrund

Applikationsutveckling för smarta telefon är dyr och det tar tid då man måste utveckla separata lösningar för varje specifik plattform (Android, iOS, Web) [3].Dessa plattformar som används idag förändras ständigt, vilket innebär att man har haft behov av att lära sig nya teknologier och språk för att hålla sig uppdaterad. Detta leder till behovet av att hitta andra lösningar, exempelvis att återanvända kod. Genom att använda samma kod i olika plattformar så vill man undersöka om det är fortare och kostnadseffektivt att utveckla applikationer. Det kan bidra till att man kan spara mer tid och pengar som är bra för utvecklingen och miljön. KTH har flera legacy-kod skrivna i C som behöver återanvändas. Legacy kod syftar oftast på källkod till en applikation har nått sitt slut. Oftast är delar eller hela applikationen är för komplex att underhålla och felsöka jämfört med moderna plattformar. Det är svårt att underhålla en legacy-kod eftersom det kan vara en utmaning att uppdatera eller lägga till nya, eftersom det kräver mycket tid att förstå och utveckla legacy-kod. Det kan hindra en smidig överföring till de nyare teknologier och ramverk på grund av brist effektivitet eller kompatibilitet. [2] C är ett programspråk som har existerat sedan 1972 och är inspiration till flera moderna programspråk som Java, C++ etc. Flera studier har visat att C är snabb och energieffektivt jämfört med andra populära programspråk, vilket ger en självklarhet att många väljer C som programspråk. Statistiken visar att C ligger plats två när det gäller strategiska beslut om vilket språk som ska användas för att bygga nytt mjukvarusystem.[1] Detta ger stöd till varför KTH vill undersöka att om det är möjligt att återanvända samma C-modellkod på Android, HTML5 och iOS för att dra slutsatsen om vad för resultat som kan uppstå vid återanvändning av kod samt kan det ge mindre kostnader vid utveckling av mobila applikationer.

## Tidigare arbeten

Det finns få studier som motsvarar exakt det arbete som författarna ska utföra. Men det finns flera studier som har möjliga kopplingar till arbete, exempelvis prestandaberäkningar och hållbar utveckling.

Y. Cheon et al. utförde en rapport [4] som handlar om hur mycket kod kan man återanvända från Java - Desktop applikation till Android-applikation. Detta gjordes genom att beräkna antalet rader av kod som har överförts. Det visade sig att ca 37%-40% av kod kunde återanvändas mellan Java-Desktop-applikation och Android-applikation. Men samtidigt så nämns det att 8% mer kod behövde tilläggas på Android då man behövde ta hänsyn till begränsningar som existerar på Android-applikationen.

M.Olsson genomförde en experimentell studie för att jämföra CPU-prestanda mellan Flutter och två nativa applikationer. En enkätundersökning genomfördes för att mäta användarnas tolkning om utseende och animationer. Resultaten som kom fram att Flutter har fördelen när det kommer till CPU-prestanda jämför med native, men är inte lika utvecklat när det gäller animationer. [5]

C.Fourniers forskning handlar om att jämföra prestandan mellan progressiva webbappar och mobila applikationer på Android, särskilt när det gäller smidighet, minne och CPU användning. Enligt rapporten har det inte funnits tidigare forskning på prestandan hos progressiva webbappar, speciellt hos användargränssnittet. Där ville man utreda responsivitet, hur snabbt applikationen kan svara på användarinmatning och smidighet, hur snabbt kan det rendera nya bilder. Hon kom fram att progressiva webbappar har stora fördelar men är inte lika smidig användarupplevelse som mobila applikationer på Android. [6]

A. Malki utförde en studie för att jämföra prestanda hos två cross-plattform ramverk, det är Apache Cordova och Flutter. Syftet med studien var att undersöka vilket av dessa ramverk passar bäst utifrån prestandaperspektiv. Båda ramverken kan använda en och samma app på olika plattformar som Android och iOS. Det som analyserades är uppstartstiden, exekveringstiden, batteriförbrukningen, CPU och RAM-användning. Metoden gick ut på att skapa två applikationer med samma funktionalitet i varje ramverk för att jämföra resultatet.

Testerna för exekveringstid och RAM utfördes 35 gånger på applikationerna. Det visade sig utifrån testerna på exekveringstid att Flutter har bättre prestanda än Apache Cordova. Flutter använde Java String Pool för att visa resultatet och Apache Cordova använde Javascript där strängarna hanterade på ett annat sätt. Flutter visade sig vara sämre än Apache Cordova. CPU-testerna utfördes genom att iterera genom olika funktioner 35 gånger, där Apache Cordova hade mindre CPU-användning jämfört med Flutter. Apache Cordova hade mindre batteriförbrukning än Flutter applikationen på grund av mindre användning av RAM och CPU. Apache Cordova hade en snabbare uppstartstid än Flutter, men Flutter visade sig att vara snabbare vid laddning i release läge. Flutter hade högre installationsstorlek än Apache Cordova men båda minskade i release läge. Uppstartstiden varierar på grund av kompilering läge och installationsstorlek. [7]

S. Herelius genomför en studie om grön kodning där flera människor använder flera enheter och webbplatser. Som utvecklare har man ett större ansvar att hålla koll på miljöpåverkan. Syftet med studien är att hitta förslag på hur webbutveckling kan göra företag mer lönsamma och attraktiva för konsumenter. Denna uppsats undersöker olika alternativ för webbutveckling för att spara mer energi och utforska programmeringsspråk, databaser, innehåll och design. Genom att undersöka dessa områden kan man få en djupare förståelse för hur miljövänliga de är. Resultaten visade att C och Rust är de bästa energieffektiva programmeringsspråken enligt flera studier. Det visar sig också att PHP och JavaScript verkar vara mer effektiva än Swift och Python i testfallen när det gäller energiförbrukning. [8]

## Teknisk bakgrund

Detta avsnitt går igenom på djupet hur dessa plattformar, språk, ramverk och verktyg fungerar. Syftet är att hitta information hur tekniken fungerar innan man går vidare till utvecklingsfasen.

### **C Programmering**

C är ett programmeringsspråk som är populärt och flexibelt att använda.  Det är ett strukturerat programmeringsspråk som används för att skriva olika applikationer, komplexa program och operativsystem. C är ett basspråk för många programmeringsspråk, genom att lära sig C så blir det enklare att lära sig andra programmeringsspråk. C kan användas brett i olika applikationer, som ger snabbare exekvering. C är ett portabelt språk som gör att program som är skrivna på C kan köras på andra maskiner, som ger möjligheten att använda koden på en annan dator. [9]

### **Android**

Android är ett Linux-baserat operativsystem och programmeringsplattform utvecklad av Google, främst för mobiltelefoner, surfplattor, smarta klockor, IoT enheter och etcetera. Den första kommersiella versionen av Android lanserades år 2008. Idag är Android den populäraste och största mobila plattformen som finns i världen. [10] Det finns cirka 2,5 miljarder enheter som har Android som operativsystem. Källkoden till Android är öppen för alla, vilket medför att alla kan både modifiera och utnyttja Android som operativsystem utifrån sina önskemål. [11]

### **Jetpack Compose**

Jetpack Compose är ett ramverk utvecklat av Google, främst gjord för att bygga användargränssnitt på Android. Det är även möjligt att använda Jetpack Compose på andra plattformar som på webben etc. Ramverket lanserade sin första version i juli 2021. Syftet med Jetpack Compose är att underlätta utvecklandet och underhållandet av användargränssnittet på Android. [12] Jetpack Compose är det rekommenderade valet att bygga Android-applikationer. [13] Tidigare så var det vy-baserade applikationer som rekommenderade valet. Problemet med vy-baserade applikationer är att det finns stora risker att göra fel om man hanterade vyer manuellt och det är komplicerat att underhålla applikationen ju fler vyer man har.

Under de senaste åren har flera industrier tagit fram ramverk som Jetpack Compose som gör det lättare att utveckla och testa användargränssnittet. Jetpack Compose använder det så kallat deklarativt API, vilket innebär att som utvecklare så ska fokuset ligga på UI-programmering sen vid körning så tar Compose-miljön hand om resten. Det som sker på bakgrunden på en Jetpack Compose applikation är att den först utför det så kallas för initial composition vilket innebär att den genererar användargränssnitt första gången på skärmen. Vid förändring på data så påverkas bara de delar av användargränssnittet som är då relaterade. Detta kallas för recomposition. [14] För att composition ska fungera så behöver man annotationen composable, vilket är fundamentalt byggblock för att konstruera och återanvända delar av användargränssnittet.[15] Figur 1 visar ett Hello-World-program på Jetpack Compose.

En bild som visar programvara, Multimedieprogram, Grafikprogramvara, Redigering

Automatiskt genererad beskrivning

**Figur 1. En simpel Jetpack Compose applikation skriven på Kotlin som visar Hello World.**

### **Kotlin**

Kotlin är ett statisk-typat programspråk utvecklat av JetBrains. Språket används till att utveckla klienter, servrar och mobila applikationer. Kotlin skapades som projekt år 2010 med syfte att skapa en modernare lösning till Java. Fördelar med att använda Kotlin är att den har null säkerhet, vilket innebär mindre risk att nullpointerexception kan dyka upp. Den kan underlätta skapandet av dataklasser. Den är samtidigt flexibel och möjliggör att kombinera objektorienterad programmering med  funktionell programmering. [16] [17]

### **Android Studio**

Android Studio är en utvecklingsmiljö skapad av JetBrains främst för utveckling av Android-applikationer. Den lanserade sitt första stabila version år 2014 och ersatte den tidigare Eclipse Android Development Tools som var då den primära utvecklingsmiljö för utveckling av Android-applikationer. Android Studio finns tillgängligt för nedladdning på Windows, MacOS och Linux. Det som finns i Android Studio är Gradle, vilket är byggsystem för utvecklingsmiljön,  Android-emulator och Github-integrering etc. [18]

### **iOS**

IOS är ett operativsystem som är utvecklat av Apple. Apple använder operativsystemet för sina egna produkter. Operativsystemet är skapat för att förenkla användargränssnitt och ser till att användaren enkelt navigerar på sina enheter för att åt funktioner och appar. Den viktigaste funktionen i iOS är integrationen med hårdvaran, som garanterar att iOS enheter fungerar väl med andra Apple produkter. En annan viktig faktor för Apple är säkerhetsfunktionerna. Apple har gjort detta till sin högsta prioritet i sina design av iOS och skapat funktioner som Touch samt Face ID för autentisering. Apple ser till att iOS enheter får regelbunden uppdatering för att hindra att känslig information läcker ut. [19]

### **Swift**

Swift är ett programmeringsspråk som lanserades av Apple år 2014 [20]. Språket är byggt med ett modernt tillvägagångssätt för att uppnå hög säkerhet och prestanda. Målet med Swift-projektet är att skapa bästa användningsområde för både mobila och stationära applikationer.

Den är skapad för att göra det enklare för utvecklare att skriva kod, göra ändringar och uppdatera systemet på ett smidigt sätt. Swift är designat för att skriva kod på ett säkert sätt. Detta inkluderar funktioner som skyddar mot vanliga säkerhetsproblem hos applikationerna.

Swift ersatte det tidigare programspråket Objective-C som då var rekommenderade språk för att utveckla iOS-applikationer. Swift har lånat många funktioner från Objective-C med en syntax som gör det lättare att läsa och underhålla. Swift måste vara jämförbar med Objective-C i prestanda för de flesta uppgifter. Prestanda ska vara förutsägbar och konsekvent.

Swift innehåller också funktioner som gör det lättare att läsa och skriva kod. Till exempel automatisk minneshantering, hitta misstag i koden samt att utvecklaren har mer kontroll över koden. Enligt Apple är Swift aldrig färdigt och de kommer att övervaka språkets utveckling, så att den hela tiden förbättras. [21]

### **SwiftUI**

SwiftUI är ett ramverk som används för att konstruera användargränssnitt på iOS-applikationer. SwiftUI använder en deklarativ syntax som gör det enkelt att specificera vad användargränssnittet ska innehålla och göra. Detta gör det enklare att förstå och underhålla koden. SwiftUI ger utvecklaren mer kontroller över applikationen och koden blir enklare att läsa vilket sparar mer tid vid underhåll. Den deklarativa stilen förenklar också komplexa koncept som animation, vilket gör det enklare att lägga till animationer och effekter i sina appar.

Figur 1 visar hur en deklarativ syntax ser ut med texten “hello world” och en bild. Deklarativa syntax används i vyer som är relaterade till användargränssnittet och avgör hur det ska  reagera på användarens interaktioner. [22]

**En bild som visar text, skärmbild, Teckensnitt, design

Automatiskt genererad beskrivning**

**Figur 1.  En simpel SwiftUI applikation som visar Hello World**

### **Xcode**

Xcode är en utvecklingsmiljö som är skapad av Apple och används av utvecklare för att skapa appar för Apple plattformar. Xcode hanterar hela utvecklingsprocessen från att skapa appar till att testa. Utvecklare kan använda en iOS simulator för att snabbt skapa prototyper och testa sina appar i en simulerad miljö när en riktig iOS enhet inte är tillgänglig. [23]

Den senaste versionen är Xcode 14, som innehåller allt som behövs för att utveckla, testa och distribuera appar för plattformar som tillhör Apple. Xcode 14 drar nytta av enkelheten och kraften hos Swift och SwiftUI med en ny app-upplevelse för flera plattformar, som gör att man kan koda snabbare med olika funktioner samt projekt byggs upp till 25 % snabbare. [24]

### **HTML5**

HTML5 är den senaste versionen av HTML-tekniken. HTML är det populäraste märkspråk vid utveckling av användargränssnittet. Själva förkortningen HTML står för HyperText Markup Language. Den tar fram strukturen för självaste webbsidan. Till skillnad från de tidigare versioner av HTML så har HTML5 stöd för bland annat multimedia-element som inbyggd ljud, video, animation etc. Men samtidigt även stöd för multitrådning med hjälp av Javascript. Kravet för att använda HTML5 är bland annat att man skriver längst uppe på koden taggen !DOCTYPE html. [25][26][27]

### **Javascript**

Javascript är ett programspråk som skapades av Brendan Eich år 1995 för Netscape Communications. [28] Javascript tillsammans med HTML och CSS utgör basen till web/front-end/klientutveckling. Javascript gör det möjligt att HTML-sidorna blir mer dynamiska och interaktiva. [29] Statistiken visar att 98,4 % av alla webbplatser använder Javascript som programmeringsspråk vid klientutveckling. Samtidigt så är Javascript det populäraste språket att lära sig utifrån alla programspråk som finns. [30]

### **Visual Studio Code**

Visual Studio Code är en enkel, snabb utvecklingsmiljö som används för att skriva kod. Den har support för hundratals olika språk. Visual Studio möjliggör att man blir mer produktiv bland annat med hjälp av syntax-highlighting, bracket-matching, code snippets etc. Visual Studio Code kan göra så att man kan navigera kod enklare med hjälp av intuitivt snabbkommando, customization etc. Visual Studio Code är installerbar på MacOS, Linux och Windows. [31]

## Återanvändning av modellkod

Att återanvända kod på ett annat språk på mobila applikationer är tekniskt möjligt. Oftast implementerar man kod på ett annat språk främst för att applikationen är prestandakrävande. Man behöver ett språk som kan köra nära processorn och exekvera instruktioner snabbt som exempelvis programspråket C. [32] Samtidigt kan det finnas flera andra skäl till att man vill återanvända kod på dessa plattformar, bland annat minimera tidsförluster vid utveckling av kod som utför samma funktionalitet [33].

För företag och organisationer så kan återanvändning av kod ha stor betydelse för att spara kostnader och energiresurser. För de flesta organisationer och företag så har man oftast mål som behöver uppnå minskning av koldioxidutsläpp för att minimera pågående klimatförändringar. Ett sätt att minska ner energikonsumtionen är att implementera det så kallas på engelska för green coding. Mer information om green coding finns på tidigare arbeten på avsnitt 2.2.

Det finns flera typer på hur man kan återanvända kod. De två vanligaste är alltså interna och externa återanvändning av kod. Intern återanvändning av kod innebär att man använder återanvänder kod på andra platser i samma projekt. Den andra är externa återanvändning av kod, vilket innebär att man placerar befintlig kod i ett annat projekt. [34]

Det man behöver tänka på vid återanvändning av kod är att den är säker, trovärdig, effektiv och kan underhållas. Återanvändning av kod kan låta simpelt men det visar sig vara svårare än det man tror. Man behöver ta hänsyn till flera utmaningar som kan förekomma vid återanvändning av kod, bland annat kan det ta längre tid att integrera den kod som ska återanvändas med själva projektet [35][36]. Det kan bero på att miljön som applikationer körs är inte anpassat för det okända programspråket eller kod som man vill implementera på applikationen eller systemet. Dessutom så behöver man även se till att självaste kod-modulerna som ska återanvändas på ett annat system har inte några buggar eller fel. Det krävs noggrannhet och omtänksam planering med arbete av återanvändning av kod. [37]

## Verktyg för att binda återanvändbar modellkod skriven i C

Det finns flera olika verktyg som kan binda återanvända modellkod med okänt programspråk med den kod som kan exekveras på miljön. Android rekommenderar att man använder Kotlin först för att utveckla Android-applikationer främst för att man behöver ett programspråk som kan hjälpa skriva mer säkrare kod samt möjlighet att integrera med existerande applikationer och ramverk. [38] Det finns möjlighet att använda C - och C++ kod med hjälp av Android Native Development Kit(NDK). NDK används till att kompilera C- eller C++ kod till ett native bibliotek som då paketeras till Android Package Kit (APK) genom Gradle, vilket är utvecklingsmiljöns byggsystem. Därefter kan Java eller Kotlin anropa C - eller C++ kod med hjälp av Java Native Interface (JNI). Denna interface förklaras mer på avsnitt 2.5.1. Detta verktyg är användbart om applikationen behöver mer prestanda eller om den har behov att återanvända C- eller C++ bibliotek. [39]

Man kan även tillägga Javascript- och Webassembly kod på Android genom med hjälp av JavascriptEngine, vilket är ett bibliotek som kan utvärdera Javascript-kod. En nackdel är att den inte är stabil och ligger sedan april 2023 på alpha-stadiet. [40]  Med Swift så är det möjligt att använda återanvända C-kod med hjälp av interoperabilitet. Detta förklaras på avsnitt 2.9.2. Men dessutom så är det också möjligt att återanvända Webassembly-kod med hjälp av SwiftWasm, vilket är ett LLVM-verktyg som används för att generera wasm-kod på Swift. [41]

På HTML5 applikation så är Webassembly enda möjlighet att binda existerande kod med okänt programspråk, vilket förklaras på avsnitt 2.9.4.1. Tillsammans med kompilatorn Emscripten så kan man generera wasm-kod från kompilerad C- och C++ kod, vilket förklaras mer på avsnitt 2.9.3.

### Java Native Interface (JNI)

Java Native Interface är en native programming interface som gör det möjligt att skapa interoperabilitet mellan språken Java och C/C++ på Java Virtual Machine (JVM). De flesta utvecklare använder JNI för att hantera situationer där Java kan inte skriva helt eller användas på applikationen. Det finns flera möjligheter till varför man använder JNI. En orsak kan vara att man vill använda C- eller C++ bibliotek på Java-applikationen. Med hjälp av JNI så kan man göra nativa biblioteket nåbar på Java-kod. Dessutom så kan JNI användas som stöd för att återanvända kod som kan förbättra prestandan på applikationen.

En bild som visar text, skärmbild, diagram, linje

Automatiskt genererad beskrivning

Figur 1: Är en illustration på hur man kan kommunicera mellan C- C++ kod med Java/Kotlin kod med hjälp av JNI

Java Native Interface (JNI) använder datatyper som finns i Java och kräver en specifik prefix notering. Exempelvis om man ska använda datatypen integer på Java Native Interface så motsvarar det datatypen jint. Mer information om motsvarande datatyper för Kotlin, Java och JNI finns på figur 2. Vid användning av JNI så finns det fall som man behöver hantera, bland annat vid överföring av strängar mellan Java och C strängar.

Man behöver använda speciella metoder för att hantera konvertering av String mellan dessa programspråk. GetStringUTFChars är en metod som då konverterar Java-sträng till C-sträng, i detta fall motsvarar sträng på C en char array. Om man vill överföra en C-sträng till Java-kod, så kan man använda funktionen NewStringUTF.

En bild som visar text, skärmbild, nummer, meny

Automatiskt genererad beskrivning

Figur 2: Är en tabell som visar motsvarande datatyper för Kotlin, Java och Java Native Interface

Om man behöver överföra objekt mellan Java och C/C++ så behöver man använda datatypen jobject för att arbeta med motsvarande Java objekt i JNI.  Om man behöver hitta en specifik klass så använder man FindClass. Vid behov av att skapa ett objekt på JNI så använder man AllocObject eller NewObject. Om har behov av att komma åt fält och metoder på en Java-klass så kan man göra det med motsvarande GetFieldID och GetMethodID funktioner. Arrayer som lagrar int-värde motsvarar då på JNI datatypen jintarray. Om man behöver lagra en lista med strängar då använder man JNIs jobjectArray. JNIEXPORT används i C/C++ för att deklarera en funktion som exporteras via JNI och det gör att man kan hitta funktionen från Java-Kod. JNICALL används tillsammans med JNIEXPORT där C/C++ kod blir tillgängliga för JVM så att man kan anropa det från Java kod. [42] För att underlätta läsaren så finns det två figurer som illustrerar och förklarar hur man kan anropa C-kod med hjälp JNI. Detta motsvarar då figur 3 och 4.

En bild som visar text, skärmbild, programvara

Automatiskt genererad beskrivning

Figur 3: Visar hur man kan anropa C-kod på Kotlin med hjälp av JNI

En bild som visar text, skärmbild, programvara, Teckensnitt

Automatiskt genererad beskrivning

Figur 4: Visar hur man kan skicka input men även JNI-konfigurering som gör att man kan anropa tillbaka variabler och objekt som motsvarar på Kotlin/Java

### **C interoperabilitet med Swift**

Det är möjligt att använda C-kod på Swift. Swift stöder det så kallas för interoperabilitet, vilket gör det möjligt att kommunicera mellan Swift och C utan större bekymmer. Det ger större möjligheter till att använda C- kod på Swift till syfte att återanvända gamla legacy-bibliotek eller kod som kan förbättra prestanda på applikationen.

En bild som visar text, skärmbild, Teckensnitt

Automatiskt genererad beskrivning En bild som visar text, skärmbild, Teckensnitt, programvara

Automatiskt genererad beskrivning

 Figur 1: h fil med C funktioner                                    Figur 2: C Kod med logik

I ett projekt kan man importera C-filer till Swift-koden genom att använda något som kallas för en bridging header. När man skapar en C-fil i en existerande Swift-app kommer Xcode att be om att skapa en header-fil till C-filen och en bridging header-fil. När bridging header-filen har tillgång till publika C-header-filen blir de synliga i Swift-appen. Alla deklarationer blir automatiskt tillgängliga i hela projektet, vilket innebär att man kan använda klasser och andra deklarationer från C-koden i Swift. [43]

En bild som visar text, skärmbild, Teckensnitt, programvara

Automatiskt genererad beskrivning En bild som visar text, skärmbild, Teckensnitt

Automatiskt genererad beskrivning

 Figur 3: Bridging fil i projekten Figur 4. Hur h fil används i bridging fil

### Webassembly

Webassembly(WASM) är ett assemblerspråk som lanserades 2016 till syfte att förbättra bland annat prestandan på dessa traditionella Javascript-applikationer. Målet med webassembly är att den ska vara snabb, effektiv, säker, läsbar, portabel och möjlighet till felsökning. Webassembly är inte primärt gjort för att skriva kod för hand utan den är designat för att vara ett effektivt kompilationsmål för högnivåspråk som bland annat C, C++, Rust etc. Det finns flera kompilatorer som kan skapa wasm-kod. Den populäraste kompilatorn är emscripten som kan kompilera och konvertera C- eller C++ kod till wasm-kod. [44]

Emscripten är en kompilator som används för att kompilera C, C++-kod till Wasm-kod. Det används allt från högpresterande grafiska spel till ljud och filer. Det finns flera sätt att anropa C-funktioner på Emscripten men det enklaste är att anropa antingen ccall eller cwrap funktioner.

En bild som visar text, skärmbild, Teckensnitt, Rektangel

Automatiskt genererad beskrivning

Figur 1: Är en illustration som beskriver hur C och C++ kod kan med hjälp av emscripten omvandla C eller C++ kod till wasm-kod som kan därefter användas på Javascript eller HTML5-fil

Funktionen ccall anropar funktioner skrivet på C eller C++ med specifika parametrar och returnerar resultatet. Funktionen cwrap returnerar en JavaScript funktion som motsvarar på C. Funktionen cwrap är bra när man behöver anropa en funktion ett antal gånger [45]. Figurerna 2, 3 och 4 demonstrerar hur man kan anropa C-kod med hjälp av emscripten.

En bild som visar programvara, Multimedieprogram, text, Grafikprogramvara

Automatiskt genererad beskrivning

Figur 2: Demonstrerar en illustration som inhämtar värdet från JS - glue code med hjälp av  och displayer värdet på användargränssnittet

En bild som visar programvara, Multimedieprogram, text, Datorikon

Automatiskt genererad beskrivning

Figur 3:  Är en abstrakt/bridge-fil som hanterar all konfigurering och nödvändigheter för att kunna kompilera och köra C-kod på HTML5 applikation

En bild som visar programvara, Multimedieprogram, text, Grafikprogramvara

Automatiskt genererad beskrivning

Figur 4: Visar en ren C funktion som inhämtar sträng data och returnerar datan tillbaka

# Metod

Detta kapitel diskuterar vi vilka metoder som har valts för att undersöka återanvändning av kod på det här projektet samt vilka resultat som har fått utifrån utveckling och tester som har utförts.

## Vetenskaplig metod

Detta arbete har till syfte att undersöka om möjligheten till överföring av modellkod skriven i C på HTML5, Android, iOS. Arbetet började med litteraturstudier om dessa plattformar och ramverk. Informationen som har inhämtats på denna arbetet kommer mestadels från officiella hemsidor exempelvis (Android, iOS, Mozilla) men även ett antal studier som har kopplingar till återanvändning av kod har lagts in på rapporten. Därefter går arbetet på djupare undersökning på varför återanvändning av kod kan vara intressant. Efter att vi har fått en inblick på vad och varför är återanvändning av kod intressant så studeras möjliga verktyg som kan binda återanvändning av kod skriven på C på dessa plattformar.

Det finns få akademiska studier som motsvarar arbetet med att återanvända kod med C programspråk på mobila applikationer. Dessa söktermer ‘Code reuse in mobile application’, ‘Code reuse in Android’,’ Code reuse in iOS’, ‘Code reuse in HTML5’, ‘Sustainable code’, ‘mobil prestanda’ har använts för att hitta relevant information. Dessa rapporter och studier som används har utvärderats utifrån att kolla på titeln och sammanfattningen. Dessa rapporter och information har hittats med hjälp av KTH Primo, Google Scholar, Diva-Portal och Google.

Valet av verktyg, programspråk och applikationer som ska användas på dessa plattformar, har diskuterats med handledaren. Det som diskuterades är att programspråket C ska användas för att skriva och återanvända modellkod till mobila applikationer. Programmeringsspråket C är inspiration till flera moderna språk och många hårdvaror har support och kompabilitet att kompilera C-kod vilket gör det möjligt att skriva C-kod från exempelvis TV till bilar.

På Android-delen kommer NDK och JNI väljas för att binda C- och Kotlin-kod, detta förklaras mer på avsnittet 2.7.1. Det går även att binda kod med hjälp av WebAssembly på Android men problemet är att den inte är stabil.

På iOS så har Swift naturligt stöd för C genom interoperabilitet, vilket innebär att man kan anropa C funktioner utan större problem.

På HTML5 så har Webassembly valts eftersom det är enda valet man kan kombinera ett okänt språk som C med Javascript och HTML5. Mer om webassembly förklaras mer på avsnitt 2.7.3.

Under utvecklingsfasen kommer en väderapp att byggas, där modellkoden är skriven i C på Android, iOS och HTML5. Anledningen till att välja väderapp som prototyp är att den anses vara en lämplig applikation som inte är för enkel eller alltför komplex att bygga. Samtidigt har författarna erfarenhet av att bygga applikationer som motsvarar väderapp från tidigare kurser. Dessutom ger det möjlighet att testa hur komplex det är att använda C-modellkod på dessa plattformar samt hur mycket data klara applikationen av. Mer information om dessa applikationer finns på nästa avsnitt 3.2.

Efter att applikationerna har utvecklats så är det dags för analysering och testning. Det som ska analyseras och testas är bland annat prestanda och kod storlek/komplexitet. Anledningen till varför prestanda har valts är för att undersöka kapaciteten på en applikation som använder C som modellkod. Vidare kommer analysering av kodens komplexitet och storlek att utföras för att bestämma om det var nödvändigt att lägga till eller minska kod under utvecklingsfasen. Mer information om prestandaanalys och kodstorlek finns i avsnitt 3.3.

## Prototyp

Prototypen som kommer att byggas i utvecklingsfasen är en väderapp. Orsaken till varför prototypen är vald nämns i slutet på avsnitt 3.1. Väderappen innehåller koordinater(longitud, latitud) och en lista för prognosdata. Inuti en prognoslista så innehåller det 10 dagars prognosdata som visar varje timme och dag för hur prognosen ser ut på platsen. Dessa data hämtas från SMHIs API. Användare får ange koordinaterna(latitud, longitud) på applikationerna, men dessa koordinater som anges på väderapplikationen eller API är begränsade inom Sverige. Om man anger fel eller koordinater som inte existerar, då visas ett felmeddelande. Felmeddelande visas även om applikationen/enheten inte har någon uppkoppling till Internet. Syftet med denna prototyp är att utforska möjligheten att återanvända samma C-modellkod på dessa plattformar och undersöka om det ger konsekvenser vid återanvändning av kod.

## Utvärdering av prototyp

### **Kodstorlek**

På tidigare arbeten så har  Y. Cheon et.al. analyserat återanvändning av kod genom att jämföra på antalet rader kod/klasser som behövdes tilläggas eller minimeras på både Android och Java Desktop applikation. Genom att följa denna tidigare arbeten så får man reda på kodkomplexiteten och kodstorleken och det ger möjligheten att dra slutsats om det var full möjligt att återanvända samma C-modellkod på HTML5, Android, iOS eller behövde koden tilläggning.

### **Prestanda**

Det är viktigt att ta hänsyn till prestanda på mobila applikationer. Användaren förväntar sig att en applikation ska vara snabb och responsiv utan att tappa kvalitet. Syftet med undersökningen är bland annat undersöka hur mycket kapacitet klarar applikationer som har modellkoden skriven i C. Dessa prestandatester kommer att utföras genom att lägga till 100, 1000 och 10000 prognosdata på dessa applikationer för att mäta hur mycket data kan applikationen klara av.

# Resultat

Detta avsnitt går igenom hur vi har implementerat applikationer samt hur dessa applikationer testades.

## Implementering av applikationer

Denna sektion presenterar implementeringen av en väderapplikation som använder modellkod för att hantera väderdata i både C och den rekommenderade språket. Den visar hur befintliga C-koden används och kan användas till de olika plattformarna som är iOS, Android och HTML5. Men samtidigt förklarar vi även hur vanliga applikationer med rekommenderade språk används på iOS, Android och HTML5 för att kunna utföra tester, analyser och dra slutsatsen utifrån frågeställningar på avsnitt 1.2.

### **Användargränsnittet**

Alla plattformar utför samma funktionalitet när de hanterar väderapplikation, oavsett om applikationen använder C-kod eller inte. Applikationen för specifika plattformar har samma utseende, tex Android applikationen har samma användargränssnitt, oavsett om det används C-kod eller inte. När applikationen startas erbjuds användaren att ange värden för longitude och latitude för att hämta väderdata. Vid en lyckad sökning visas 10 dygnsprognos för det angivna området. Om sökningen misslyckas visas en varning som uppmanar användaren att ange giltiga värden. När användaren återvänder till applikationen visas det senaste som söktes på applikationen. Om användaren matar in felaktiga värden eller anger en destination som inte existerar, kommer en varning att visas som uppmanar användaren att genomföra en ny sökning.

Det är vanligt att plattformar som Android och iOS rekommenderar vilka språk och ramverk man ska använda för att bygga mobila applikationer. Dessa applikationer är strukturerade där användargränssnittet och logiken hålls separerade. Det som användes på alla applikationer är MVVM (Model-View-ViewModel). Orsaken till att MVVM används är för att få mer struktur och renare kod på projektet. Figur 12 visar hur ViewModel används för att hålla kommunikationen mellan vyn och självaste modellkoden skriven i C.

En bild som visar skärmbild, programvara, Datorikon, Multimedieprogram

Automatiskt genererad beskrivning

Figur 1: Visar användargränssnittet på alla mobila applikationer.

### **Modellkod**

Modellkod för denna väderapplikation är skriven i C och beskriver logiken som bygger på SMHI 10 dygnsprognos. På figur 1 och 2 så kan man se mer i detalj hur modellkoden har implementerats och använts på Android, iOS och HTML5 oföränderligt. Modellkoden består av två strukturer, vilket är väderdata och timdata. Väderdata innehåller godkänd tid samt en list över timdata, vilken hanterar datum och tid, temperaturen och väder symbol. Förutom det så innehåller det metoder som gör det möjligt att hämta, lägga och skapa struktur från modellkoden.

En bild som visar text, skärmbild

Automatiskt genererad beskrivning

Figur 2: Visar alla definition av datatyper och deklareringar på modellkod i en headerfil skriven i C

**En bild som visar text, skärmbild, Teckensnitt

Automatiskt genererad beskrivning**

Figur 3: Visar fortsättning på figur 1

En bild som visar text, skärmbild

Automatiskt genererad beskrivning

Figur 4: Visar detaljer på hur implementering av modellkod kan ske på C

En bild som visar text, skärmbild

Automatiskt genererad beskrivning

Figur 5: Fortsättning av figur 4

### **Bindning av modellkod skriven i Swift**

När man integrerar C-kod i iOS-projektet genereras en “bridging-fil"  som faciliterar kopplingen mellan Swift och C. Denna här filen möjliggör att komma åt C-koden  direkt, utan att behöva förlita på externa ramverk. För att hantera Hourly Weather data från C-kod, skapas en Swift-struktur med hjälp av UnsafeMutablePointer, som ger direkt åtkomst till timdata i C-koden. Data modellerna är i en wrapper-kod för att koppla ihop C-modellkoden med Swift och gör att det är enklare att arbeta med. Genom wrapper-koden kan man läsa och ändra värden samt hantera det på ett effektivt sätt. Figur 6 illustrerar hur man kan använda funktionerna i C-modellkoden med wrapper samt exempel på funktionerna setTime och getTime hämtar och sätter värden för tiden med hjälp av de relevanta C-funktionerna.

En bild som visar text, skärmbild, Teckensnitt

Automatiskt genererad beskrivning

Figur 6: Visar Swift-kod som motsvarar i C

### **Bindning av modellkod skriven i Android**

Implementeringen av väder applikationen involverar användning av Java Native Interface (JNI) och den nativa kodmodulen som är WeatherNative, som binder metoder som finns i C med hjälp av JNI. WeatherNative hanterar vår C-modellkod och möjliggör synlighet av data i Android-applikationen. Koden utvecklades med hjälp av Android Studio och NDK för att underlätta integrationen av den nativa kodmodulen. Funktionen system.loadLibrary(“JniBridge”) är nödvändigt för att kunna ladda en specifik JNI bibliotek i Android-applikationen. Denna funktion möjliggör användningen av extern C kod applikationen genom JNI, vilket ger åtkomst till det som är definierat i C-koden. I figur 8 visar WeatherNative där den använder sig av JNI bibliotek och extern från JNI.

I figur 9 så tillhandahåller den nativa kodmodulen viktiga funktioner för att extrahera och manipulera väderdata, inklusive att hämta och ställa in godkänd tid. Efter att ha implementerat den nativa kodmodulen, med användning av 'external' på alla JNI-funktioner, kan man få tillgång till exempelvis getApprovedTime och setApprovedTime. För att få åtkomst till WeatherNative-objekten används jClass genom funktionen GetObjectClass. Genom användning av jfieldID kan vi komma åt matchande namn på fält i WeatherNative, som visas i Figur 9 där approvedTime hämtas genom GetFieldID och söker efter fältet i WeatherNative-klassen och kräver att det är en sträng. För att hämta approvedTime-strängen används GetObjectField, som är kopplad till GetFieldID i WeatherNative. SetObjectField används för att ändra värden i objektet genom att ta emot objektet och de fält som ska ändras, tillsammans med de nya värdena. Datatyperna i WeatherNative används inom JNI för att göra data synliga i Android-applikationen när externa funktioner används. Genom att använda dessa funktioner kan vi effektivt extrahera och manipulera godkänd tid i Android-applikationen. Det är viktigt att funktionerna matchar både i JNI och WeatherNative för att möjliggöra korrekt kommunikation mellan de två.

En bild som visar text, skärmbild, programvara, Teckensnitt

Automatiskt genererad beskrivning

Figur 8: Visar WeatherNative, som gör att funktionerna från C-modellkod blir synliga i Kotlin och kan användas i Android-applikationen

**En bild som visar text, skärmbild, Teckensnitt

Automatiskt genererad beskrivning**

Figur 9: Visar hur man sätter och lägger tidsstämpel på Java Native Interface

### **Bindning av modellkod på HTML5**

För att integrera C koden i Javascript och Wasm (Webassemlby) miljön används emcc kommandot för att kompilera C koden till en Javascript/Wasm-fil. Genom att använda EMSCRIPTEN\_KEEPALIVE kan man säkerställa att det deklareras C funktionerna exporteras och blir synliga för JavaScript efter att man har skrivit in emcc kommandon.  Den process möjliggör att man kommer åt C-kodens funktioner och datastrukturer direkt från JavaScript sidan. Figur 11 visar hur man kan hämta och ändra värdet på tidsstämpel med C funktionen, som gör det möjligt att manipulera och använda C kodens funktionalitet inom Javascript miljön. För att ändra värdet på longitude i Weather objekten som man använder i applikationen så används funktionen setLon. Denna funktion namn måste matcha den motsvarande C funktionen och tar två argument som är weatherObj som representerar Weather objektet i C koden och lon som innehåller det nya värdet för longitude.

En bild som visar text, skärmbild, programvara

Automatiskt genererad beskrivning

Figur 10: Visar exempel hur header-fil ser ut på C i modellen

En bild som visar skärmbild, text

Automatiskt genererad beskrivning

Figur 11: Visar hur man kan anropa C-funktionen genom JS glue code som automatisk genereras när Emscripten kompilerar C-kod

## Kodstorlek

För att dra slutsatser om det var full möjligt att återanvända samma C-modellkod på HTML5, Android, iOS utan att tillägga någon kod så behövs det undersökning av kodstorlek på applikationerna. Beräkning av kodstorlek görs genom att beräkna hur många antal rader av kod som författarna har skrivit själv för att avgöra om det har funnits behov av att tillägga extra kod på applikationen för att den ska fungera. Resultatet visade sig att det fanns behov av att lägga till kod på alla plattformar.  Hur mycket kod som skiljer mellan varje plattform varierar från olika plattform om man utgår från tabellerna 1, 2, 3. Orsaken är att det går inte binda direkt C kod med exempelvis Javascript eller Kotlin utan man måste använda verktyg som JNI eller WebAssembly för att manuellt binda mellan dessa språken samt behövde man lägga också fokus för manuell hantering av minne av vilka objekten.

Tabell 1 visar jämförelse på antal rader av kod på hela applikationen som författarna har skrivit själv på en vanlig Android-app som har en modell kod skriven i Kotlin jämfört med modellkod skriven i C.

Tabell 1: Jämförelse antal rader av kod på hela applikationen som författarna har skrivit själv på en vanlig Android-app som har en modell kod skriven i Kotlin jämfört med modellkod skriven i C

|  |  |
| --- | --- |
| Android Applikation | Hur många rader av kod |
| Modellkod skriven i Kotlin | 759 |
| Modellkod skriven i C | 1230 |

Tabell 2: Jämförelse på antal rader av kod på hela applikationen som författarna har skrivit själv på en vanlig iOS-app som har en modell kod skriven i Swift jämfört med modellkod skriven i C

|  |  |
| --- | --- |
| IOS Applikation | Hur många rader av kod |
| Modellkod skriven i Swift | 826 |
| Modellkod skriven i C | 1019 |

Tabell 3:  Jämförelse antal rader av kod på hela applikationen som författarna har skrivit själv på en vanlig HTML5-app som har en modell kod skriven i JS  jämfört med modellkod skriven i C

|  |  |
| --- | --- |
| HTML Applikation | Hur många rader av kod |
| Modellkod skriven i JS | 310 |
| Modellkod skriven i C | 493 men inkluderar inte JS lim-kod och Wasm-kod eftersom det var autogenererat av Emscripten. |

## Prestanda

Prestandatester är viktiga för att undersöka om applikationen är stabil och trovärdigt. I denna testning kommer vi att utgå från C. Fournier et al. formel för att beräkna CPU-beräkning.  Tabellerna nedan visar resultatet av CPU-beräkningar med hjälp av C.Fournier-formeln i olika scenarier för de olika plattformarna. De ger en jämförelse mellan plattformar som använder C-modellkod och det rekommenderade programmeringsspråket.

Tabell 1: Dummy data med C modell kod

|  |  |  |  |
| --- | --- | --- | --- |
| **Dummy i C** | **iOS** | **Android** | **HTML5** |
| 100 | 5,3 % | 6,3 % | 4,5 % |
| 1000 | 29,3 % | 32,6 % | 24,1 % |
| 10000 | 91,2 % | 92,3 % | 87,2 % |

Tabell 2: Dummy data utan C modell kod

|  |  |  |  |
| --- | --- | --- | --- |
| **Dummy utan C** | **iOS** | **Android** | **HTML5** |
| 100 | 4,8 % | 4,1 % | 3,5 % |
| 1000 | 25,9 % | 28,2 % | 21,1 % |
| 10000 | 86,2 % | 88,7 % | 79,6 % |

Tabell 3 API data

|  |  |  |  |
| --- | --- | --- | --- |
| **API** | **iOS** | **Android** | **HTML5** |
| C | 3,8 % | 5,3 % | 5,5 % |
| Utan C | 3,3 % | 4,2 % | 4,8 % |

# Analys och diskussion

Detta avsnitt analyserar och diskuterar vad för resultat har vi fått från utvecklingen och testerna. Det som har undersökts är kodstorleken på applikationer för att se hur mycket kod skiljer det mellan en vanlig applikation och en applikation som använder modellkod skriven i C. Dessutom så undersöks även prestandan mellan applikationer för att ta reda på om det tar mer eller mindre prestanda om man har en applikation som har modellkod skriven i C jämfört med en applikation som har modellkod skriven i den rekommenderade språket som används på HTML5, Android och iOS.

## Utveckling av applikationer

Mycket av den tid som användes under projekten var att lära sig och söka fakta om dessa verktyg som Java Native Interface (JNI), WebAssembly och interoperabilitet på Swift, eftersom att författarna inte hade någon erfarenhet att om dessa verktyg för att återanvänder modellkod sedan tidigare.

En utmaning för oss författarna var att hitta information om hur man kan använda C-kod i Android och HTML5. På iOS fanns det tillgängligt material som vi kunde lära oss av. Det gjorde vårt arbete lättare då Swift har naturligt stöd för programspråket på C. Den svåraste utmaningen var Android, där koden är skriven i Java eller Kotlin. För att använda C-kod i Android behövde vi att lära oss om JNI (Java Native Interface), och det var svårt att hitta tillräckligt med information om JNI för Android. Vi var tvungna att läsa dokumentationen och experimentera mycket, eftersom det var svårt att hitta och lösa fel. När vi stötte på fel var det svårt att identifiera orsaken och det fanns begränsad information om JNI och felmeddelanden, vilket gjorde arbetet svårare. Det krävde noggranna tester av applikationen för att hitta och åtgärda problemen. För att implementera C-kod i HTML5 använde vi WebAssembly, vilket är en relativt ny teknik. Även här fanns det begränsat med information och det var nytt för oss att använda det.

En annan utmaning som var svår vid utvecklingen av applikationerna med C-modellkod var att hantera minnet korrekt för att undvika onödig belastning på systemet. Det var viktigt att se till att minnet rensades ordentligt efter att data har lagrats, särskilt när vi hanterade listor i C-kod. Jämfört med rekommenderade språk var det enklare att implementera i de språk där man inte behövde bekymra sig om att hämta kod från andra platser eller manuell hantering av minnet, vilket sparade tid.

Det fungerar att använda äldre C-modellkod för olika plattformar, men vi anser att det inte är värt det på grund av den ökade komplexiteten och behovet av att ha djupare kunskap inom områden som JNI och WebAssembly. Det var mycket smidigare och enklare att använda de rekommenderade språken eftersom det fanns gott om information tillgänglig och det var mer direkt att börja koda utan att behöva genomgå flera steg för att få applikationen att fungera. Att använda en äldre C-modellkod till plattformarna kräver mer tid och energi, vilket kan förlänga arbetsprocessen. Även studier och andra källor påpekar att återanvändning av kod vara utmanande på olika projekt. Vissa projekt kan det vara omöjligt att använda återanvändning av kod eftersom projektet är för stort och komplex. Samtidigt så visar det sig även från externa källor att utvecklare föredrar att bygga applikationen från början. Orsaken kan vara brist på intresse och kunskap på återanvändning av kod.

## Diskussion om resultaten från undersökningen av kodstorlek och prestandatester

Det som har kommit fram från undersökningen på kodstorleken är att det förekom 471 rader mer kod om man hade modellkod skriven i C på Android-applikation jämfört med om man hade Android-applikation skriven i Kotlin. På iOS var det mindre kod som behövdes tilläggas eftersom Swift har en naturlig support för C. Det som resulterade var att ca 193 rader mer kod förekom på iOS app med modellkod skriven i C jämfört med iOS app skriven i ren Swift kod. HTML5 förekom cirka 180 rader mer kod på modellkod skriven i C jämfört med en ren HTML5 applikation med Javascript.

Utifrån undersökningen på kodstorleken och utvecklingen så har det kommit fram att oavsett vilken applikation man arbetar med så fanns det behov för att lägga till extra-kod om modellkoden var skriven på C jämfört med en applikation som hade modellkoden skriven i rekommenderad språk eftersom vissa plattformar som Android och HTML5 applikation så behövdes omvandlingen mellan C och Kotlin hanteras manuellt inklusivt minneshantering. Vilket gjordes så att mycket av det tid spenderades på att fixa en fungerande applikation som kan felfritt skicka data mellan C och Kotlin/Swift och Javascript kod med en fungerande minneshantering.

Prestandamätningar varierar beroende på faktorer som typen av applikation, vilken plattform som användes och om det fanns andra körande program i bakgrunden. För att mäta prestanda använde vi en formel från tidigare studier och använde plattformens inbyggda verktyg för att övervaka CPU-användning. Ett av verktygen Android Studio profiler CPU-användningen förändras baserat på våra inmatningar och data hämtningar. På liknande sätt erbjöd Xcode liknande verktyg för att övervaka CPU-användning. Inom HTML5 programmerade vi in egna CPU-beräkningar för att utvärdera prestandan. När det gäller prestanda var det viktigt att ta hänsyn till operativsystemets version och att våra tester utfördes via simulatorer. Resultatet skulle troligen vara annorlunda på en riktigt enhet beroende på hur mycket CPU arbetar och belastade. Vi genomförde olika CPU-tester genom att lagra 100, 1000 och 10000 väderdata för att se hur det påverkar prestandan. Enligt Camille Fourniers formel visade det sig att användningen av C inte skilde sig markant mellan de olika plattformarna. En annan faktor som påverkar prestandan är vilken typ av plattform som används och om den har den senaste operativsystemversionen och moderna datorverktyg, vilket också påverkar prestandan.

## Hållbar utveckling och kostnader

I detta projekt användes programspråket C för att utveckla modellkoden. Detta val av språk kan bidra till minskade koldioxidutsläpp och minskad processorkraft genom gröna kodningsmetoder då den är snabb när det gäller prestanda. Emellertid kräver detta en noggrann hantering av kodbindningen för att kunna återanvända koden på ett korrekt och tydligt sätt, särskilt när man försöker binda C-modellkod med de rekommenderade språken för mobila plattformar. Det finns risker i samband med detta, såsom manuell hantering av minnesresurser, vilket kräver en stabil förståelse inom detta område. Om hanteringen inte utförs korrekt kan det leda till ökad processorkraft och minnesanvändning, vilket kan orsaka bland annat att batteriet dräneras, vilket innebär att användare behöver konsumera mer energi.

Ur ett ekonomiskt perspektiv kan återanvändning av kod vara fördelaktigt om man har utvecklare med rätt kompetens och erfarenhet. Det är viktigt att nämna att dessa verktyg och själva kod överföringen är komplexa i sig. Om kompetens och erfarenhet saknas kan det medföra ökad risk för buggar och försämrad säkerhet inom applikationen. Detta kan i sin tur leda till att företagets kunder och användare väljer att sluta använda applikationen, vilket kan resultera i förlust för företaget.

# Slutsatser

Målet med det här arbetet är att undersöka möjligheten att återanvända kod på Android, iOS och HTML5. Det har visat sig att det är möjligt att använda C modellkod för plattformarna men det kräver en noggrann planering och omfattande arbete eftersom verktygen som används för att binda C och native applikationer är komplicerade och tidskrävande. Det kan behövas tillägga extra kod på applikationen för att kunna hantera bindningen korrekt. Med rätt kompetens och erfarenhet kan processen för återanvändning av kod hanteras mer noggrannare och utförligt. Om inte kan det bli ökad risk för bland annat mer energikonsumtion, mjukvarubuggar, försämrad säkerhet som kan orsaka då förluster utifrån ett ekonomiskt perspektiv. Fortsatta arbete kan vara kring säkerhetsaspekter vid återanvändning av C-modellkod. Det kan vara viktigt att analysera och utvärdera potentiella säkerhetsrisker och sårbarheter som kan uppstå vid återanvändning av C-modellkod och utveckla lämpliga säkerhetsåtgärder och bästa praxis för att minimera dessa risker.

# Källförteckning

[1] TIOBE, TIOBE Index for April 2023 [Online]

Available: <https://www.tiobe.com/tiobe-index/> (hämtad 2023-04-17)

[2] Gorbachenko Pavel, Everything You Need to Know About Legacy Code [Online]

Available: <https://enkonix.com/blog/legacy-code/>  (hämtad 2023-04-17)

[3]  Strizic Marko, Top factors influencing the cost of native app development [Online]

Available: <https://decode.agency/article/native-app-development-cost/>

(hämtad 2023-04-17)

[4] Cheon Y, Carlos V. Chavez and  Castro U, Code Reuse between Java and Android Applications. The University of Texas at El Paso, El Paso, Texas, U.S.A; 2019 [Online]. Available: <https://www.scitepress.org/Papers/2019/78437/78437.pdf>

(hämtad 2023-03-23).

[5] Olsson M, A Comparison of Performance and Looks Between Flutter and Native Applications, Karlskorna: Blekinge Institute of Technology, 2020 jun 13

[6] Fournier Camille, Comparison of Smoothness in Progressive Web Apps and Mobile Applications on Android, Stockholm: KTH, 2020

[7] Malki A, Hybrid applikationer med Apache Cordova och Flutter, Karlstad:  Karlstad Universitet, 2021 VT.

[8] Herelius S, Green coding, Karlskrona:  Blekinge Institute of Technology, 2022 juni 10.

[9]Thompson B , C Programming Language Tutorial, 2023 mars 04 [Online][updated: 2023-03-04].

Available: <https://www.guru99.com/c-programming-language.html>

(hämtad: 2023-03-25)

[10] Google Developer Training, Android Developer Fundamentals Course Concepts: Unit 1 - Get started. Lesson 1: Build your first app. Introduction to Android, 2018 Sept [Online][updated:  2018-09].

Available:<https://google-developer-training.github.io/android-developer-fundamentals-course-concepts-v2/unit-1-get-started/lesson-1-build-your-first-app/1-0-c-introduction-to-android/1-0-c-introduction-to-android.html>  (hämtad: 2023-03-26)

[11]Android, Vad är Android [Online].

 Available: <https://www.android.com/what-is-android/> (hämtad: 2023-03-26)

[12] Google Android Developers, Jetpack Compose is now 1.0: announcing Android’s modern toolkit for building native UI, 2018 Juli 28 [Online].

Available:<https://android-developers.googleblog.com/2021/07/jetpack-compose-announcement.html> (hämtad: 2023-03-26)

[13] Jetbrains, Compose Multiplatform [Online].

Available: <https://www.jetbrains.com/lp/compose-mpp/> (hämtad 2023-03-28)

[14]  Google Android Developers, Why adopt Compose, 2023 Mars 01 [Online].

Available:<https://developer.android.com/jetpack/compose/why-adopt>

(hämtad 2023-03-28)

[15] Google Android Developers, Thinking in Compose, 2023 April 01 [Online][updated:  2023-03-01].

Available:<https://developer.android.com/jetpack/compose/mental-model>

(hämtad 2023-03-28)

[16] Kotlin, Kotlin Evolution, 2023 April 11 [Online].

Available:<https://kotlinlang.org/docs/kotlin-evolution.html#principles-of-pragmatic-evolution> (hämtad 2023-03-28)

[17] Kotlin, FAQ [Online] [updated:  2023 Mars 25].

Available: <https://kotlinlang.org/docs/faq.html#what-can-i-use-kotlin-for>

(hämtad 2023-03-29)

[18] TechTarget, Android Studio [Online].

Available:<https://www.techtarget.com/searchmobilecomputing/definition/Android-Studio> (hämtad: 2023-03-29)

[19] Apple, New features available with iOS 16 [Online].

Available: <https://www.apple.com/ios/ios-16/features/> (hämtad 2023-03-29)

[20] Apple, Swift Has Reached 1.0, 2014 Sep 09 [Online].

Available: <https://developer.apple.com/swift/blog/?id=14> (hämtad 2023-03-39)

[21]  Swift, About Swift [Online].

Available: [https://www.swift.org/about](https://www.swift.org/about/)/ (hämtad 2023-03-29)

[22] Apple, Exploring the structure of a SwiftUI app [Online].

Available:<https://developer.apple.com/tutorials/swiftui-concepts/exploring-the-structure-of-a-swiftui-app?changes=_3>  (hämtad 2023-03-29)

[23] Apple, Documentation Xcode [Online].

Available: <https://developer.apple.com/documentation/xcode> (hämtad 2023-03-30)

[24] Apple, Xcode 14 [Online].

Available: <https://developer.apple.com/xcode/> (hämtad 2023-03-30)

[25] Mozilla, HTML5 [Online] [Updated: 2023 Feb 21].

Available:<https://developer.mozilla.org/en-US/docs/Glossary/HTML5>(hämtad 2023-04-01)

[26] W3schools, HTML Tutorial [Online].

Available:  <https://www.w3schools.com/html/> (hämtad 2023-04-02)

[27] Barron Brenda, Kinsta: HTML vs HTML5: Lär dig de avgörande skillnaderna mellan dem, 2021 Aug 10 [Online].

Available: <https://kinsta.com/se/blog/html-vs-html5/> (hämtad 2023-04-02)

[28] W3schools, JavaScript History [Online].

Available: [https://www.w3schools.com/js/js\_history.as](https://www.w3schools.com/js/js_history.asp)p (hämtad 2023-04-03)

[29] W3schools, HTML JavaScript [Online].

Available: <https://www.w3schools.com/html/html_scripts.asp> (hämtad 2023-04-04)

[30] Gitnux,JavaScript Statistics 2023: Key Insights And Trends, 2023 Mars 3 [Online].

Available: <https://blog.gitnux.com/javascript-statistics/> (hämtad 2023-04-05)

[31] Visual Studio Code, Why did we build Visual Studio Code? [Online] [Updated: 2023 03 30].

Available <https://code.visualstudio.com/docs/editor/whyvscode> (hämtad 2023-04-05)

[32] GNU Astronomy Utilities, Why C programming language? 2022 Oct [Online]

Available:<https://www.gnu.org/software/gnuastro/manual/html_node/Why-C.html> (hämtad 2023-04-06)

[35]Sakshi Pandey, The Importance Of Code Reusability In Software Development, [Online]. BrowserStack; [cited: October 3, 2022]

Available:<https://www.browserstack.com/guide/importance-of-code-reusability>

(hämtad 2023-04-01)

[35]Richard Bellairs, What is Code Reuse? Code Reuse Best Practice, [online]. Perforce; [cited: 2017-07-07]

Available:<https://www.perforce.com/blog/qac/what-code-reuse-code-reuse-best-practices>(hämtad: 2023-03-20)

[36]DJ Wardynski, 8 Ways To Ensure Code Reusability In A Project, [online].Brainspire;[cited: 2022 02 21].

Available:<https://www.brainspire.com/blog/maximize-your-ability-for-code-reusability-8-effective-ways> (hämtad 2023-03-28)

[37]Google Android Developers, Android´s Kotlin-first approach, [online].2023[updated: 2023-03-01]

Available: <https://developer.android.com/kotlin/first> (hämtad: 2023-04-05)

[38]Google Android Developers, Get started with NDK, [online].[updated: 2022-10-12].

Available: <https://developer.android.com/ndk/guides> (hämtad 2023-04-10)

[39] Google Android Developers, JavascriptEngine, [online].2023[updated: 2023-04-05].

Available:<https://developer.android.com/jetpack/androidx/releases/javascriptengine>

(hämtad: 2023-04-08)

[40] Swift, Swift and Webassembly,[online].

Available:<https://book.swiftwasm.org/index.html> (hämtad 2023-04-06)

[41]Oracle, Introduction Java Native Interface, [online], 2020

Available:<https://docs.oracle.com/javase/7/docs/technotes/guides/jni/spec/intro.html#wp16230> (hämtad 2023-03-30)

[42]Apple Inc., Importing Objective-C into Swift, [Online],

Available:<https://developer.apple.com/documentation/swift/importing-objective-c-into-swift> (hämtad 2023-03-29)

[43] Mozilla Inc, WebAssembly Concepts,[Online],2023[updated 2023-03-23],

Available:<https://developer.mozilla.org/en-US/docs/WebAssembly/Concepts>

(hämtad 2023-04-02)

[44] Emscripten, About Emscripten [Online], 2015,

Available:<https://emscripten.org/docs/introducing_emscripten/about_emscripten.html>

(hämtad 2023-04-02)

[45]Emscripten, Interacting with code [Online], 2015,

Available:<https://emscripten.org/docs/porting/connecting_cpp_and_javascript/Interacting-with-code.html?highlight=library> (hämtad 2023-04-02)

# Bilagor

# Bilaga A – Rubrik för bilaga A

# Bilaga B – Rubrik för bilaga B