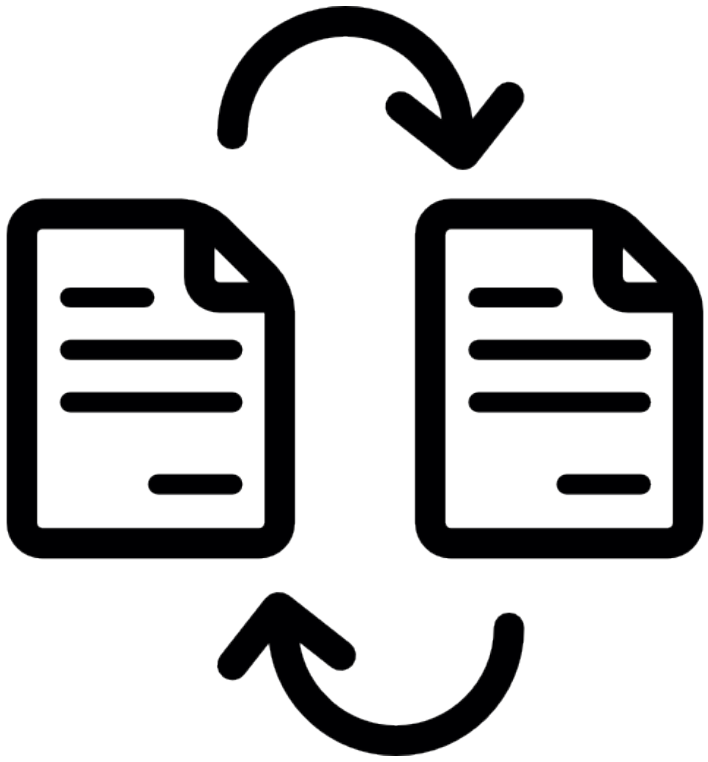
CHALMERS
UNIVERSITY OF TECHNOLOGY



Filsynkronisering i Realtid

Plugin till Eclipse för synkronisering av programkod

Examensarbete inom Högskoleingenjörsprogrammet i Datateknik

Magnus Källtén

David Strömner

Institutionen för data- och informationsteknik
CHALMERS TEKNISKA HÖGSKOLA
Göteborg, Sverige 2015

Filsynkronisering i Realtid
Plugin till Eclipse för synkronisering av programkod

Handledare: Uno Holmer, Institutionen för data- och informationsteknik.

Alixandér Ansari, Sigma Technology

David Ryden, Sigma Technology

Examinator: Christer Carlsson, Institutionen för data- och informationsteknik.

Kandidatarbete 2015:06
Högskoleingenjörsprogrammet i Datateknik
Chalmers Tekniska Högskola
SE-412 96 Göteborg
Telefon +46 31 772 1000

Typsatt i LaTeX
Göteborg, Sverige 2015

Abstract

The intention of this project was to create an improvement for code collaboration. Code collaboration is something developers often struggle with and none of today's programs and methods remove the problem entirely. A solution is to introduce a program that runs in the background that can modify files' content. This proposed program would allow developers to connect to each other to share and work simultaneously on the same projects and files. The program we developed became a plugin to the IDE (Integrated Development Environment) Eclipse and it can be used by two developers to write in shared files while being able to see what the other developer is writing in real time. The program we developed has the potential to improve the way code is developed in the near future.

Sammanfattning

Målet med projektet var att skapa en bättre metod för att samarbeta inom kodutveckling. Att samarbeta inom kodutveckling är något programmerare ofta har problem med; en perfekt lösning till problemet existerar inte idag. En lösning är att introducera ett program som körs i bakgrunden som kan modifiera innehållet av filer. Programmet skulle tillåta utvecklare att koppla upp sig mot varandra för att sedan kunna dela och arbeta i samma filer samtidigt. Programmet vi utvecklade blev ett plugin till IDE:en (Integrated Development Environment) Eclipse och kan användas av två utvecklare för att skriva i delade filer och kunna se vad den andra utvecklaren skriver i realtid. Programmet vi har utvecklat har potentialen att förbättra sättet kod utvecklas på inom en snar framtid.

Keywords: file, synchronise, synchronisation, myers, diff, difference, bitap, fuzzy, match.

Förord

Vi vill tacka våra handledare David Ryden och Alixander Ansari för deras konsultation under projektets gång och för att de gav oss möjligheten att göra projektarbetet på Sigma Technology. Stort tack till Uno Holmer, vår Chalmershandledare för hans hjälp med korrekturläsning och rådgivning. Ytterligare ett tack till alla på Sigma Technologys kontor på Lindholmen som gjorde tiden där minnesrik.

Titeln, Nätverk och Dator ikonerna tagna ifrån Freepik från www.flaticon.com
Moln ikonen tagen ifrån SimpleIcon från www.flaticon.com

Innehållsförteckning

Figurer

Tabeller

Ordlista

ASF The Apache Software Foundation, en orginisation som tillhandahåller Apache Software License.

Bitap Algoritm som implementerar match och patch.

Diff Differensalgoritm, algoritm som räknas ut skillnaderna mellan två strängar.

Editscript Ett skript som innehåller ändringarna som behöver göras för att göra om en sträng till en annan.

IDE Integrated Development Environment, utvecklingsmiljö för kodning.

IP Internet Protocol, protokoll som används för överföring av information.

Match Matchningsalgoritm, Algoritm jämför en text med ett mönster.

Moln IT-tjänst som tillhandahålls över internet, ofta något tillämpningsprogram, serverprogram eller lagring av data.

NAT Network Adress Translation, ett protokoll som manipulerar IP-adresser i IP-paket.

PAT Port Address Translation, ett tillägg till NAT som manipulerar portnummer i IP-paket.

Patch Algoritm som klistrar in tecken i en text.

Path Ett klassobjekt som representerar en filväg.

Varargs En funktion som tar in ett obestämt aritet.

1
Inledning

1.1 Bakgrund

I dagens samhälle blir det vanligare med samarbete över Internet då tekniken utvecklas och blir billigare samt att kontakter utomlands blir allt mer vanliga. Detta leder till ett behov av nya tekniska lösningar som kan göra kontakt och samarbete över Internet smidigare och lättare. Teknikutvecklingen har gett oss många ovärderliga verktyg som används varje dag, till exempel röst- och videokonferensverktyg, Github och Google Drive. Men det finns fortfarande nischer som inte har fyllts ännu.

Projektet görs i samarbete med Sigma Technology, ett konsultföretag som är en global leverantör av produktinformation, mjukvara och inbyggda systemlösningar. Sigma Technology är en del av Sigma Group som är en ledande konsultgrupp med kontor i Sverige, Ukraina, Kina, Ungern och USA.

1.2 Syfte

Syftet med projektet är att ta fram en prototyp till en produkt som kan utföra synkronisering av texter i realtid mellan två användare; alltså att två användare kan sitta med varsin kopia av ett dokument och se vad den andra personen skriver i realtid. Konceptet existerar redan och används flitigt med till exempel verktyg som Google Docs. Där är det möjligt skapa dokument och se vad den andra skriver. Skillnaden mellan dessa och vårt projekt är att de existerande verktygen är molnbaserade. Ett molnbaserat verktyg lagrar datan på molnet och hanteras av en tredje part. Detta är inte önskvärt då användare är beroende av deras tjänst. Det kan dessutom leda till stöld av immaterialrätt om säkerheten inte är tillräckligt bra.

Vi vill dessutom vidareutveckla vår förmåga att ta fram en produkt och att få djupare kunskaper av synkroniseringsalgoritmer.

1.3 Mål

Under projektets gång ska en fungerade prototyp för att synkronisera filer mellan två olika användare skapas. Det vill säga att två användare skall kunna redigera i en och samma fil samtidigt och se uppdateringar i realtid. Tanken är att programmet ska kunna användas i IDE:er för att kunna föra ett närmare samarbete under

programutveckling. Programmet kommer att fungera enligt klient-server-modellen för att klienter ska kunna hitta varandra. Filerna kommer endast att lagras lokalt och inte på serverenheten.

1.4 Avgränsningar

Prototypen kommer att använda sig av en differensalgoritm för att hitta skillnader mellan användarnas text och sedan beräkna var ändringarna ska ske. På grund av den strama tidsramen finns det inte tid nog att skriva en egen differensalgoritm utan vi kommer att använda oss av färdigskrivna open source-alternativ.

Utöver det finns det några funktioner som kommer att bortprioriteras. På grund av tidsbrist kommer vi inte att hinna implementera allt som finns i vår vision av ett färdigt program. Dessa inkluderar:

Kryptering
Kryptering är en viktig funktion för skydd av både konfidalitet och integritet. Det är inte önskvärt att vem som helst som kan lyssna på nätverket kan få tillgång till datan som skickas och inte heller att det är möjligt att ändra i paketen som skickas.

Fler användare
Det hade varit önskvärt att kunna sitta fler än två med samma text. Vi kommer dock att koncentrera oss på att få det att fungera mellan två användare. Att kunna ha fler användare kräver mer tid för att skriva en mer robust serverarkitektur medan två användare endast behöver skicka data mellan varandra.

Fler IDE:er
Det är meningen att kunna använda vilken IDE som helst, det är dock skillnad mellan olika IDE:er och hur de behandlar ändringar som ännu inte har sparats, så prototypen kommer att fokusera på Notepad och Eclipse.

2
Metod

2.1 Arbetsmetod

Det valda arbetssättet är Scrum. Scrum är ett modernt arbetssätt som är till för att utveckla produkter på ett flexibelt sätt med fokus på öppenhet, kontroll och anpassning. Scrum kräver dock fler projektmedlemmar för att kunna användas som tänkt. Vi kommer därför att använda oss av de delar som kan vara till nytta för färre medlemmar. Dessa inkluderar User Stories, ett sätt att bestämma arbetsuppgifter och sätta storlek och prioritet, samt morgonmöten där vi diskuterar vad som ska göras under dagen [1]. Vi kommer använda Java som programmeringsspråk eftersom vi har mest erfarenhet av det språket.

2.2 Vektyg

GitHub - Waffle
För att dela och hålla reda på källkoden kommer GitHub att användas. GitHub är ett webbaserat versionshanteringsverktyg som saknar ett centralt arkiv vilket gör det möjligt att skapa en egen kopia [2]. Det går att skapa både publika och privata repositories men är främst ett verktyg för privatpersoner eller för open source-projekt. Det är mer vanligt för företag att använda sina egna mer privata alternativ.

Vi kommer även att använda Waffle vilket är ett GitHub-kopplat verktyg som gör att man kan skriva upp vad som ska göras och hur varje arbetsuppgift ligger till, om den är påbörjad, pågående eller färdig [4].

Eclipse
Eftersom vi kommer att använda Java som programmeringsspråk för att skriva programmet använder vi IDE:en Eclipse. Vi använder Eclipse eftersom det är den IDE:en vi har mest erfarenhet av, den är gratis och för att det är en av de mest använda IDE:erna för Java [3]. Eclipse är en gratis och utbyggbar IDE, vilket syftar på att den bygger till stor del på användares egna tillägg kallat insticksprogram.

Google Docs - LaTeX
För att ha en gemensam plats där vi skriver projektrapporten använder vi Google Docs. Med Google Docs är det möjligt för flera personer att redigera i samma dokument samtidigt. För att sedan skriva rent texten kommer vi att använda LaTeX (Lamport TeX). LaTeX är ett verktyg för sammanställning av dokument och hjälper

till att göra akademiska rapporter strukturerade och stilrena.

2.3 Litteraturstudier

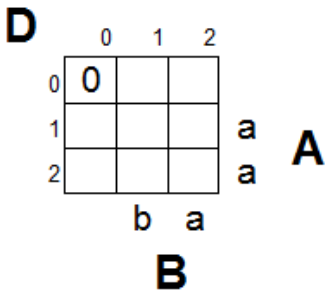
Programmet som vi kommer att utveckla behöver använda sig av någon slags sorteringsalgoritm för att avgöra var tillägg och reduktioner ska göras i texten. För att förstå och kunna göra ett beslut om valet av algoritm kommer vi att lägga tid på litteraturläsning inom dessa områden.

3
Teknisk Bakgrund

3.1 Myers Algoritm

Myers differensalgoritm används för att hitta skillnaderna mellan strängarna **A** och **B**. Längden av **A** kommer benämnas som **m**, respektive **n** för **B** och strängarnas tecken som $\mathbf{a_1a_2...a_m}$ för **A**, respektive $\mathbf{b_1b_2...a_n}$ för **B**. Myers differensalgoritm tar fram ett editskript **S** som beskriver det mest effektiva sättet att omvandla strängen **A** till strängen **B**. I de fall **A** och **B** inte innehåller många skillnader kommer editskriptet att bli litet i förhållande till texten. Algoritmen har en komplexitet av **O(NE)**, där **N** = **m+n** och **E** är storleken av det minsta editskriptet från omvandlingen av **A** till **B** [5].

För att demonstrera hur Myers differensalgoritm fungerar återgår vi till strängarna **A** och **B**. Låt **A** = **aa** och **B** = **ba**. Strängarna sätts upp i en tvådimensionell elementmatris, **D**, där varje element består av ett editskript och ett heltalsvärde, **d**. **d** står för hur många ändringar som behöver göras för att omvandla den ena strängen till den andra strängen medan editskriptet säger vilka ändringar som behöver göras. Viktigt att observera är att matrisen har en extra inledande rad och kolumn, den extra raden och kolumnen är till för fallen då tomma filer jämförs. Samtidigt tillkommer en fast ingångspunk med ett specifikt **d**-värde, 0. Figuren 3.1 visar matrisen **D**:s startläge med strängarna **A** och **B**.



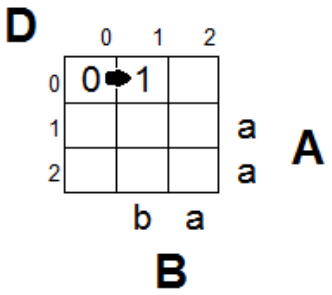
Figur 3.1: Matris **D**.

3.1.1 Regler

För att hitta det optimala skriptet för omvandlingen och fylla i matrisen använder algoritmen tre regler:

Regel ett - Höger
Regel ett beskriver villkoren som krävs för att beräkna värdena i elementet till höger om elementet **D[i,j]** i matrisen. För att beräkna värdena i elementet **D[i,j+1]** behöver **D[i,j]**:s **d**-värde och editskript vara kända. Om kraven uppfylls sätts **D[i,j+1]**:s **d**-värde till **D[i,j]**:s **d**-värde adderat med 1. **D[i,j+1]**:s editskript sätts till **D[i,j]**:s editskript samt att "Lägg till $\mathbf{b}_j$" läggs till i slutet av editskriptet.

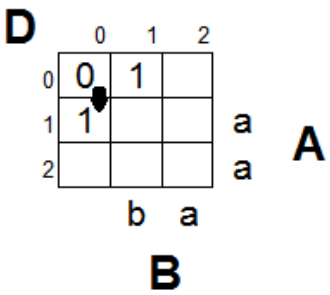
För att demonstrera regel ett fortsätter vi på matrisen i figuren 3.1. Vi vill beräkna vad elementet till höger om elementet **D[0,0]** kommer att innehålla. Eftersom elementet **D[0,0]**:s **d**-värde är 0 och dess editskript är känt är kraven uppfyllda. Elementet **D[0,1]**:s **d**-värde tas från **D[0,0]** och adderas med 1 och i dess editskript läggs det till: "Lägg till **b**". Matrisen illustreras i figur 3.2.



Figur 3.2: Regel ett exempel.

Regel två - Neråt
Regel två beskriver villkoren som krävs för att beräkna värdena i elementet nedanför elementet **D[i,j]** i matrisen. För att beräkna värdena i elementet **D[i+1,j]** behöver **D[i,j]**:s **d**-värde och editskript vara kända. Om kraven uppfylls sätts **D[i+1,j]**:s **d**-värde till **D[i,j]**:s **d**-värde adderat med 1. **D[i+1,j]**:s editskript sätts till **D[i,j]**:s editskript samt att "Ta bort $\mathbf{a}_i$" läggs till i slutet av editskriptet.

För att demonstrera regel två fortsätter vi på matrisen i figuren 3.2. Vi vill beräkna vad elementet nedanför elementet **D[0,0]** kommer att innehålla. Eftersom elementet **D[0,0]**:s **d**-värde är 0 och dess editskript är känt är kraven uppfyllda. Elementet **D[1,0]**:s **d**-värde tas från **D[0,0]** och adderas med 1 och i dess editskript läggs det till: "Ta bort **a**". Matrisen illustreras i figur 3.3.

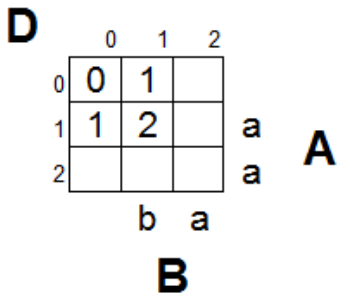


Figur 3.3: Regel två exempel.

Regel tre - Diagonalt
Regel tre skiljer sig från regel ett och två i och med att regeln har ett extra krav och att regeln prioriteras före de andra två. Det innebär att algoritmen börjar med att beräkna så många element som möjligt med hjälp av den tredje regeln innan den beräknar element med hjälp av regel ett och två.

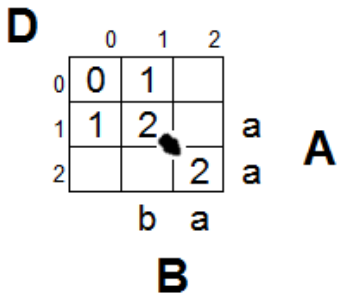
Regel tre beskriver villkoren som krävs för att beräkna värdena i elementet diagonalt ner till höger i matrisen från elementet **D[i,j]**. För att beräkna värdena i elementet **D[i+1,j+1]** behöver **D[i,j]**:s **d**-värde och editskript vara kända. Utöver det måste $\mathbf{a}_{i+1} = \mathbf{b}_{j+1}$. Om kraven uppfylls sätts **D[i+1,j+1]**:s **d**-värde till **D[i,j]**:s **d**-värde och **D[i+1,j+1]**:s editskript sätts till **D[i,j]**:s editskript.

För att demonstrera regel tre fortsätter vi på matrisen i figuren 3.3. En 2:a på platsen **D[1,1]** i figuren 3.4 har blivit beräknad med hjälp av regel ett eller två för att komma till ett scenario där regel tre kan demonstreras.



Figur 3.4: Förberedelse för regel tre exempel.

Vi vill beräkna vad elementet diagonalt nedanför till höger från elementet **D[1,1]** kommer att innehålla. Eftersom elementet **D[1,1]**:s **d**-värde är 2, dess editskript är känt och att $\mathbf{a}_2 = \mathbf{b}_2$ är kraven uppfyllda. Elementet **D[2,2]**:s **d**-värde tas från **D[1,1]** och **D[2,2]**:s editskript sätts till **D[1,1]**:s editskript. Matrisen illustreras i figur 3.5.

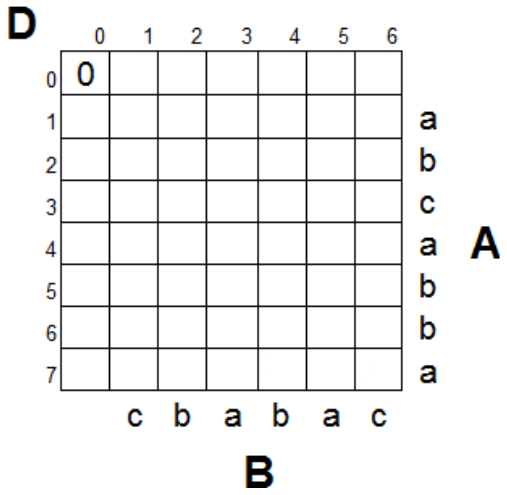


Figur 3.5: Regel tre exempelet.

3.1.2 Shortest Edit Script - Demonstration

I detta delkapitlet introduceras en större matris, **D**, som använder sig av strängarna **A** och **B**. Låt **A = abcabba** och **B = cbabac**. Med de tre reglerna som grund går

det att beräkna alla element i matrisen som illustreras i figur 3.6, men innan dess görs ett par observationer.



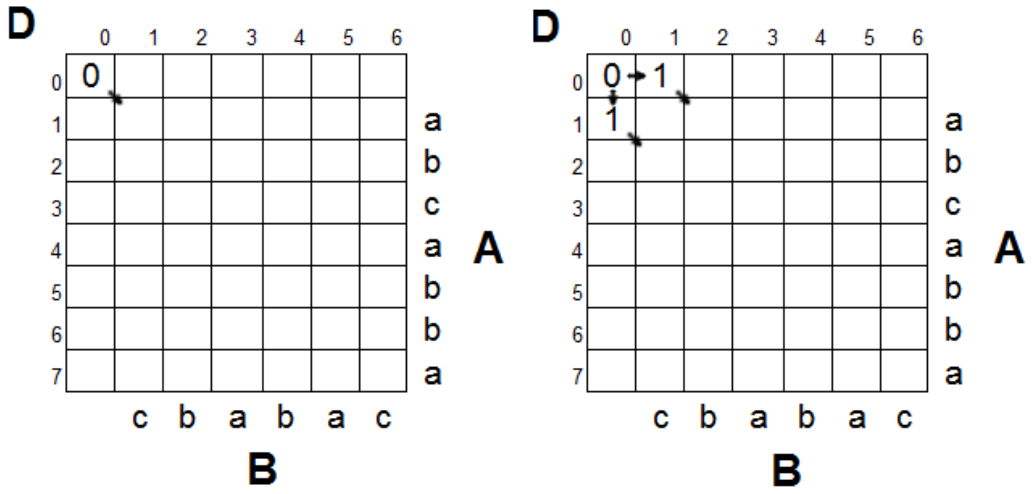
Figur 3.6: Matris **D**.

En helt ifylld matris kommer att ha ett så kallat **SES** (Shortest Edit Script) på platsen **D[m,n]** som alltid beskriver det kortaste editskriptet som omvandlar strängen **A** till strängen **B**. Elementen i matrisen beräknas i turordning efter växande **d**-värde. Algoritmen börjar med att beräkna alla element som kommer att få **d**-värdet 0. När inga fler 0:or kan beräknas ökas värdet och algoritmen fortsätter med att beräkna alla element i matrisen som kommer att få **d**-värdet 1. Detta fortsätter tills elementet **D[m,n]** har blivit beräknat, det kortaste editskriptet är då funnet. Som en följd av detta kan algoritmen beräkna **SES** innan alla element i matrisen har beräknats. Det innebär att det är fördelaktigt om strängarna är lika varandra då fler element beräknas med hjälp regel tre, vilket gör att algoritmen når **D[m,n]** tidigare.

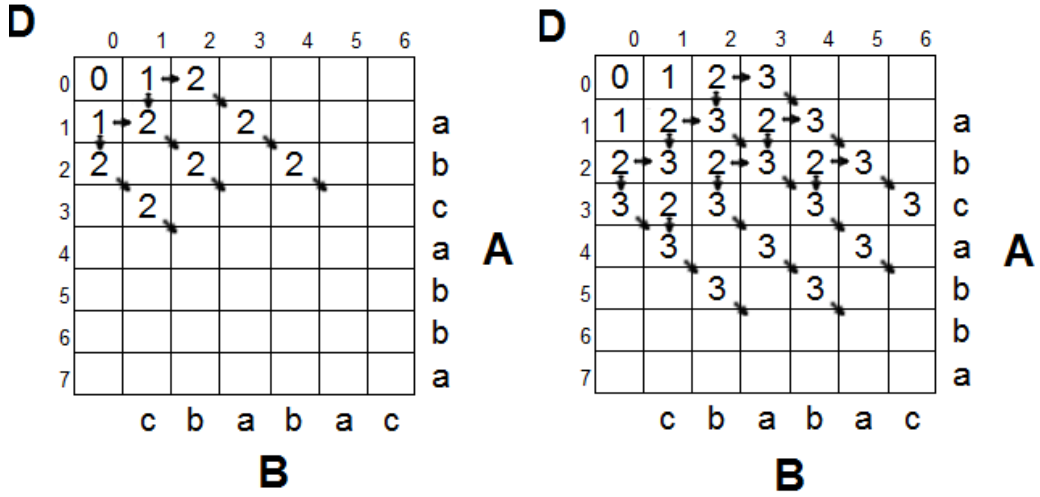
Vi återgår till exemplet med strängarna **A** och **B**. Elementet på platsen **D[0,0]** har alltid ett fixt **d**-värde, 0. Algoritmen börjar med att beräkna alla element som kommer att få **d**-värdet 0. De enda elementen som kommer att få värdet 0 är de som befinner sig på diagonalen från **D[0,0]**, men för att beräkna elementet **D[1,1]** med hjälp av den tredje regeln behöver $\mathbf{a}_1 = \mathbf{b}_1$, vilket inte stämmer då $\mathbf{a}_1 = \mathbf{a}$ och $\mathbf{b}_1 = \mathbf{c}$. Eftersom det inte går att beräkna fler element med **d**-värdet 0 kommer algoritmen därför att fortsätta med att beräkna alla element med **d**-värdet 1.

Från elementet **D[0,0]** går det att beräkna elementet **D[0,1]** med hjälp av regel ett och elementet **D[1,0]** med hjälp av regel två. Eftersom **D[0,0]** har ett **d**-värde och ett editskript som är tomt och att **D[0,1]** är okänt sedan tidigare uppfylls kraven för regel ett. Element **D[0,1]** kan fyllas i med elementet **D[0,0]**:s **d**-värde plus 1, 0 + 1 = 1, och lägga till "Lägg till **c**" i dess editskript. Samma sak gäller för **D[1,0]**, elementet är okänt och de andra kraven för regeln har vi redan konstaterat att de gäller. 1 adderas till **d**-värdet och sätts som **D[1,0]**:s **d**-värde och i editskriptet läggs "Ta bort **a**" till. En illustration sker i figur 3.7.

Efter beräkning med hjälp av regel ett eller två ser algoritmen om det går att beräkna element med hjälp av regel tre. I iterationen med **d**-värdet 1 hittas däremot inga element som kan beräknas med regel tre. Algoritmen fortsätter då att beräkna element som kommer att få **d**-värdet 2. Närmare beskrivning av den andra, tredje och fjärde iterationen beskrivs inte eftersom de följer samma steg som redan beskrivits. Iterationerna illustreras däremot i figurerna 3.7, 3.8 och 3.9.

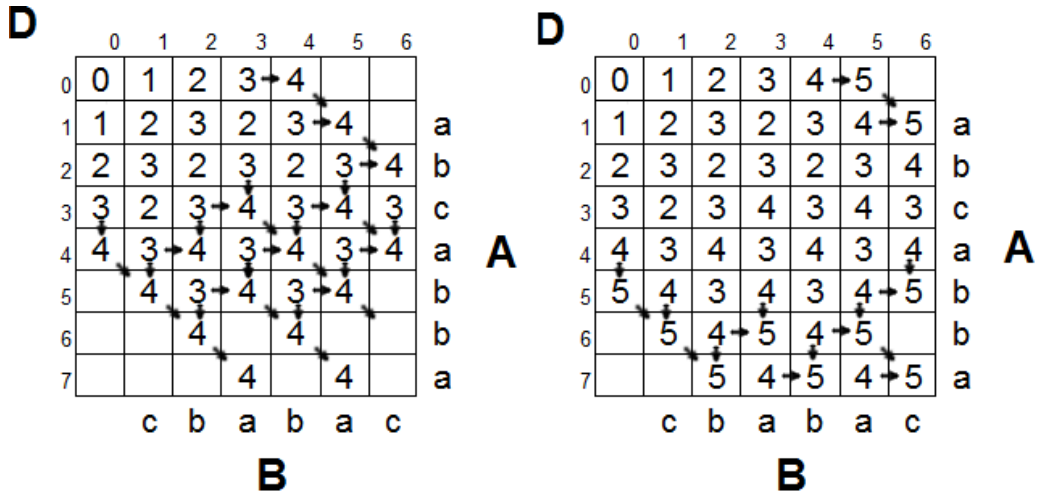


Figur 3.7: Illustration om hur algoritmen fyller ut 0:orna och 1:orna för exemplet.



Figur 3.8: Illustration om hur algoritmen fyller ut 2:orna och 3:orna för exemplet.

Den sista iterationen är däremot intressant. När algoritmen beräknat elementet **D[7,6]** har **SES** tagits fram. Algoritmen har då fått fram slutresultatet och kommer då att avsluta utan att ha beräknat alla element i matrisen. Den slutgiltiga matrisen kan ses i figur 3.9.



Figur 3.9: Illustration om hur algoritmen fyller ut 4:orna och 5:orna för exemplet.

3.2 Bitap Algoritm

Bitap är en metod för att matcha strängar approximativt, den utför match och patch utifrån editeringsskriptet som Myers differensalgoritm beräknar (se kapitel 3.1) och beskriver hur editeringsskriptet ska appliceras på texten.

Strängmatchning är något som har varit ett problem sedan 70-talet [7]. Över åren har flertal algoritmer föreslagits och algoritmerna delas upp i två grupper: online och offline. Online är algoritmer som inte använder sig av index (pekare till olika delar i texten för snabbare åtkomst) och är därmed sämre när dokumenten blir stora. Offline är algoritmer med index och är långsammare än online när det är korta strängar [8]. Bitap får sin hastighet genom att utnyttja datorns skiftregister [9].

3.2.1 Levenshteinavstånd

Levenshteinavståndet, också känd som editeringsavståndet, definieras som det minsta antalet insättningar, borttagningar och substitueringar av enskilda tecken som behöver göras för att modifiera om en sträng till en annan [10].

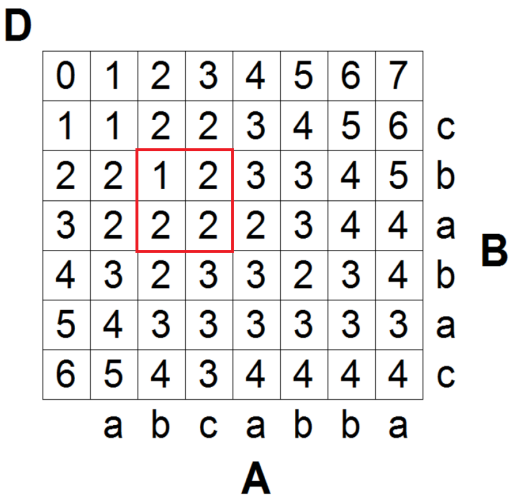
Sättet som Levenshteinavståndet fungerar påminner om Myers differensalgoritm, men Levenshteinavståndet räknar diagonal flytt annorlunda eftersom substitutering av tecken stöds. Eftersom reglerna för Levenshteinavståndet [11] är jämförbara med reglerna för Myers differensalgoritm sammanfattas de här:

1) D[i,0] = i, D[0,j] = j

2) D[i,j] = $\begin{cases} D[i-1, j-1], if(a_1 = b_j) \\ 1 + min(D[i-1, j-1], D[i-1, j], D[i, j-1]), otherwise \end{cases}$

Demonstration
Som exempel används två strängar, **A** och **B** som sätts upp i elementmatrisen **D**. Låt **A** = **abcabba** och **B** = **cbabac**. Enligt den första regeln kommer **D[0,0]** få

värdet 0 medan de resterande elementen i den första raden och den första kolumnen kommer sättas till heltal i ökande ordning från 0. De resterande elementen i matrisen beräknas med hjälp av den andra regeln. Den andra regeln jämför först om $\mathbf{a}_i = \mathbf{b}_j$. Stämmer det får elementet **D[i,j]** samma värde som **D[i-1,j-1]**:s värde, annars sätts elementet **D[i,j]** till det värdet som är minst av värdena från elementen **D[i-1,j-1]**, **D[i-1,j]** eller **D[i,j-1]**, adderat med 1. Ett exempel finns inom den röda rutan i figur 3.10; elementet **D[3,3]** har satts med hjälp av den andra regeln. $\mathbf{a}_3$ var inte lika med $\mathbf{b}_3$ och värdet fick sättas genom att välja det minsta värdet från elementen **D[2,2]**, **D[2,3]** och **D[3,2]**. Det minsta värdet var från elementet **D[2,2]**, 1, vilket innebär att elementet **D[3,3]** fick värdet 2 enligt den andra regeln.



Figur 3.10: Exempel på en Levenshteinavtåndsmatris.

3.2.2 Exakt Sträng Matchning

Exakt sträng matchning är det lättare fallet av approximativ strängmatchning. Som namnet antyder kan algoritmen endast hitta matchningar i en text om mönstret matchar exakt. Algoritmen använder sig av datorns skiftregister vilket tillåter algoritmen att kontrollera samtliga tecken för ett givet mönstret samtidigt.

Algoritmen utgår från en array **t** som har en given längd **m**, samt ett mönster **P** som ska matchas med **t**. Det första steget är att skapa en array **T**, där varje plats i arrayen motsvarar chiffret för varje unikt tecken som förekommer i **P** eller **t**. Chiffrerna har samma längd som **P** och består av 1:or och 0:or. Varje chiffer har 0:or där dess tecken förekommer i **P** och 1:or på resterande platser. Tecken som finns i **t** men inte i **P** kommer att ha ett chiffer som består av enbart 1:or. Alla chiffer är spegelvända eftersom datorns skiftregister skiftar till vänster.

För att demonstrera hur chiffrerna ser ut sätt $\mathbf{t} = \mathbf{ABCAA}$ och $\mathbf{P} = \mathbf{ABAB}$. Mellan **t** och **P** finns det tre unika tecken: **A,B,C**. Eftersom det finns tre unika tecken så kommer **T** ha tre stycken arrayer. Varje array kommer ha längden fyra eftersom det finns fyra tecken i **P**:

T[A] = 0101
T[B] = 1010
T[C] = 1111

Spegelvända:

T[A] = 1010
T[B] = 0101
T[C] = 1111

Innan algoritmen kan börja jämföra mönstret mot texten behövs det ett sätt för algoritmen att hålla reda på resultatet från en jämförelse. Statusarrayen **S** används för detta ändamålet. **S** kommer att ha samma längd som **t**, det vill säga längden **m**. **S** initieras med 1:or vilket motsvarar att inget matchar ännu. Algoritmen behöver slutligen en variabel **i** för att hålla reda på vilket tecken i **t** det är som den ska läsas av härnäst. **i** initieras till 0.

När väl initieringen är färdig kommer algoritmen att börja leta efter matchningar. Algoritmen skiftar värdena i **S** ett steg till vänster vilket gör att det kommer in en 0:a längst till höger i arrayen. Algoritmen genomlöper arrayen **t** genom att läsa tecknet på plats **i**. Med hjälp av tecknet kan chiffret fås ut ur **T** via $\mathbf{T}[\mathbf{t}_i]$. Chiffret som fås ut läggs ihop med **S** med hjälp av OR-operationen och resultatet stoppas tillbaka i **S**. Om 0:an fortfarande finns kvar i **S** betyder det att början av en matchning har hittats. **i** ökas sedan med ett och algoritmen börjar om med att skifta vänster. Algoritmen är färdigt när i = **m** eftersom det då inte finns något mer tecken från **t** att läsa.

Om en 0:a inte försvinner efter en OR-operation kommer den att skiftas ett steg till vänster. Vilket betyder att nästa tecken i mönstret kommer att avgöra ifall nollan kommer att finnas kvar även nästa varv. När **i** = **m** avslutas algoritmen och 0:or som finns kvar i **S** kommer att motsvara matchningar som algoritmen hittat. För att det ska räknas som en fullständig matchning behöver matchningen vara minst lika långt som mönstret.

Demonstration
För att förtydliga algoritmen ges ett exempel nedan:
Låt **t = ABBCAB**, **P = ABB** och **m = 6**. Som nämndes ovan behövs det ett chiffer för varje unikt tecken som förekommer i **P** eller **t**. Eftersom det finns totalt tre unika tecken mellan **t** och **P** behövs det tre platser i **T**. Varje array kommer ha längden tre eftersom det finns tre tecken i **P**:

T[A] = 011
T[B] = 100
T[C] = 111

Chiffren ovan har inte blivit spegelvända ännu, efter spegelvändning ser de ut som följande:

T[A] = 110
T[B] = 001
T[C] = 111

Kvar att initiera är **S** och **i**. $\mathbf{S_0} = \mathbf{111111}$ eftersom **t** är 6 tecken lång och i **S** initiala tillstånd ska arrayen endast innehålla 1:or. $\mathbf{i = 0}$ vilket gör att algoritmen hämtar första tecknet i **t** när den börjar utföra matchningen. Algoritmens huvudloop sammanfattas nedan i punktform:

- Skifta $\mathbf{S}_i$ till vänster, en 0:a kommer in längst bort till höger. (Skiftad: $\mathbf{S}_i'$)
- Få fram chiffret för nuvarande tecken: $\mathbf{T}[\mathbf{t}_i]$.
- Utför OR-operationen mellan $\mathbf{T}[\mathbf{t}_i]$ och **S**. Resultatet stoppas tillbaka i **S**.
- Öka **i** med 1. Ifall $\mathbf{i = 6}$ avbryt annars börja om från första punkten.

Genom att följa punkterna ovan skiftas $\mathbf{S}_0$ till vänster en gång och blir $\mathbf{S}_0' = \mathbf{111110}$. $\mathbf{t}_0 = \mathbf{A}$ och **T[A]=110**. Kvar att göra är att använda OR-operationen mellan $\mathbf{S}_0'$ och chiffret **110**. Resultatet av OR-operationen kan ses i tabell 3.1.

$T[t_0]$	110
S_0'	111110
S_1	111110

Tabell 3.1: Exakt matchning exempel för i = 0.

Nollan som skiftades in i $\mathbf{S}_0$ är fortfarande kvar, vilket betyder att första tecknet i **P** stämmer överens med $\mathbf{t}_0$. **i** ökas med 1 och blir $\mathbf{i = 1}$. Eftersom 1 är mindre än 6 kommer algoritmen börja om på den första punkten. $\mathbf{S}_1$ skiftas till vänster och blir $\mathbf{S}_1' = \mathbf{111100}$. Chiffret fås ut genom $\mathbf{T}[\mathbf{t}_1]$ vilket är **001**. Algoritmen använder sedan OR-operationen på chiffret och $\mathbf{S}_1'$. Resultatet av OR-operationen kan ses i tabellen 3.2.

$T[t_1]$	001
S_1'	111100
S_2	111101

Tabell 3.2: Exakt matchning exempel för i = 1.

Nollan som skiftades in i $\mathbf{S}_1$ blev en 1:a vilket betyder att tecknet **B** inte är början på en ny matchning. Däremot är 0:an från den tidigare skiftningen kvar, vilket tyder på att det andra tecknet i **P** fortsätter stämma överens med **t**. **i** ökas med ett och blir $\mathbf{i = 2}$, vilket är mindre än sex. Algoritmen fortsätter då och börjar om från den första punkten. Eftersom de resterande fyra varven kommer att utföra samma arbete som de två första varven sammanfattas de i tabellen 3.3.

$T[t_2]$	001
S'_2	111010
S_3	111011

(a) i = 2

$T[t_3]$	111
S'_3	110110
S_4	110111

(b) i = 3

$T[t_4]$	110
S'_4	101110
S_5	101110

(c) i = 4

$T[t_5]$	001
S'_5	011100
S_6	011101

(d) i = 5

Tabell 3.3: Exakt matchning exempel för i = 2 till i = 5.

Efter det sista varvet kommer **i = 6** och algoritmen kommer att avslutas. $\mathbf{S_6}$ = **011101** betyder att en exakt matchning och början på en matchning funnits av mönstret **P** i texten **t**. För att det ska räknas som en exakt matchning behöver nollan i **S** efterföljas av längden av **P**, minus 1, 1:or. Är så fallet betyder det att hela mönstret hittas i texten. Illustration kan ses i tabell 3.4.

t	A	B	B	C	A	B	
S_6	0	1	1	1	0	1	
P	A	B	B		A	B	B

Tabell 3.4: Insättning av mönstret på nollorna i S_6.

3.2.3 Inexakt Strängmatchning

Inexakt strängmatchning, mer känt som Fuzzy String Match, är en utökning av exakt sträng matchning som tillåter matchningar med **k** antal fel. Inexakt sträng matchning stödjer samma typer av operationer som Levenstein: insättningar, borttagningar och substitutioner. Antalet fel är definierat som antalet insättningar, borttagningar eller substitutioner operationer som behöver göras för att omvandla en delsträng i **t** till **P**. Utöver den vanliga resultatarrayen för exakt strängmatchning behövs **k** stycken extra arrayer. Den första av dessa extra arrayer är resultatarrayen för matchning med ett fel och byggs med hjälp av resultatarrayen för exakt matchning. Arrayen för två fel byggs på arrayen med ett fel och så vidare [12].

Demonstration
För att förtydliga hur arrayerna med upp till **k** antal fel ser ut ges ett exempel nedan: Låt **t = Katten såg den bruna musen**, **P = den** och **k = 3**. Den exakta arrayen skapas på samma sätt som beskrevs i kapitel 3.2.2 och arrayerna för ett, två och tre fel kan ses i tabellen 3.5.

Antal fel	K	a	t	t	e	n		s	å	g		d	e	n		b	r	u	n	a		m	u	s	e	n
0	1	1	1	1	1	1	1	1	1	1	1	0	1	1	1	1	1	1	1	1	1	1	1	1	1	1
1	1	1	1	0	1	1	1	1	1	1	0	0	0	1	1	1	1	1	1	1	1	1	1	0	1	1
2	1	1	0	0	0	1	1	1	1	0	0	0	0	0	1	1	0	0	0	1	1	1	0	0	0	1
3	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Tabell 3.5: Exempel på inexakt strängmatchning.

Vid noll antal fel, det vill säga exakt sträng matchning, hittas bara en matchning

eftersom **P** endast finns på ett ställe i **t**. När ett fel tillåts hittas det däremot fem matchningar:

- '**ten**' substitutiera **t** mot **d**.
- ' **de**' lägg till **n** efter **de**.
- '**den**' exakt matchning.
- '**en** ' lägga till ett **d** före **en**.
- '**sen**' substitutiera **s** mot **d**.

På liknade sätt hittas det fjorton matchnignar vid två fel. Vid tre eller fler fel kommer hela texten att matcha eftersom det går att substituera vilka tre tecken som helst i **t** för att få fram **P**.

3.2.4 Patch

Patch är algoritmen som jämför inexakta matchningar (se kapitel 3.2.3) med Myers differenser (se kapitel 3.1). Varje rad i differenslistan behandlas enskilt av matchningsalgoritmen, det vill säga att varje rad i differenslistan kommer att få **k** antal matchlistor. Alla matchningarna i dessa listorna tilldelas poäng beroende på hur långt från differensradens ursprung matchningen ligger och hur många fel det finns i matchningen. Lägre poäng för en matchning betyder att det är en bättre matchning. Formeln ser ut som följande:

$$s = \frac{\frac{k}{p}}{c} + \frac{\frac{d}{m}}{1-c}$$

Där:

s = Totala poängen.
k = Antalet fel.
p = Längden av mönstret **P**.
d = Avståndet mätt i antalet tecken från differensradens förväntad position.
m = Längden av texten.
c = Konstant som avgör inverkan mellan **d** och **k**.

Eftersom $\frac{k}{p}$ divideras med **c** medan $\frac{d}{m}$ divideras med **1-c** kommer ett lågt **c** värde innebära att algoritmen favoriserar mer exakta resultat, medan ett högt **c** värde favoriserar matchningar närmare differensradens ursprung [9].

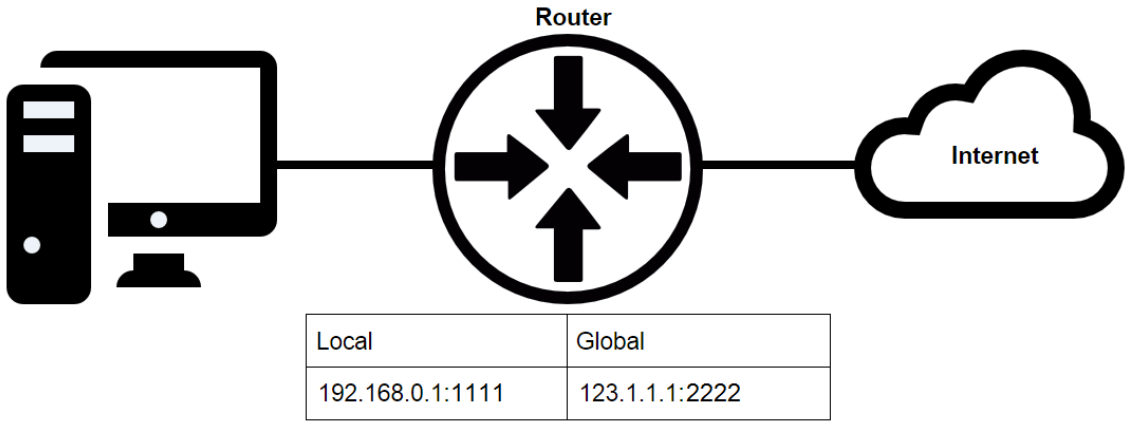
3.3 NAT/PAT

NAT (Network Adress Translation) är ett protokoll som översätter IP-adresser (Internet Protocol)[14]. Oftast gäller det översättning av privata adresser i en LAN-miljö (Local Area Network) till publika adresser som kan användas globalt. Protokollet byggs in i routrar eller brandväggar med den främsta avsikten att reducera

antalet använda publika adresser. En NAT-router översätter adressen på IP-paket genom att byta ut källadressen från datorns IP-adress till routerns IP-adress. Detta öppnar möjligheten för fler datorer att använda färre publika IP-adresser.

PAT (Port Address Translation), eller NAT/PAT, är ett tillägg till NAT och fungerar genom att manipulera källporten i IP-paketen utöver källadressen. Routern sparar de utgående IP-paketens källadress och källport i en tabell i minnet och sätter ett nytt värde som paketets källport och routerns egna IP-adress som källadress. Värdena sparas för att kunna styra svarspaket tillbaka till rätt adress.

I figur 3.11 visas ett exempel på NAT/PAT. Datorn har den lokala privata adressen 192.168.0.1 och när den skickar ett paket från port 1111 till en adress utanför det lokala nätverket kommer IP-paket att skickas från datorn med källadress 192.168.0.1 och källport 1111. När paketet kommer fram till routern sparar routern paketets källadress och källport i en tabell tillsammans med ett annat värde som routern bestämmer, i detta exemplet är det värdet 2222. Routern sätter paketets källadress till sin egen, 123.1.1.1, och källporten till 2222. När mottagaren svarar kommer mottagaren att skicka tillbaka till paketets källadress och källport. När routern tar emot svarspaketet söker den igenom sin tabell efter ett portvärde som stämmer överens med IP-paketets destinationsport. Finns det inget värde som stämmer överens med värdet slängs paketet, men i detta exemplet hittas port 2222 i tabellen. Routern ändrar då paketets destinationsadress och destinationsport till de tillhörande värdena i tabellen och sänder vidare paketet.



Figur 3.11: NAT/PAT exempel.

Detta innebär att det blir problem om man vill sätta upp kommunikation mellan datorer på andra sidan av en NAT/PAT-router. Om vi använder exemplet i figur 3.11: Utanför NAT/PAT-routern ser det ut som datorn har IP-adressen 123.1.1.1. Men skulle någon skicka till den adressen och använda ett annat portnummer än 2222 kommer de paketen att slängas i routern då det portvärdet inte finns i routerns NAT/PAT-tabell. Detta försvårar direktkommunikation.

4

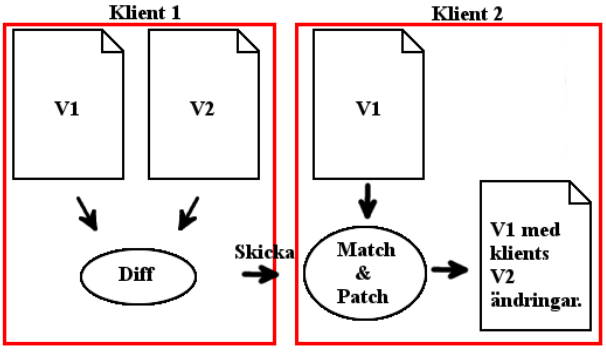
Genomförande

Projektets inleddes med att undersöka vilka lösningar som redan fanns för att synkronisera kod. Vi kunde endast hitta molntjänster som använde en liknande lösning som vi tänkt oss. Detta bekräftade att det fanns en nisch att fylla med vår produkt. Vi diskuterade om vi skulle implementera synkroniseringsalgoritmerna själva eller om vi skulle använda oss av open source-lösningar. På grund av den begränsade tiden valde vi att använda olika open source-projekt och algoritmer.

4.1 Differens, Match och Patch

När vi var färdiga med litteraturstudierna visste vi att det behövdes tre typer av algoritmer för att synkronisera dokument: Differens, Match och Patch.

Det kommer att finnas två klienter med samma dokument från början. Båda klienter har två versioner av dokumentet, en grundversion och en version de gjort ändringar i. Differensalgoritmen används för att ta fram en differenslista som beskriver hur grundversionen av dokumentet kan omvandlas till den nya versionen av dokumentet. Listan skickas sedan till klientens partner. Eftersom partnern kan ha gjort egna ändringar i sitt dokument innan differenslistan kan appliceras behövs en match- och patchalgoritm. Matchningsalgoritmen beräknar var i dokumentet differenserna kan appliceras och patchalgoritmen beräknar sedan vilka matchningar som ska appliceras på dokumentet. Figuren 4.1 illustrerar hur algoritmerna samarbetar.



Figur 4.1: Illustration av Differens-Match-Patch problemet.

Det finns två populära algoritmer för att räkna ut differensen mellan två dokument, Myers och UNIX. Myers fungerar bättre för små ändringar medan UNIX oftare används när filerna förväntas vara till större del olika [15]. Eftersom programmet ska ge

illusionen av att ändringar i ett dokument sker i realtid så kommer tiden mellan två skickade differenser att vara liten. Som en följd av detta kommer antalet ändringar som en användare hinner göra till dokumentet mellan varje sändning oftast att vara liten, borsett från fallen då användaren klistrar in längre stycken i dokumentet. Av dessa anledningar bestämde vi oss för att använda Myers differensalgoritm.

Matchningsalgoritmer är uppdelade i två grupper, online och offline. Kapitlet 3.2 Bitap Algoritm har mer information om skillnaden mellan dem. Vi bestämde oss för att välja en online-algoritm. Mer specifikt valde vi Bitap-algoritmen som är den snabbaste online-algoritmen [9].

Patch är simpel i dess konstruktion, vilket betyder att även långsamma patchalgoritmer kommer att ha liten inverkan på programmets svarstid. På grund av detta spelade det mindre roll vilken typ av patchalgoritm open source-biblioteket implementerade. Vi hittade ett bibliotek från Google, diff-match-patch, som implementerade exakt det vi sökte.

4.1.1 Licens

Googles open source-bibliotek diff-match-patch kommer med licensen Apache 2.0 som skrivits av den icke-vinstdrivande orginisationen The Apache Software Foundation (ASF) [16]. ASF använder licensen för att distribuera programvaror, men är även till för att appliceras på andras programvaror vars skapare vill använda samma villkor. För att använda mjukvara med Apache licensen måste en del regler följas. Licensen måste vara inkluderad i modifierad mjukvara, stora ändringar måste vara angivna och om det finns en NOTICE-fil med noteringar måste även den vara inkluderad om programvaran distribueras vidare. Man får inte använda originalförfattarens namn, varumärke eller logotyp. Originalförfattaren kan heller inte hållas ansvarig för skador som orsakats av programvaran. Mycket frihet tillåts, bland annat att modifiera programvaran, använda den till kommersiella ändamål och distribuera den ursprungliga eller modifierade programvaran. Detta passade vår agenda.

4.2 Struktur

Programmet är strukturerat enligt MVC (Model View Controller), vilket är en vanlig struktur för mindre program då de olika delarna kan operera självständigt. Att strukturera programmet enligt MVC gör att delarna blir mindre kopplade, de blir lättare att testa, det blir lättare att bygga ut och individuella delar kan bytas ut utan att behöva göra stora ändringar i den resterande koden. MVC innebär att man försöker att hålla det visuella(GUI:et) och datadelarna isär och använda kontrollklasser för att förmedla data mellan olika delar. Kontrollklasser styr flödet i programmet och förser GUI:et med ett sätt att fråga efter information från andra klasser.

4.3 Klient och Server

Från början hade vi tänkt oss att programmet skulle vara helt klientbaserat utan någon server. Från tidigare erfarenheter visste vi att det är svårt att komma åt enskilda datorer bakom en enhet, oftast en router, som använder sig av NAT/PAT protokollet (se kapitlet 3.3 NAT/PAT). Majoriteten av datorer använder sig av privata IP-adresser. Från utsidan av nätverket går det inte att kontakta en dator med en privat IP-adress direkt. Vad som behövs för att skapa kontakt mellan dem är ett mellansteg. I de flesta fall gäller det en dator som agerar som server. Med hjälp av en server kan datorer koppla upp sig mot varandra genom att använda servern som mellansteg.

Det första som behövde göras var att skapa en kontakt mellan två noder. Vi studerade hur sockets fungerar och dess begränsningar. Servern behöver använda en ServerSocket. ServerSocketen lyssnar efter anslutningar på en port. När en anslutning sker på servern skapas en socket via ServerSocketen och binds till en ny tråd. Med hjälp av den nya socketen kan en InputStream och en OutputStream skapas. Med dessa strömmar går det att skicka och ta emot serialiserade paket mellan noderna.

För att klienterna skulle kunna kommunicera med varandra lagrades varje klient till en början med dess användarnamn och socket på servern. Detta fungerade bra vid testning då det var lätt att etablera kontakt med en annan klient. Problemet var att när klienterna skulle bilda tvåvägskommunikation fanns det inget bra sätt för den första klienten att kommunicera tillbaka. För att ordna detta ändrades sättet servern lagrar uppkopplade klienter. Istället för att lagra en klients socket lagras hela tråden. Med denna metoden sätts trådarnas partners till varandra och kan därmed använda varandras send-metoder för att kommunicera. Detta förbättrade strukturen samt tog hand om problemet där kommunikationen endast fungerade en väg.

För att skicka data till den andra noden användes till en början en klass som hade en paketvariabel och en konstruktor för varje pakettyp. Ett problem vi upptäckte var att det var problematiskt att skala upp det då nya konstruktorer med fler datafält och nya get-metoder behövde läggas till. Vägen runt det var att använda sig av varargs. Varags tillåter en metod att ta emot ett obestämt antal argument som metoden sedan kan iterera igenom. Med Javas varargs kunde en generell sendmetod konstrueras som tog emot en pakettyp och all övrig data som tillhörde den pakettypen, till exempel en sträng om pakettypen är av typ *CHAT*.

4.4 Synkronisering

För att få bättre förståelse för hur man använder Googles Diff-Match-Patch bibliotek så genomfördes tester. Meningen med testerna var att se vilka bibliotekets begränsningar var, att se hur differenserna såg ut och om det skulle vara möjligt att sända dem över nätet. Efter att vi hade bekräftat att det skulle fungera med Goog-

les bibliotek konstruerade vi klasser för att kunna synkronisera enstaka dokument. För att spara tid skapades tester som verifierade att klasserna fungerade på korrekt sätt. När vi var nöjda med resultaten från testerna byggde vi ut klasserna för att synkronisera en katalog. Tester skrevs även för att verifiera de nya funktionerna.

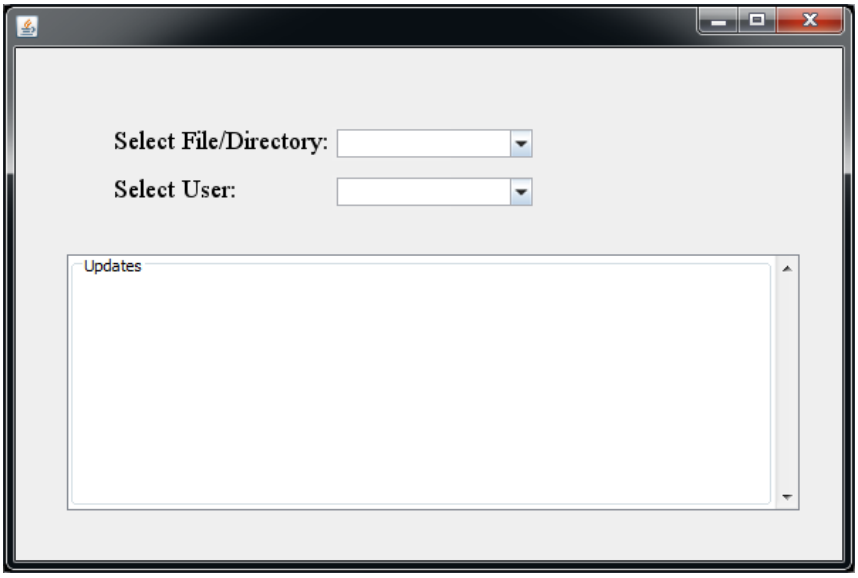
När synkroniseringsklasserna kändes färdiga började vi undersöka hur man modifierade temporära filer från olika textredigeringsprogram. Det visade sig att nästan inga textredigeringsprogram använder temporära filer utan har ej sparad text i RAM-minnet. Detta var problematiskt då programmet inte skulle vara användbart om det bara kunde hitta ändringar efter att man sparat ett dokument. Ett annat problem är att majoriteten av textredigeringsprogram kan märka av ifall texten har ändrats utanför dess miljö. Varje gång vårt programmet patchar in en ny differens måste användaren godkänna det och ladda om dokumentet. Det blir då upp till användaren att bestämma ifall egna icke sparade ändringar är viktigare än de som ens partner har skrivit. För att få det att fungera enligt målet, att användarna ska kunna skriva och se vad den andra skriver utan avbrott, valde vi att skriva programmet som ett insticksprogram till en IDE. Anledningen till det valet var att alla stora IDE:er har stöd för att hämta och modifiera filer som är öppna via dess editor. Eftersom Eclipse är en populär IDE samt att vi själva använde den under utvecklingen av programmet valde vi att skriva insticksprogrammet till Eclipse. Endast några få redigeringar behövde göras för att se om ett dokument är öppet i Eclipse eller om det inte är det.

4.5 GUI

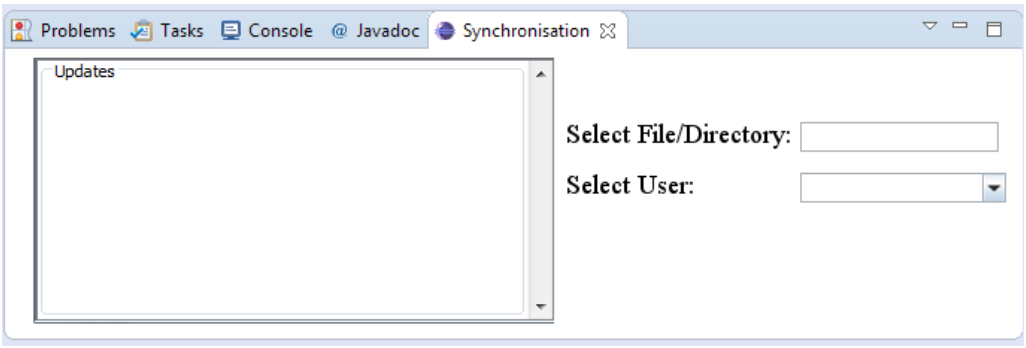
GUI:et utvecklades tidigt för att få en bättre idé om vilka funktioner som behövdes i programmet. GUI:et gick igenom tre iterationer innan det ansågs vara färdigt.

Första iterationen av GUI:et
Den första iterationen av GUI:et var simplistisk och innehöll endast två fält och en panel att visa uppdateringar i. Det första fältet var till för att låta användaren välja vilken fil eller katalog användaren vill synkronisera. Det andra fältet var till för att välja vilken användare användaren vill synkronisera mot. Figuren 4.2 visar hur GUI:et var tänkt att se ut.

Andra iterationen av GUI:et
Den andra iterationen kom som en konsekvens av bytet från en självständig applikation till ett insticksprogram. Eftersom ett insticksprogram har möjligheten att vara en del av IDE:ens huvudfönster ville vi att vårt program skulle vara en del av Eclipse redigeringspanel. Mer specifikt ville vi att när insticksprogrammet startas så skulle vårt program initieras som en ny flik intill Problems, Tasks, Console och Javadoc. Fliken skulle ha samma innehåll som det föregående GUI:et. Figuren 4.3 visar hur GUI:et var tänkt att se ut.



Figur 4.2: Illustration av första iterationen

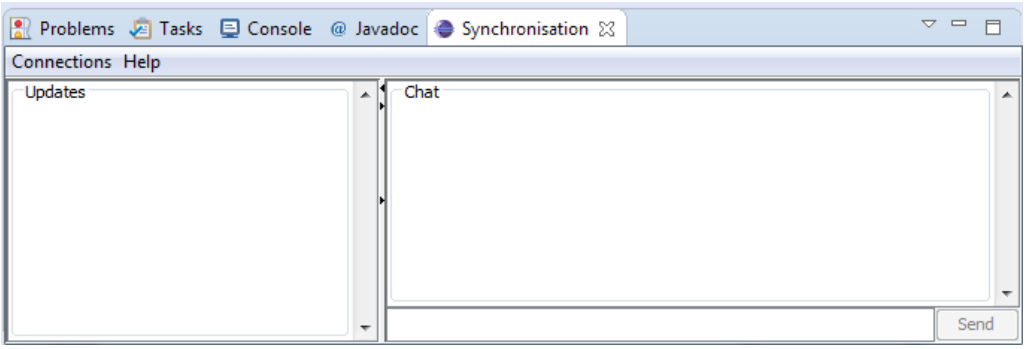


Figur 4.3: Illustration av första iterationen som en del av IDE:en.

Under utvecklingen av andra delar av programmet upptäcktes fler uppgifter som GUI:et behövde behandla, till exempel upp- och frånkoppling till och från servern. Vi ville även implementera en chat för att lättare kunna samarbeta under tiden programmet körs. GUI:et designades om till att komma innehålla en menyrad. Menyalternativen tillåter användaren att koppla upp sig till och från servern samt tillåter användare att ansluta sig till och från andra användare på servern. Menyalternativen som kräver indata, såsom *Connect to server* och *Connect to user*, byttes ut mot dialoger som öppnas vid menyval. Ytan som frigjordes med hjälp av menyn kunde göras om till en chat. Figur 4.4 visar hur det var tänkt att GUI:et skulle se ut.

Tredje iterationen av GUI:et

Efter mycket sökande och testande kom vi fram till att dokumentationen för hur man arbetar med Eclipses visuella komponenter inte var tillräcklig för att åstadkomma vad vi var ute efter. Bland annat fanns det inget stöd för att ha en egen meny i en flik, så det skulle behöva skrivas från grunden eller att menyvalen flyttades ner igen och återgick till att vara fält. Då gjordes den tredje och slutgiltiga iterationen av



Figur 4.4: Illustration av andra iterationen

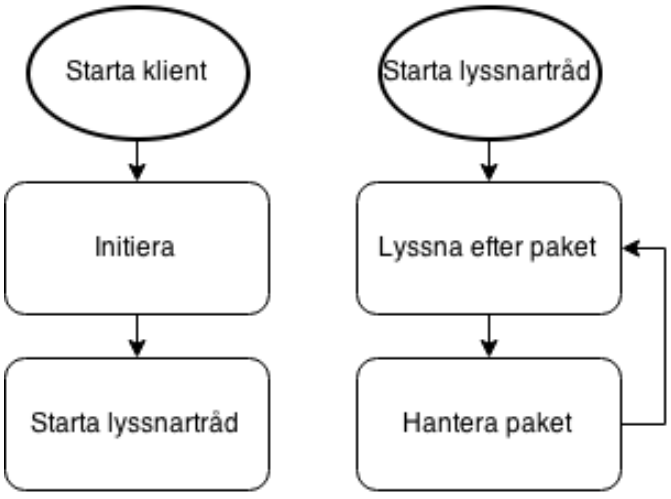
GUI:et. Vi använde den första iterationen som var ett löst fönster och uppdaterade den med hur den andra iterationen var tänkt att se ut.

5
Systembeskrivning

Programmet initieras som ett instickspogram. Instickspogrammet gör det möjligt att komma åt den osparade texten i filerna som är öppnade i utvecklingsmiljön. Instickspogrammet skapar kontrollern som har hand om GUI:et, klienten och synkroniseringsklasserna.

5.1 Klient

Innan initieringen av klienten sker behöver en användare skapa en anslutning till servern. Anslutningen sker via en dialog i GUI:et där användaren får skriva in ett användarnamn, IP-adress till servern och portnumret som servern lyssnar på. Har IP-adressen och portnumret angivits vid ett tidigare tillfälle kommer de att ha sparats i en textfil och kommer vara inskrivna när dialogen öppnas. Vid ändring skrivs värdena över i textfilen. Under initieringen hämtas IP-adressen och portnumret från textfilen och en socket skapas mot servern. Efter att socket:en har etablerats skapas en ObjectOutputStream och en ObjectInputStream för att kunna skicka och ta emot objekt. En tråd skapas därefter som lyssnar efter nya paket från servern och förmedlar paketen vidare till andra komponenter i programmet. Figur 5.1 visar klientens flödesschema.



Figur 5.1: Flödesschema för klienten.

Paketen skickas med hjälp av en send-metod som tar in en Enum-typ och en varargs. Enum-typen används för att avgöra vad resten av paketet kommer att innehålla och

därmed hur paketet kommer att behandlas. Detta görs med hjälp av en switch-metod som har ett case för varje pakettyp. För att illustrera hur paketen tas emot visas ett utdrag från klientents recieve-metod i figur 5.2. Alla pakettyper kan ses i appendix A.

```
private void receive(){
    while(!receiveThread.isInterrupted()){
        try {
            List<Object> argsList = (ArrayList<Object>) in.readObject();

            switch((Packets)argsList.get(0)){
                case GRANTACCESS:
                    Controller.getInstance().getLauncher().serverGrantedConecction();
                    break;
                case DENYACCESS:
                    Controller.getInstance().getLauncher().serverDeniedConnection();
                    break;
                case NEWUSER:
                    Controller.getInstance().updateUserList(Packets.NEWUSER, (String)argsList.get(1));
                    Controller.getInstance().getLauncher().updatePane("User " + argsList.get(1) + " connected to the server.");
                    break;
                case DISCONNECTUSER:
                    Controller.getInstance().updateUserList(Packets.DISCONNECTUSER, (String)argsList.get(1));
                    Controller.getInstance().getLauncher().updatePane("User " + argsList.get(1) + " disconnected from the server.");
                    break;
```

Figur 5.2: Utdrag från klientens receive-metod.

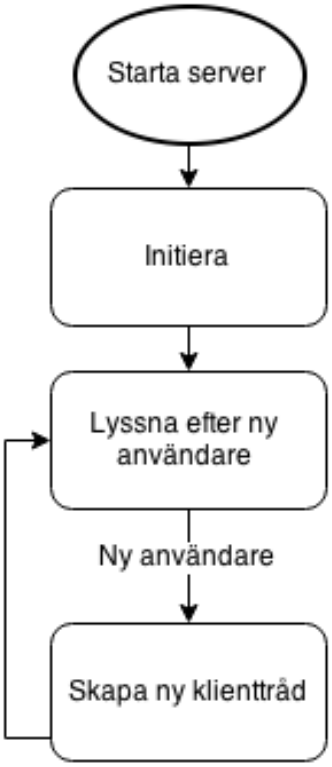
5.2 Server

Servern är ett separat program som inte behöver köras på samma dator som klienten. Detta var ett designval vi gjorde då det är vanligt att placera servrar på särskilda enheter som är anpassade för att köra servrar. Vid serverns initiering skapas en lista för att hålla reda på alla klienter som den ansluter under programmets gång.

Efter initiering lyssnar servern efter nya anslutningar. När en ny klient skapar en anslutning till servern så skapar servern en ny ClientThread. ClientThread är en trådklass som tar hand om kommunikationen med klienten tills klienten kopplar bort sig från servern eller tills servern stänger ner tråden. När servern stängs ner stängs alla klienttrådar för att undvika problem med klienternas instickspogram. Figur 5.3 visar serverns flödesschema.

5.2.1 ClientThread

ClientThread är utvecklad enligt figur 5.4. En ClientThread är en trådklass som servern skapar för att behandla en klient när en ny anslutning sker. Det första tråden gör är att lyssna efter ett paket som innehåller användarnamnet som användaren angivit. Tråden söker igenom serverns lista med användare och om användarnamnet är upptaget kommer trådklassen att stängas ner för att spara resurser. Om användarnamnet däremot är ledigt kommer tråden, tillsammans med användarnamnet, att läggas till i serverns lista. Användarnamnet blir då upptaget och klienten blir tillgänglig för andra klienter att koppla upp sig mot. Tråden är då etablerad och



Figur 5.3: Flödesschema för servern.

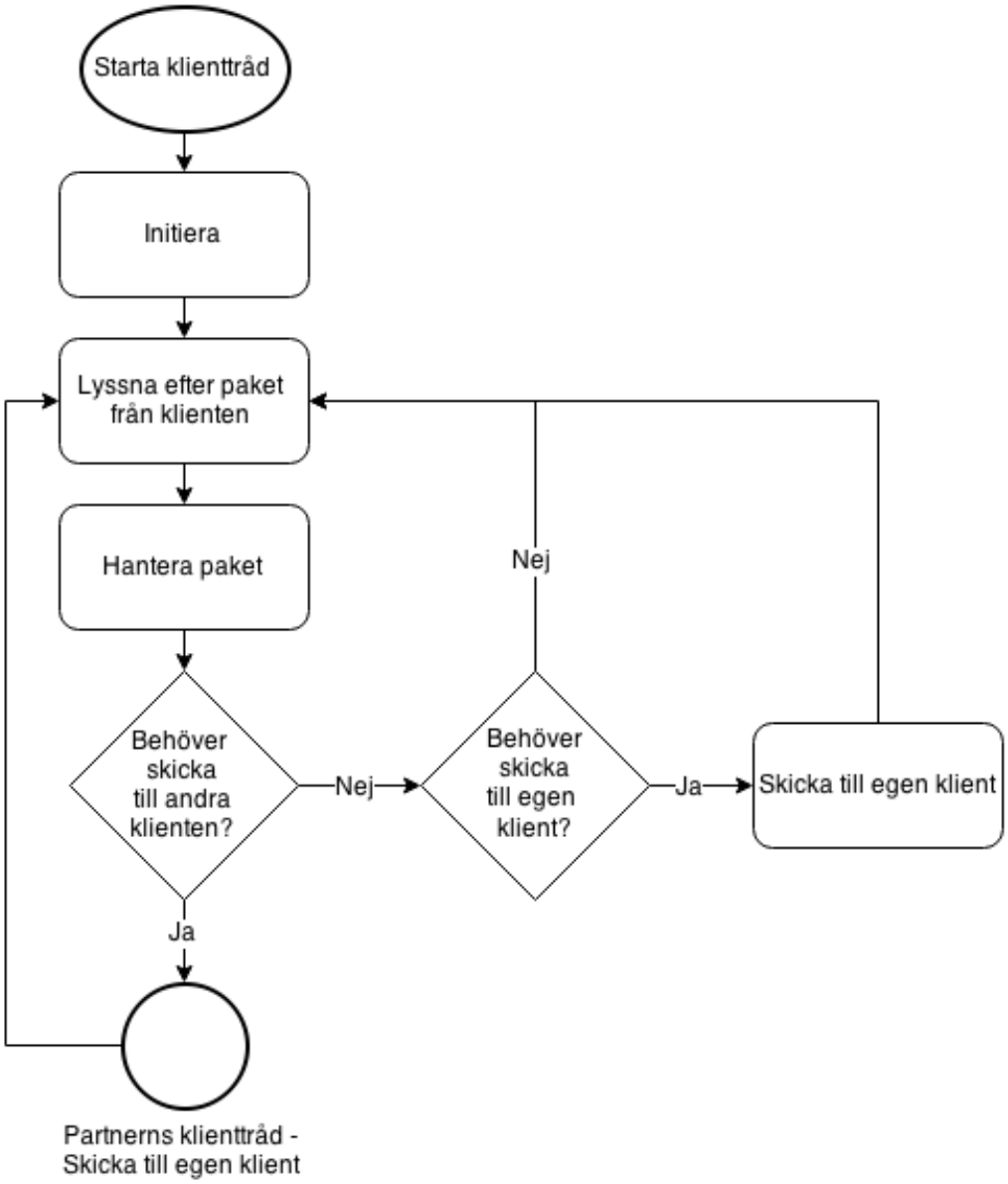
väntar på fler meddelande från klienten.

För att två klienter ska kunna kommunicera med varandra måste båda klienterna vara uppkopplade mot servern och vara lediga, det vill säga inte redan kommunicerar med en annan klient. Den ena klienten börjar med att fråga den andra om de kan börja kommunicera. Användaren som får förfrågningen kan neka eller acceptera kopplingen. Om användaren nekar så skickas ett meddelande som antyder detta och det förmedlas till den förfrågande klientens användare. Om användaren däremot accepterar kopplingen så kan klienterna börja kommunicera med varandra. Under uppkopplingen mellan klienterna kan klienterna skicka chattmeddelanden och differenser för de delade filerna. När en av klienterna vill koppla ifrån meddelar den det till den andra klienten så att den kan stänga ner funktioner på sin sida.

När en klient vill koppla bort sig ifrån servern skickar den ett slutligt meddelande som berätter det. ClientThread meddelar då alla andra uppkopplade klienter så att de inte försöker kommunicera med den nerstängda klienten. Slutligen plockar tråden bort användarnamnet från serverns lista så att andra klienter kan använda sig av namnet.

5.3 Synkronisering

För att synkronisera filer används två klasser. Huvudklassen är SynchroniseRoot som utför majoriteten av arbetet för synkroniseringen medan SynchroniseDocument

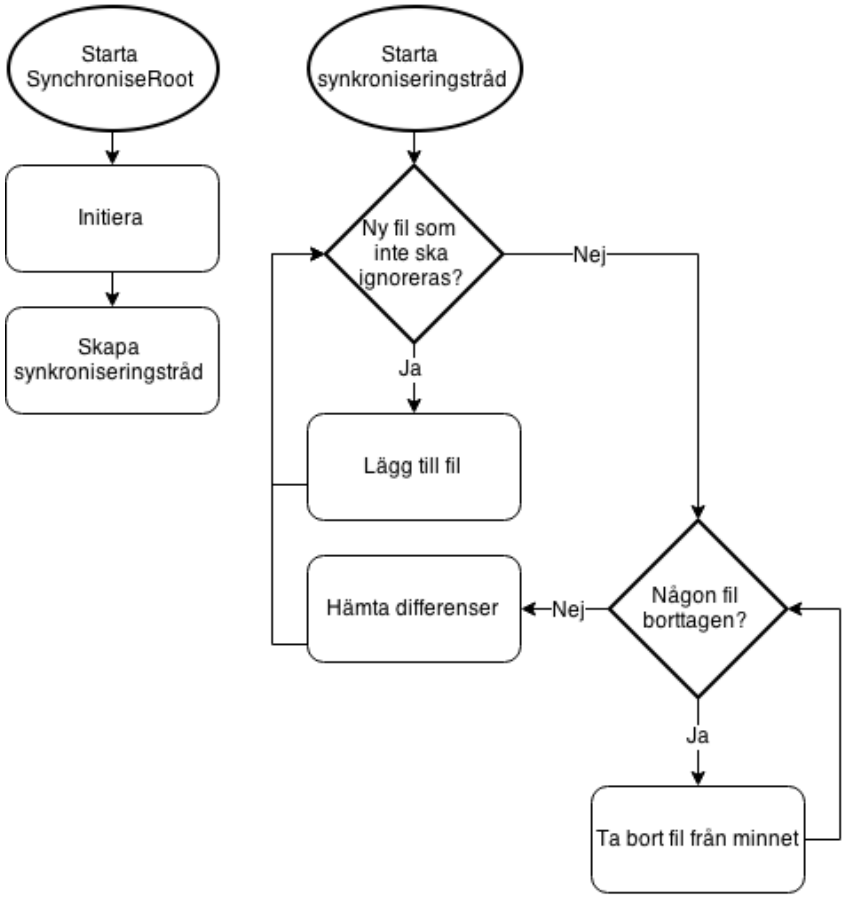


Figur 5.4: Flödesschema för serverns klienttråd.

är en hjälpklass som underlättar genom att spara data och differenser för varje fil som ska synkroniseras.

Under initieringen tar SynchroniseRoot emot en Path, vilket är ett klassobjekt som representerar en filväg, som säger vilken mapp eller fil som programmet skall hålla reda på samt en lista med filer som SynchroniseRoot skall ignorera inuti katalogen. Listan är till för att kunna ignorera filer som inte behöver synkroniseras, detta kan vara binärfiler som varierar mellan varje kompilering. Om ingen lista tillkommer ignoreras inga filer. För att hitta alla filer som ska synkroniseras itererar programmet igenom katalogen och om filen inte finns på ignoreringslistan skapas ett nytt SynchroniseDocument för filen. Datan som för nuvarande finns i filen sparas till klassen så att det finns en utgångspunkt att räkna ut skillnader mellan filens versio-

ner. När alla filer i katalogen har lagrats skapas en ny tråd. Den nya tråden består av en evighetsloop som periodiskt inspekterar ifall användaren har tagit bort eller lagt till filer och hämtar differenserna för alla filer i katalogen. Flödesschemat för synchroniseringen är given i figuren 5.5.



Figur 5.5: Flödesschema för Synkroniseringen.

För att programmet ska räkna ut differenserna för en fil måste två versioner av filen finnas, en äldre version och en nyare. Den äldre versionen är datan som finns i filens SynchroniseDocument medan den nyare versionen läses in direkt från filen. Om filen är öppen i Eclipse så hämtas datan där ifrån. Efter att differensen mellan de två versionerna är beräknade uppdateras filens SynchroniseDocument med datan från den nyare versionen. Med den här lösningen registreras bara ändringarna som användaren har gjort mellan varje anrop av metoden. Koden för hur differensen hämtas är givet i figur 5.6.

Programmets patch-metod klistrar in ändringar och en lista som in-parameter. Listan innehåller alla ändringar som ska göras på en fil och vilken fil det ska göras på. Eftersom användare kan skapa nya filer i katalogen måste patch först alltid försäkra sig om att filen existerar. Om filen inte existerar gör inte patch någonting då det är nätverkets uppgift att se till att den skapas. Om däremot filen existerar hämtar patch filens differenslista från dess SynchroniseDocument och gör om listan till en

```
SynchronizeDocument file;

// Is the file opened in Eclipse? If so read the data from there
String eclipseString = Util.openEclipseGetContent(root.resolve(doc).toString());
String modifiedString;
modifiedString = eclipseString != null ? eclipseString : Util.openReadFile(root.resolve(doc));

LinkedList<Diff> list = dmp.diff_main(file.getData(),modifiedString, true);
file.setData(modifiedString);
```

Figur 5.6: Utdrag från SynchroniseRoots getDiff metod

patchlista. Om filen är öppen i Eclipse hämtas datan därifrån. Patchen appliceras då på den datan innan den skrivs tillbaka till Eclipse. I andra fall läses datan in direkt från filen och utför samma procedur. Slutligen måste filens SynchroniseDocument uppdateras så att den inte tror att ändringen som precis patchades är en ändring som användaren själv gjort. Annars hade ändringar registrerats på nytt varje gång. Koden för hur patch utförs är givet i figur 5.7.

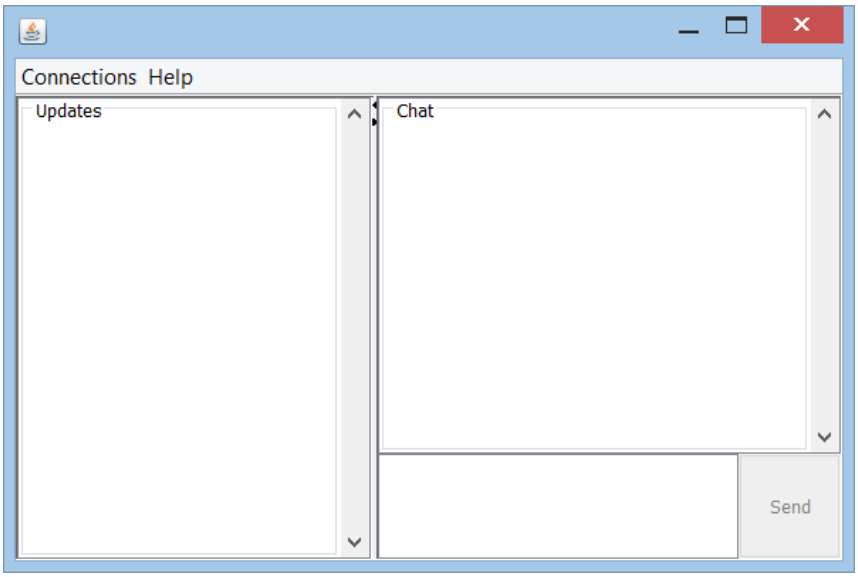
```
LinkedList<Patch> patchList = dmp.patch_make(diffList);
Object o[] = null;

// Is the file opened in Eclipse? If so patch the file through the IDE
String eclipseString = Util.openEclipseGetContent(root.resolve(doc).toString());
String s;
if(eclipseString != null){
    o = dmp.patch_apply(patchList, eclipseString);
    s = (String)o[0];
    Util.openEclipseWriteContent(root.resolve(doc).toString(), s, diffList);
}
else{
    o = dmp.patch_apply(patchList, Util.openReadFile(root.resolve(doc)));
    s = (String)o[0];
    Util.openWriteFile(root.resolve(doc), s);
}

// Update the internal file
o = dmp.patch_apply(patchList, rootMap.get(doc.toString()).getData());
s = (String)o[0];
SynchronizeDocument sd = rootMap.get(doc.toString());
sd.setData(s);
```

Figur 5.7: Utdrag från SynchroniseRoots patch metod

6
Resultat

För att starta programmet måste det först installeras. Programmet är ett insticksprogram till Eclipse och startas genom att trycka på ikonen som efter installation befinner sig på verktygsfältet.

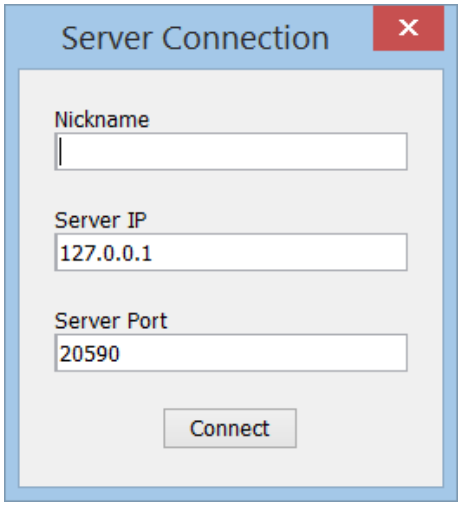
GUI:et kommer upp efter att programmet startats, se figur 6.1. Efter att ha tryckt på menyraden *Connections* finns menyvalen *Connect to server*, *Disconnet from server*, *Connect to user* och *Disconnet from user*. Från början är endast *Connect to server* valbar. Den vänstra panelen med rubriken *Updates* kommer att visa uppdateringar som till exempel att användaren anslutits till servern, att en annan användare har anslutits till servern eller att man skickat eller tagit emot en ny fil.



Figur 6.1: Huvudfönstret för programmet.

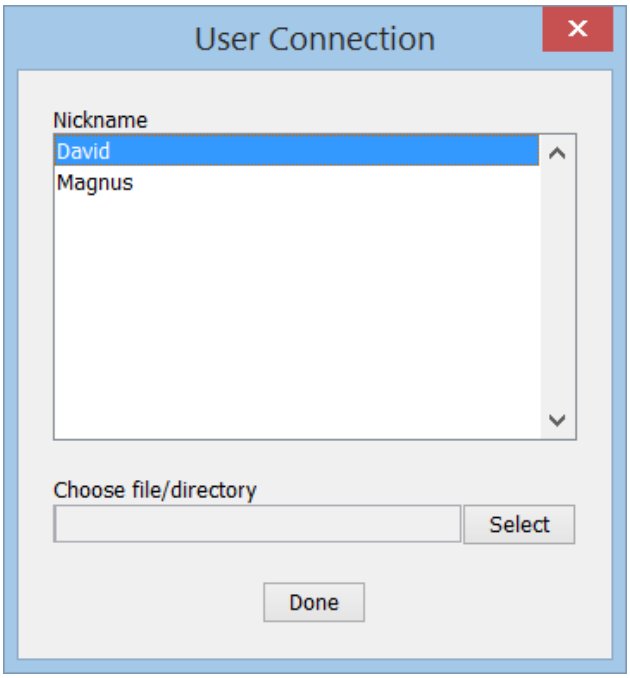
Efter att ha tryckt på *Connect to server* öppnas dialogen *Server Connection*, se figur 6.2. Där tillåts användaren skriva in ett användarnamn, IP-adressen till servern och porten som servern lyssnar på. Om dessa har skrivits in vid ett tidigare tillfälle kommer de att ha sparats och vara ifyllda när dialogen skapas.

Om allt går bra, det vill säga att servern är igång, att det är rätt IP-adress och portnummer samt att det valda namnet inte är upptaget kommer användaren att bli ansluten till servern. Efter anslutning blir menyvalen *Connect to user* och *Disconnet from server* aktiva val. För att ansluta sig till en annan användare väljs menyvalet



Figur 6.2: Server Connection dialogen.

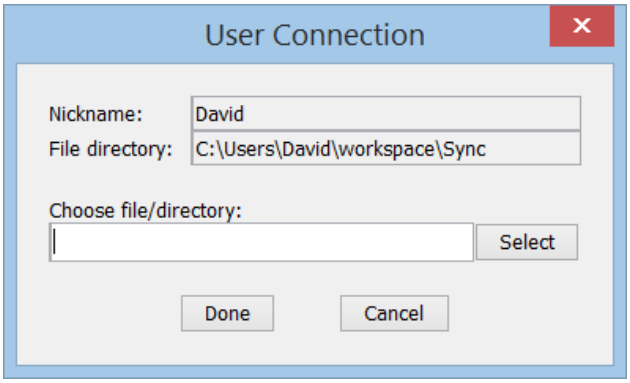
Connect to user; då öppnas dialogen som visas i figur 6.3. I dialogen visas vilka användare som är anslutna till servern. Användaren får välja en användare i listan och en fil eller katalog som kommer att synkroniseras.



Figur 6.3: User Connection dialogen.

Om en annan användare vill starta en uppkoppling kommer dialogen i figur 6.4 visas. I dialogen står det vem det är som vill starta uppkopplingen och filen eller katalogen som den användaren vill synkronisera. Valet är då att neka anslutningen genom att trycka på *Cancel* eller att själv välja en fil eller katalog och acceptera anslutningen. Om allting stämmer skall programmet fungera och de två anslutna användarna skall kunna skriva i samma dokument samtidigt i Eclipse. I huvudfönstret som kan ses i figur 6.1 är det, efter anslutning till en användare, möjligt att

skicka chattmeddelanden till den andra användaren.



Figur 6.4: User Connection dialogen på mottagande sida.

7
Diskussion

Vi har tagit fram en produkt som utför målen vi satte. Det går att koppla upp sig mot en annan användare och se koden som båda skriver i realtid i Eclipe.

Tidsplanen vi satte för oss innan projektet har vi hållit relativt bra. Vissa delar tog mycket kortare tid än vi planerat för, medan andra delar tog längre tid. Schemat vi satte från början (se appendix B) var inte helt genomtänkt. Då det var svårt att jobba på samma sak samtidigt utan att arbetet blev ineffektivt ledde det till att vi jobbade på olika delar. I och med detta blev det stor skillnad mellan schemat vi skapade för oss innan projektet och hur arbetet utfördes i verkligheten. Däremot anpassade vi oss och kunde ändå arbeta på ett strukturerat sätt.

Hade vi gjort om projektet hade vi gjort några saker annorlunda. Det första vore att planera bättre och tänka i mer detalj om vad som behövs och hur mycket tid det skulle kräva. Vi skulle tänkt på att vi inte skulle kunna jobba på samma sak samtidigt och planerat därefter. Vi hade även gjort paketstrukturen lite annorlunda. Programmet använder varargs för att skicka olika antal och olika sorters objekt (se delkapitel 5.1). Detta gör att det är kritiskt att hålla reda på vilken ordning som argumenten skrivs in i paketen såväl som vilken ordning de läses från paketen. Ett bättre alternativ hade varit att skriva en separat klass för de olika paketen med get- och set-metoder.

7.1 Insticksprogram

Anledningen till att vi gjorde ett insticksprogram till Eclipse var att vi ville läsa texten som ännu inte sparats i IDE:en. Att göra det utan ett insticksprogram hade krävt massivt arbete med minnet som det inte fanns tid för. Att istället skriva ett insticksprogram till Eclipse gav oss möjligheten att läsa texten genom Eclipse.

Resultatet är att vårt program är bundet till Eclipse och därmed även de språk som Eclipse klarar av. Nu klarar Eclipse av många av de vanligaste språken, bland annat Java, C/C++, Javascript, PHP, Python, Ruby och Pearl [17]. Eclipse har till och med stöd för att lägga till språk med hjälp av verktyget Xtext [18]. Det är däremot en stor nackdel att vara bunden till Eclipse jämfört med att inte vara det. Eclipse är inte det första valet för många utvecklare och vi hade önskat att man inte skulle behöva göra det valet för att kunna använda produkten.

7.2 NAT/PAT

Det uppstår problem om man vill skapa direkt kontakt mellan två datorer som befinner sig bakom NAT/PAT-routrar, ett problem som vi har stött på tidigare. På grund av det antog vi att det inte fanns någon lösning som kringgår problemet. Detta ledde till att vi skrev en server. Hade vi undersökt saken närmare kunde vi möjligen ha funnit en annan lösning. Det vi har hittat i efterhand är STUN (Session Traversal Utilities for NAT)[19]. Verktyget kan användas av en ändnod för att avgöra IP-adressen och porten som NAT-routern allokerat till sig. Detta upptäcktes dock väldigt sent när kommunikationsdelen av programmet redan var skriven och tid fanns inte för att undersöka om det var en bättre metod för att hantera kommunikationen.

8

Slutsats

I slutändan fick vi fram en produkt som uppfyller huvudmålet, att två personer samtidigt ska kunna editera i samma text i en IDE på varsin dator. Den största skillnaden i hur det blev och de satta målen är att det blev ett insticksprogram till Eclipse istället för ett fristående program. Visionen var att det skulle vara ett mer universellt program som skulle fungera till alla de populäraste IDE:erna.

8.1 Fortsatt Utveckling

Programmet är långt från att vara en färdig produkt. Många funktioner saknas som i sig kräver omfattande testning och verifiering. Vi har skrivit en grundversion som fungerar under givna förutsättningar. Förutsättningarna är bland annat att filerna är lika från början och att uppkopplingen är stabil så att inga större fördröjningar uppstår. För att vara en färdig produkt behöver programmet vara säkrare, mer användarvänligt och så buggfritt som möjligt. Funktioner som saknas nämns nedan.

Initial synkronisering
Programmet fungerar endast under rätt förutsättningar, detta inkluderar bland annat att filernas innehåll är lika vid start. En lösning till detta vore att programmet läser igenom filerna på båda sidor och ser till att de är lika innan den startar synkroniseringen.

Kryptering
Differenserna sänds i dagsläget i klartext. Det gör det möjligt att med hjälp av till exempel en *Man in the middle* attack att få tag i allt som skrivs mellan användare [20]. Kryptering kan vara en lösning på problemet då attackeraren inte längre skulle kunna läsa differenserna. Att implementera ett stöd för att använda asymmetriskt kryptering, alltså att man använder sig av varandras privata och publika nycklar för att både kryptera och signera differenserna, hade erbjudit ett bra sätt att öka konfidaliteten och integriteten [21].

Stödja fler användare
Grundversionen stödjer bara att två användare skriver i samma dokument samtidigt. GUI:et behöver ändras för att kunna stödja och skapa grupper samt att servern behöver ett sätt att lagra grupperna så att den kan skicka till alla användare i gruppen.

Stödja fler program
Från början var tanken att det skulle vara ett självständigt program som skulle fungera till de flesta textredigeringsprogram. Då det upptäcktes att det inte var möjligt inom tidsramen och att vi då löste problemet genom att skriva ett insticksprogram till Eclipse, betyder det att programmet är bundet till Eclipse. Vi hade önskat att det stödde fler IDE:er, men det kräver antagligen ett unikt insticksprogram och GUI för varje IDE.

Pausfunktion
Något som diskuterats är en pausfunktion. Det hade tillåtit den ena användaren att stoppa den andres ändringar för att till exempel kompilera koden. Det finns dock risk att båda användarna kan göra ändringar under tiden som programmet är pausat, som kan göra det svårt för programmet att räkna ut differensen efter återstart. För att lösa detta skulle någon versionshantering i stil med Git behöva implementeras.

GUI i Eclipse
Vi hade önskat att GUI:et var en del av Eclipse-fönstret istället för att vara fristående. Hade GUI:et varit en del av Eclipse-fönstret hade det inte varit nödvändigt att byta fönster under tiden man programmerar.

Stödja ignorering av filer
Som nämnts i systembeskrivningskapitlet 5.3 finns det implementerat att hantera en ignoreringslista, alltså en lista med filer som man inte vill ska synkroniseras. Detta är däremot inte implementerat i GUI:et då vi inte hade tid att skriva den lösningen vi tänkt oss. Lösningen kan vara att potentiellt skriva en sub-klass till klassen JFileChooser som gör det möjligt att välja flera olika filer i olika undermappar.

Ångra
Ångra-funktionen i textredigeringsprogram tar bort det senaste som skrivits. Eclipse har däremot inget sätt att skilja på om användaren har redigerat texten själv eller om texten har redigerats av den andra användaren med hjälp av vårt insticksprogram. Detta innebär att om den andra användaren har redigerat texten senast kommer ångra-funktionen att ta bort det som den andra användaren skrev. En lösning till detta behöver implementeras innan det blir en färdig produkt.

8.2 Potential

Tanken är att programmet ska tillåta och stödja ett närmare samarbete när det gäller programmering än vad som är vanligt idag, men det finns även fler användningområden. Några scenarion där insticksprogrammet skulle vara användbart beskrivs nedan.

Vanligt samarbete
Då två personer kan jobba i samma text går det att antingen samarbeta med en del eller arbete på varsin del. Arbetar programmerarna på samma del går det att få ut samma fördelar som vid användning av parprogrammering, men med insticks-

programmet är det möjligt för båda programmerarna att skriva. Parprogrammering har visats öka kvalité av design, reducera mängden fel, förstärka teknisk förmåga och öka kommunikation mellan projektmedlemmar [22]. Insticksprogrammet gör det även möjligt att använda parprogrammering på distans, vilket är positivt ur en miljöaspekt och kan samtidigt spara på resekostnader för företag.

Hjälp
Om en programmerare arbetar ensam och behöver hjälp finns det inte många effektiva sätt att få hjälp av någon på distans. Med vårt insticksprogram kan man däremot enkelt koppla upp sig mot en annan användare, dela projektfiler och få hjälp direkt. I och med att programmet skickar över filer som inte finns på båda sidor kan den ena användaren välja en tom katalog och låta programmet skicka över projektfilerna från den andra användaren. På så sätt är det ett enkelt och direkt sätt att få hjälp då båda får direkt tillgång till koden.

Utnyttja mer personal
I programmeringprojekt idag är det svårt att använda mycket personal och när det gäller projekt som är sena skriver Frederick Brooks: *"Adding manpower to a late software project makes it later."* [23]. Det beror på att det inte funnits något bra sätt att utnyttja ytterligare personal. Vi tror att vårt program skulle göra det möjligt att utnyttja mer personal utan att orsaka förseningar då mer personal kan arbeta på samma delar.

Litteraturförteckning

[1] scrumguides.org. (2014). Scrum Guide.
ONLINE
http://www.scrumguides.org/scrum-guide.html. [Besökt 8 Juni 2015]

[2] git-scm.com. (NA). About - Distributed.
ONLINE
http://git-scm.com/about/distributed. [Besökt 8 Juni 2015]

[3] Zeichick Alan. (2012). Zeichick's Take: Java, Java [Besökt 29 Maj 2015].

[4] Rally Software. (2014). Waffle.io Work Better on Github Issues.
ONLINE
https://waffle.io/. [Besökt 8 Juni 2015]

[5] Myers Eugene. (1986). An O(ND) Difference Algorithm and Its Variations. Algorithmica. 1 (1-4), 251-266. PP:2
ONLINE
http://www.xmailserver.org/diff2.pdf [Besökt 5 Maj 2015]

[6] Miller Webb och Myers Eugene. (1985). A File Comparison Program. Software-Practice And Experience. 15 (11), 1025-1040.
ONLINE
http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.189.70rep=rep1type=pdf [Besökt 22 Mars 2015]

[7] Faro Simone och Lecroq Thierry. (2010). The Exact String Matching Problem: a Comprehensive Experimental Evaluation. PP: 1
ONLINE
http://arxiv.org/abs/1012.2547v1 [Besökt 1 Juni 2015]

[8] Boytsov Leonid. (2011). Indexing methods for approximate dictionary searching: Comparative analysis. Journal of Experimental Algorithmics. 16 (1.1). PP: 2

[9] Fraser Neil. (2006). Writing: Fuzzy Patch.
ONLINE
https://neil.fraser.name/writing/patch/ [Besökt 1 Juni 2015]

[10] Pieterse Vreda och Black Paul. (2013) Algorithms and Theory of Computation Handbook, CRC Press LLC, 1999, Levenshtein distance", ifrån Dictionary of Algorithms and Data Structures.
ONLINE
http://www.nist.gov/dads/HTML/Levenshtein.html [Besökt 8 Juni 2015]

[11] Hyyrö Heikki, Fredriksson Kimmo och Navarro Gonzalo. (2005). Increased Bit-Parallelism for Approximate and Multiple String Matching. Journal of Experimental Algorithmic. 10 (2.6), 1-27.

[12] Smetanin Nikita. (2011). Fuzzy string search.
ONLINE
http://ntz-develop.blogspot.se/2011/03/fuzzy-string-search.html [Besökt 29 Mars 2015]

[13] Geoff Huston. (2015). IPv4 Address Report.
ONLINE
http://www.potaroo.net/tools/ipv4/. [Besökt 19 Maj 2015]

[14] Srisuresh och Holdrege. (1999). RFC 2663.
ONLINE
http://tools.ietf.org/html/rfc2663. [Besökt 19 Maj 2015]

[15] Miller Webb och Myers Eugene. (1985). A File Comparison Program. Software-Practice And Experience. 15 (11), 1025-1040. PP:10-11
ONLINE
http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.189.70rep=rep1type=pdf [Besökt 22 Mars 2015]

[16] Apache Software Foundation. (2004). Licenses.
ONLINE
http://www.apache.org/licenses/. [Besökt 20 Maj 2015]

[17] Sugrue James. (2008). Eclipse - So Much More Than Just Java.
ONLINE
http://eclipse.dzone.com/news/eclipse-much-more-java. [Besökt 21 Maj 2015]

[18] Eclipse Organisation. (2015). Xtext - Language Development Made Easy!.
ONLINE
https://eclipse.org/Xtext/index.html. [Besökt 21 Maj 2015]

[19] Rosenberg, Cisco, Mahy, Matthews, Wing. (2008). RFC 5389.
ONLINE
http://tools.ietf.org/html/rfc5389. [Besökt 21 Maj 2015]

[20] owasp. (2014). Man-in-the-middle-attack.
ONLINE
https://www.owasp.org/index.php/Man-in-the-middle_attack. [Besökt 28 Maj 2015]

[21] Apache Software Foundation. (2013). SSL/TSL Strong Encryption: An Introduction.
ONLINE
http://httpd.apache.org/docs/2.2/ssl/ssl_intro.htmlcryptographictech. [Besökt 29 Maj 2015]

[22] Alistair Cockburn och Laurie Williams. (2001). The Costs and Benefits of Pair Programming. Extreme programming examined, 1025-1040.
ONLINE
http://collaboration.csc.ncsu.edu/laurie/Papers/XPSardinia.PDF [Besökt 23 Juni 2015]

[23] Brooks, F (1975). The Mythical Man-Month. Reading, MA: Addison-Wesley. s25.

A

Pakettyper för nätverk

Paket-typ	Beskrivning/Innehåll
GRANTACCESS	Servern har accepterat registreringen.
DENYACCESS	Registreringen har blivit nekad.
NEWUSER	En ny användare har blivit registrerad på servern, användarlistan uppdateras.
DISCONNECTUSER	En användare har kopplat bort sig från server, användarlistan uppdateras.
STARTCOLLABORATION	En annan användare vill skapa en kollaboration. En ruta kommer då upp med användarnamnet och filen/katalogen som valts. Man får då själv välja fil/katalog och acceptera eller neka.
STOPCOLLABORATION	Avslutar en påbörjad kollaboration.
ACKUSERCONNECT	Att den man försökt kollaborera med har accepterat kollaboreringen.
DENYUSERCONNECT	Att den man försökt kollaborera med har nekat kollaboreringen.
SYNCFILE	En synkroniseringsfil som sedan används för att ändra text.
CHAT	Ett chatmeddelande som sedan visas i chatrutan.
ERROR	Skickas från servern då något fel skett tillsammans med en sträng med information om vad som gått fel.
OK	Används endast för testning.

Tabell A.1: Beskrivning av paketen som behandlas på klientsidan.

B

Gantt schema