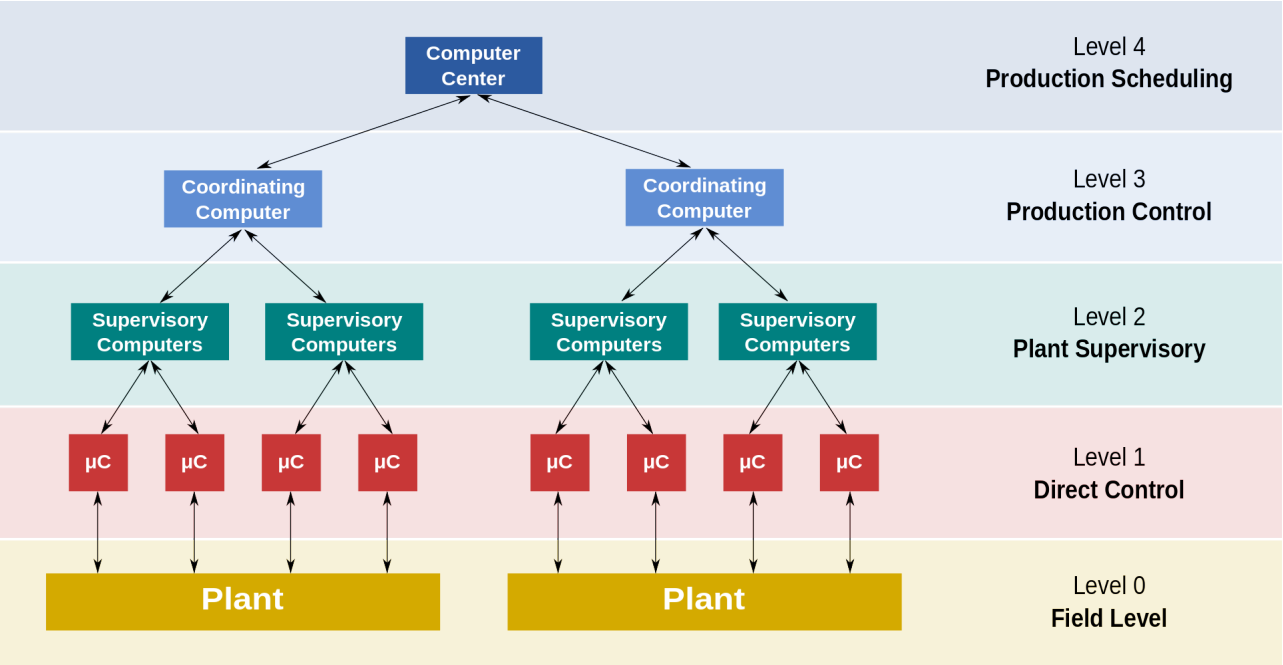
CHALMERS



Filhantering i SCADA-projekt

Versionshantering, jämförelse och kvalitetsgranskning av konfigurationsfiler i CitectSCADA

Examensarbete inom Data- och Informationsteknik

Jennifer Fredberg
Johanna Persson

Institutionen för Data- och Informationsteknik
CHALMERS TEKNISKA HÖGSKOLA
GÖTEBORGS UNIVERSITET
Göteborg, Sverige 2021

EXAMENSARBETE

Filhantering i SCADA-projekt

Versionshantering, jämförelse och kvalitetsgranskning av konfigurationsfiler i CitectSCADA

Jennifer Fredberg
Johanna Persson

Institutionen för Data- och Informationsteknik
CHALMERS TEKNISKA HÖGSKOLA
GÖTEBORGS UNIVERSITET

Göteborg 2021

Filhantering i SCADA-projekt
Versionshantering, jämförelse och kvalitetsgranskning av konfigurationsfiler i CitectSCADA
JENNIFER FREDBERG
JOHANNA PERSSON

Examinator: Peter Lundin
Handledare: Robin Bandgren

Institutionen för Data- och Informationsteknik
Chalmers Tekniska Högskola / Göteborgs Universitet
412 96 Göteborg
Telefon: 031-772 1000

Omslag: Nivåer i ett SCADAsystem
D. Pugliesi, ”Functional levels of a Distributed Control System (DCS),” 2014. [Elektronik bild]. Tillgänglig: https://upload.wikimedia.org/wikipedia/commons/thumb/1/10/Functional_levels_of_a_Distributed_Control_System.svg/1280px-Functional_levels_of_a_Distributed_Control_System.svg.png, hämtad: 2021-05-06.

Institutionen för Data- och Informationsteknik
Göteborg 2021

SAMMANFATTNING

Acobia är ett företag som arbetar med SCADA-system inom fastighetsautomation där systemen består av alla de sensorer och styrdon som är placerade i fastigheterna. Sensorerna och dess konfigurationsdata läggs löpande till i systemet. I samband med detta sker dock ingen kontroll om konfigurationsdata följer fastställda regler för metadata eller syntax. Acobia önskar en lösning som kan kvalitetssäkra filerna i deras system samt ett versionshanteringssystem för att kunna följa när ändringar skett.

Arbetet innebar att skapa en lösning som kontrollerar givna filer i ett fastighetsprojekt och som validerar om dessa följer syntax samt meddelar var eventuella avvikelser är. Ett versionshanteringssystem applicerades på projektet för att kunna jämföra olika versioner, vilket resulterar i att datan kan analyseras i efterhand. Arbetet resulterade i idéer på hur versionshantering skulle kunna implementeras hos Acobia men kunde inte hitta en bra förutsättning för hur det skulle implementeras för ett helt fastighetsprojekt i dagsläget. Arbetet har delvis skett hos Acobia men mestadels hemifrån på grund av rådande pandemi.

Nyckelord: SCADA, Versionshantering, Python, Datastruktrer

ABSTRACT

Acobia is a company that works with SCADA-systems within real estate automation where the systems contain of all the sensors and control devices that are placed in the real estates. The sensors and their configuration data are put in the system but are not immediately controlled if following syntax. Acobia requires a solution that can assure the quality of the files in their systems, and a version control system to be able to follow up when changes have been made.

The thesis work created a solution that validated given files in a real estate project in terms of following syntax where eventual deviations occur. A version control system was applied to a project to be able to compare versions, which resulted in that the data could be analysed later. The work found ideas on how version control could be implemented at Acobia but could not find a prerequisite on how to implement a whole real estate project. The work has partly been made at the office at Acobia, but mostly from home because of the ongoing pandemic.

Keywords: SCADA, Version Control, Python, Data structures

FÖRORD

Examensarbetet utfördes av två mekatronikstudenter på Chalmers Tekniska Högskola vid Institutionen för Data- och Informationsteknik under vårterminen 2021. Dessa två mekatronikstudenter har genom vått och torrt kämpat sida vid sida, det är vi Johanna och Jennifer. Det är det sista arbetet innan vi får egna vingar ut till arbetslivet. Tyvärr har rådande pandemi inneburit att arbetet inte kunnat utföras på Acobia så mycket som egentligen hade önskats. Men med teknologins hjälp har det mesta arbetet kunnat genomföras på distans.

Stort tack Sakib Sistek för alla hejarrop och vägledning i projektet. Stort tack Mats Persson för att ha agerat bollplank. Och stort tack till Robin Bandgren, vår handledare på Acobia, för din expertis.

Ordlista

Citect-SCADA - Ett SCADA-program utvecklat av Schneider Electric SA

GDPR - General Data Protection Regulation

JVM - Java Virtual Machine

Klient - Dator som är uppkopplad till en server via ett nätverk. I detta fall till en server med ett repo

Notepad - Anteckningsapplikation i Windows

Notepad++ - Texteditor för programmering

ORM - Objekt-relationell mappning

PHP - Hypertext Preprocessor

Repo - Lagringsplatsen på huvudservern vid versionshatntering där versionerna lagras

SCADA - Supervisory Control and Data Acquisition

Server - Nätansluten dator med installerat serverprogram. I detta arbete server med ett repo

Widget - Window gadget, Fönsterpryl, Användargränssnittskomponent

Innehåll

1 Inledning

I dagens uppkopplade värld sker stora arbeten för att automatisera system och processer. Acobia är ett företag som jobbar med automation inom fastigheter, infrastruktur och industri. När stora projekt ska automatiseras krävs system och enheter som samlar in data kring processer för att samla in information och kunna skapa automatisk styrning. Enheter som samlar in data uppdateras, installeras eller tas bort från systemet löpande vilket resulterar i att konfigurationsfilerna för enheterna också ändras. Det är underentreprenörer som monterar och installerar som har tillgång till dessa filer samt de som administrerar systemet och hanterar datan. Det är därmed många personer som är involverade och som kan ändra i filerna som projektet består av. För att på ett strukturerat och mer tidseffektivt sätt kunna överblicka de förändringar som sker och vem som gjort dem kan versionshantering tillämpas. Det innebär att alla ändringar kan spåras. Hos Acobia i deras fastighetsprojekt används inte något sådant system och de upplever svårigheter i att följa upp ändringarna som gjorts.

1.1 Bakgrund

Inom fastighetsautomation används ett överordnat system som kan koppla ihop och samla in data från de olika enheterna som är kopplade till systemet. I detta fall används systemet CitectSCADA. CitectSCADA delar upp installationen i projekt där ett projekt kan motsvara en fastighet. En fastighet blir då i systemet en projektmapp där filer som hör till fastigheten sparas. Filerna kan innehålla bilder, olika typer av larm men också enheternas konfigurationer. Datan i projekten kan ändras av de intressenter som har tillgång till datan vilket innebär att det är flera olika typer av personer och roller som har möjlighet att manuellt påverka systemets konfiguration. Vid den nuvarande inmatningen kan man inte säkerställa att datan matas in på ett korrekt sätt och datan bör därför granskas för att säkerställa kvaliteten. Fel inmatning kan resultera i att monterade sensorer inte används till sitt syfte och resulterar i att datan inte kan samlas in.

I nuläget kontrolleras all inmatade data manuellt och detta är mycket tidskrävande för de anställda på Acobia. För att automatisera kontrollen av datakvaliteten behövs ett skript och användargränssnitt (GUI) samt versionshantering. Kontrollen ska se skillnader mellan två filer samt kvalitetsgranska filerna mot givna standarder och definierade regelverk. På så sätt kan man få en överblick kring vad som ändrats i projekten i form utav nya larm och ändrade larmprioriteter. Det är i dessa filer som behov av automatisk kontroll och automatiskt kunna jämföra två önskade filer för att se om rätt syntax applicerats vid ändring. Kvaliteten på SCADA-systemets funktion är beroende av kvaliteten på ändringarna av filerna. Om datan i konfigurationsfilerna inte följer standard kan det bli risk att exempelvis larm inte larmar när de ska.

1.2 Syfte

Syftet med arbetet är att skapa ett verktyg som jämför och kontrollerar kvaliteten på filer för olika versioner av konfigurationsfiler i ett CitectSCADA-projekt. Kontrollen av datan automatiseras så att mindre arbetstid behöver läggas på att kontrollera inmatade och ändrade värden i projektfilerna. Utvecklingen av programmet kommer resultera i en bättre datahantering och möjlighet för Acobia att integrera versionshantering av

konfigurationsfiler i fler projekt. Med hjälp av versionshanteringen kommer tiden som tidigare användas till att manuellt granska filerna kunna användas till något mer värdeskapande.

1.3 Mål

Målet är att med hjälp av implementation av ett versionshanteringssystem och logik kunna jämföra två versioner av given konfigurationsfil och visa skillnaderna mellan dessa. Det skall även vara möjligt att kontrollera kvaliteten på filerna, det vill säga att alla prioriterade kolumner är ifyllda enligt standard.

Mål:

1. Integrera versionshantering på ett fastighetsprojekt
2. Kvalitetsgranskning genom att prioriterade kolumner är ifyllda enligt standard
3. Jämföra skillnader mellan olika versioner av konfigurationsfiler
4. En helhetslösning av de tre ovanstående målen

1.4 Frågeställningar

Frågeställningar som fanns när projektet startade beskrivs här.

1. Är det möjligt att på ett användarvänligt sätt integrera versionshantering i nuvarande arbetsflöde?
   (a) Hur integreras ett fastighetsprojekt i ett versionshanteringsprogram?
   (b) Vilket versionshanteringssystem är lämpligt att använda?
   (c) Hur ska man välja ut och jämföra två versionsfiler från en versionshantering?
   (d) Hur ska man välja ut och säkerhetsställa kvaliteten på en versionsfil från en versionshantering?
2. Vilket programmeringsspråk har egenskaper som gör det enkelt med filhantering?
   (a) Vilka paket gör språket användningsbart?
   (b) Finns det alternativa sätt att lösa problemet på?
3. Hur skapas ett användargränssnitt som visar kvalitet och skillnader mellan filer?

1.5 Avgränsningar

Här presenteras de avgränsningar som projektet gjort för att arbetet skulle kunna genomföras.

1. Endast angivna databasfiler från ett fastighetsprojekt kommer tas i beaktning
2. Tar ej hänsyn till konfigurationsfiler av typen .ctg
3. Lösningen ska ej implementeras i Acobias nuvarande system

4. Bara versionshantering på en enhet
5. Endast ett grundläggande användargränssnitt kommer göras för att visa möjlig funktion

2 Teknisk bakgrund

Under detta avsnitt beskrivs den information som krävdes för att genomföra projektet.

2.1 SCADA

SCADA är ett överordnat IT-system som kopplar ihop kontrollenheter, sensorer och andra komponenter i automatiserade processer. Det kan vara inom fastighetsautomation, infrastruktur eller en industriell miljö. SCADA-systemets funktionalitet innebär att alla uppkopplade enheter kan manövreras från samma system. All data som samlas in visualiseras och kan styras från ett ställe. Genom att koppla sensorer och flöden till grafiska vyer ger systemet möjlighet till god överblick över de fysiska och logiska miljöer som systemet hanterar. SCADA-system har även larmhantering som ger operatören en enkel, överskådlig bild var larmet kommer ifrån eftersom den är kopplad till den grafiska layouten. SCADA-system möjliggör bearbetning och tolkning av stora mängder data. Flöden för analys, beräkningar och simuleringar bearbetas för att förstå hur de ingående komponenterna fungerar och skapa beslutsunderlag för optimering och framtida förändringar [1].

2.1.1 Arbetsflöde

För att åstadkomma ett stort och komplex SCADA-system är det många personer som samverkar för att uppnå en hög effektivitet i arbetet med systemet. De driftbilder som finns bör följa samma standard så att personalen i anläggningarna inte känner sig osäkra på hur bilderna är uppbyggda. Exempel på personal som det är betydelsefullt för är drift- och felsökningspersonal som direkt kan lokalisera hus, våning och system som berörs vid larm. Enheter, som till exempel sensorer, ska planeras och bestämmas, monteras och matas in i systemet och därefter sker kontroll att de är inmatade i systemet på korrekt sätt. Beteckningsnamnen på enheterna är en viktig faktor för att underlätta framtida arbete. Filerna ska godkännas och kompileras in i projektet. En besiktning krävs av konfigurationsfilerna för att kontrollera ännu en gång att allt är godkänt. För att åstadkomma ett bra arbetsflöde krävs bra riktlinjer där alla är medvetna om vad som är viktigt och hur man ska samarbeta på ett bra sätt [2].

2.1.2 Konfigurationsfiler

I mappstrukturen som skapas vid fastighetsprojekt i CitectSCADA tillkommer olika typer av konfigurationsfiler, filerna är av typen .dbf och .ctg. Flera personer kan göra ändringar direkt i databasfilerna (dbf). Konfigurationsfilerna beskrivs nedan [2].

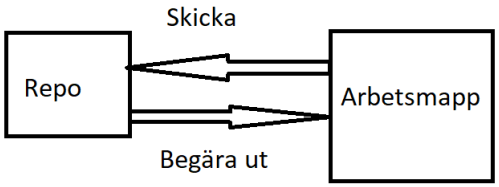
- **Variable.dbf**: Här sparas de variabler som tillkommer i projekten.
- **Tsdig.dbf**: Här finns det inbyggda larm som tidstämplas av enheten på millisekunders-nivå och används mestadels inom elkraft.
- **Advalm.dbf**: Här återfinns de mer avancerade larmen.
- **Trend.dbf**: Variabler definieras som har data som kan vara viktiga att använda i framtiden
- **Pagemenu.dbf**: Definierar trädmenyn

- **Diagalm.dbf**: Larm definieras som inte har möjlighet att kommunicera klockslag. När ett larm slås på anger CitectSCADA tidsstämpel i stället.
- **.ctg**: Sparas filen som en .ctg är det en grafik fil. Dessa typer av filer dyker upp i projekten när de sparas.

2.2 Versionshantering

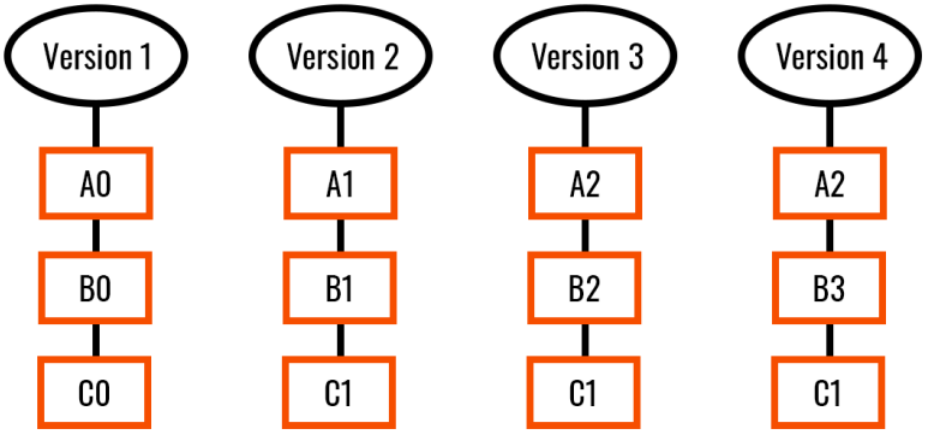
Ett system för versionshantering av filer möjliggör att kunna gå tillbaka och se ändringar som genomförts tidigare. Versioner som skapas får olika versionsnummer och det skapas ett strukturerat sätt att få en överblick på projektets framsteg som ger möjlighet att på ett effektivt sätt felsöka om problem uppstår [3].

Versionshanteringssystem exekveras på en huvudserver där datan huvudsakligen lagras. För att ta del av versionerna begärs (checkout) den önskade datan ut från huvudservern. När en version har begärts ut och sedan ändrats lokalt skickas (commit) den tillbaka till huvudservern och en ny version skapas och sparas. Till varje version lagras också vem som gjort ändringen, när ändringen gjordes och eventuellt också en beskrivande text av versionen. Lagringsplatsen på huvudservern där versionerna lagras kallas för ett repo (repository). Genom att begära ut en kopia av originalarbetet betyder det att datan som finns inte försvinner eller fördärvas när en ny kopia begärs ut och ändras. När ändringen är gjord sparas den på nytt i en ny version och på så vis kan arbetets gång följas hela tiden och för att vid ett senare tillfälle ha möjlighet att gå tillbaka och kontrollera vad som ändrats [4]. Förhållandet mellan repo och kopia visas i figur 1.



Figur 1: Koncept av versionshantering

När projektmapp skickas tillbaka till repot, sparas den momentana datan som filen innehåller. Till de filer som inte uppdaterats skapas i stället en ”länk” till den senaste uppdaterade versionen, för att minimera behovet av lagringsutrymme [5]. I figur 2 visas en förenklad bild över hur dessa versioner kan se ut med filerna A, B och C.



Figur 2: Förenklad bild över versionshantering

I figuren studeras att fil A inte har ändrats från version tre till version fyra. Detta innebär att A2 bibehåller sin länk till då den senast var ändrad, mellan version två och tre. För B som uppdateras vid varje version kommer den erhålla ett nytt nummer B1, B2 och B3. Versionshanteringen möjliggör att redigera i äldre versioner, men även att hoppa fram och ändra i de nyare versionerna.

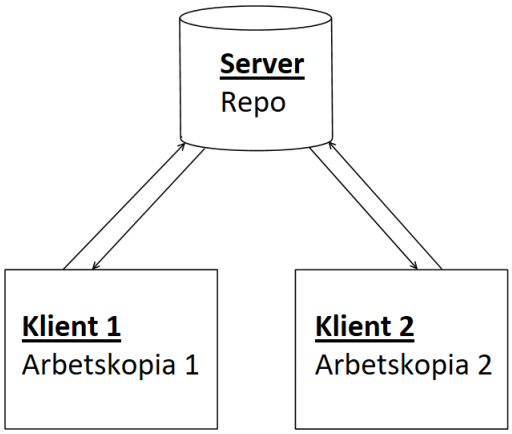
Versionshanteringssystem är i huvudsak uppbyggda på två olika sätt, centraliserade eller distribuerade. Distribuerade versionssystem innebär att projekt kan hämtas ner lokalt och köras i sin helhet på ett lokalt repo. I de centrerade systemen skrivs alla ändringar direkt på det gemensamma repot vilket innebär att klienten hela tiden måste vara uppkopplad för att kunna skicka in ändringarna i systemet. Det innebär i det centrerade fallet att om ett fel skickas in påverkas alla som har tillgång till repot medan det vid distribuerade system kan varje klient provköra så att den nya koden är kompatibel med resten av programmet innan det skickas tillbaka till det gemensamma repot [6].

Valet av centraliserat eller distribuerade system bör grunda sig på hur förutsättningarna ser ut, tex hur många som har tillgång och arbetar från repot samt hur nätverksmiljön är uppbyggd. Centraliserade system är mer lämpade om det är färre personer som jobbar på samma data eftersom ändringarna är direkta medan det distribuerade system appliceras bättre på stora projekt där många jobbar på olika delar. Det kan vara enklare att ha hög säkerhet på ett centraliserat repo än att ha flera distribuerade lokala repon som måste hålla en hög säkerhet om det är viktiga data som finns i repot. De centraliserade systemen kan vara långsammare eftersom varje gång datan skickas görs det till en annan enhet medan det för distribuerade sker det på samma enhet [7],[8].

2.2.1 Apache Subversion

Apache Subversion (SVN) är ett centraliserat versionshanteringssystem. Det innebär att repot finns på ett ställe där endast arbetskopior kan begäras ut. För att testa ändringar som gjorts på en arbetskopia skickas det tillbaka till repot och kan köras med resterande data därifrån. Inom mjukvaruutveckling skulle det innebära att om en ändring som är gjord inte är kompatibel med övrig kod eller inte följer syntax förstör det koden för de andra som arbetar med den. I och med att repot är på en annan enhet innebär det att vid varje ändring i koden som görs och ska skickas in till systemet krävs

det att enheten är uppkopplad [9]. Visualisering av struktur av ett centraliserat system visas i figur 3.



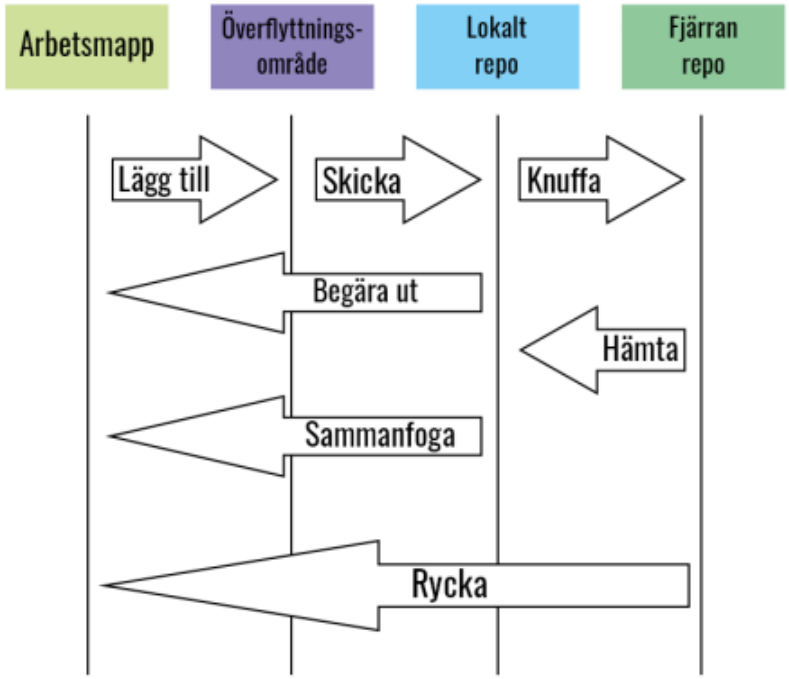
Figur 3: Förenklad bild av SVN

För att enkelt kunna använda SVN finns grafiska tillägg att installera, ett är TortosieSVN. Tillägget möjliggör att skapa arbetskopior, skicka tillbaka ändringar och se skillnader mellan textfiler som finns i repot. Genom att markera ”command line client tools” i installationen kan SVN-kommandon också användas i cmd vilket innebär att all kontakt med repot kan ske både grafiskt och via cmd [10].

2.2.2 Git

Git är ett distribuerat versionshanteringssystem (DVCS). Skaparen förklarar git som ”the stupid content tracker” [11]. Systemet är utvecklat för hantering av större projekt, men fungerar även bra till mindre projekt [12]. Git är efterföljaren, till det redan beskrivna SVN. Sedan start år 2005 har utveckling av Git fortsatt framåt och används idag av många företag [5].

Fördelarna med användning av Git är att den har en enkel design, snabb, fungerar bra för icke linjär utveckling och effektiv för större projekt. Git kan användas i cmd eller via ett GUI (GitHub desktop) [5]. Arbetsflödet för användning av Git beskrivs i figur 4. Det finns fyra olika steg i arbetsflödet, arbetsmapp (working directory), överflyttningsområde (staging area), lokalt repo (local repo) och fjärran repo (remote repo). Det går att redigera filerna lokalt innan de läggs upp i fjärran repo, och sedan sammanfoga (merge) med det som gjorts innan eller som någon annan klient ändrat.



Figur 4: Förklaring över Git

Som det går att studera i figuren är arbetsmappen platsen där alla mappar och filer finns som tillhör projekten i repot. Vid ändringar läggs dessa i överflyttningsområdet, denna ses som en zon mellan arbetsmappen och det lokala repot som studeras i figuren. Genom att skicka tillbaka ändringarna läggs dessa i ens lokala repo. För att alla användare skall kunna få de senaste ändringarna används knuff (push) till fjärran repo. Men hjälp av hämta (fetch) skickas de senaste ändringarna till det lokala repot [13].

2.3 Batchfiler och cmd

Kommandotolken (cmd) i Windows kan köra skript som utför handlingar vid tillfällen som önskas. Skripten skrivs i notepad eller notepad++ som är kompatibla med textfiler i Windows. Filerna kallas för batchfiler och kan utföra uppgifter som kopiera eller flytta filer. Genom att spara filen med ändelsen .bat markeras filen som körbar för Windows. Skript kan användas för diverse enkla filhanteringsutövningar eller stänga av enheten.

Med en plugin från TortoiseSVN kan cmd också användas för SVN kommandon vilket möjliggör att hanteringen av SVN kan ske mer automatiskt. Detta går sedan att koppla ihop med programmeringsspråket Python. Eftersom .bat filer kan hanteras i Python går det även att använda dessa SVN kommandon för att skapa en integrerad helhetslösning med versionshantering och ett programmeringsspråk [14].

2.4 Virtuell maskin

En virtuell maskin (VM) är ett program som körs på en dator med ett eget operativsystem. Programmet skapar en simulerad eller virtuell datormiljö i vilken det går att installera ett annat operativsystem. Detta kan vara användbart om man har

behov av ett annat operativsystem men inte vill installera om sin dator eller om man vill prova något utan att påverka sin vanliga dator. Den virtuella maskinen använder hårdvaran som finns på datorn men virtualiseringsprogrammet hanterar kommunikation mellan den virtuella maskinen och hårdvaran. Den fysiska maskinen kan vara på samma enhet eller på en annan enhet som nås via nätverk [15],[16]. En VM fungerar bra även om hårdvaran slutar fungera. Det är då enkelt att flytta över programmet med innehållet till en annan enhet och är då inte beroende av att den gamla hårdvaran behöver fungera igen [2],[17].

2.5 Python

Python är ett interpreterande programmeringsspråk som kan användas för webbutveckling, mjukvaruutveckling, GUI, matematik, nätverks- och systemskripts. Språket beskrivs ofta som ett skriptspråk. Python kan till exempel läsa och ändra i filer samt koppla upp sig mot databaser. Programmeringsspråket har likheter med det engelska språket vilket gör inlärningströskeln låg.

Med hjälp av språkets enkla syntax kan utvecklare skriva program med mindre rader kod än i andra programmeringsspråk. Koden som skrivs kan exekveras direkt, vilket gör det enkelt att arbeta med prototyper [18].

Python är fördelaktigt att använda om hantering av cmd och SVN önskas eftersom det är möjligt att använda cmd- och SVN-kommandon direkt i Python. Språket är även uppbyggt på sådant sätt att den kan användas på olika plattformar [19].

2.5.1 Pandas

Pandas är en tilläggsmodul utvecklad för att hantera och ändra datastrukturer på ett enkelt sätt. I Pandas kan datastrukturer hanteras genom Series eller DataFrame. DataFrame är en datastruktur som är uppbyggd som en två-dimensionell matris, som visualiseras i figur 5. I matrisen kan det finnas värden av olika datatyper. En DataFrame kan skapas genom att skicka in exempelvis en dbf- eller excel-fil och skapa ett DataFrame-objekt av filen. Detta ger möjligheten att använda objektets olika funktioner på objektet som skapats av filen. I en DataFrame kan datan nås vid given kolumn och rad där ett visst värde finns [20].

	A	B	C	D	E
0	1	2	3	4	0
1	6	7	8	0	10

Figur 5: Visualisering av ett DataFrame-objekt

I figuren, går det att se att kolumnerna namngavs som A till E och raderna 0 till 1. DataFramen skapas med hjälp av kommandot pd.DataFrame('Tabell - namn'). Då kan attribut som "iloc[ ]" användas, där olika positioner kan erhållas i en DataFrame och tabellinformationen blir därmed lätthanterlig.

2.5.2 Tkinter

För att skapa ett grafisk utseende kan biblioteket tkinter användas. Tillägget gör det möjligt att skapa ett grafiskt gränssnitt till Pythonkoden. Det finns andra moduler i Python som kan skapa ett grafiskt gränssnitt men tkinter är det som har flest användare [21]. Tkinter möjliggör att kompilera en applikation till en binärfil, det vill säga att göra programmet körbart utan tillgång till Python-tolken på enheten [22].

För att få tillgång till att grafiskt välja mapp i Pythonprogrammet från utforskaren går detta att göra genom att hämta filedialog-funktionen från tkinter. Ett kodexempel på hur tkinter importeras och möjliggör att öppna mappar direkt från enheten visas nedan.

```
import tkinter as tk
from tkinter.filedialog import askdirectory
```

I tkinter finns widgets som input-rutor, knappar, skroller, textrutor och mycket mer. Dessa kallas för GUI element. Elementen kan användas och manipulerats på olika sätt, som exempelvis ändra färg på knappar, ändra font och så vidare [23].

2.5.3 Andra moduler

Här beskrivs andra moduler som möjliggör hantering av data och versionshantering i Python.

- **shutil:** Shutil är ett tillägg som möjliggör att hantera filer och mappar i Python. Shutil kan bland annat kopiera, flytta och ta bort filer [24].
- **dbfread:** För att kunna läsa databasfiler kan modulen dbfread användas. Den kan exempelvis avkoda en fil om den innehåller tecken som inte kan läsas samt öppna dbf-filer [25].
- **SVN remote:** SVN modulen möjliggör att hantera SubVersion genom Python kod. Python i sin tur använder koden i cmd vilket medför att SVN-tillägg för cmd måste vara installerad på enheten [19].
- **PySVN:** Ett tillägg i Python är PySVN som möjliggör att skriva SVN-kommandon i Python koden som hanterar repot. Detta är fördelaktigt för att skapa ett alternativ för användaren för att kunna påverka vilka filer i repot som ska hanteras [26].

2.6 Alternativa programmeringsspråk

Alternativa programmeringsspråk till Python beskrivs i följande avsnitt.

2.6.1 Java

Programmeringsspråket Java är ett objektorienterat språk. Java och Python har länge konkurrerat kring vilket som är populärast bland mjukvaruutvecklare. De två olika språkens syntax skiljer sig åt då Java har striktare syntaxregler än vad Python har. Vid deklaration av variabler i Java måste anges vilken typ dessa är av. Detta kan innebära

att det kan upplevas svårare för nybörjare att lära sig syntaxen, medan andra anser att det ger en större tydlighet [27].

En annan egenskap med Java är att den med hjälp av JVM kan köras på alla maskiner, vilken även Python har möjlighet att göra [28]. Vid webbutveckling används Spring och Blade i Java, där Spring är det mest använda. En av fördelarna av användning av Spring vid webbutveckling är att det finns ett stort nätverk av användare, samt att ett flertal företag använder sig av detta. Pythons backend webbutvecklingsverktyg Django, är både säker och robust. Även om Spring också är ett robust alternativ vid applikationsbyggnad på en företagsnivå, är Django det som idag används mest. En av Djangos fördelar är att den integrerar bra med databaser och utför operationer på olika typer av data, tack vare ORM-lagret [27]. Java har också bibliotek som möjliggör att använda SVN kommandon i koden [29], samt öppna och läsa dbf-filer [30].

2.6.2 PHP

Ett annat programmeringsspråk som används till webbutveckling och annan programmering är PHP. En av fördelarna är att den integrerar bra med HTML. Som med Java och Python har språket många användare vilket gör att det finns mycket material att tillhandahålla. En fördel med PHP är att den laddar webbsidor snabbt, och kan ladda ungefär tre gånger så snabbat jämför med Python [31]. PHP har tillägg som kan hantera SVN [32].

2.6.3 SQL

För att hantera relationsdatabaser används SQL. Det är ett verktyg som organiserar, hanterar och hämtar data som lagrats av en databas. Det är ett standardiserat programmeringsspråk. Språket i sig är inte ett databashanteringssystem, utan en integrerad del och ett verktyg för att kommunicera med en databashanterare (DBMS) [33]. Några av dessa databashanterare som använder SQL är Oracle, Microsoft Accsess och MySQL [34].

3 Metod

Metoden har delats upp i fem delar för att förtydliga vad som behövs för att genomföra projektet. De fem delarna är faktainsamling, versionshantering, jämförelse, kvalitetsanalys samt integration.

3.1 Faktainsamling

Först görs en datainsamling om hur företaget önskar att slutprodukten skall se ut, och därmed förstå vilken uppgift som ska lösas. Miljön som aktuella projekt körs i är CitectSCADA och därav krävs informationssökning om programmet för att förstå hur datan processas. Det krävs också kunskap i hur inmatning sker och hanteringen av datan ser ut i dagsläget då den matas in av underentreprenörerna som installerar enheterna i fastigheterna. Eftersom det är efterfrågat att implementera en versionshantering av konfigurationsfiler krävs kunskap inom området, då ingen i projektet har erfarenhet av det sedan tidigare. Det krävs förståelse för vilket sätt filerna klarar av att bli hanterade samt vilket sätt som är hållbart för användarna och systemet. Faktainsamlingen sker genom intervjuer med personal på Acobia, informationssökning om programmen samt analys av programvaror som möter kraven på informationsinhämtning och anpassning till uppgiften.

3.2 Versionshantering

Det finns flera olika versionshanteringssystem på marknaden. Genom att undersöka, prova och utvärdera de dominerade systemen och förstå hur de är uppbyggda och fungerar skapar det förutsättningar för att applicera ett system på ett fastighetsprojekt. En viktig aspekt som ska undersökas är svårighetsnivån på programmet så att lösningen på projektet lyckas hanteras i en rimlig omfattning av nybörjare. Vidare sker undersökningen med hjälp av intervjufrågor till en anställd hos Acobia för att veta hur versionshanteringen ska appliceras på deras system.

Ett par olika versionshanteringssystem jämförs och utvärderas för att se vilket av dem som passar bäst till projektets syfte. Det krävs god kännedom om hur den valda versionshanteringen fungerar och vilka förutsättningar som finns för att använda systemet, till exempel tillgång till nätverk. Resonemang bör föras kring säkerhetsaspekter eftersom det kan vara känsliga data som sparas i systemet. Analys ska göras för att säkerställa att rätt förutsättningar finns för att kunna integrera versionshanteringen med övrig funktionalitet. Slutligen behövs även mängdträning i versionshanteringssystemen så att kunskap fås kring dess strukturer och hur verktygen fungerar, samt en samlad förståelse om för och nackdelar med olika system.

3.3 Jämföra versioner

För att få en förståelse för hur filerna ser ut och hur dessa kan hanteras bör arbete göras med hantering av enkla filer i utvecklingsmiljön. Testprojekt skapas med enkla filer för att enkelt kunna analysera vilken metodik som krävs för att kunna jämföra två filer. Tester kan utföras på dessa såsom ändring av syntax för att enkelt analysera vad som händer. Efter att hanteringen av enkla filer lyckats kan detta appliceras på större och mer komplexa filer.

Givet är att filerna som ska hanteras är av typen dbf. Till en början hanteras textfiler eftersom den endast är uppdelad i rader och därav bör denna simpla fil hanteras först för att lära sig programmeringsspråket och utvecklingsmiljön. En undersökning behöver sedan göras för att komma fram till hur dbf-filerna kan hanteras och hur datan skall kunna läsas och redigeras. Det behöver undersökas om det finns tillägg i valt programmeringsspråk som kan hantera sådana filer och vilka funktionaliteter som finns. Data från en given position behöver erhållas på specifika platser i filen och då måste ett sätt hittas där det går att ange vilken position som önskas och hämta ut värdet därifrån. Även stora filer med många poster behöver också undersökas för att säkerställa att även dessa går att jämföras. För att kunna studera filer behöver en undersökning göras för att bestämma lämpligt programmeringsspråk, vilka tillägg och om det finns olika sätt att hantera filen.

3.4 Kvalitetsanalys av filer

Då filernas kvalitet skall granskas innebär detta att innehållet i filerna måste kontrolleras. Datan som ska kontrolleras står angivet i ett styrdokument från Acobia. Samma programmeringsspråk, tillägg och utvecklingsmiljö ska användas för att kvalitetsgranskas samt jämföra versioner.

Ett program skrivs genom att starta med enklare filer och enbart leta efter en cell i filen. Sedan kan detta program appliceras på mer komplexa filer samt där fler celler ska kontrolleras.

3.5 Integration

En helhetslösning ska implementeras bestående av versionshantering, jämföra och kvalitetsgranska versioner.

Funktioner och egenskaper bör under arbetets gång valideras om dessa är kompatibla med resterande funktioner i de andra utvecklingsområdena. Eftersom programmet ska kunna integrera de tre olika delarna är det av vikt att dessa kan samverka. Integreringen bör göras steg för steg för att undvika oreda och förvirring i koden.

4 Genomförande

I genomförande beskrivs hur projektet fortgick för att lösa uppgiften att implementera versionshantering, jämföra versioner samt kvalitetsgranskning av projektfiler.

4.1 Informationssökning

Ett frågeformulär utarbetades för undersöka Acobias önskemål och klargöra var informationssökningen skulle startas. Olika tankar kring vad uppgiften innebar samt vilket tillvägagångssätt som skulle användas var frågor som konkretiserades och lades till i frågeformuläret. Det skapade en gemensam bild för de deltagande i projektet för hur uppgiften var utformad och skulle angripas. Några av frågorna var:

- Vad är CitectSCADA och dess funktion, kan programmeringen ske direkt i CitectSCADA?
- Hur fungerar versionshantering och vad innebär det?
- Hur datan i filerna ser ut och hur filstrukturen i fastighetsprojekten ser ut som ska hanteras?
- Vilket språk/miljö är det lämpligt att behandla datan i?
- Hur fungerar en virtuell maskin?

För att undersöka svaren på dessa frågor intervjuades handledaren på Acobia, samt att projektdeltagarna själva sökte svar på lämpligt sätt. Resultatet av intervjun klargjorde vilken lösning som önskades av Acobia samt gav kännedom om vilket typ av projekt och filer som ska hanteras.

4.2 Versionshantering

En analys gjordes för att undersöka vilka versionshanteringssystem som fanns på marknaden och utvärdera hur fördelar och nackdelar med dessa påverkade arbetet. Målet är att kunna jämföra två olika versioner av konfigurationsfilerna.

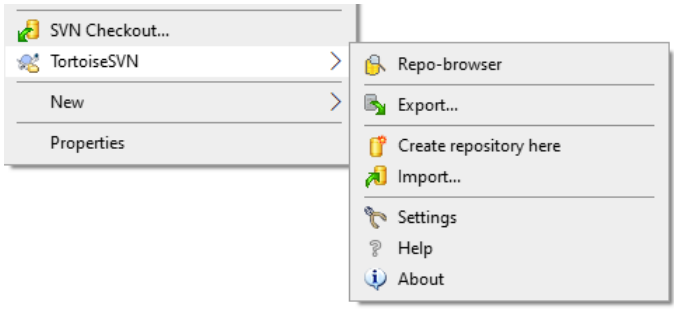
4.2.1 Val av versionshantering

Konceptet med versionshantering var nytt för projektdeltagarna och därav krävdes en informationssökning om ämnet. Metodiken var att titta på videos om Git och efterlikna miljön som sattes upp i dessa. Utifrån miljöerna kunde vidare undersökning ske och leta efter funktioner och möjligheter att använda programmet. Funktionerna som undersöktes gjordes med enkla textfiler för att det skulle vara enkelt att följa och analysera vad som hände och hur det hände. Vidare undersöktes Subversion Apache för att förstå hur det skiljde sig från Git. På Acobia arbetar ett fåtal personer på samma data vilket innebär att SVN är det versionshanteringssystem som är mest lämpat för ändamålet. Projektdeltagarna hade ingen tidigare erfarenhet av versionshantering och det var SVN som hade enklare nivå för nybörjare för att komma igång, därav valdes SVN. SCADA-systemet körs på en virtuell dator där det endast finns en inloggning, därmed kan alla som har access till den ändra som inloggad på samma användare på den enheten. Eftersom det endast är ett fåtal som kan ändra datan skulle det fungera

att ha en versionshantering lokalt på den virtuella maskinen och även kunna hantera datan på ett säkrare sätt.

4.2.2 Implementering av versionshantering

Det första steget vid implementering var installation av den senaste versionen av SVN samt GUI:t TortoiseSVN. En mapp skapades i utforskaren för att inleda användningen av SVN. Två undermappar skapades, genom högerklick på den ena kunde ett repo skapas, alternativen som öppnas vid högerklick syns i figur 6.



Figur 6: Alternativ vid högerklick på mapp i utforskaren när TortoiseSVN är installerad

I högerklickfunktionerna i figur 6 skapas ett repo genom att vänsterklicka på ”Create repostiory here”. I en av de två mapparna har ett repo skapats. Det kommer upp en dialogruta som frågar om en mappstruktur ska skapas och den rutan ska kryssas för. Det skapas en struktur i repot med mapparna braches, tags och trunk som är delarna som möjliggör versionshantering, se figur 7a.

Ett högerklick på den andra mappen möjliggör istället att skapa en arbetsmapp, genom ett tryck på SVN checkout som visas i figur 6. I arbetsmappen hamnar den angivna versionen från repot och det är där som ändringar i arbetet kan göras. Hela filstrukturen från repot (figur 7a) hamnar i arbetsmappen och nu kan filer och projekt läggas till samt ändras. Hur de båda mapparna ser ut efter föregående handlingar visas i figur 7b.

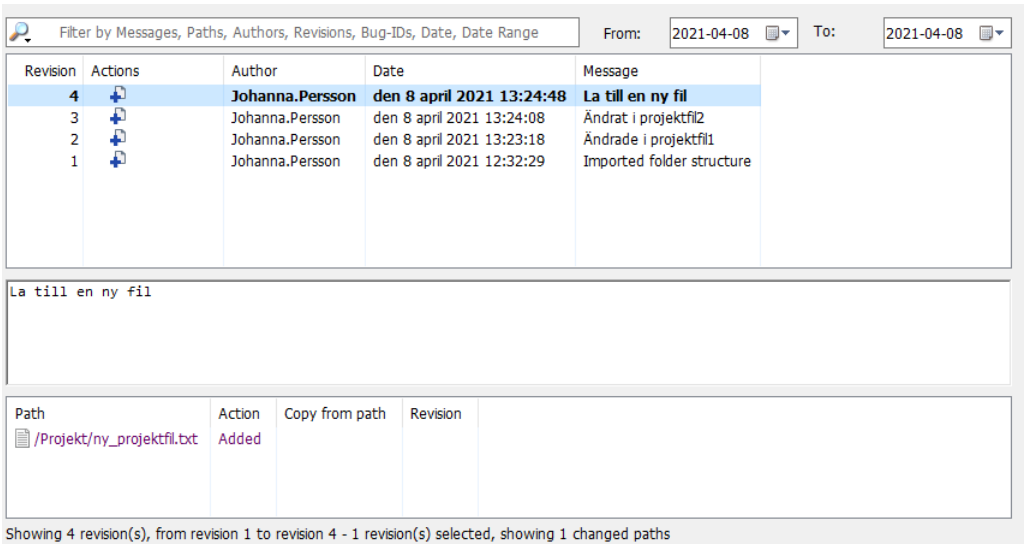
(a) Repots struktur skapad av tortousieSVN (b) Repot och Arbetsmappen

Figur 7: Repot och arbetsmapp i utforskaraen

När ett repo och en arbetsmapp skapats är det fördelaktigt om de inte är placerade i närheten av varandra i filstrukturen, alltså inte som de är i figur 7b. Repot bör placeras så att det är avskilt från alla arbetskopior, så att en användare inte av misstag ändrar i repot. Om det sker ändringar i repot förstörs filhanteringen. För en oerfaren användare som inte förstår principen kring versionshantering kan det vara lätt att tro att ändringar ska ske i repot. Med tanke på att det är flera personer som kommer att

hantera datan i versionshanteringen och att det är viktig data som finns där är det av högsta intresse att repot bevaras. Repot skulle kunna läggas på en separat dator och där också behörighetskontroll implementeras. Från denna dator kan endast arbetskopior hämtas och uppdateringar lämnas in.

Varje gång en ändring görs i arbetsmappen skall det avslutas med att skicka ändringen till repot. När datan skickas till repot anges vilka mappar eller filer som har ändrats, vem som gjort ändringen samt en kommentar om vad som ändrats. När ändringar gjorts och de har skickats till repot kan loggen i TortoiseSVN öppnas för att överblicka ändringarna som gjorts, se figur 8.



Figur 8: TortoiseSVNs logg-fönster

I figuren har kommentarer lagts till under ”Message”. Här är det önskvärt att underentreprenören skriver ett ID för att det skall förenkla en eventuell felsökningsprocess.

Ett mål med projektet var att jämföra innehållet i två olika versioner. Ett verktyg i TortoiseSVN som utforskades uppgav sig för att kunna jämföra två versioner. Det verktyget kunde endast jämföra två textfiler vilket gjorde att andra lösningar behövde undersökas för att kunna jämföra två dbf-filer.

Efter installation av TortoiseSVN, versionshanterades först enkla filer, både av typen text- och databasfiler. Genom att öva på att versionshantera filerna med olika kommandon, blev det enklare att förstå hur programmet hanterade större filer och projekt.

För att vara säker på att implementationen av versionhateringen fungerar för ett helt projekt, testades även detta med ett fastighetsprojekt. I projektet fanns alla intressanta konfigurationsfiler. Detta innebär att alla olika typer av filer som existerar i ett verkligt fastighetsprojekt används även här. Genom denna kontroll kunde slutsatsen dras att det är möjligt att föra över och följa ett helt projekt i TortoiseSVN.

Ett krav var att programmet ska kontrollera versionerna om alla angivna kolumner enligt standard är ifyllda, exempelvis om ett larm inte är korrekt ifyllt kommer detta inte kunna larma när den ska. För att kunna jämföra två filer i projektet som är i olika

versioner behöver dessa kunna begäras ut från versionshanteringen. Installationen av TortoiseSVN kördes igen för att kunna installera tillägget för att manövrera SVN i cmd. Detta möjliggör att skapa ett skript som kan begära ut de önskade versionerna. Nedan visas kommandot som exporterar en kopia av repot av en given version.

```
svn export "sokvag_till_svn" namn_kopia.dbf
```

Genom att i cmd stå på den sökväg som filen ska kopieras till används kommandot för att hämta rätt version.

Versionshanteringen som implementeras fungerar endast om en person arbetar på projektet åt gången. Det är inte lämpligt att flera användare delar på samma användarnamn. Det är bättre att arbetskopiorna är separerade från där repot är.

4.3 Val av filhanteringsmiljö och programmeringsspråk

Det finns många olika typer av program för filhantering av databasfiler. I början av projektet evaluerades databasspråket SQL. Det studerades om SQL kunde användas direkt i CitecSCADA eller i ett oberoende databasprogram som Microsoft Accsess. Däremot var detta inte det mest effektiva sättet att för att hantera dbf-filer och integration av versionshanteringssystem eftersom databasfiler inte kräver användning av SQL, alltså kunde andra sätt evalueras. En efterforskning gjordes på hur cmd fungerar eftersom det går att hantera och kopiera filer där. Till en början verkade det som att det var ett bra val att hantera dbf-filer, dock var det inte det på grund av att det var svårt att hantera filerna i den dynamiska miljön. Då det upplevdes mer komplext att skriva skript som hanterar filerna på önskat sätt i cmd, även hur utdatan skulle kunna visas på ett försåtligt sätt. Därför evaluerades andra sätt att hantera detta på.

Genom att undersöka flera olika programmeringsspråk som kan hantera filer, som Java och Python kunde ett kvalitativt val göras om vilket programmeringsspråk som skulle användas. Python blev språket som använts i detta projekt för att det var enkelt att förstå och det fanns mycket information att söka på eftersom det finns många användare. Projektdeltagarna hade aldrig jobbat i Python innan så tid lades ner på att förstå hur språket fungerar och vilka fördelar det skulle innebära. När programmeringsspråket Python var bestämt användes utvecklingsmiljön PyCharm.

4.3.1 Val av filbearbetning - problem och lösning

Till en början skapades DataFrame-objekt av de inmatade dbf-filerna med hjälp av modulen Pandas i Python. Detta fungerade på små egendesignade testfiler men inte på de korrekta testfilerna. Mycket tid lades på att förstå varför det var på detta sätt, samt hur detta skulle kunna motverkas. DataFrames är fördelaktigt för att cellerna i dbf-filerna förblir i matrisformat, det blir då enkelt att loopa igenom tabellvärdena. Ett annat alternativ som studerades var att göra om filerna till csv-filer. Detta löste problemet med att öppna filerna men det blev svårt att loopa igenom eftersom datan nu bara var i rader som var separerade med komman, se figur 9.

```
#namn,efternamn,ålder
"Jennifer","Fredberg",23
"Johanna","Persson",23
"Lars","Larsson",12
"Kajsa","Kajsadotter",45
```

Figur 9: Exempel på csv-fil

När hanteringen av csv-filer blev mycket svårhanterlig, gjordes ännu mer eftersökning på hantering av dbf-filer. I programmet läses databasfiler in med hjälp av modulen dbfread. När de har lästs in korrekt av Python användas sedan pandas för att skapa DataFrame objekten. Anledningen till att de riktiga testfilerna inte kunde läsas berodde på att det fanns tecken som inte kunde tolkas av dbfread. Detta problem löstes genom att modulen uppdaterades till version 2.0.7, där det finns en inbyggd avkodare som byter ut alla specialtecken (utökade ASCII-tecken) (char_decode_errors='replace'), till frågetecken. Uppgiften i sig är enbart en kvalitetsanalys och jämförelse mellan två filer.

När det gick att läsa in komplexa filer med specialtecken, behövs en kod som letar efter önskade filer. Det är sex stycken olika konfigurationsfiler som måste hanteras, dessa ligger i en specifik mapp på datorn.

4.4 Kod

Efter att filhanteringen var avklarad och inläsning av filerna var möjlig kunde arbetet på att skapa en kod som klarar önskade mål påbörjas. I de två underrubrikerna nedan kommer genomförandet för jämförelse och kvalitetsanalysen gås igenom.

Idén bakom hur programmet ska fungera arbetades fram under tidens gång och när koden påbörjats såg planen ut enligt följande: Programmet hämtar repot som är angivet i koden och hämtar en arbetskopia av den senaste versionen. Ett fönster kommer upp som anger vilken den senaste versionen i repot är samt gör det möjligt att välja de två versioner som skall jämföras. Sen kommer utforskaren upp och i mappstrukturen läggs de exporterade versionerna i en jämförelsemapp. I mappen väljs önskad projektmapp. Därefter hämtas alla de konfigurationsfilerna som är angivna för att bearbeta datan. En ny ruta kommer upp som är indelad i tre. En del kommer visa skillnaderna mellan versionerna medan de två andra visar kvaliteten för respektive version. Genom att ha fil val väljs en önskad konfigurationsfil åt gången för att enkelt kunna överskåda datan för respektive fil.

4.4.1 SVN i Python

Till en början utvärderades om en .bat fil kunde redigeras och ändras i Python eftersom kommandon i cmd och .bat filerna var möjliga att hantera SVN. Dock fanns det tillägg i Python som kunde göra de önskade handlingarna som underlättade hanteringen av SVN. Genom att ha aktiverat alla funktioner vid installation av TortoiseSVN kan kommandon skrivas i cmd och SVN-tillägget kan då användas i Python. Tillägget möjliggör att Python agerar en extern klient som hittar repot och begär ut en arbetskopia. Koden visar hur ett en arbetskopia begärs ut från repot.

```
r = svn.remote.RemoteClient('file:///filsokvag_repo')
r.checkout('sokvag_arbetsmapp')
```

Om inget versionsnummer anges när en arbetsmapp begärs ut, hamnar den senaste versionen i arbetsmappen. För att kunna jämföra två olika versioner behöver de båda hämtas ut från repot. Genom att ange vilken version som önskas, kan den önskade versionen exporteras. För att exportera en version anges vilket repo som filerna ska exporteras från, filsökvägen var det ska hamna samt vilket nummer på version som ska exporteras. Koden som gör detta visas nedan där revision1 och revision2 är versionsnummer.

```
def export_from_svn(revision1, revision2):
    r.export('sokvag_var_den_ska_hamna', revision=revision1)
    r.export('sokvag_var_den_ska_hamna', revision=revision2)
```

Genom att hantera SVN i Python kunde mycket av filhanteringen förenklas eftersom filerna som skulle arbetas med direkt hamnade på ett angivet ställe med SVN kommandona och kunde då lätt lokaliseras av resten av koden.

4.4.2 Kod för att jämföra

Pythonprojektet startades med att kunna jämföra två enkla dbf-filer, där filerna var lika stora och utan specialtecken. Svaret som matades ut bestod av ettor och nollor, en etta betydde att raden var densamma för de två filerna och en nolla det motsatta. Eftersom filerna inte kan garanteras vara lika stora måste två olika stora filer kunna jämföras. Även mer komplexa filer som har specialtecken. Problemet i dessa filer som hanteras är grader Celsius-tecknet. Som tidigare beskrivet måste dbf-filerna avkodas och detta görs med hjälp koden nedan.

```
file = DBF('C:fil\\variable.dbf',
load=True, char_decode_errors='replace')
```

När hanteringen av enkla filer och inmatningen av stora filer fungerade startade hantering av två olika stora filer. Detta görs med hjälp av .ne i Pandasmodulen. Funktionen gör en elementär operation och jämför DataFrames-objekten efter ojämlikheter. Om en rad är olika i de två filerna kommer utdatan vara True.

Om en rad i två filer är lika kommer utdatan att vara False. Genom att sätta de rader som skilde sig åt till False användes följande kod: x = df1.ne(df2) == False. Detta gjordes för det var mer behändigt att tolka svaret, se figur 10.

	ADDR	CLUSTER	COMMENT	CUSTOM	...	REF2	TAGGENLINK	TYPE	UNIT
0	True	False	False	False	...	False	False	True	True
1	True	False	False	False	...	False	False	True	True
2	True	False	False	False	...	False	False	True	True

Figur 10: Visulisering av tomma celler i run-fönstret

I figuren visas två olika dbf-filer i PyCharms run-fönster och exempelvis skiljer sig kolumnen CLUSTER sig åt för de två filerna från rad 0 till 2.

För att skriva ut på vilka rader som det faktiskt skiljer sig åt skapades två if-villkor. Koden i figur 11 skriver ut datan med hjälp av en textwidget i tkinter, denna möjliggör att på ett grafiskt sätt visa skillnaderna mellan de två filerna. I början skrevs datan ut i run-fönstret i PyCharm.

```
def check_if_equal(df_1, df_2):
    n = 1
    x = df_1.ne(df_2) == False
    if df_1.equals(df_2):
        same = ("\nDom är samma")
        text1.insert(tk.END, same)
    if df_1.equals(df_2) == False:
        not_same = ("Dom skiljer sig på följande rader\n")
        text1.insert(tk.END, not_same)
        for i in range(len(x)):
            n = n + 1
            y = x.iloc[i].eq(False).sum()
            if y != 0:
                info = ("Rad:", n, "\n")
                text1.insert(tk.END, info)
```

Figur 11: Kod som jämför

I koden (figur 11) visas att variabel y (y = x.iloc[i].eq(False).sum()) används i ytterligare ett if-statment. Denna säger om summan av hela kolumnen är lika med noll kommer den inte att gå vidare. Däremot om summan blir större än noll kommer den skriva ut på vilken rad de skiljer sig åt. Eftersom datan som skall matas ut enbart är att veta om de två dbf-filerna skiljer sig åt, räcker det med att skriva ut på vilken rad detta sker på. Koden används som ett funktionsanrop som i Python skrivs med hjälp av en definition.

4.4.3 Kod för kvalitetsgranskning

För att kvalitetssäkra filerna, det vill säga leta efter tomma celler, görs detta först genom att leta i ett DataFrame-objekt skapad av en enklare fil. Ett sätt att lösa detta på är att leta tomma celler i ett specifikt kolumnnamn genom att sätta följande, namn = df1['NAMN'] == ''. Här har DataFrame, df1, en kolumn som kallas 'NAMN' och kollar om cellen är True respektive False. Vid True är en cell tom, denna datan används sedan för att kunna skriva på vilken rad detta sker på. Innan denna koden går att applicera på de verkliga filerna måste styrdokument granskas. Denna dokumentation studeras för att veta vilka kolumner som skall granskas i varje fil, exempelvis skiljer sig variable.dbf från advalm.dbf kolumner. När detta möjliggjorts går det sedan att applicera koden på de riktiga filerna.

Till en början var koden lång och svårläslig då det skapades 12 olika DataFrames, eftersom sex olika konfigurationsfiler ska kvalitetsgranskas samt jämföras mot varandra. Detta löstes genom att skapa en definition (figur 12) för att kunna använda koden flera gånger.

```
def check_col(df, col, table, file_pos, wid, name):
    counters = reset_counter()
    for i in range(len(df.index)):
        counters[1] = counters[1] + 1  # n
        counters[2] = counters[2] + 1  # x
        if col.iloc[i]:
            test = (
            "Rad: ", counters[1], "Kolumn: ", name, "Namn: ",
            table[file_pos].records[counters[2]]['NAME'], "\n")
            wid.insert('2.0', test)
```

Figur 12: Kod som kvalitetsgranskar

I koden (figur 12) används en räknare som återställs varje gång definitionen startas. Räknaren skapades av två anledningar. För det första kunna skriva ut rätt radnummer, då den utan räknare har startvärdet noll, men i dbf-filerna är första raden med data på plats två. För det andra kunna hämta rätt namn till rätt kolumn så att rätt data kan hamna i utdatan, se figur 13.

```
{Tomma celler i version } 26 { :
{Rad: } 5 {Kolumn: } RAW_ZERO {Namn: } Sensor_4 {
}{Rad: } 3 {Kolumn: } ADDR {Namn: } Sensor_2 {
}{Rad: } 2 {Kolumn: } ADDR {Namn: } Sensor_1 {
}}
```

Figur 13: Utdata som visar kvaliteten på en fil

I bilaga A, visas de kolumnnamn av vikt givet från styrdokumentet för konfigurationsfilen variable.dbf. I DataFrame kan inte indexering av kolumner användas för att loopa-igenom, därav krävdes definiering av alla namn på kolumnerna. I varje konfigurationsfil skiljer sig namnen på de kolumner som ska kontrolleras. För att definiera varje kolumn av vikt i alla konfigurationsfiler blev koden lång då det var svårt att hitta en autogenererande lösning.

4.5 Skapa användargränssnitt

Genom diskussioner med handledare på företaget visades en första prototyp på hur GUI:t skulle kunna se ut. Det ska vara lätt att gå mellan de sex olika filerna, och visa upp skillnader mellan två filer och tomma celler i dessa två. I ett förslag från Acobia, visades det upp bland annat i vilka filer som ändringar skett samt vad som ändrats. För att kunna skapa ett GUI som visar den önskade utdatan evaluerades flera olika lösningar.

Den första koden som skapades i Python var att utdatan skrevs över till en textfil. Ett skript skapades för att kunna skriva data som skulle presenteras till en textfil. Idén var att datan som skulle presenteras skulle sparas i en fil för att få en enkel överblick. Det

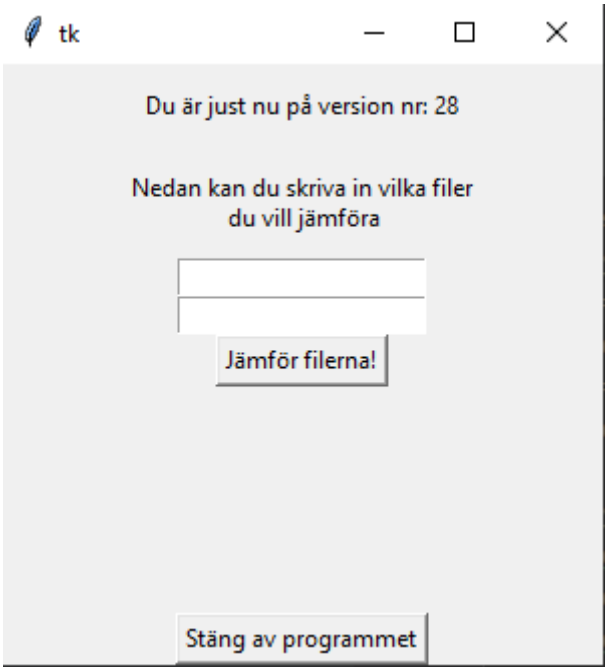
var ett enkelt sätt att hantera utdatan men det var svårt att få en bra överblick över varje fil. Därav utvärderades andra alternativ för att visa datan.

En idé var att skapa en körbar kod direkt från datorn som en applikation i form av en binärfil. Ett första test gjordes med en enkel knapp. Att skapa en applikation skulle innebära att Python inte behöver vara installerad på de enheter som använder programmet. Utifrån de förutsättningar i arbetet var detta inte möjligt då det inte verkade som att Pandas och dbfread var kompatibla med att skapa en binärfil.

4.5.1 Tkinter som användargränssnitt

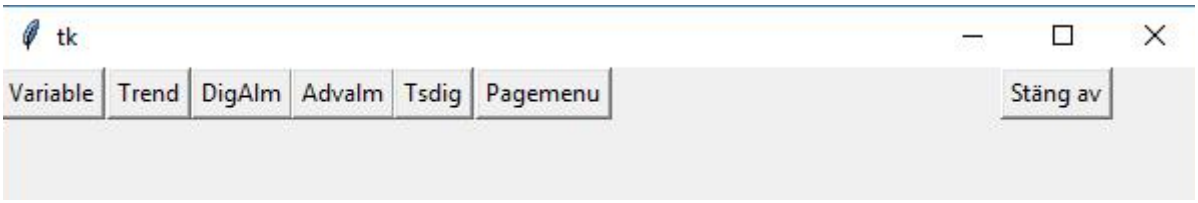
Genom informationssökning hittades Python-modulen tkinter som användas för att skapa grafiska gränssnitt. Den är enkel att implementera samt har mycket dokumentation. Det första som skapades var en enkel knapp utan någon logik bakom.

Det första GUI:t skulle bestå av sju knappar (inklusive stänga av knapp) i startfönstret, på det sättet möjliggöra att välja mellan de konfigurationsfiler som finns i projekten. Då det ska vara möjligt att ange två versioner skapades två entryboxes i tkinter, se figur 14.



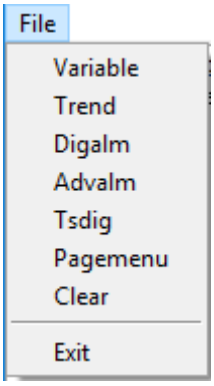
Figur 14: Programmets första fönster

För att enkelt gå mellan de olika filerna skall ett nytt fönster öppnas, där utdatan ska visas. Till en början användes knappar längst upp i fönstret som en header, se figur 15, för att möjliggöra att välja utdata för en konfigurationsfil åt gången.



Figur 15: Knappar i header

Det var komplicerat att använda tkinter-knappar på ett snyggt sätt för att bläddra mellan filerna på sättet ovan. Det var komplicerat att manipulera knapparna på det sätt som önskades, istället användes en filemenu där ingen hänsyn behövdes till placering av knappar, se figur 16. Genom att skapa en filemenu som går det att bläddra mellan de olika filerna på ett enkelt sätt.



Figur 16: Filemenu

För att kunna visa datan och bläddra igenom denna, skapades en texteditor. Datan skrevs enbart ut direkt på texteditorn. Det skapade problem då datan måste kunna bläddras igenom. Eftersom filerna består av flera rader kan det uppstå skillnader på flera ställen. I texteditorn implementerades en scrollbar som möjliggör överblick när det är många rader utdata.

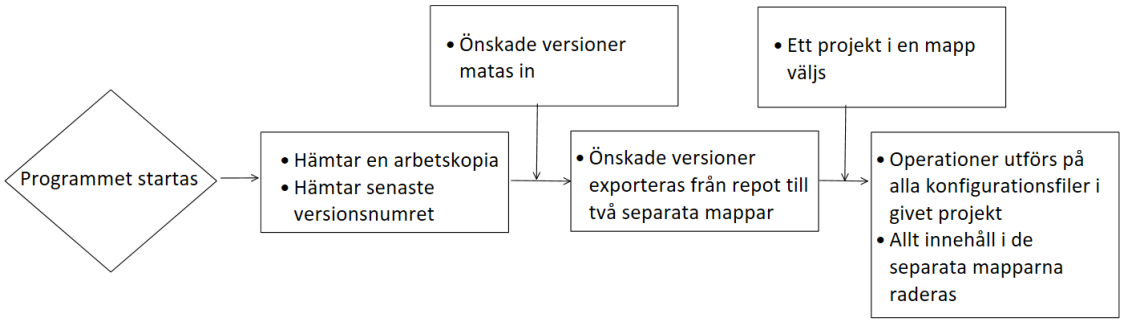
4.6 Integration av SVN, kod och användargränssnitt

I de tidigare avsnitten har utvecklingen av komponenterna samt arbetets genomförande beskrivits. De olika delarna skall fungera ihop och nedan beskrivs hur de har integrerats med varandra.

Integrering har skett i olika stadier av projektet. I det första försöket att sammankoppla de olika delarna skapades en fristående .bat kod som flyttade över och sparade filerna i en mapp som kallades ”mapp-for-compare”. Filerna sparades med en tidsstämpel för att kunna validera vilken fil som skulle vara av intresse. Batch-filen skulle förenkla kopieringen av de filer som sedan skulle jämföras i TortoiseSVN. Genom att ha filerna i en specifik mapp med ett specifikt namn kunde dessa sedan hämtas i Pythonkoden för att göra jämförelsen. Efter diskussion med Acobia och mellan projektdeltagarna beslutades att sammanställa delarna på ett annat sätt.

När det konstaterades att Pythonkod kunde användas för att kommunicera med SVN skapade det en möjlighet att enkelt kombinerad de två. Genom att först testa en

uppsättning av nästan identiska filer för jämföra, och sedan använda versioner från SVN, kunde funktionen stegvis implementeras. Det möjliggjorde att använda versionerna av de nu versionshanterade projekten i Pythonkoden. För att tekniskt kunna använda SVN-kommando och koden för att göra upprepande operationer krävdes ett flöde för att hantera filer och mappar, se figur 17.



Figur 17: Flödesschema med alla funktioner

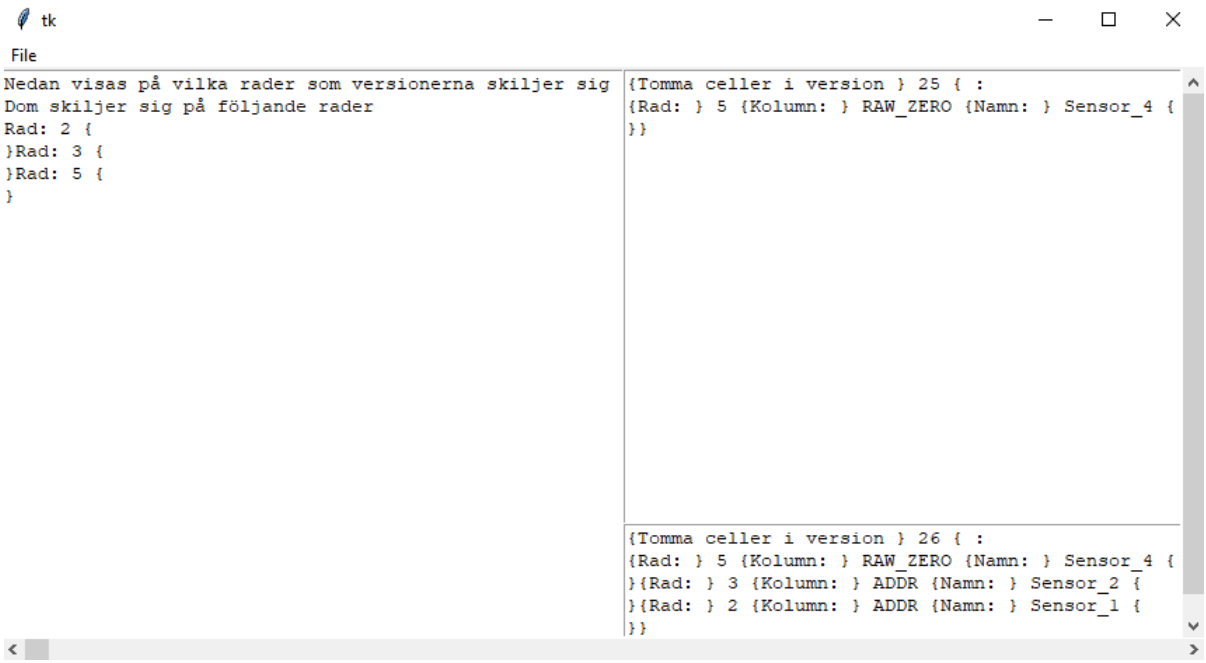
Flödet visar hur de olika komponenter integrerar med varandra, samt med ett användargränssnitt, för att kunna skapa en helhetslösning kring de givna frågeställningarna.

5 Resultat

Under projektet har ett repo satts upp för att kunna versionshantera filer. Ett fastighetsprojekt lades in i repot för att kunna versionshanteras. Repot möjliggör att gå tillbaka och undersöka när ändringar har skett och vem som gjort dem se figur 8. Versionshanteringssystemet har installerats på samma enhet som allt arbete gjorts på vilket endast möjliggjort att en klient kan skapa arbetskopior och exportera filer av repot.

Ett användargränssnitt skapades för att kunna integrera flera funktionaliteter. Programmet tar in vilka två versioner som är av intresse och exporterar dem så att ett projekt kan väljas. Versionerna matas in i första fönstret, se figur 14.

Första funktionaliteten är att två stycken versioner av samma konfigurationsfil kan jämföras för att se om ändringar skett. I det valda projekt jämförs alla konfigurations filer mot respektive i den andra versionen och skillnaden visas. Om det är något som skiljer sig åt i filerna kommer raden skrivas ut som skiljer sig samt i vilken kolumn för att enkelt identifiera vilka celler som skiljer sig åt. Utdatans karaktär visas i figur 18.



Figur 18: Andra fönstret och jämförelse, kvalitetsgranskning för dbf-filen variable

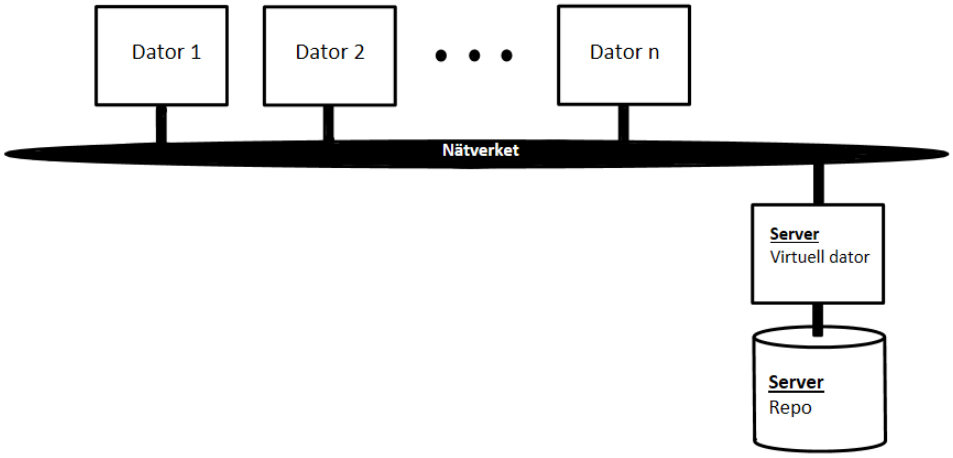
Utdatan anger även kvaliteten på filerna, som är den andra funktionen. I varje konfigurationsfil kontrolleras att de celler som enligt standarden ska vara ifyllda är det. Om en cell inte är det kommer rad och kolumn visas för att lokalisera där kvaliteten inte är bra.

6 Diskussion

Lösningen har utifrån krav som ställdes upp i början av arbetet kunna genomföras på tillfredsställande sätt. En helhetslösning har implementerats och diskuterats, där det är möjligt att jämföra och kvalitetsgranska de sex olika konfigurationsfiler av intresse. En versionshantering har på ett enkelt sätt implementerats så att versioner kan användas i koden.

6.1 SVN

Projekten som ska versionshanters finns på en virtuell dator där en användare finns med en inloggning. De fysiska datorerna loggar då in på den virtuella maskinen och där de kommer kunna skapa en arbetsmapp för att ändra datan. Strukturen visas i figur 19.



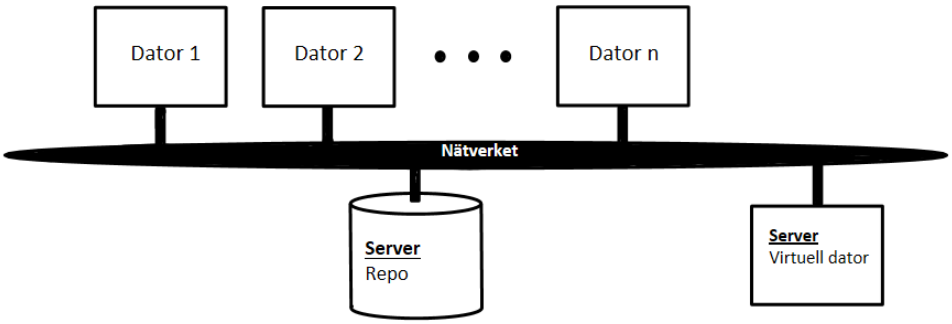
Figur 19: Visualisering hur virtuell dator är placerad i nätverket

Om SVN implementeras är den enklaste lösningen att lägga report direkt på den virtuella datorn eftersom det är där all data lagras, vilket skulle motsvara projekts angreppsätt att ha repot på samma dator som arbetet sker. Detta är dock inte att föredra då det kan medföra att flera personer är inne och ändrar samtidigt eftersom det endast finns ett användarkonto på datorn. Med enkelhet går också att nå repot som skall lämnas orört. Under arbetet var detta ett problem som uppstod och då kunde inte filerna längre användas.

Repot bör därför lagras på ett ställe som gör att användarna inte riskerar att blanda ihop report med arbetskopiorna. Detta innebära att repot inte ska lagras någonstans i användarens hemmakatalog till exempel i ”Mina Dokument” på Windows. Genom att placera repot på ett annat ställe än den virtuella datorn, minskar risken ytterligare för sammanblandning. Att placera repot på en egen server skulle möjliggöra att varje klient har en egen användare och att den är separerad från där arbetsmappen är. Då kan klienterna inte komma åt själva versionshanteringssystemet och repot om till exempel servern är låst.

För att komma ifrån problemet ovan kan repot läggas på ett annat ställe, till exempel en annan server, där kan filerna begäras ut till respektive dator. Då kan klienterna inte

komma åt själva versionshanteringen på repot om förslagsvis servern är låst. Då kan en lösning skapas där en administratör har tillgång till servern med repot men att medarbetarna endast kan begära ut arbetskopior. Det skapar också möjlighet att begära ut kopior till respektive dator istället för att ändra på samma virtuella dator. Detta minskar risken att förstöra det verkliga projektet. Hur det skulle kunna se ut visualiseras i figur 20. En sådan lösning skulle innebära en stor omstrukturering i hur arbetet sker kring arbetet i CitectSCADA hos Acobia. Egenskaperna hos ett versionshanteringssystem samt hur åtkomst sker till datan skulle behöva utvärderas för att på ett bra sätt integrera versionshanteringen på deras CitectSCADA-projekt.



Figur 20: Visualisering hur virtuell dator är kan placeras i nätverket

Som visas i figuren önskas att versionshanteringssystemet skall köras på en speciell server varifrån arbetskopiorna kopieras. Då har den virtuella datorn access till repot och dess data. Då är den virtuell datorn sin egen enhet som accessar repot utan att ha tillgång till all data, utan endast det som har möjlighet att ändrats.

SVN:s centraliserade system har fördelen att den kan vara mer säker än Git eftersom repot endast lagras på ett ställe och då säkerheten kan koncentreras dit. Datan i filerna kan vara känsliga och skall inte spridas vidare. När valet mellan dessa två gjordes var det viktigt att ha säkerheten i åtanke, därför var SVN att föredra i detta projekt.

Problematik som rör all versionshantering är när fler kopior begärs ut av samma version och sedan ska skickas tillbaka. Då måste de olika ändringarna sammanfogas, och i värsta fall riskerar viss data att skrivas över, exempelvis om två klienter lagt till data på samma rad. Eftersom arbetet endast sker av ett fåtal personer behöver det inte vara ett problem om alla är medvetna om situationen. Då skulle förslagsvis ett schema kunna göras när det är ok att ändra, för att slippa svåra tekniska situationer.

6.2 Programmering och kod

Programmeringsspråket som valdes var Python, vilket kändes som ett bra val då filer skulle hanteras. De tre huvuddelarna var möjliga att skriva i programmeringsspråket som valdes. Hade annat språk använts hade kanske inte denna möjlighet funnits. Däremot hade problemet kanske kunnat lösas med att använda ett annat programmeringsspråk som Java. Valet föll på Python utifrån den enkla syntaxen, samt att det finns mycket dokumentation.

Till en början löstes problemet på ett annat sätt och bestod av flera olika delar, detta innan integrering av SVN i Pythonkoden. Den första lösningen bestod av en extern

batch-fil, som kopierade och flyttade över filer från TortoiseSVN, och ett externt Python-program som jämförde och kvalitetsgranskade dbf-filerna. Med kunskapen från den första lösningen skapades en större förståelse för hur problemet skulle lösas på ett bättre sätt. En av frågeställningarna var om det fanns alternativa sätt att lösa problemet på, vilket testades genom att undersöka olika alternativ. Genom att använda lösningen med SVN-kommandon i Pythonkoden blev lösningen väl integrerad och enkel att hantera.

Som tidigare beskrivet i genomförande byttes alla specialtecken ut till frågetecken. De specialtecken som fanns i filerna var tecknet för grader Celsius. Det kan ställa till med problem om filerna som skall jämföras innehåller fler specialtecken, som till exempel om inmatningsfel sker eller att det är ett nytt specialtecken som matas in, eller om specialtecknet har någon form av betydelse. Det skulle innebära att två likdana tecken har samma ersättningstecken vilket gör att filerna inte kan jämföras på ett säkrat sätt. Om en enkel lösning hade funnits för att undvika att ersätta specialtecken med frågetecken, skulle det gjorts, men risken bedömdes som låg att detta skulle påverka.

De konfigurationsfiler som ska undersökas har olika kolumner som behöver kontrolleras. Eftersom innehållet i filerna är divergerande blir programkoden också lång och svårläslig. Om mer programmeringserfarenhet hade funnits hade detta problem kunnat lösas på ett effektivare sätt. Genom att istället vara mycket noggrann med att kommentera koden och lägga upp koden på liknande sätt för de sex olika konfigurationsfilernas definitioner blev det lite mer lättläst.

En förutsättning för att versionshanteringen ska fungera i den programmerade lösningen. När ett projekt läggs in i versionshanteringen krävs att alla konfigurationsfiler finns där i, på grund av att alla konfigurationsfiler kontrolleras varje gång ett projekt väljs i koden. Detta för att undvika att en version väljs som inte är komplett för då kraschar programmet. Med stor sannolikhet är inte detta ett problem om Acobia vill integrera en sådan lösning, men det var problematisk i första delen av utvecklingen av programmet.

6.3 Användargränssnitt

Användargränssnittet som utformats är i sin yttersta enkelhet och har bara det nödvändigaste. Då ingen av projektmedlemmarna tidigare designat ett GUI, är det svårt att veta vad som är enklast för en användare. För Acobia fungerar användargränssnittet bra om det är fåtal projekt åt gången som ska kontrolleras, men kan vara jobbigare att använda under en längre tid. Är det många skillnader mellan versioner, kan det bli svårt att överblicka innehållet i utdatan. Om det däremot är några få skillnader samt inte så många tomma celler, är det tillgängligt och användarvänligt att använda programmet.

Det är enbart standardutseendet i tkinter som använts. En fördel med att använda tkinter är att den har flera GUI element som kan manipuleras genom att exempelvis ändra font, ändra bakgrundsfärg och ändra färg på texten. Under arbetsgången var inte det största fokuset ett ”snyggt” GUI utan det låg på koden bakom GUI:t, som tidigare beskrivits. Hade det funnits mer tid skulle mer efterforskning gjorts kring hur ett användarvänligt gränssnitt faktiskt ser ut, vilka färger, vart knappar ska sitta och så vidare. En tanke som diskuterades var om det skulle vara bättre med större knappar,

samt även om det skulle vara fet text för att förtydliga vissa ord.

Nu används en filemenu, som är användarvänligt att använda för att gå mellan de olika filerna. Däremot även här påverkas att kunskaper saknas inom att skapa ett användarvänligt GUI. Exempel på tankar som diskuterats angående GUI:t visas nedan.

- Vad är det bästa för Acobia?
- Hur mycket tid kommer spenderas i applikationen?
- Är användargränssitett tillräckligt tydligt?

Som visas i figur 14 kommer ”måsvingar” med vid utskrivning i textwidget. Det hade varit enklare att studera utskriften om dessa inte hade kommit med. Även om funderingar diskuterades hur GUI:t skulle kunna förbättras, blev det slutgiltiga GUI:t fungerande och tillfredsställande för ändamålet.

6.3.1 Webbapplikation

För att använda programmeringslösningen på olika enheter, evaluerades flera förslag, till exempel som att skapa en webbapplikation. En webbapplikation kan skapas genom att starta en IIS-server för att hosta en webbserver lokalt på datorn. Genom att integrera detta med Django kan en enkel webbplats skapas för att visa utdata. Fördelen med en webbplats är att flera har möjligheten att nyttja denna, istället för att ha Pyhton installerad på varje enskild enhet. Att integrera detta hade tagit för lång tid för examensarbetet, men Acobia kan integrera det själva om de så önskar.

6.4 Miljö, ekonomi, juridik och etik

Ur ett miljöperspektiv, går det inte säkert att dra slutsatsen om lösningen påverkar miljön positivt eller negativt, utifrån den fakta som projektet innehar. Det går dock att reflektera utifrån flera olika perspektiv på hur lösningen skulle kunna påverka miljön. Ett sätt att reflektera kring är att mindre energi kommer användas eftersom en dator behöver arbeta under en längre tid när kontrollen sker manuellt. Däremot måste Python laddas ner på redan befintliga arbetsdatorer och köras direkt därifrån, där det inte finns någon uppfattning om hur energiåtgången kommer skiljas i de båda fallen. Larm som inte är konfigurerade på rätt sätt har en energiförbrukning även då enheterna inte fungerar som de ska.

Anledningen att Acobia ville skapa denna lösning var för att minska den arbetstid som det idag tar att jämföra och kvalitetsgranska filerna. Tiden kommer istället kunna läggas på annat arbete vilket drar in mer pengar till företaget.

Datan för fastigheterna är känsliga handlingar och bör ej hanteras av obehöriga. Juridiskt sett skulle det innebära att säkerheten bör vara omfattande för att säkerställa att inga intrång görs. Det skulle kunna innebära att information om anställda hos underentreprenörer och på Acobia skulle kunna spridas om ID skrivs när datan skickas till repot, vilket är emot GDPR. GDPR innebär skydd för fysiska personer och hantering av deras personuppgifter, detta är viktigt för att bibehålla en persons integritet [35]. Acobias kund kräver en hög säkerhet för att informationen endast ska behandlas av behöriga.

Utifrån ett etiskt perspektiv kommer förhoppningsvis konflikter att minskar mellan uppdragstagarna på fastigheterna men även på Acobia. I nuläget sker det konflikter i form av vem som gjort ändring i konfigurationsfilerna som kan ha blivit fel. Med versionshanteringen kommer förhoppningsvis arbetarna på fastigheterna att skriva in sitt ID och då blir det inga konflikter om vem av underentreprenörer som gjort vilken uppdatering, utan det enkelt går att felsöka om någon gör systematiska fel.

7 Slutsats och förslag på vidareutveckling

En versionshantering har implementerats och versioner från denna har används till vidare funktioner. Kod har skapats som kvalitetsgranskar filer samt jämför skillnader mellan två versioner som visualiseras i användargränssnitt.

För vidare utveckling av versionshantering bör analys göras av hur Acobias förutsättningar ser ut för att implementera en versionshantering som motsvarar de krav som finns. De bör ta hänsyn till var den ska vara placerad och på vilket sätt versionshanteringen ska användas i deras arbete.

Användargränssnittet som skapas bör göras så att inte PyCharm behöver vara igång när programmet ska köras alternativt att skapa en annan applikation för att använda den logik som skapats. Det skulle kunna innebära ett skapa en webbplats eller ett körbart program. Arbete skulle även kunna göras med hur utdatan presenteras.

8 Källor

[1] Schneider electric, ”CitectSCADA,” UÅ. [Online]. Tillgänglig: https://www.se.com/se/sv/product-range-presentation/60288-citectscada/ (hämtad: maj 2021)

[2] R. Bandgren, privat kommunikation, apr. 2021.

[3] Microsoft Inc., ”Introduktion till versionshantering”, UÅ. [Online]. Tillgänglig: https://web.archive.org/web/20131219113923/http://office.microsoft.com/sv-se/windows-sharepoint-services-help/introduktion-till-versionshantering-HA010021576.aspx (hämtad: maj 2021)

[4] K. Azad, ”A Visual Guide to Version Control,” BetterExplained, [Online]. 2007. Tillgänglig: https://betterexplained.com/articles/a-visual-guide-to-version-control/ (hämtad: maj 2021)

[5] S. Chacon, B. Straub, Pro Git: Everything you need to know about Git, 2. uppl., USA: Apress, 2014. [Online]. Tillgänglig: https://github.com/progit/progit2/releases/download/2.1.303/progit.pdf, Hämtad: maj 2021

[6] B. Collins-Sussman, B. W. Fitzpatrick, C. Michael Pilato, Version Control with Subversion: For Subversion 1.7, Sebastopol, CA, USA: O´Reilly, UÅ. [Online]. Tillgänglig: http://svnbook.red-bean.com/, Hämtad: maj 2021

[7] Nulab, inc., ”Git vs. SVN: Which version control system is right for you?,” 2020. [Online]. Tillgänglig: https://backlog.com/blog/git-vs-svn-version-control-system/ (hämtad: maj 2021)

[8] UO, ” Subversion vs. Git: Myths and Facts,” 2016. [Online]. Tillgänglig: https://svnvsgit.com/ (hämtad: maj 2021)

[9] Perforce Software, Inc, ”Git vs. SVN – What Is The Difference?,” 2018. [Online]. Tillgänglig: https://www.perforce.com/blog/vcs/git-vs-svn-what-difference (hämtad: maj 2021)

[10] THE TORTOISESVN TEAM, ”About TortoiseSVN,” UÅ. [Online]. Tillgänglig: https://tortoisesvn.net/about.html (hämtad: maj 2021)

[11] L. Torvalds, ”Initial revision of ”git”, the information manager from hell”, Github, [Online], apr. 08, 2005. Tillgänglig: https://github.com/git/git/commit/e83c5163316f89bfbde7d9ab23ca2e25604af290 (hämtad: maj 2021)

[12] MindRoad, ”Git för utvecklare,” 2020. [Online]. Tillgänglig: https://www.mindroad.se/sv-SE/v%C3%A5ra-utbildningar/verktyg-och-metodik/git-22242842 (hämtad: maj 2021)

[13] G. Li, ”Quick Start Guide to Git,” DEV, [Online], jun. 15, 2020. Tillgänglig: https://dev.to/tauag/quick-start-guide-to-git-2of5 (hämtad: maj 2021)

[14] Microsoft Inc, ”Using batch files,” 2009. [Online]. Tillgänglig: https://docs.microsoft.com/en-us/previous-versions/windows/it-pro/windows-server-2003/cc758944(v=ws.10)?redirectedfrom=MSDN (hämtad: maj 2021)

[15] Microsoft Inc, ”Vad är en virtuell dator?,” UÅ. [Online]. Tillgänglig: https://azure.microsoft.com/sv-se/overview/what-is-a-virtual-machine/#what-used-for (hämtad: maj 2021)

[16] Computer Sweden, ”IT-ord,” 2017. [Online]. Tillgänglig: https://it-ord.idg.se/ord/virtuell-maskin/ (hämtad: maj 2021)

[17] M. Mills, ”Hur man klonar eller flyttar en virtuell maskin med VirtualBox,” ITGIC, [Online], apr. 13 2021. Tillgänglig: https://itigic.com/sv/how-to-clone-or-move-a-virtual-machine-with-virtualbox/ (hämtad: maj 2021)

[18] Python software foundation, Python is powerful... and fast; plays well with others; runs everywhere; is friendly easy to learn; is Open.,"UÅ. [Online]. Tillgänglig: https://www.python.org/about/ (hämtad: maj 2021).

[19] D. Oprea, ”svn 1.0.1,” Python Package Index, [Online], feb. 2020. Tillgänglig: https://pypi.org/project/svn/ (hämtad: maj 2021)

[20] the pandas development team, ”User Guide,” 2020. [Online]. Tillgänglig: https://pandas.pydata.org/docs/user_guide/index.html (hämtad: maj 2021)

[21] ShadowClaw20017, ”tkinter,” Pyhton, [Online]. Feb 2021. Tillgänglig: https://wiki.python.org/moin/TkInter (hämtad: maj 2021)

[22] S. Gupta, ”Executable GUI with Python,” Life and Tech, [Online], jun. 28, 2019. Tillgänglig: https://medium.com/lifeandtech/executable-gui-with-python-fc79562a5558 (hämtad: maj 2021)

[23] CoderLegacy, ”Python GUI with Tkinter,” UÅ. [Online]. Tillgänglig: https://coderslegacy.com/python/python-gui/ (hämtad: maj 2021)

[24] Python Software Foundation, ”shutil — High-level file operations,” UÅ. [Online]. Tillgänglig: https://docs.python.org/3/library/shutil.html (hämtad: maj 2021)

[25] O.M. Bjørndalen, ”Introduction,” dbfread, [Online], UÅ. Tillgänglig: https://dbfread.readthedocs.io/en/latest/introduction.html (hämtad: maj 2021)

[26] B. A. Scott, ”pysvn - Programmer’s reference”, pysvn - Programmer’s reference, [Online], UÅ. Tillgänlig: https://tools.ietf.org/doc/python-svn/pysvn_prog_ref.html (hämtad: maj 2021)

[27] Y. Nader, ”Python vs Java in 2021”, hacker, [Online], jan. 08, 2021. Tillgänglig: https://hackr.io/blog/python-vs-java, (hämtad: maj 2021)

[28] DataFlaire, ”Pros and Cons of Java | Advantages and Disadvantages of Java,” UÅ. [Online]. Tillgänglig: https://data-flair.training/blogs/pros-and-cons-of-java/ (hämtad: maj 2021)

[29] TMate Software, ”SVNKit: Subversion for Java,” UÅ. [Online]. Tillgänglig: https://svnkit.com/ (hämtad: maj 2021)

[30] AppDoc, ”Class DBFReader,” UÅ. [Online]. Tillgänglig: https://appdoc.app/artifact/com.github.albfernandez/javadbf/1.6.2/com/linuxense /javadbf/ DBFReader.html (hämtad: maj 2021)

[31] A. Roznovsky, ”Why use PHP main advantages and disadvantages,” LightIT, [Online], UÅ. Tillgänglig: https://light-it.net/blog/why-use-php-main-advantages-and-disadvantages/ (hämtad: maj 2021)

[32] The PHP group, ”SVN Functions,” UÅ. [Online]. https://www.php.net/manual/en/ref.svn.php (hämtad: maj 2021)

[33] J.R. Grolf, P.N. Weinberg, SQL: The Complete Reference. California, USA: Osborne/McGraw-Hill, UÅ. [Online]. Tillgänglig: http://englishonlineclub.com/pdf/SQL%20-%20The%20Complete%20Reference%20[EnglishOnlineClub.com].pdf, Hämtad: maj 2021

[34] SQLCourse, ”What is SQL?,” UÅ. [Online]. Tillgänglig: http://www.sqlcourse.com/intro.html (hämtad: maj 2021)

[35] Europaparlamentets och rådets förordning (EU) 2016/679 av den 27 april 2016 om skydd för fysiska personer med avseende på behandling av personuppgifter och om det fria flödet av sådana uppgifter och om upphävande av direktiv 95/46/EG (allmän dataskyddsförordning) (Text av betydelse för EES) (2016), EUT L119/1. [Online]. Tillgänglig: https://eur-lex.europa.eu/legal-content/SV/TXT/?uri=celex%3A32016R0679, Hämtad: maj 2021.

9 Bilaga A

Definitionen som initierar variablerna i filen variable.dbf

```
def variable():
    cluster1_var = df1['CLUSTER'] == ''
    comment1_var = df1['COMMENT'] == ''
    name1_var = df1['NAME'] == ''
    addr1_var = df1['ADDR'] == ''
    type1_var = df1['TYPE'] == ''
    raw_zero1_var = df1['RAW_ZERO'] == ''
    raw_full1_var = df1['RAW_FULL'] == ''
    eng_full1_var = df1['ENG_FULL'] == ''
    eng_zero1_var = df1['ENG_ZERO'] == ''
    eng_units1_var = df1['ENG_UNITS'] == ''
    format1_var = df1['FORMAT'] == ''

    cluster2_var = df2['CLUSTER'] == ''
    comment2_var = df2['COMMENT'] == ''
    name2_var = df2['NAME'] == ''
    addr2_var = df2['ADDR'] == ''
    type2_var = df2['TYPE'] == ''
    raw_zero2_var = df2['RAW_ZERO'] == ''
    raw_full2_var = df2['RAW_FULL'] == ''
    eng_full2_var = df2['ENG_FULL'] == ''
    eng_zero2_var = df2['ENG_ZERO'] == ''
    eng_units2_var = df2['ENG_UNITS'] == ''
    format2_var = df2['FORMAT'] == ''

    colName_Variable = ["CLUSTER", "COMMENT", "NAME",
                        "ADDR", "TYPE", "RAW_ZERO",
                        "RAW_FULL", "ENG_FULL",
                        "ENG_ZERO", "ENG_UNITS", "FORMAT"]
    colVar1_Variable = [cluster1_var, comment1_var,
                        name1_var, addr1_var, type1_var,
                        raw_zero1_var, raw_full1_var,
                        eng_full1_var, eng_zero1_var,
                        eng_units1_var, format1_var]
    colVar2_Variable = [cluster2_var, comment2_var,
                        name2_var, addr2_var, type2_var,
                        raw_zero2_var, raw_full2_var,
                        eng_full2_var, eng_zero2_var,
                        eng_units2_var, format2_var]

    clearTextInput()
    check_if_equal(df1, df2)
    check_file(df1, df2, colVar1_Variable,
               colVar2_Variable, colName_Variable)
```