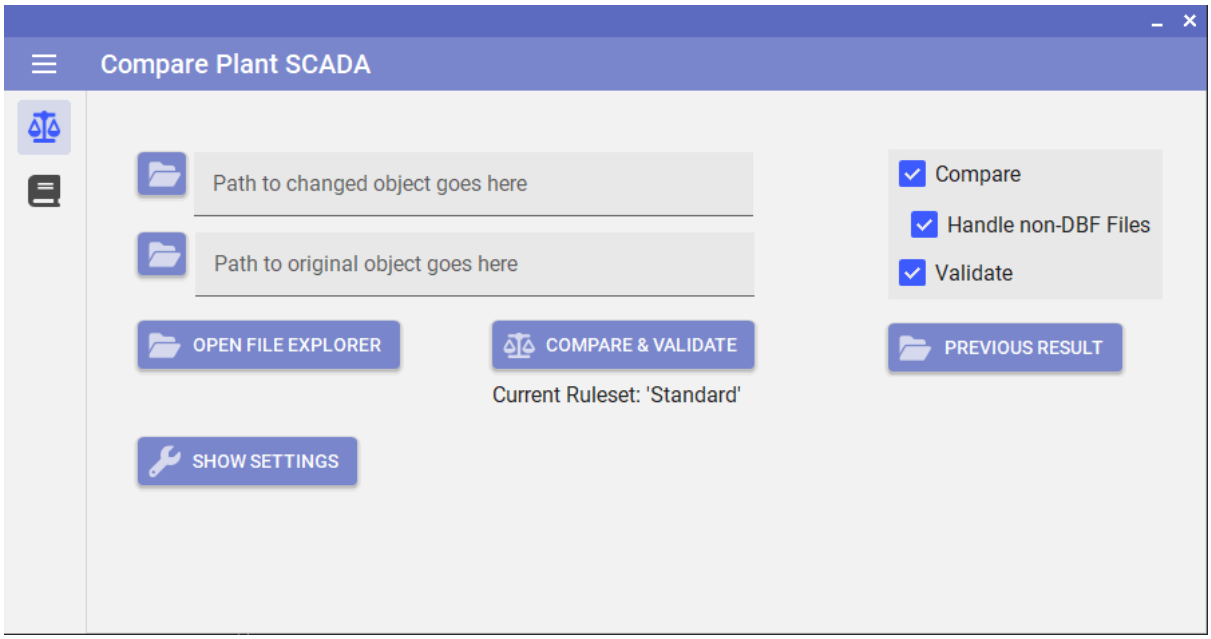


Jämförelse- och valideringsapplikation för SCADA-projekt

Examensarbete inom högskoleingenjörsprogrammet Mekatronik

Joel Ahlinder
Pontus Carlsson

INSTITUTIONEN FÖR Data- och Informationsteknik
CHALMERS TEKNISKA HÖGSKOLA
Göteborg, Sverige 2023
www.chalmers.se

EXAMENSARBETE 2023

Jämförelse- och valideringsapplikation för SCADA-projekt

JOEL AHLINDER
PONTUS CARLSSON

Institutionen för Data- och Informationsteknik
Chalmers tekniska högskola
Göteborg, Sverige 2023

Jämförelse- och valideringsapplikation för SCADA-projekt
JOEL AHLINDER
PONTUS CARLSSON

Handledare:
Sven Knutsson, *Chalmers tekniska högskola*
Andreas Karlsson*, Acobia*

Examinator:
Lars Svensson, *Chalmers tekniska högskola*

Institutionen för Data- och Informationsteknik
Chalmers tekniska högskola / Göteborgs Universitet
SE-412 96 Göteborg
Telefon: 031-772 1000

Omslag: Förstasida av applikation som skapats.

SAMMANFATTNING

Acobia är ett företag som arbetar med automationssystem. Företaget är ej knutet till ett varumärke utan jobbar med de mjukvaror kunden efterfrågar. En mjukvara som används regelbundet på Acobia är AVEVA Plant SCADA. Mjukvaran har problemet att det idag inte finns något sätt att se ändringar mellan olika versioner av arbete. Därmed litar utvecklare på varandra att noggrann testning av utfört arbete har skett. Om fel påträffas måste dessa hittas manuellt**,** vilket är mycket tidskrävande. För att öka säkerheten och upptäcka potentiella fel tidigare har Acobia efterfrågat en applikation där användaren kan se skillnader mellan olika projektversioner samt validera den senaste versionen.

Arbetet innefattar utveckling av en applikation som tar data som används av automationsprogrammet AVEVA Plant SCADA och jämför denna med en tidigare version av datan samt validerar den efter användarinställda regler. Slutligen genererar applikationen ett resultat, innehållande versionernas förändringar samt brutna valideringsregler. Arbetet resulterade i en funktionell applikation med ett modernt och användarvänligt gränssnitt som underlättar felsökningsprocessen markant.

Nyckelord: SCADA, Jämförelse, Validering, Applikation, Gränssnitt

ABSTRACT

Acobia is a company that works with automation systems. The company is not bound by any specific brand and can therefore work with any software the customer requests. One software that is used regularly at Acobia is AVEVA Plant SCADA. The software has a problem where there is no way to see changes made between different versions of the work. Therefore, developers must rely on each other to thoroughly test the implemented work. If errors occur these must be found manually, which takes a lot of unnecessary time. To improve the validity of these changes and to find potential errors earlier, Acobia has requested an application with the purpose of presenting changes and validation errors that have come with the most recent version.

This project consists of the development of an application that takes data used by the software AVEVA Plant SCADA and compares it with an older version of the data, it also validates the data based on user-selected rules. Lastly the application generates a result which includes all the changes made and any validation rules that have been broken in the latest data. The project resulted in an application with a modern and simple user interface which will significantly simplify the process of troubleshooting.

Keywords: SCADA, Comparison, Validation, Application, User interface

FÖRORD

Examensarbetet är utfört under vårterminen 2023 av Joel Ahlinder och Pontus Carlsson på högskoleingenjörsprogrammet, Mekatronik på Chalmers tekniska högskola.

Vi vill ge ett stort tack till handledaren Andreas Karlsson samt hans kollega Anton Andersson på Acobia. Med deras kontinuerliga hjälp och stöd blev arbetet möjligt att utföra. Vi vill även tacka Acobia som gav oss möjligheten att göra vårt examensarbete hos dem. Slutligen vill vi ge ett stort tack till handledaren på Chalmers, Sven Knutsson som väglett oss genom projektet, särskilt vid rapportskrivningen.

Ordlista

SCADA – Supervisory control and data acquisition
AVEVA Plant SCADA – Ett SCADA-program utvecklat av företaget AVEVA
HMI – Human machine interface
CLR - Common language runtime
SQL – Structured query language
.BAT – Batchfil
Dictionary - Grundläggande datastruktur, uppbyggd av nyckel och värde

1. Inledning

Idag används intelligenta styr- och övervakningssystem inom en mängd olika branscher. Utvecklingen av dessa system inom exempelvis infrastruktur och industri är en del av arbetet som sker på företaget Acobia. När stora projekt automatiseras med styr- och övervakningssystem finns det tillgång till mängder av data. Denna data förändras sedan kontinuerligt av olika människor i samband med att systemen uppdateras. Med det sagt krävs det att alla som hanterar datan är ytterst noggranna och att inga fel sker. Idag dokumenteras förändringarna av datan manuellt på Acobia. För att öka säkerheten och användarvänligheten av datahanteringen, önskar man införa ett jämförelseverktyg. En applikation som visar förändringar mellan olika versioner av filer med data och validerar den senaste versionen av datan.

1.1 Bakgrund

Vid utveckling av styr- och övervakningssystem är en av mjukvarorna som Acobia använder AVEVA Plant SCADA. Detta är en mjukvara för att arbeta med Supervisory control and data acquisition (SCADA), där SCADA är ett system som används inom styrning samt övervakning av processer [1] [2]. Vidare beskrivning av SCADA finns under rubriken 2.1. Mjukvaran AVEVA Plant SCADA använder en mängd olika filer för varje projekt där ett projekt exempelvis kan vara en produktionslina eller en fastighet. Det finns flera olika människor som har tillgång till och kan ändra datan som används av AVEVA Plant SCADA, utan tillgång till versionshantering. Med det sagt finns det alltid en risk att förändringar leder till fel, med svårigheter att spåra felen. Något som kan orsaka problem i projekten som datan är relaterad till.

Det finns inget säkert och effektivt verktyg Acobia känner till, som kan kontrollera data använd av AVEVA Plant SCADA. Därmed skulle en applikation införas, för att kontrollera filers förändringar, samt validera filerna baserat på användarinställda regler. Detta leder till ökad effektivitet och säkerhet av kontrollprocessen.

1.2 Syfte

Syftet med arbetet är att ta fram en applikation som kan jämföra och validera filer från projekt av AVEVA Plant SCADA. Utvecklingen av verktyget kommer leda till en effektivare samt mer standardiserad process vid kontrollering av ändringar mellan filversioner.

1.3 Mål

Målet med projektet är att ta fram ett verktyg som jämför och validerar filer generade i AVEVA Plant SCADA. Verktyget ska generera ett resultat som presenteras för användaren så att hen lätt förstår vad som förändrats, exempelvis i Excel. Resultatet ska visa vilka ändringar som skett samt vilka regler som ej är uppfyllda enligt valideringen.

Validering syftar till att specifika regler uppfylls, exempelvis att variablers enheter stämmer överens i den senaste versionen av filerna. Jämförelse syftar till att jämföra olika versioner av samma filer som AVEVA Plant SCADA använder.

1.4 Frågeställningar

Här beskrivs de frågeställningar som fanns vid projektstart.

1. Går det att skapa en applikation som är tillräckligt användarvänlig och effektiv för att utvecklare på Acobia ska använda den?

    a) Hur bör gränssnittet för verktyget se ut?
    b) Hur lång tid bör en jämförelse med applikationen ta?

2. Hur ska verktygets resultat presenteras för användaren?

    a) Hur mycket data, relaterad till resultatet ska användaren få tillgång till, i resultatet som presenteras?
    b) Bör en extern mjukvara användas för att presentera resultatet, exempelvis Excel?

3. Går det att göra applikationen tillräckligt dynamisk för att den ska vara användbar då AVEVA Plant SCADA utvecklas?

1.5 Avgränsningar

Nedan förklaras de avgränsningar som gjordes i projektet.

1. Verktyget kommer att utvecklas i språket C#
2. För att bygga verktygets gränssnitt kommer Windows Forms användas.
3. Förändringar i filer som inte är dBase-filer vilka används av AVEVA Plant SCADA kommer att presenteras med hjälp av mjukvaran WinMerge.

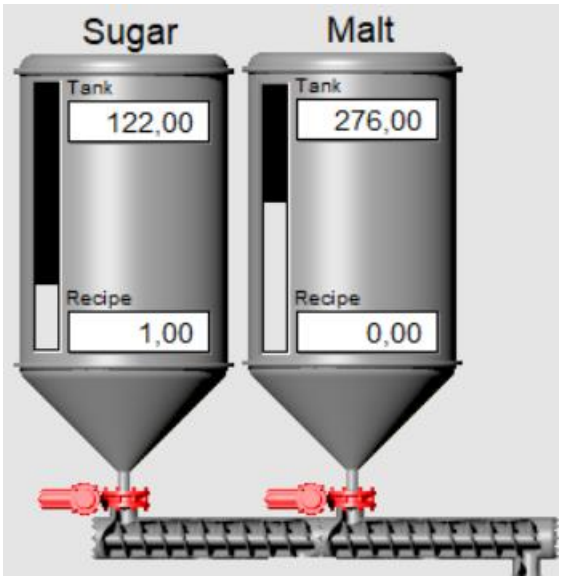
Windows Forms, C#, dBase-filer samt WinMerge kommer förklaras vidare i kapitel 2.

2. Teknisk bakgrund

Nedan beskrivs teknisk information som behövdes för att utföra arbetet.

2.1 SCADA

SCADA är ett system som används för att kontrollera processer. Detta sker i form av styrning eller övervakning, bland annat inom industri. Systemet består huvudsakligen av tre huvudkomponenter: utrustningen som ska övervakas och styras, fältenheter i form av mindre datorer som exempelvis består av programmable logic controllers (PLC) eller remote terminal units (RTU), samt human machine interface (HMI) som även kan kallas för gränssnitt. Exempel av ett gränssnitt syns i Figur 1 [1].



Figur 1: Exempel på en processöverblick i AVEVA Plant SCADA

Förenklat fungerar systemet på följande vis: utrustningen skickar data till fältenheterna som värderar datan och automatiskt bestämmer vad som ska hända härnäst. Därefter vidarebefordras datan och styrsignalerna till gränssnittet för att exempelvis en tekniker ska kunna få en överblick av systemet. I gränssnittet kan teknikern även styra processer direkt som exemplet i Figur 1 med ventiler men också ändra hur systemet beter sig automatiskt över tid [3].

På så vis gör SCADA-system det möjligt att exempelvis stoppa en process på distans med hjälp av HMI om datan som skickats tillbaka är alarmerande [4].

2.1.1 Filer som används av AVEVA Plant SCADA-projekt

Det finns flera typer av filer som används i AVEVA Plant SCADA-projekt. Nedan nämns de som behöver förstås för att förstå applikationens användningsområde.

- dBase-filer: Relationsdatabasfiler som innehåller varierande mängder data, se exempel i Figur 2.

Majoriteten av ändringar i AVEVA Plant SCADA leder till ändringar i de tillhörande dBase-filerna. Om ändringar utförs felaktigt kan det leda till att styrning slutar att fungera och viktiga larm ej aktiveras. Ändringar på fel sätt kan därmed leda till problem om de inte blir upptäckta i tid [3].

	A	B	C	D	E	F	G
1	NAME,C,79	TYPE,C,	UNIT,C,31	ADDR,C,254	RAW_Z	RAW_FU	ENG_Z
2	Agitator_Cmd	DIGITAL	DISK_PLC	D302			
3	Arrow_fiber_optical	DIGITAL	IODevInternal	D24			
4	Arrow_Network3	DIGITAL	IODevInternal	D15			
5	BATCH	INT	DISK_PLC	I9	0	32000	0
6	BatchTotal	INT	DISK_PLC	I10	0	32000	0

Figur 2: Urklipp av dBase-fil

- CI-filer: Dessa filer innehåller programmeringskod i språket cicode som utvecklare skrivit i AVEVA Plant SCADA [3].

- XML- och INI-filer: Dessa filer är konfigurationsfiler som skapas och används inom AVEVA Plant SCADA på olika vis [3].

- CTG-filer: Dessa är grafiska sidor i AVEVA Plant SCADA som exempelvis visar drift för processer inom industri och fastighet. Från dessa sidor går det att se processen, den kan ibland även styras [3].

- CTF-filer: Dessa filer är kompilerade CTG-filer [3].

2.2 .NET Framework

.NET Framework är ett ramverk utvecklat av Microsoft som används för att bygga webbaserade datorprogram samt Windowsapplikationer. Ramverket består av CLR samt .NET Frameworks egna klassbibliotek. CLR hanterar bland annat minne samt exekvering och kompilering av kod. Klassbiblioteket innefattar en mängd olika färdiga klasser. Med hjälp av dessa klasser kan Windows Forms som används för att bygga Windowsapplikationer användas [5].

Utöver de klassbibliotek som finns i ramverket automatiskt finns även ett verktyg i .NET Framework som heter NuGet. NuGet tillåter användare att ladda ned och ladda upp paket med kod som sedan kan återanvändas [6]. Exempelvis för att öppna och extrahera innehållet ur ZIP-filer eller XLSX-filer. (XLSX-filer är Microsoft Excels huvudsakliga filformat [7].)

2.3 Windows Forms

Windows Forms är ett ramverk för gränssnitt som används för att skapa datorapplikationer för Windows. En stor fördel med Windows Forms är att det har en mycket enkel drag-and-drop-funktionalitet med färdiga klasser för att skapa gränssnittet till applikationen som utvecklas. Där klasserna bland annat kan vara knappar eller menyträd av olika slag. Dessa klasser kopplas sedan till olika händelser

med kod så att utvecklaren av en applikation exempelvis kan bestämma vad som händer vid ett knapptryck [8].

2.4 Externa applikationer

Nedan beskrivs externa applikationer relaterade till projektet.

2.4.1 WinMerge

WinMerge är en applikation med funktion för att visa ändringar mellan filversioner samt lägga samman ändringar mellan versioner. Dessa funktioner fungerar på en mängd olika filtyper men även mappar [9].

2.5 Programmeringsspråk

Här beskrivs programmeringsspråk som är relevanta för projektet.

2.5.1 C# och objektorienterad programmering

C# är ett objektorienterat programmeringsspråk utvecklat av Microsoft. Objektorienterad programmering är en typ av programmering baserad på en slags hierarki. Denna består av klasser och instanser av klasserna som är oberoende av varandra, vilka kallas objekt. Utvecklaren kan själv skapa klasser som innehåller olika variabler samt metoder, där metoder är funktioner som kan användas av klasserna. På så vis kan utvecklaren bygga ett stort program med många mindre byggstenar i form av klasser. Exempel på redan existerande klasser i C# är Array och String [10].

C# är ett språk som påminner mycket om C och Java. Program skrivna i C# kan köras i .NET och därmed är C# ett passande språk för att bygga applikationer till Windows. Detta på grund av .NET:s kompatibilitet med C# och Windows Forms [11].

2.5.2 Databaser och SQL

Databaser är samlingar med information som lagras i ett datorsystem. Det finns relations- och icke-relationsdatabaser, skillnaden mellan dessa är deras struktur samt att relationsdatabaser kan använda SQL vilket icke-relationsdatabaser inte kan. Strukturskillnaderna är att relationsdatabaser är ordnade i tabeller medan icke-relationsdatabaser har en struktur baserad på specifika krav för den lagrade datan [12]. SQL är ett programmeringsspråk som använder kommandon för att hantera data i relationsdatabaser. Exempel på vad SQL-kommandon kan göra med data i dessa databaser är: hämta, uppdatera, ta bort eller ändra data [13].

2.5.3 Java

Java är ett språk som bygger på objektorienterad programmering. När det började utvecklas var språken C och C++ väldigt populära. Dessa språk var stora men hade problem när de skulle användas till exempelvis molnapplikationer. Java skapades med

målen att vara lika kraftfullt som andra objektorienterade språk men simplare att använda, och även vara mycket enklare att exportera till andra operativ system [14].

2.6 Batchfiler

En batchfil är en textfil som innehåller minst ett kommando. Ett kommando kan exempelvis vara att öppna en fil eller en hemsida, se Figur 3.

När filen körs kommer operativsystemet utföra alla kommandon som filen innehåller. Därmed går det exempelvis att öppna en mjukvara samt ge den indata på en gång genom att endast köra en batchfil med kommandon relaterade till mjukvaran och indata [15].

```
@echo off
chcp 65001
"C:\Program Files\SourceGear\Common\DiffMerge\sgdm.exe" "C:\TestFolder\AnläggningOld\threedBase\files.txt" "C:\TestFolder\AnläggningOld\threedBase\files.txt"
```

Figur 3: Batchfil för att öppna WinMerge med två filer.

2.7 EPPlus

EPPlus är ett NuGet-bibliotek som används för att hantera data i Excelformat för .NET-applikationer. Biblioteket introducerar en klass kallad ExcelPackage som möjliggör att skapa XLSX-filer från grunden med data och design. Det går även att läsa in existerande XLSX-filer för att läsa datan som finns och redigera filen eller formatet [16].

2.8 MaterialSkin 2

MaterialSkin 2 är ett NuGet-bibliotek som finns tillgängligt för Windows Forms. Biblioteket introducerar nya grafiska objekt som påminner om grafiska objekt från Windows Forms men med utökad funktionalitet. Biblioteket skapades för att förenkla design i Windows Forms [17].

2.9 Virtuell maskin

En virtuell maskin är en slags dator uppbyggd av mjukvara i stället för hårdvara [18]. Den virtuella maskinen exekverar på en existerande dator där båda har individuella operativsystem. Därmed är det möjligt att använda en virtuell maskin med Linux som operativ system på en Windows-dator. Ett av flera användningsområden för virtuella maskiner är att kunna testa applikationer på olika operativsystem samt i en tom och säker miljö [19].

3. Metod

Nedan beskrivs metodiken som använts vid utvecklingen av verktyget under projektet.

3.1 Inledande undersökning

Projektet kommer inledas med en kort undersökning kring Windows Forms, SQL, AVEVA Plant SCADA samt ett ”proof of concept”-verktyg som Acobia framställt.

Information kring Windows Forms och SQL krävs för att få en förståelse för hur ett enkelt gränssnitt kan utvecklas samt hur data ur dBase-filer kan hanteras.

Mjukvaran AVEVA Plant SCADA kommer undersökas för att få en djupare förståelse för vilka typer av filer den använder samt hur deras innehåll ser ut. Specifikt upplägget av dBase-filerna; kräver mer förståelse då dessa kommer hanteras på en djupare nivå.

Det finns även ett existerande ”proof of concept”-verktyg; detta kommer undersökas grundligt för att få en överblick av hur vissa problem kan lösas samt för att skapa inspiration till projektets verktyg.

3.2 Utvecklingsprocess

Utvecklingen av verktyget kommer ske stegvis, där dess funktionalitet delas upp i mindre delar, då det anses mer effektivt att lösa flera små problem i stället för ett stort på en gång. Uppdelningen av verktygets funktionalitet kommer ske på följande vis:

1. Jämförelse av dBase-filer.
2. Utveckling av grundläggande gränssnitt.
3. Funktionalitet för presentation av resultat.
4. Jämförelse av alla filer som används av AVEVA Plant SCADA.
5. Validering av dBase-filer.
6. Funktionalitet för redigering av CSV-filer genom verktyget.
7. Vidareutveckling av gränssnitt.
8. Testning med virtuell maskin.

Detta underlättar även vidareutveckling av verktyget då det blir lättare att uppdatera specifika delar av det.

3.3 Förbättringsprocess

När programmet börjar bli färdigt kommer handledaren från Acobia hjälpa till att testa verktyget i sitt dagliga arbete. Detta för att upptäcka fel och potentiella förbättringar. Åsikter från handledaren kommer efterfrågas, för att programmet ska kunna bli mer anpassat till en relevant användare.

4. Genomförande

Genomförandet inleddes med frågor till handledaren på Acobia för att få en djupare förståelse av det efterfrågade verktygets funktion samt hur det skulle kunna skapas. Några av dessa frågor var följande:

- Vilka huvudsakliga funktioner ska verktyget ha?
- Hur kan de filer som ska hanteras se ut?
- Hur ska användaren få ta del av resultatet?
- Vilket språk bör verktyget skapas i?

Efter att ha fått svar på dessa frågor var det klargjort vad som efterfrågades samt hur verktyget potentiellt kunde skapas.

4.1 Val av programmeringsspråk och utvecklingsmiljö

Acobia hade redan utvecklat ett verktyg som liknade det efterfrågade men det hade en låg användarvänlighet, var långsamt samt var svårt att uppdatera. Prototypprogrammet var skapat i C#. Projektdeltagarna hade tidigare programmerat i Java och C men inte C#, men när dessa tre jämfördes togs beslutet att C# ändå skulle passa bättre för projektet. Detta eftersom det fanns mer hjälp att få från handledaren på Acobia, samt att inspiration kunde tas från det redan existerande verktyget. Därmed valdes C# som språk för examensarbetet.

Eftersom erfarenhet saknades i C# lades till en början tid på att utöka förståelsen inom språket, men eftersom det påminde mycket om C samt Java var detta en snabb process. Utvecklingsmiljön som valdes var Visual Studio. Detta för att deltagarna använt miljön tidigare samt att där fanns tillgång till Windows Forms. Detta underlättade skapandet av en applikation med hjälp av dess drag-and-drop-funktionalitet.

4.2 Funktionalitet

Underrubrikerna nedan beskriver applikationens huvudsakliga funktionalitet.

4.2.1 Inläsning av indata

Vid inläsning av indata kollas först indatans fil-ändelse. Om den slutar på .CTZ eller .ZIP måste datan först extraheras för att sedan kunna läsas in igen. Där .CTZ är en form av komprimerad fil som ibland exporteras från AVEVA Plant SCADA; dessa påminner om ZIP-filer. Extraheringen sker automatiskt i applikationen med hjälp av NuGet-biblioteket: DotNetZip.

Vid inläsning av dBase-filer används två olika metoder med olika för- och nackdelar. De två metoderna är Jet OLEDB 4.0, vilket är ett bibliotek som finns redo i Visual Studio samt NDbfReader, vilket är ett bibliotek som finns att ladda ned genom NuGet.

Fördelen med OLEDB Jet 4.0 var att det gick att använda SQL-kommandon på filer och därmed läsa in datan ur filen på specifika vis med SQL-kommandon om detta efterfrågades. Exempelvis att sortera all data alfabetiskt vid inläsning. Problemet med OLEDB Jet 4.0 var att det inte gick att läsa in filer med namn längre än 8 tecken, då kraschar programmet.

För att läsa in filer med namn längre än 8 tecken hittades biblioteket NDbfReader som fungerade bra oavsett längd på namn. Problemet med denna metod var att det inte gick att använda SQL-kommandon vid inläsning.

Efter samtal med handledaren på Acobia togs beslutet att kompromissa och använda NDbfReader i de fall där filnamnet hade fler än 8 tecken, samt använda OLEDB Jet 4.0 i resterande fall. Oavsett vilken metod som används vid inläsning av dBase-filer sparas datan i en DataTable-tabell, likt en matris.

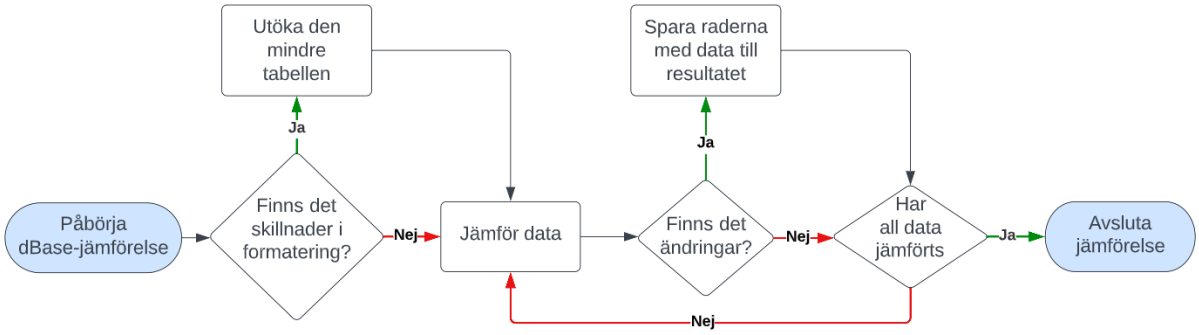
Beroende på vilket språk som använts i AVEVA Plant SCADA kan dess filer ha konstiga tecken. Om användaren exempelvis använt bokstäverna ”å”, ”ä”, ”ö” kan dessa bli inlästa som andra tecken. För att hantera detta finns en metod som letar efter de inkorrekta tecken som upptäckts. Baserat på testning gick det att se vilka tecken ”å”, ”ä”, ”ö” blev omskrivna till. Därefter gick det att ersätta de tecknen med den korrekta motsvarigheten av ”å”, ”ä”, ”ö”. Denna metod krävs eftersom mycket av Acobias arbete sker i Skandinavien, där dessa tecken är vanliga.

4.2.2 Jämförelse av dBase-filer

Vid jämförelse av två dBase-filer har först två filer blivit inlästa till DataTable-tabeller i form av matriser. Först undersöks formaten av de båda matriserna. För att jämförelsen mellan de två matriserna ska kunna ske, krävs det att de har samma format. Om formaten inte är likadana för de båda tabellerna måste formaten därmed ändras för att jämförelsen ska bli korrekt.

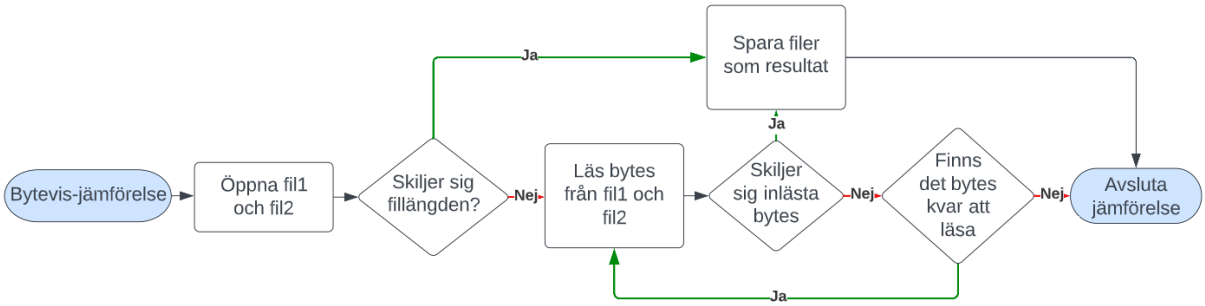
Formatförändringen sker genom att lägga till rader/kolumner i den matris där de saknas. Om rader saknas läggs tomma rader till i slutet av matrisen medan kolumner som saknas, kopieras från den större matrisen och läggs till på rätt position i den mindre matrisen.

När formaten stämmer överens i de båda matriserna jämförs varje element på samma position i de båda matriserna. Om två element skiljer sig åt, sparas dessa specifika element samt hela raden där de befinner sig, för att sedan visas i resultatet som något som har förändrats mellan de två filerna. Se Figur 4 för hela jämförelseprocessen.



Figur 4: Flödesschema som visar processen för att jämföra enskilda dBase-filer

Vid jämförelse av filer kan det vara upp till flera 100 dBase-filer som ska jämföras. Därför spenderas mycket tid på detta, då filerna ofta innehåller stora mängder data. En stor flaskhals för applikationen var därmed jämförelse av filer där det ej fanns skillnader. Detta är eftersom applikationen behövde jämföra alla positioner i två matriser varje gång två filer skulle jämföras. Något som var onödigt tidskrävande i de fall där det inte fanns några skillnader mellan de två filerna. För att förbättra applikationens prestanda insåg deltagarna att det gick att först jämföra alla dBase-filer bytevis, för att sedan endast jämföra matriserna mellan de filer där den bytevis-jämförelsen upptäckt skillnader. Bytevis-jämförelse innebär att filerna jämförs byte för byte och om två bytes skiljer sig, har filerna inte samma innehåll**.** Förändringen ökade applikationens prestanda markant, då bytevis-jämförelse är mycket effektivt gentemot jämförelse av matriser. Se flödesschema kring bytevis-jämförelse för ett förtydligande i Figur 5.



Figur 5: Flödesschema som visar processen för att jämföra filer bytevis

4.2.3 Resultatsgenerering

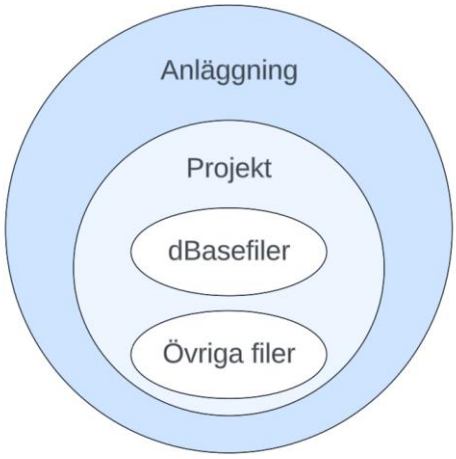
Till en början utvärderades hur resultatet av jämförelse och validering skulle visas för användaren, de två metoder som diskuterades var XLSX-filer samt någon typ av tabell i gränssnittet. Tabellen hade fördelen att den inte krävde att användaren hade mjukvara för att hantera XLSX-filer. Dessa filer hade fördelen att det var möjligt att skapa stora filer och bygga ut verktyget med mer funktioner på ett smidigare sätt. Därmed valdes XLSX-filer till att användas för resultatsgenerering.

Resultatet från jämförelse och validering presenteras till användaren i en XLSX-fil. För detta används biblioteket EPPlus som via OfficeOpenXML skapar XLSX-filer med datan som tagits fram. Första kalkylbladet i alla XLSX-filer innehåller en sammanfattning av alla jämförelser som gjorts. Bladet innehåller även en guide för vad de olika färgerna i resultatsfilen står för och vilka filer utöver dBase-filer som jämförts. De andra kalkylbladen innehåller data från filerna som jämförts. Varje gång ett nytt resultat genereras, raderas föregående resultat. Detta för att spara plats på användarens dator. Utseendet av XLSX-filerna förklaras vidare under rubriken 4.3.

4.2.4 Jämförelse av projekt och anläggning

På Acobia hanteras projekt och anläggningar, där ett projekt innehåller dBase-filer samt en mängd andra filer från AVEVA Plant SCADA och en anläggning består av flera projekt, se visualisering i Figur 6.

För att kunna jämföra projekt och anläggningar krävdes fler metoder utöver de som användes för att endast jämföra dBase-filer.



Figur 6: Visualisering över relation mellan projekt och anläggningar

För att jämföra två projekt med varandra jämförs alla filer med samma namn i de två projektmapparna. När två dBase-filer med samma namn hittats, jämförs dessa på samma vis som tidigare. Den stora skillnaden vid projektjämförelse är att en loop körs som går igenom alla filnamn i de båda projekten och ser till att alla dBase-filer jämförs med motsatt dBase-fil. XLSX-filen som genereras av applikationen ser likadan ut som den vid jämförelse av endast en dBase-fil, förutom att det finns fler kalkylblad, där varje fil som jämförts har ett eget blad med information om jämförelsen.

Vid jämförelse av anläggning finns oftast men inte alltid en dBase-fil med namnet: MASTER i anläggningen. Denna innehåller sökvägar till alla projekt som anläggningen innehåller. För att jämföra en anläggning med MASTER-fil blir först alla sökvägar till projekt som anläggningen innefattar inlästa från denna. Därefter jämförs dessa projekt med varandra likt beskrivningen ovan. För att jämföra en anläggning utan MASTER-fil sparas i stället alla undermappar från indatan för att jämföras mot motsvarande

undermapp. När alla projekt har jämförts så är jämförelsen av anläggningen klar. Eftersom varje projekt skapar en XLSX-fil, skapas flera XLSX-filer vid en jämförelse av anläggning. Därmed skapas även en extra XLSX-fil med en överblick över alla projekt som jämförts där det även finns hyperlänkar för att navigera mellan alla XLSX-filer.

4.2.5 Specifika jämförelseregler för speciella dBase-filer

Det finns många dBase-filer som skapas av AVEVA Plant SCADA som behöver hanteras annorlunda. Som exempel finns det filer där flera kolumner av data behöver kombineras innan de jämförs. I dessa fall kan specifika SQL-kommandon användas för att läsa in datan ur dBase-filerna på efterfrågat sätt.

På Acobia vet man vilka dBase-filer som kräver specifika SQL-kommandon, därmed fungerade det att läsa in en CSV-fil med information om alla välkända dBase-filer. I CSV-filen inkluderas information som beskriver om ett annorlunda SQL-kommando utöver standardkommandot ska användas, baserat på dBase-filers namn. CSV-filen lästes in som en dictionary med en filnamns-sträng som nyckel samt en struct som värde. Structen bestod av en sträng i form av ett SQL-kommando samt en bool med information om filen skulle bli inläst eller ej.

Om ett filnamn inte existerar i dictionaryn antas det att filen är skapad av någon på Acobia som en typ av kopia eller liknande och därmed används standard-SQL-kommandot.

4.2.6 Jämförelse av filtyper utöver dBase-filer

AVEVA Plant SCADA använder inte enbart dBase-filer utan även filtyper som exempelvis .CI och .XML. Även dessa filer kan innehålla ändringar som bör presenteras för användaren. För detta skapades en metod som jämför filer utöver dBase-filer.

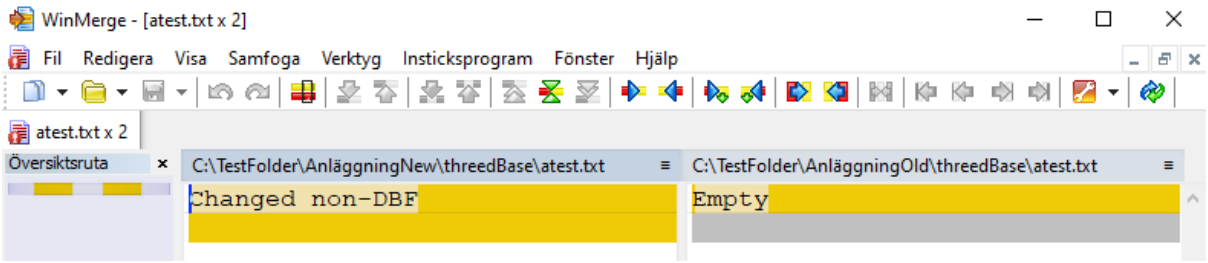
Först utvärderades två olika metoder. Den ena presenterades av projektets handledare på Acobia och hette checksum. Detta innebar att gå igenom hela filerna och skapa en kontrollsumma baserad på innehållet i filerna. Detta hade fungerat, men det andra alternativet som hittades ledde till samma resultat fast snabbare. Den andra metoden är bytevis-jämförelse som beskrivs mer utförligt ovan, under rubrik 4.2.2.

För att visa för användaren vilka de specifika ändringarna i filerna är så används mjukvaran WinMerge. Programmet körs inte automatiskt, i stället finns hyperlänkar på förstasidan av XLSX-filen för projektet i en egen lista, detta visas i Figur 7.

Hyperlänken öppnar WinMerge med de två filerna som hade ändringar och presenterar ändringarna, se Figur 8.

File	Result	External Hyperlink
atest.txt	See hyperlink for changes.	Link
ctest.txt	See hyperlink for changes.	Link

Figur 7: Hyperlänkar till WinMerge från XLSX-fil



Figur 8: Ändringar av två filer, presenterade i WinMerge

4.2.7 Hantering av CTG- och CTF-Filer

CTG-filer kan medföra problem relaterade till AVEVA Plant SCADA. Vanligtvis får filer som ligger i olika projekt under samma anläggning använda samma namn. Exempelvis så har varenda projekt en dBase-fil med namn ”pages.dbf”. Men specifikt CTG- och CTF-filer får enbart finnas en gång i en anläggning, annars blir det fel vid körning av SCADA-anläggningen. Dubbletter av dessa filer kan förekomma då utvecklare kopierar innehåll från andra SCADA-anläggningar för att effektivisera sitt arbete, utan vetskapen att den kopierade filen redan existerade. För att hantera detta problem sparar applikationen namn på alla CTG- och CTF-filer vid validering av en anläggning. Om det finns dubbletter kommer dessa sedan visas i det slutgiltiga resultatet. För att bekräfta vilken av dubbletterna som var korrekt finns en dBase-fil som heter ”pages.dbf”. Från denna kan korrekta filsökvägar extraheras till CTG- och CTF-filer. Dessa visas i resultatet och på så vis vet användaren vilken fil av dubbletterna som bör raderas ur anläggningen, se Figur 9.

CTF-filer är kompilerade CTG-filer och därför finns även ett problem om det finns en CTF-fil utan relaterad CTG-fil. Detta orsakar fel i projektet i AVEVA Plant SCADA. För att hantera detta fel jämför applikationen vid validering av anläggning alla filnamn av CTF- och CTG-filer för att se att alla CTF-filer har en motsvarighet i form av CTG-fil. Om inte så påvisas vilken CTF-fil som saknar CTG-fil i det slutgiltiga resultatet, se Figur 9.

CTF/CTG-duplicates in facility		
File	Path	Is in pages.dbf
!alarmcomments.ctf	D:\ExJobb\TestFolder\User\Example\!AlarmComments.ct	YES
!alarmcomments.ctf	D:\ExJobb\TestFolder\User\ExampleSA\!AlarmComment	NO
!alarmcomments.ctg	D:\ExJobb\TestFolder\User\Example\!AlarmComments.ct	YES
!alarmcomments.ctg	D:\ExJobb\TestFolder\User\ExampleSA\!AlarmComment	NO
CTF-files found without corresponding CTG-file		
File	Path	
!sg1_xpitune.ctf	D:\ExJobb\TestFolder\User\Include\!sg1_xpitune.ctF	

Figur 9: Resultatet med dubbletter av CTG- samt CTF -filer finns samt ett exempel där en CTF-fil saknar tillhörande CTG-fil

4.2.8 Generering av batch-filer

Vid jämförelse av filer som inte var dBase, skulle dessa om de var förändrade kunna öppnas i WinMerge genom hyperlänkar i Excel. Hyperlänkar i Excel kan endast öppna ett program eller en länk i taget och därmed inte WinMerge samt de två filerna som jämförts.

För att lösa detta skapades en metod som skapar batch-filer, dessa öppnar WinMerge samt de jämförda filerna. För varje fil som inte är dBase där det finns förändringar mellan de två jämförda eller filen endast finns i nya eller gamla versionen av datan, skapas en batch-fil. Denna batch-filen kopplas sedan till hyperlänken i XLSX-filen som ska öppna WinMerge.

4.2.9 Validering av dBase-filer

I vissa fall är det viktigt att specifika dBase-filer uppfyller olika regler. För att säkerställa detta går det att validera filerna med hjälp av olika SQL-kommandon. Användaren av verktyget kan själv styra över vilka valideringsregler som ska användas och vilka filer som reglerna gäller. Frågorna anpassar användaren med hjälp av redigerbara CSV:er som innehåller reglerna. Dessa CSV:er förklaras mer utförligt under rubriken 4.4.3.

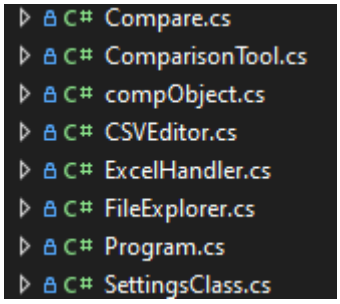
Valideringen fungerar på så vis att applikationen först tar fram namn på alla dBase-filer som blivit inmatade. Därefter kollas vilka valideringsregler som är relaterade till dessa filer.

Då valideringen körs, försöker denna spara data i en DataTable-tabell baserat på SQL-kommandot relaterat till valideringen. Om denna tabell efter körningen har någon form av data i sig innebär det att valideringsregeln är bruten. Då sparas datan i tabellen och visas sedan i det slutgiltiga resultatet.

4.2.10 Dokumentation och struktur av kod

Applikationens struktur är uppbyggd på en metodik med många små metoder i stället för få stora metoder. Något som anses vara mer lättförståeligt samt dynamiskt och effektivt, då många metoder kan återanvändas för olika syften.

Koden är även uppdelad i flera olika klasser där varje klassnamn är relaterat till vad dess metoder gör. Exempelvis befinner sig alla metoder som läser från eller skriver till filer i klassen ”FileExplorer”. Detta för att underlätta förståelse för utvecklare som ska navigera i koden. Se Figur 10 för alla projektets klasser.



Figur 10: Klasser i projektet

Metodiken som använts för att dokumentera koden är att kortfattat beskriva varje metod med en kommentar kring dess funktionalitet vid dess metodsignatur.

4.3 Presentation av data

Som nämnt i kodavsnittet så sparas data som är ändrad för att presenteras i en XLSX-fil. Varje gång applikationen körs söker den först efter Excel. Om Excel finns kommer Excel användas för att öppna XLSX-filerna. Om inte Excel finns, söker applikationen efter Libre Office Calc och använder i stället den mjukvaran för att öppna XLSX-filerna. Detta eftersom alla inte har tillgång till Excel på grund av licenskrav.

Om två anläggningar har jämförts kommer användaren först presenteras med en sammanfattningsfil som innehåller en överblick över de separata projektjämförelserna. I Figur 11 syns sammanfattningsfilen som innefattar en summa av alla ändringar i ett projekt samt en hyperlänk till XLSX-filen där projektjämförelsens resultat presenteras.

Compared Facilities: AnläggningNew/AnläggningOld
Date and time of comparison: 2023-05-23 09:42:22
Ran by user: Joel

Changed projects with DBF-files are listed below

Project	Result	Hyperlink
ööåProjekt	8 Changes.	Link
threedBase	16 Changes, 1 Validation-rules broken.	Link
Kommunikation	Project was added in the new facility	
ProjektAndreas	Project was added in the new facility	

CTF-files found without corresponding CTG-file

File	Path	
l.ctf	C:\TestFolder\AnläggningNew\threedBase\L.CTF	

Figur 11: Sammanfattningsfil i Excel

Framsidan av XLSX-filen till en projektjämförelse innehåller information om vilka filer i projektet som har ändringar, hur många ändringar som fanns i projektet samt en hyperlänk till kalkylbladet som presenterar ändringarna. Allt detta syns i Figur 12.

.

Compared Projects: threedBase/threedBase
Date and time of comparison: 2023-05-04 10:51:33
Ran by user: Joel

Changed DBF-files are listed below

File	Result	Worksheet Hyperli
created.dbf	9 Comparison Change(s), 2 Columns(s) Added/Removed	Link
created2.dbf	2 Comparison Change(s)	Link
variable.dbf	5 Comparison Change(s)	Link
extra.dbf	File was added in the new project	
extra2.dbf	File was added in the new project	

Changed non-DBF-files are listed below

File	Result	External Hyperlink
atest.txt	See hyperlink for changes.	Link
ctest.txt	See hyperlink for changes.	Link
btest.txt	NonDBF-File was added in the new project	Link
X.ctg	NonDBF-File was added in the new project	Link
files.txt	NonDBF-File was removed in the new project	Link
more.txt	NonDBF-File was removed in the new project	Link
testing.txt	NonDBF-File was removed in the new project	Link

Validation warnings are listed below

Rulename	Result	Worksheet Hyperli
RULENAME1	127 error(s), Warning. Variable without Equiment	Link
RULENAME2	160 error(s), Warning. No Cluster in variable.dbf	Link

Color Representation	
Added/New Data	Deleted/Old Data
A row has this color if it contains Deleted/Old data	
A row has this color if it contains Added/New data	

If the Tag has been Added/Deleted, or if an entire row of data has been Added/Deleted, the entire row will be colored accordingly.If a column has been Added/Deleted the header will be colored.

NON-DBF-Details

File-extension	Description
.ci	Cicode
.ctg	Page
.ctl	Symbol
.ctm	Genie
.ctt	Template
.txt	
.xml	

Figur 12: Projektfil

Det finns två typer av ändringar i dBase-filer som presenteras på olika sätt, dessa förklaras nedan.

Ändring av ”tagg”/nyckel: Alla rader data i en dBase-fil har ett namn eller en nyckel som första kolumn för att kunna identifieras. Om en nyckel enbart finns i den gamla filen presenteras det i resultatet som en borttagen rad. Raden blir i resultatet färgad enligt användarens inställning på variabeln ”colorDeleted”. Om en nyckel i stället endast finns i den nya filen, presenteras det i resultatet som en tillagd rad. Raden blir då i stället färgad enligt variabeln ”colorAdded”. Se Figur 13 för visualisering.

NAME	TYPE	UNIT	ADDR	RAW_ZERO	RAW_FULL
BatchTotalXX	INT	DISK_PLC	I10	0	32000
BatchTotal	INT	DISK_PLC	I10	0	32000
ääö	DIGITAL	DISK_PLC	D11		

Figur 13: Ändring av nyckel (NAME) där "colorDeleted" är röd och "colorAdded" är grön

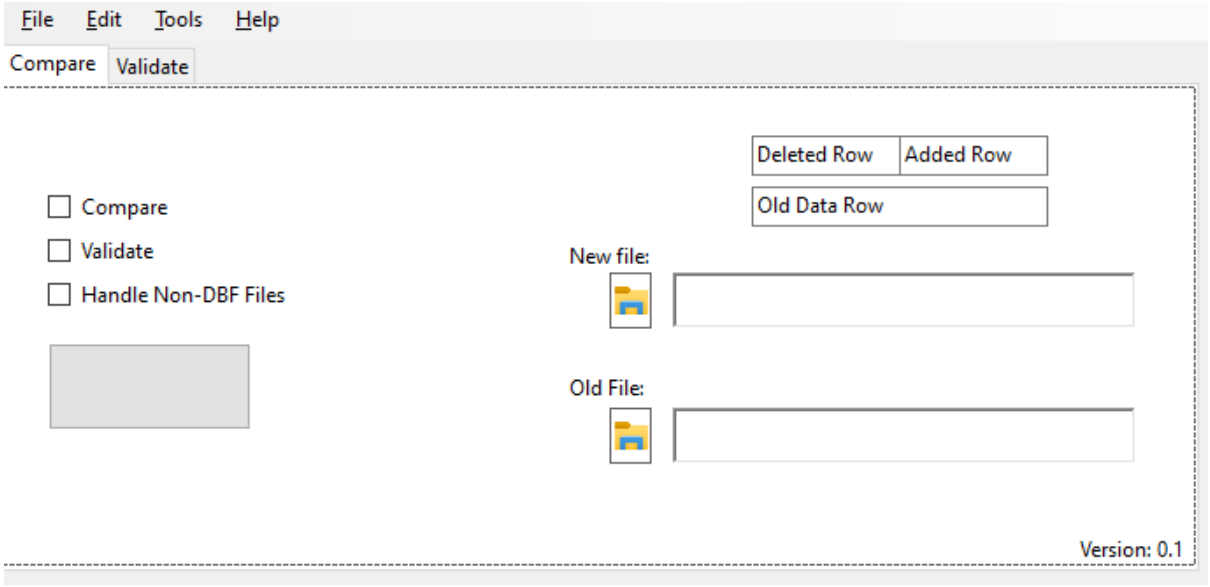
Ändring av data i rad: Om nyckeln inte är ändrad men ett eller flera element på en rad är ändrade så presenteras både den gamla och nya raden där elementen befann sig i resultatet. Raden med den gamla datan blir färgad enligt variabeln ”colorOld” och de specifika elementen med förändringar har färg från variabeln ”colorDeleted”. Raden med ny data får endast de ändrade elementen färgade enligt variabeln ”colorAdded” och bakgrundsfärgen är vit. Detta visas i Figur 14.

	A	B	C	D
1	NAME	TYPE	UNIT	ADDR
2	Arrow_Network3	DIGITAL	IODevInternalxx	D15
3	Arrow_Network3	DIGITAL	IODevInternalchanged	D15
4	BATCH	INT	DISK_PLCxx	I9
5	BATCH	INT	DISK_PLC	I9
6	BIT_10	DIGITAL	DISK_PLC14	D10
7	BIT_10	DIGITAL	DISK_PLC	D10

Figur 14: Förändringar i data där "colorDeleted" är röd, "colorAdded" är grön och "colorOld" är grå

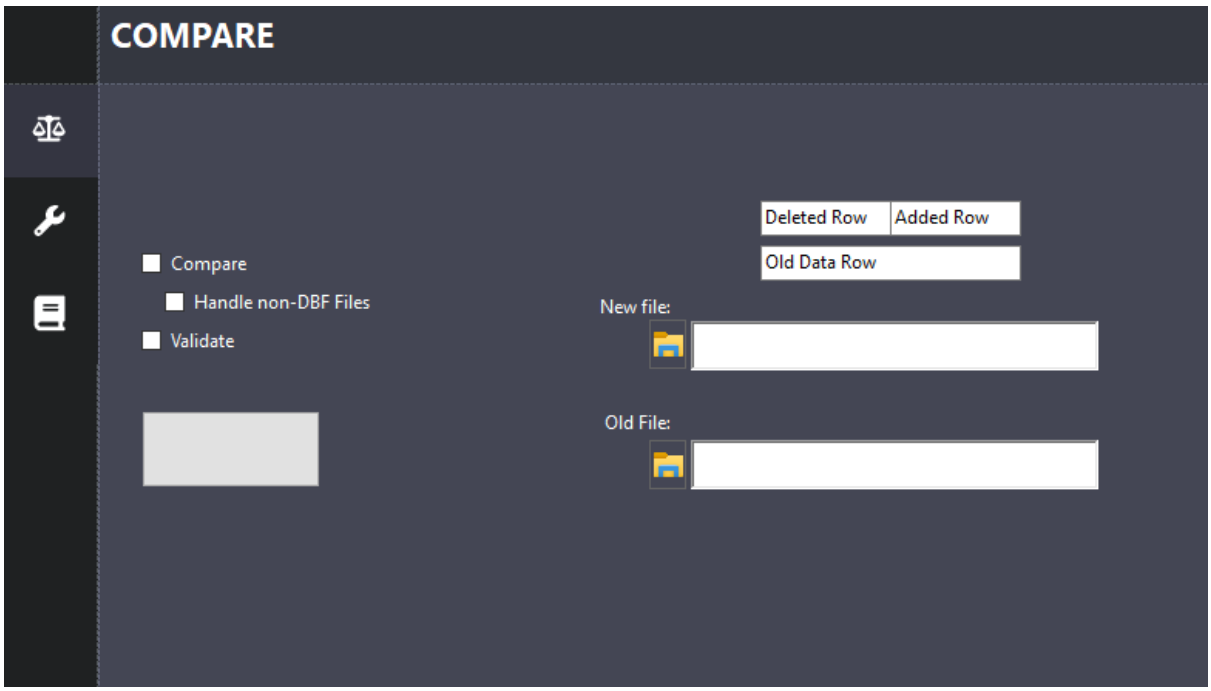
4.4 Gränssnitt

Till en början användes Windows Forms standardobjekt för att skapa ett enkelt gränssnitt som syns i Figur 15. Målet var att skapa ett gränssnitt som var lättförståeligt vilket var möjligt. Men det fanns svårigheter att göra det snyggt och inbjudande med endast den funktionalitet Windows Forms erbjöd.



Figur 15: Första version av gränssnitt

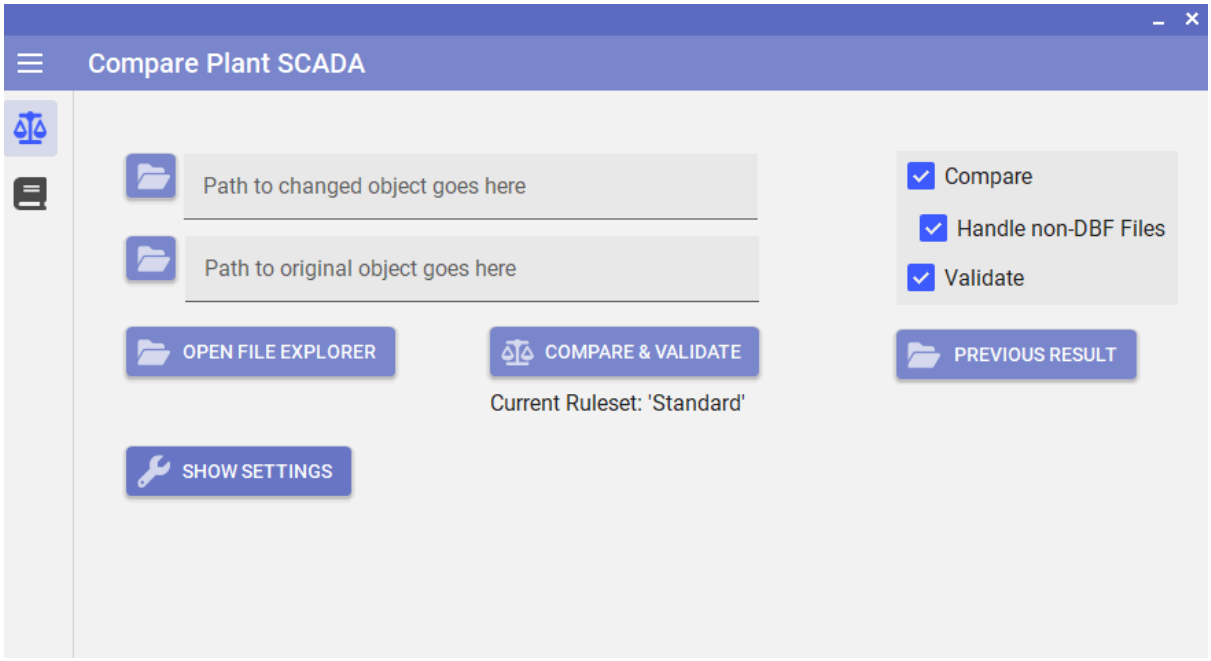
Version två av gränssnittet såg lite modernare ut med hjälp av ikoner från NuGet-biblioteket FontAwesome i sidomenyn till vänster med alla programmets flikar. Version två av gränssnittet syns i Figur 16.



Figur 16: Version två av gränssnitt

Trots uppdateringen kändes gränssnittet fortfarande väldigt gammeldags och det slutgiltiga resultatet vidareutvecklades med hjälp av NuGet-biblioteket MaterialSkin 2, vilket hade större designmöjligheter. Med hjälp av MaterialSkin 2 fick det slutgiltiga

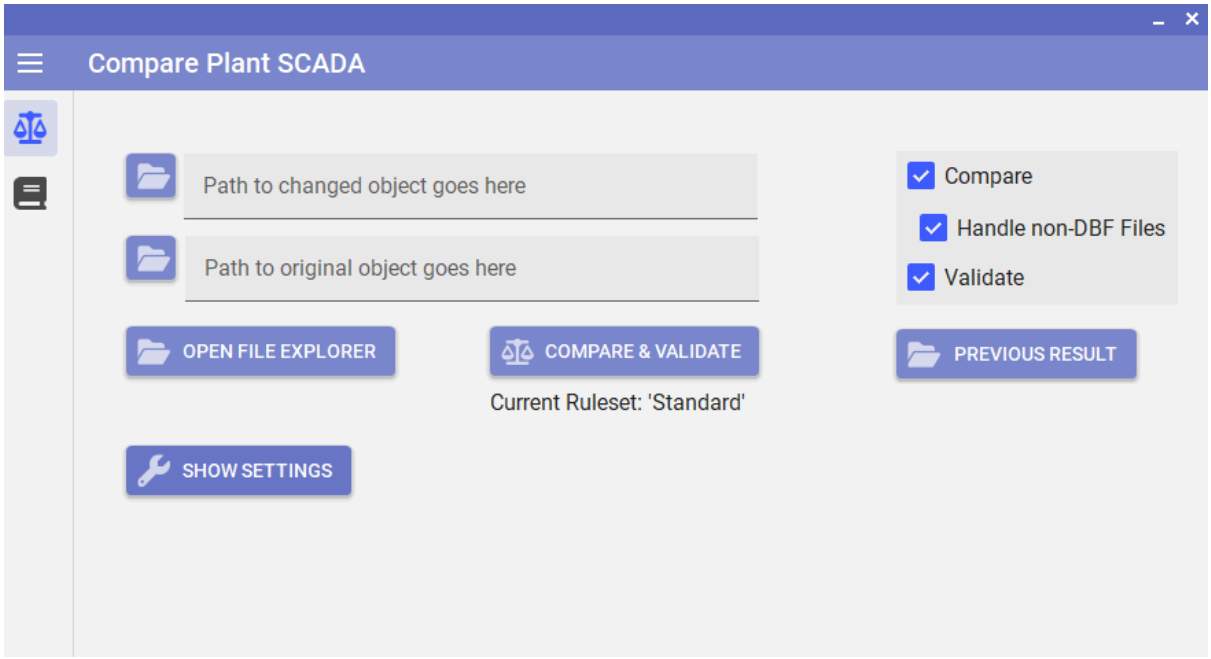
gränssnittet ett mer modernt och inbjudande utseende. Något som kändes viktigt för användarvänligheten. Det slutgiltiga gränssnittet syns i Figur 17.



Figur 17: Slutgiltigt gränssnitt

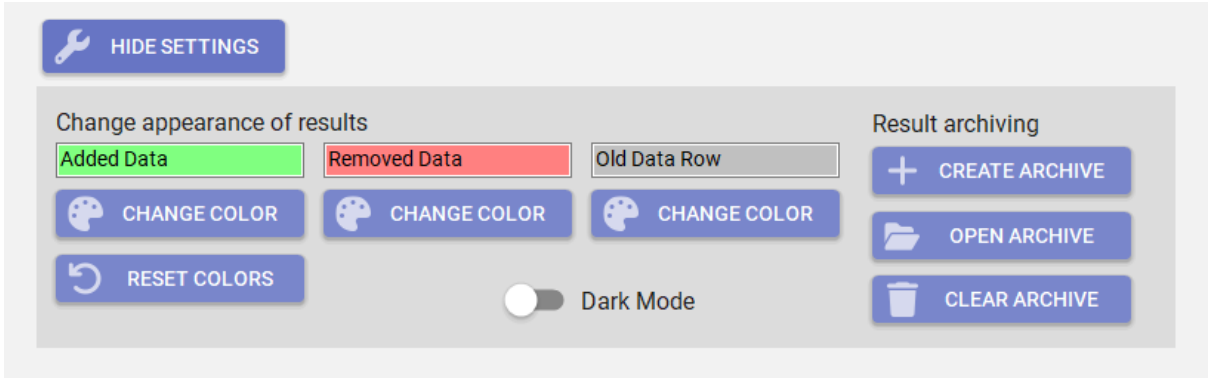
4.4.1 Huvudflik

När applikationen startas visas huvudfliken för användaren vilket syns i Figur 18. Den här fliken används majoriteten av tiden då programmet är aktivt eftersom det är härifrån jämförelse och validering av indata sker. Kryssrutorna till höger är inställningarna på vad som är aktiverat under körningen. Den första och andra inställningen uppifrån handlar om jämförelse, varav den första enbart aktiverar jämförelse för dBase-filer medan den andra hanterar alla andra filtyper användaren tidigare valt att aktivera. Den tredje rutan kryssas i om användaren vill validera det objekt som ligger i den övre textrutan. När användaren har valt vad som ska göras så göms de element av programmet som inte är relevanta för att tydliggöra vad som efterfrågas av användaren. Exempelvis syns endast den övre textrutan om användaren bara valt att kryssa i ”Validate”.



Figur 18: Huvudsidan av applikationen med två textrutor som ska innehålla sökväg för de objekt som ska hanteras

Det enda som ej är relaterat till körningen av programmet är knappen ”SHOW SETTINGS”. Knappen tar fram inställningar gällande färg samt arkivering av senaste resultat, se Figur 19. Färginställningarna påverkar vilka färger som sedan används i resultatet, detta eftersom det kan existera användare som har svårigheter för vissa färger. Om användaren vill spara undan det senaste resultatet går även det att göra härifrån genom att arkivera det via ”CREATE ARCHIVE”.



Figur 19: Knapptryckning av "SHOW SETTINGS"

4.4.2 Progressbar

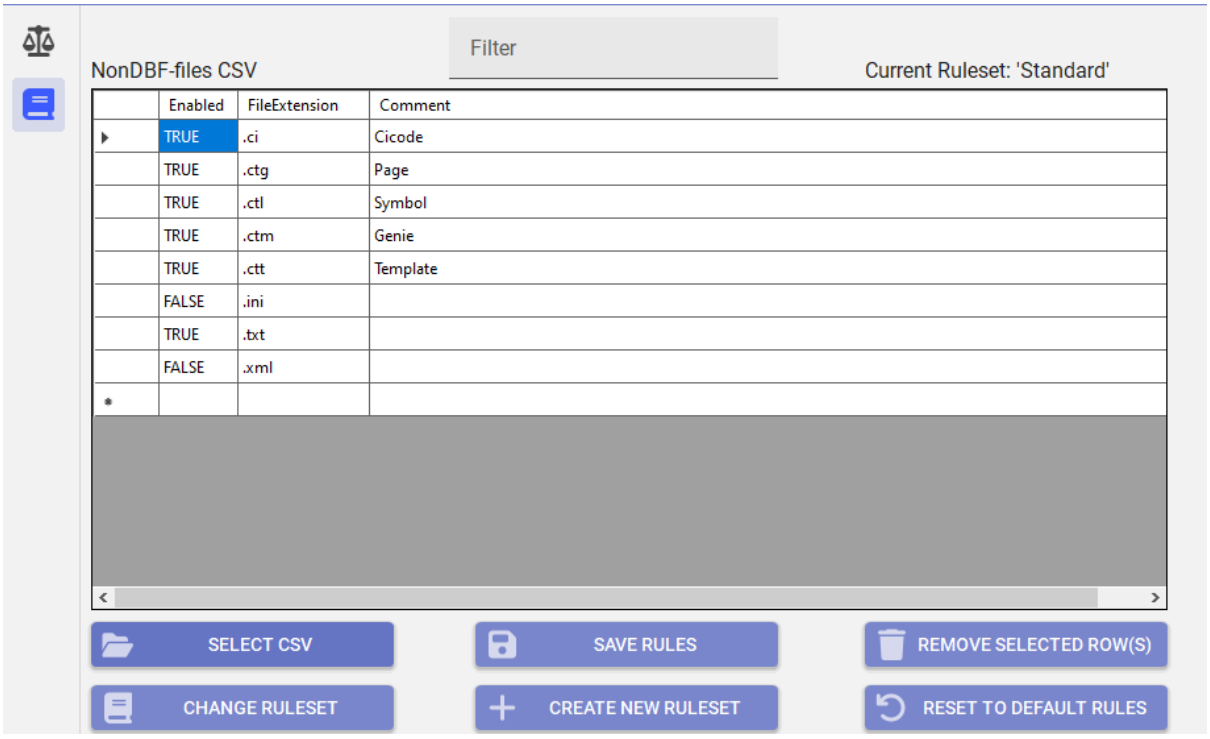
Det är viktigt för applikationens användarvänlighet att det går att följa programmet när det arbetar. Detta förebygger förvirring kring om programmet har fryst, men även att användaren stänger av programmet innan det arbetat klart. Ett vanligt sätt att

presentera detta är med hjälp av en progressbar som under arbete visar ett uppskattat värde på hur mycket arbete som genomförts samt hur mycket som finns kvar.

Projektets progressbar visas inte innan dess att användaren har startat en körning. När programmet arbetar rapporteras status när den kommer förbi vissa viktiga punkter i programmet och även relativt kontinuerligt under huvudarbetet som innefattar jämförelsen och valideringen av filerna. När körningen är färdig och progressbaren visas som full så skrivs det ut att en XLSX-fil genereras och öppnas, då detta kan ta tid på vissa datorer som kör applikationen.

4.4.3 Regelflik

Programmets andra flik är regelfliken som syns i Figur 20. Denna innehåller funktionalitet för att ändra de fundamentala regler som finns sparade i CSV-filer. När en fil valts för att redigeras så kommer dess data upp i en ruta som efterliknar ett Excel-ark, med rader och kolumner. Större andelen av datan som ska sparas härifrån valideras direkt när användaren ändrar den, då programmet bygger på att dessa filer är uppbyggda på ett korrekt sätt. Eftersom datan valideras automatiskt här får användaren direkt respons om något hen skrivit inte fungerar.



Figur 20: Funktionalitet för att uppdatera applikationens regler

Under regelfliken finns tre olika CSV-filer med regler som används vid varje körning och som kan redigeras genom ett eget gränssnitt. Dessa CSV-filer innefattar följande:

DBase-filer: CSV-filen innehåller namn på alla filer som automatiskt genereras av AVEVA Plant SCADA. Filen har fyra kolumner som innefattar om filen skall läsas in vid jämförelse/validering, filens namn, om filen har en specifik regel samt en kommentar relaterad till filen.

Valideringsregler: Information relaterad till valideringsregler finns i en CSV-fil med fyra kolumner. Dessa innehåller information huruvida regeln ska användas, vilka filer regeln ska användas på, SQL-kommandot för regeln samt en kommentar.

Ej dBase-filer: CSV-filen med information angående filer som inte är dBase-filer har tre kolumner. Dessa innefattar om filtypen är aktiv vid jämförelse, filändelsen samt en kommentar.

Alla dessa regler kan ändras av användaren genom ett gränssnitt i applikationen. Exempelvis om en ny valideringsregel ska läggas till eller specifika dBase-filer ska hanteras annorlunda vid jämförelse. Detta sker med hjälp av en DataGridView i C# som påminner om en matris till utseendet vilket visas i Figur 21.

.

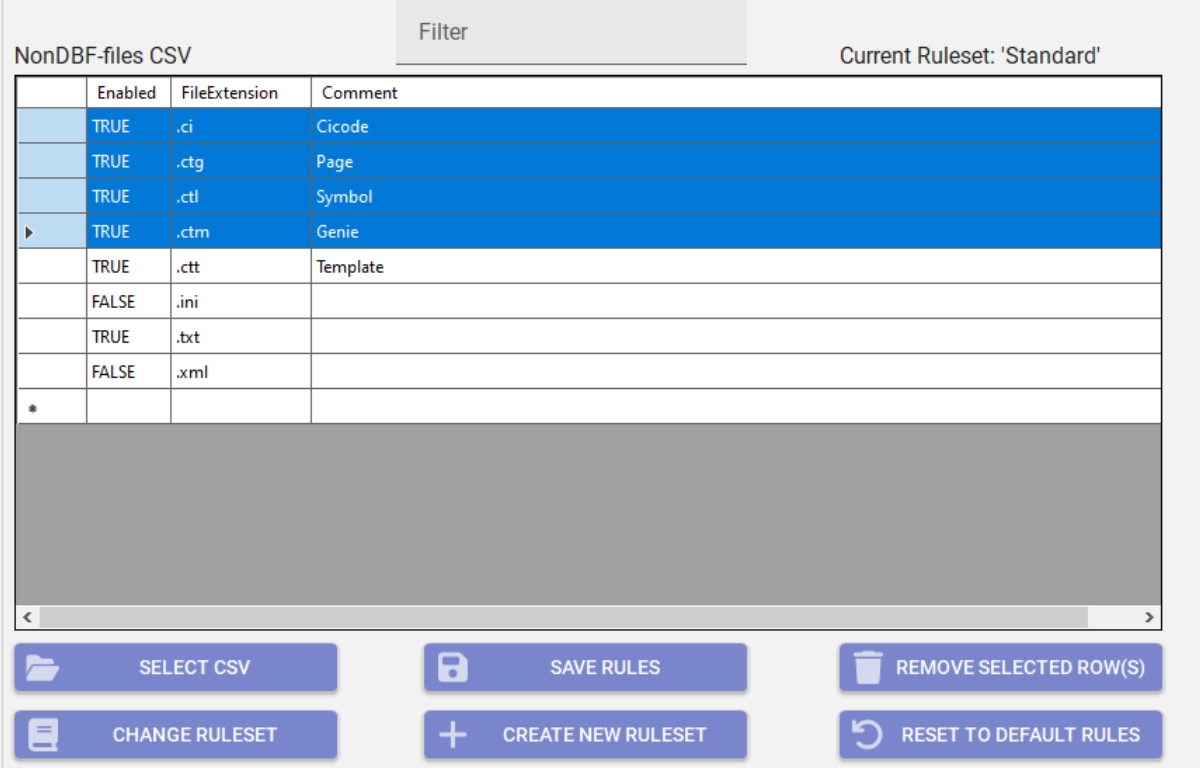
	Enabled	File	SQL-Query	Comment
	FALSE	eqgenie.DBF	SELECT * FROM {0} order by 1 asc	
	FALSE	LibHelp.DBF	SELECT * FROM {0} order by 1 asc	
	FALSE	trntype.DBF	SELECT * FROM {0} order by 1 asc	
	FALSE	Hebrew(Israel)....	SELECT * FROM {0} order by 1 asc	
	FALSE	Chinese(Simplif...	SELECT * FROM {0} order by 1 asc	
	FALSE	French(France)....	SELECT * FROM {0} order by 1 asc	

Figur 21: DataGridView med information relaterad till dBase-filer

Då användaren försöker ändra existerande- eller lägga till ny data, valideras denna innan den accepteras. Exempelvis måste dBase-filer sluta på ”.dbf” för att kunna läggas till, annars får användaren ett felmeddelande och ändringen annulleras. Liknande validering sker vid all form av ändring/tillägg av data i CSV-filer genom gränssnittet. På så vis kan inte användaren skriva in felaktig data som sedan vid körning av applikationen resulterar i en krasch.

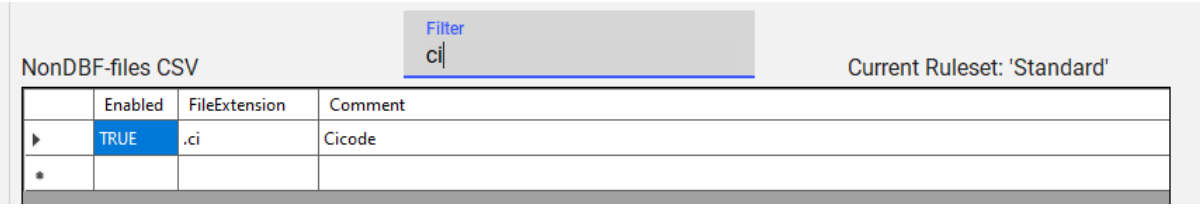
Det finns möjlighet att markera rader i gränssnittet och ta bort dessa genom knappen ”REMOVE SELECTED ROW(S)”. Det existerar även en knapp för att återställa reglerna till utgångsläget via ”RESET TO DEFAULT RULES”. Knappen återställer nuvarande CSV-fil till ett förbestämt standardläge. Detta händer genom att nuvarande CSV-fil blir överskriven med innehåll från en annan existerande CSV-fil som har all standard-data sparad. Markerad data samt knappar syns i Figur 22.

.



Figur 22: Gränssnitt för redigering av CSV-fil

Om CSV-filerna är stora kan det vara svårt att manövrera all data, därför finns det även en filter-funktion för att ändra specifika rader med information. Där filtret baseras på fil- eller regelnamn, beroende på vilken CSV-fil som är aktiv. Se Figur 23 för ett exempel.



Figur 23: Filtrera efter "ci" bland filer som inte är dBase-filer

För att slutligen spara de förändringar användaren gjort i regel-gränssnittet får hen trycka på ”SAVE RULES”-knappen. Först då uppdateras nuvarande CSV-fil med de nya ändringarna. Om användaren ej sparat ändringarna och försöker öppna huvudfliken eller byta CSV-fil, uppmanas den först att spara ändringarna då de annars annulleras.

Det finns också möjlighet att spara olika uppsättningar med regler som kallas ”rulesets” vilket syns längst ned i Figur 22. När ett nytt ”ruleset” skapas så bildas det tre nya CSV:er för dBase-filer, valideringsregler samt filer som inte är dBase-filer. Deras ursprungliga innehåll är baserat på ett förbestämt standardläge. Funktionaliteten finns för att olika anläggningar/projekt kan kräva jämförelse/validering på olika vis. Då är

det mycket effektivare att kunna skapa och spara uppsättningar med regler för varje anläggning/projekt i stället för att innan varje körning behöva skriva nya- eller ändra existerande regler. Innan körning kan användaren då välja mellan de olika ”rulesets” som skapats och välja det som passar bäst för nästkommande körning.

4.5 Utveckling av installation

När verktyget fungerade som förväntat i utvecklingsmiljö började arbete med att exportera det till en separat applikation för att användare ska kunna använda den utan en utvecklingsmiljö. I Visual Studio finns en projekttyp som kallas ”Setup Project” och denna användes för att skapa installationsfiler.

Vid körning av installationsfilerna kollar installeringen om användaren har .NET 7 installerat, annars efterfrågas det att användaren ska installera det. Detta för att applikationen inte fungerar annars. Det följer även med CSV-filer samt XLSX-filer i installationen som automatiskt placeras under användarens AppData-mapp. Detta för att applikationen ska veta var dessa filer finns vid uppstart samt ska ha tillgång att skriva till dem.

I installationen medföljer även setup-filer till mjukvaran WinMerge, då denna mjukvara utökar applikationens användningsområde. Varje gång applikationen körs kollas det om WinMerge finns installerat, annars blir användaren ombedd att installera mjukvaran och en mapp där dess setup-filer finns, blir öppnad.

4.6 Testning med virtuell maskin

När det var möjligt att installera applikationen och använda den utan Visual Studio upptäcktes det flera fel för handledaren som inte upptäcktes under utvecklingen. Troligen beroende på att projektdeltagarna både hade applikationen installerad separat samt alla filer relaterade till Visual Studio-projektet. Något handledaren inte hade, då han endast hade den separata applikationen installerad.

För att testa applikationen med samma förutsättningar som en ny användare användes då en virtuell maskin med hjälp av programvaran VMware. För att på så vis se hur applikationen fungerade på en dator utan någon tidigare relation till applikationen. Detta underlättade testning genom att fel upptäcktes självmant i stället för att handledaren behövde upptäcka dem vid sin testning. Detta sparade tid för alla involverade parter samt var lärorikt.

5. Resultat

Under projektet har en applikation skapats som kan jämföra och validera filer som används av AVEVA Plant SCADA. Om det finns förändringar eller regler blir brutna i valideringen av filerna så genererar applikationen XLSX-filer som presenterar detta. I XLSX-filerna visas vad som skiljer mellan de jämförda filerna samt vilka regler som är brutna och var i de validerade filerna som dessa regler bryts, se Figur 12 för en överblick.

DBase-filer med förändringar får egna flikar i de genererade XLSX-filerna där datan visas som i Figur 13.

Andra typer av filer med ändringar presenteras genom hyperlänkar i XLSX-filerna som externt visar skillnaderna med hjälp av mjukvaran WinMerge, detta visas i Figur 8.

Vilka filer och vilka valideringsregler som ska användas på dessa filer är upp till användaren. Med hjälp av ett CSV-redigerings-gränssnitt som finns i applikationen kan användaren fritt redigera hur och vilka filer som ska jämföras och valideras, se Figur 20.

6. Diskussion

Applikationen har uppfyllt de inledande krav som ställdes och därmed har projektet utförts som efterfrågat. Programmet hanterar filer från olika versioner av AVEVA Plant SCADA och jämför samt validerar dessa filer baserat på användarens förfrågan. Gränssnittet applikationen använder är av modern och enkel karaktär för att tilltala användaren.

6.1 Utvecklingsprocess

Angreppssättet som användes fungerade väl. Till en början tog det ganska lång tid att implementera ny funktionalitet då alla metoder behövde skapas från grunden och extra arbete lades på att göra metoderna generella. Mot slutet av arbetet kunde många metoder återanvändas och på så vis blev implementering av ny funktionalitet mer och mer effektivt med tiden.

Vid starten av arbetet planerades ungefär hälften av tiden till jämförelse av dBase-filer och hälften till validering av dBase-filer. Planeringen för jämförelsedelen av arbetet stämde väl och tog hälften av tiden. Implementering av validering gick mycket bättre än planerat och gick väldigt fort. Därmed lades tid som blev över på att förbättra grundfunktioner och att skapa ett modernare gränssnitt.

6.2 Programmering och kod

Programmeringsspråket som användes under projektet var C#. Potentiellt hade projektet även kunnat utföras i Java men fördelarna med C# var övervägande. Mycket beroende på att ett tidigare verktyg med snarlika egenskaper hade skapats på Acobia i C#. Det visade sig vara ett mycket bra val då C# har en stor mängd bibliotek tillgängliga genom NuGet vilket kan underlätta många problem och skapa många lösningar.

Arbetssättet har under hela projektet varit att börja med små byggstenar och sedan återanvända dessa för att lösa större problem. På så vis kan små och stora problem lösas med samma metoder. Med det sagt har väldigt lite av arbetet behövt ändras eller tas bort, då tänket hela tiden har varit att bygga ut på det som redan existerar. Känslan är att arbetssättet har fungerat bra men det finns också en risk att lösningen därmed kunde varit bättre om fler reflektioner gjorts regelbundet i stället för att alltid satsa på vidareutveckling av den existerande lösningen.

För att få mer input från handledaren på Acobia och därmed anpassa applikationen mer efter en ”korrekt” användare hade det varit möjligt att tidigare dela med sig av versioner som handledaren kunde testa under sin arbetstid. Det mest effektiva arbetet under hela projektet har varit då handledaren kommit med konkreta förslag på förbättringar som då fort kunnat implementeras baserat på beskrivningar. Dessa förslag är lättare att komma fram till med verktyget framför sig, därmed borde det exporterats tidigare för att på så vis möjliggöra för mer och tidigare input på förbättringar och förändringar.

Generellt sett är det viktigt att hålla kod kommenterad och läsbar, även om det bara är utvecklaren själv som ska hantera koden. I detta projekt är det extra viktigt att hålla koden läsbar då programvaran samt koden ska lämnas över till Acobia efter att exjobbet är färdigställt. Den använda metodiken för dokumentation var att varje metod skulle vara lättförståelig nog för att det skulle räcka med en kommentar vid dess metodsignatur. Denna metod tvingar utvecklare att skriva metoder simpla nog att förklaras med en kortfattad beskrivning, något som inte alltid går. Därmed hade koden potentiellt kunnat dokumenteras mer genomförligt i vissa fall.

Det finns vissa delar i koden som hade kunnat förbättrats. Den nuvarande lösningen för att hantera specialtecken i filer från AVEVA Plant SCADA är hårdkodad och baserad på de fall som hittats under projektets gång. Detta är en mycket kortsiktig lösning och ej dynamisk. Eftersom det med stor sannolikhet kommer upptäckas fler specialtecken då fler användare har tillgång till verktyget. En bättre lösning hade varit att hantera detta redan vid inläsning av filen eller att tillåta användare att kontinuerligt uppdatera de fall som hittas under användning av applikationen.

På Acobia används idag fler SCADA-system utöver AVEVA Plant SCADA, beroende på arbetsområde och utvecklare. Alla utvecklare arbetar inte i AVEVA Plant SCADA och arbetar i stället bland annat med mjukvarorna: Webport [20] och iFix [21]. Därmed hade det varit fördelaktigt om applikationen även kunde hantera jämförelse samt validering av dessa mjukvarors filer och generera resultat på liknande vis. Alla SCADA-system använder filer på olika vis, därmed hade detta varit en ganska stor utökning av applikationen men även en stor förbättring, då utvecklare på Acobia hade kunnat använda en applikation för all versionshantering samt validering. Detta hade även skapat en större grupp användare av applikationen vilket leder till mer respons och åsikter för att vidareutveckla verktyget ännu mer.

Det hade även gått att skapa en webbaserad lösning så att användaren inte behövde ladda ner någon mjukvara. Detta hade potentiellt varit smidigare för användarna men förkunskaperna som krävts för detta var inte något som fanns vid projektstart.

Något som hade underlättat och effektiviserat testning under de tidigare stadierna av applikationens utveckling är unittester. Detta är något som används under utveckling för att se att program fungerar som förväntat när man fortsätter att bygga vidare på ett program med ny funktionalitet. Detta sker genom programmerbara tester, där indata och förväntat resultat matas in i testet. Fås inte det förväntade resultatet efter testet innebär det att testet misslyckats och att en bugg existerar. Unittesting försökte implementeras under projektet, men fungerade inte som förväntat och orsakade problem med applikationen. Tester gjordes i stället manuellt med egna filer.

6.3 Etiska och ekologiska aspekter

En del av den data som verktyget kan komma att hantera ligger under sekretess, därmed är det viktigt ur en etisk och även legal aspekt att verktyget inte sparar någon data externt.

Målet med verktyget är att effektivisera redan existerande arbetsuppgifter som sker manuellt vid dator. Ur en ekologisk aspekt är det svårt att avgöra hur detta kommer påverka miljön, ett resonemang är att det nuvarande manuella arbetet kräver mer datortid och därmed mer energi. På så vis har verktyget en positiv inverkan på miljön. Men, det krävs även energi att utveckla, underhålla samt använda verktyget och därmed är det svårt att avgöra hur energiåtgången kommer se ut i de olika fallen.

6.4 Gränssnitt och användarvänlighet

För att applikationen ska användas måste den se inbjudande ut och får inte vara för krånglig för användaren att förstå vid en första överblick. Till en början användes Windows Forms standardobjekt då de var direkt åtkomliga utan extra bibliotek, vilket underlättade utvecklingen. Men att designa ett program med ett modernt gränssnitt genom detta visade sig vara svårt, då standarden är gammal. Att få objekten att se bra ut var besvärligt och tidskrävande. Därför söktes ett bibliotek som kunde underlätta processen att skapa ett snyggt gränssnitt.

Biblioteket som valdes var MaterialSkin 2 av anledningen att dess inkluderade design var enkel och modern. Det gick lätt att bygga ett projekt med ett sammanhängande tema då alla objekten var länkade till samma temainställning, som lätt kunde uppdateras. Dessutom fanns större delen av alla objekt från Windows Forms, fast med utökad funktionalitet. Exempelvis textrutor som möjliggjorde tips som står kvar i rutorna även medan användaren skriver i dem. Det tillät även för en lätt övergång till nattläge där alla programmets färger följer ett mörkare tema, vilket kan komma att föredras av vissa användare.

Dock orsakade biblioteket även vissa problem för projektet. Det låste vissa tidigare tillgängliga egenskaper relaterade till exempelvis dynamik. Att få objekt att flytta sig och ändra storlek baserat på applikationsfönstrets storlek blev mycket svårare och vissa objekt krävde med biblioteket mer arbete än de från Windows Forms krävt för att få samma resultat. Men det slutgiltiga resultatet är ändå att föredra då applikationens moderna utseende gör den mer påtaglig och inbjudande för nya användare.

Acobia ägs av koncernen Init som har utvecklare i Sverige, Norge samt Danmark. Därför är det flera olika språk som talas och används dagligen på företaget. Detta är också anledningen till att applikationen är på engelska, för att möjliggöra för maximalt antal användare med det mest generella språket. Applikationen hade dock varit mer användarvänlig om det fanns språkval som lät användaren välja mellan svenska, norska och danska utöver endast engelska. Det är en relativt enkel förbättring men det är också något som tar en hel del tid. Särskilt att hålla alla språk uppdaterade med varje förändring av verktyget.

Under majoriteten av projektet användes mjukvaran DiffMerge för presentation av ändringar i filer utöver dBase-filer då den hade ett smidigt gränssnitt. Men mot slutet av projektet upptäcktes det att denna mjukvaras senaste uppdatering skedde 2013. Därmed valdes då att byta mjukvara till WinMerge i stället som fortfarande uppdateras regelbundet med sin senaste uppdatering under 2023. Detta var en ändring som ledde till lite sämre användarvänlighet på kort sikt men potentiellt leder till att applikationen kan användas till sin fulla potential över en längre tid.

7. Slutsats

Baserat på frågeställningarna kunde följande slutsatser dras:

En applikation med ett enkelt gränssnitt har skapats. Denna kan på kort tid generera ett lättläst resultat som påvisar skillnader samt brutna regler mellan filer som används av AVEVA Plant SCADA.

Resultatet presenteras för användaren i en XLSX-fil, där all relevant data presenteras med olika färger baserat på användarens val. När applikationen körs söker den efter en mjukvara som ska användas för att hantera XLSX-filer. Huvudsakligen Excel eller Libre Office Calc.

Applikationen använder CSV-filer med regler relaterade till filer som används av AVEVA Plant SCADA. Om filerna från AVEVA Plant SCADA uppdateras eller ändras kan användare av applikationen uppdatera CSV-filerna med regler och då kommer applikationen fungera som förväntat. Därmed är applikationen dynamisk nog för att hantera potentiella uppdateringar av AVEVA Plant SCADA.

8. Referenser

[1] AVEVA, ”aveva.com,” AVEVA, i.å.. [Online]. Tillgänglig: https://www.aveva.com/en/products/plant-scada/. [Använd 24 januari 2023].

[2] Inductive Automation, ”inductiveautomation.com,” Inductive Automation, 2018. [Online]. Tillgänglig: https://inductiveautomation.com/resources/article/what-is-scada. [Använd 2023-01-24].

[3] A. Karlsson, Interviewee, *privat kommunikation.* [Intervju]. 16 mars 2023.

[4] SCADA International, ”scada-international.com,” SCADA International, i.å.. [Online]. Tillgänglig: https://scada-international.com/what-is-scada. [Använd 2023-03-15].

[5] Microsoft, ”learn.microsoft.com,” Microsoft, 2021. [Online]. Tillgänglig: https://learn.microsoft.com/en-us/dotnet/framework/get-started/overview. [Använd 2023-03-15].

[6] Microsoft, ”learn.microsoft.com,” Microsoft, 2022. [Online]. Tillgänglig: https://learn.microsoft.com/en-us/nuget/what-is-nuget. [Använd 2023-03-15].

[7] ”leadtools,” leadtools, i.å.. [Online]. Tillgänglig: https://www.leadtools.com/help/sdk/v21/dh/to/file-formats-microsoft-excel-spreadsheet-xlsx-xls.html. [Använd 4 maj 2023].

[8] Microsoft, ”learn.microsoft.com,” Microsoft, 2023. [Online]. Tillgänglig: https://learn.microsoft.com/en-us/dotnet/desktop/winforms/overview/?view=netdesktop-7.0/. [Använd 2023-03-15].

[9] ”winmerge,” winmerge, [Online]. Tillgänglig: https://winmerge.org/?lang=sv. [Använd 26 april 2023].

[10] Codebean, ”codebean.se,” Codebean, i.å. [Online]. Tillgänglig: https://www.codebean.se/objektorienterad-programmering-klasser-och-objekt/. [Använd 16 03 2023].

[11] Microsoft, ”learn.microsoft.com,” Microsoft, 13 02 2023. [Online]. Tillgänglig: https://learn.microsoft.com/en-us/dotnet/csharp/tour-of-csharp/. [Använd 2023-03-15].

[12] Microsoft, ”azure.microsoft.com,” Microsoft, i.å. [Online]. Tillgänglig: https://azure.microsoft.com/sv-se/resources/cloud-computing-dictionary/what-are-databases/. [Använd 2023-05-24].

[13] Amazon, ”aws.amazon.com,” Amazon, i.å. [Online]. Tillgänglig: https://aws.amazon.com/what-is/sql/. [Använd 2023-03-17].

[14] H. McGilton och J. Gosling, ”oracle.com,” maj 1996. [Online]. Tillgänglig: https://www.oracle.com/java/technologies/introduction-to-java.html. [Använd 2023-03-17].

[15] D. Thakur, ”ecomputernotes.com,” Ecomputernotes, i.å. [Online]. Tillgänglig: https://ecomputernotes.com/fundamental/introduction-to-computer/batch-file. [Använd 16 03 2023].

[16] EPPlus Software AB, ”epplussofware.com,” 2023. [Online]. Tillgänglig: https://www.epplussoftware.com/en. [Använd 2023-03-16].

[17] ”nuget.org,” nov. 2021. [Online]. Tillgänglig: https://www.nuget.org/packages/MaterialSkin.2/#readme-body-tab. [Använd 2023-04-17].

[18] Oracle, ”oracle,” i.å. [Online]. Tillgänglig: https://www.oracle.com/se/cloud/compute/virtual-machines/what-is-virtual-machine/. [Använd 12 april 2023].

[19] VMware, ”vmware.com,” i.å. [Online]. Tillgänglig: https://www.vmware.com/topics/glossary/content/virtual-machine.html. [Använd 12 april 2023].

[20] Kiona, ”kiona.com,” Kiona, i.å. [Online]. Tillgänglig: https://kiona.com/sv/produkter/web-port. [Använd 2023-04-12].

[21] GE Digital, ”ge.com,” GE Digital, i.å. [Online]. Tillgänglig: https://www.ge.com/digital/applications/hmi-scada/ifix. [Använd 2023-06-07].


AVANCEZ
1829
CHALMERS