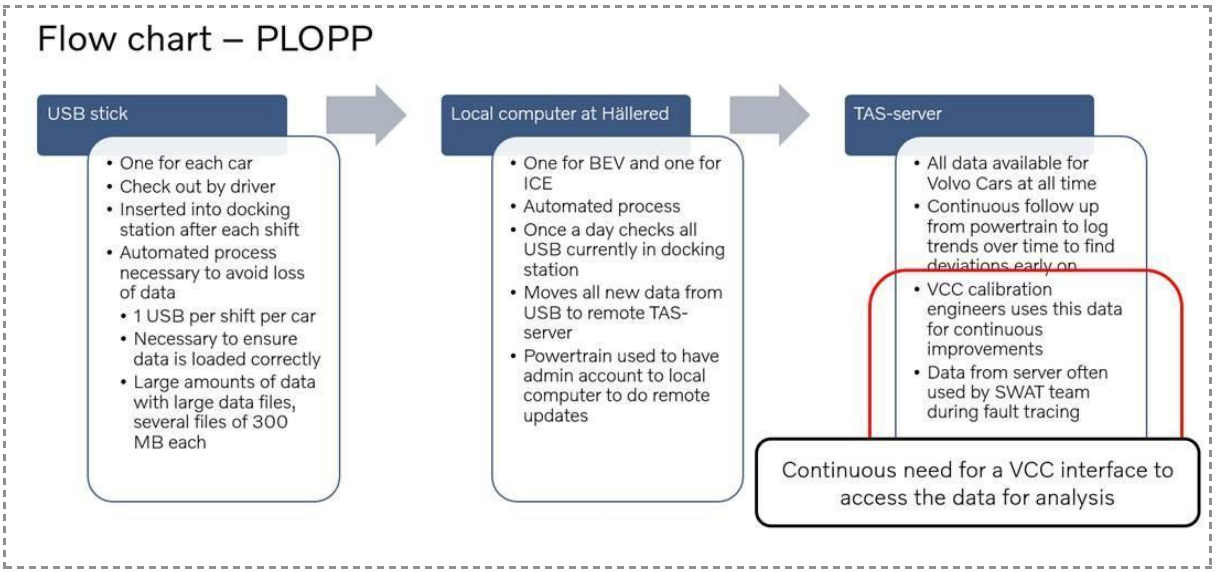
Automatiserad filöverföring och metadatahantering



- Projektsponsor (ägare): Volvo Cars
- Utförare: Fakhreddin Kabawe
- Handledare: Christian Magnusson, Team Manager
- Kontaktperson: Johan Storm, Analysingenjör
- Kontaktperson för PLOPP: Niklas Westring, Analysingenjör SW

Sammanfattning

Syftet med detta exjobb var att skriva en kod för att flytta filer och mappar från externa enheter till en specifik mapp på servern, och att göra det på ett effektivt och säkert sätt. Detta uppnåddes genom att skriva en Python-kod som kunde hantera olika fil- och mappstrukturer, ignorera vissa filer och mappar och endast flytta filer som inte redan fanns i målmappen.

För att uppnå detta genomförde jag fyra faser. I den första fasen planerade och förberedde genom att identifiera våra mål och behov, skapa en tidsplan och en lista över nödvändiga resurser, installera nödvändiga verktyg och mjukvaror, och skapa en struktur för projektet.

I den andra fasen samlade jag in och förberedde data för analysen genom att sortera och organisera data på ett lämpligt sätt.

I den tredje fasen skrev jag koden för att uppnå vårt mål. Vi testade koden för att se till att den fungerade korrekt och förbättrade den baserat på feedback och tester.

Slutligen, i den fjärde fasen, rapporterade jag om processen och resultaten, utvärderade resultaten och jämförde dem med våra ursprungliga mål, skapade en presentation och presenterade resultaten för intressenter och sammanfattade projektet.

Resultatet av detta exjobb är en fungerande Python-kod som effektivt och säkert kan flytta filer och mappar från externa enheter till en specifik mapp på serven. jag hoppas att denna kod kommer att vara till nytta för användare som behöver hantera och organisera filer från olika källor.

Innehållsförteckning

Introduktion

Denna rapport beskriver arbetet med att utveckla en kod för att flytta filer från en testbil till en server. Syftet med arbetet är att underlätta överföringen av testresultat från bilens usb-sticka till servern och därmed effektivisera processen för att analysera och utvärdera testresultaten.

Koden som har utvecklats är skriven i Python och den har utvecklats i fyra faser under en åtta veckors period. Varje fas fokuserade på en specifik del av arbetet och hade en specifik målsättning.

I den första fasen av arbetet, utvecklades funktioner för att flytta filer från en källmapp till en målmapp och dessa funktioner testades sedan för att säkerställa att de fungerar korrekt.

I den andra fasen av arbetet, implementerades en algoritm för att hitta enheter som inte är C-disken och för att flytta filerna från dessa enheter till en målmapp plus det har tillagts en säkerhetsfunktion som kontrollerar namn på usb-sticka som kopplas in och om usb-stickans namn har de önskade namn då började flytt av filer för att säkerställa att inte flytta filer från usb stickor som inte tillhör processen.

I den tredje fasen av arbetet, utvecklades en funktion för att skapa en ny mapp på servern för flyttade filer och det namnet skapades med hjälp av usb-stickans namn som används för att skapa mappen och funktionerna testades för att säkerställa att mappen skapades korrekt.

I den fjärde fasen av arbetet, integrerades alla de tidigare funktionerna i en sammanhängande kod som kunde köras automatiskt. Koden testades för att säkerställa att alla filer flyttades korrekt och att koden kunde köras utan problem.

Syfte och mål

Syftet med detta examensarbete var att undersöka och utveckla en metod för att automatisera backup och säkerhetskopiering av filer och mappar på servern.

Målet var att skapa en fungerande och pålitlig lösning som kan användas av både enskilda användare och företag för att skydda sina viktiga filer och data från förlust på grund av hårddisk kraschar, virusattacker eller andra oförutsedda händelser. Genom att automatisera backup processen skulle användarna kunna spara tid och undvika manuella fel samtidigt som de säkerställer att deras data alltid är skyddade och tillgängliga när de behövs.

Tidsplan

Vecka 11-12:

- Läsa på om filöverföring och skriva kod för att flytta filer från en enhet till en annan.
- Testa koden på olika enheter och filtyper för att säkerställa att den fungerar korrekt.

Vecka 13-14:

- Implementera möjligheten att överföra filer från flera enheter samtidigt.
- Skapa en mappstruktur på servern för att organisera de överförda filerna.

Vecka 15-16:

- Lägg till felhantering i koden för att undvika att filer går förlorade eller flytta felaktigt.
- Testa felhanteringen och se till att den fungerar som förväntat.

Vecka 17-18:

- Optimering av koden för att förbättra prestanda och minska överföringstiden.
- Dokumentation av koden och dess funktioner.
- Skriva en rapport om arbetet och dess resultat.

Design av programmet:

Programmet består av en Python-skriptfil som används för att flytta filer från en lokal enhet till en målenhet. Programmet består av följande huvudsakliga funktioner:

Programmet använder också följande Python-moduler:

1. os - för att arbeta med filsystemet.
2. shutil - för att flytta filer och mappar.
3. filecmp - för att jämföra filer.
4. win31api - för att identifiera USB-enheter, hämta information om enheten
5. time - för att hantera tiden.

Programmets design säkerställer att filerna och mapparna som flyttade är de som inte redan finns i destinationsmappen och att vissa filer och mappar (som System Volume Information och Recycler-mappar) flytta till else mapp.

Programmets design säkerställer också att filerna och mapparna sparas i mappar som är organiserade efter enhetens namn(bilens id nummer). Om en mapp för den aktuella bilen inte redan finns skapas en ny mapp. Detta säkerställer enkel åtkomst och översiktlig organisation av filerna.

Koden i detaljer

Import av moduler:

```
import os               # Importerar funktioner för att interagera med operativsystemet
import shutil           # Importerar funktioner för att hantera filer och mappar
import filecmp          # Importerar funktioner för att jämföra filer
import win32api         # Importerar funktioner för att interagera med Windows-API:et
import time             # Importerar funktioner för att hantera tid och vänta mellan operationer
```

I programmet används fem moduler: os, shutil, win32apim time och filecmp. Genom att importera dessa moduler får programmet tillgång till en uppsättning funktioner och metoder som förenklar och effektiviserar kod skrivningen.

Funktionen move_files:

```
def move_files(source_folder, dest_parent_folder):
    total_files = 0
    num_files = 0

    for root, dirs, files in os.walk(source_folder):
        total_files += len(files)

    dest_folder = os.path.join(dest_parent_folder)
    os.makedirs(dest_folder, exist_ok=True)

    for root, dirs, files in os.walk(source_folder):
        for item in files:
            source_item = os.path.join(root, item)
            extension = os.path.splitext(item)[1].lower()

            if extension in ('.rar', '.zip'):
                move_zip_file(source_item)

            elif extension in ('.mf4', '.dat'):
                move_mf4_file(source_item)

            else:
                move_else_file(source_item)

            num_files += 1
            if num_files != total_files:
                print(f"{num_files}/{total_files} Ta inte bort {volume_info[0]}, Flyttade filen : {item[:20]}")

    remove_empty_folders(source_folder)

    if num_files == total_files:
        print(f"\nAlla filer har nu flyttats från {volume_info[0]} till destinationmappen. \nDu kan nu tryggt ta bort  {volume_info[0]}  :)")
```

Funktionen "move_files" tar två argument: "source_folder" och "dest_parent_folder". Den använder modulen "os" för att navigera genom filsystemet och hantera filer.

1. Först skapar funktionen två variabler: "total_files" och "num_files". "total_files" kommer att hålla antalet filer som finns i källmappen och "num_files" kommer att räkna antalet filer som har flyttats.

2. Sedan går funktionen igenom alla underkataloger och filer i källmappen genom att använda "os.walk" funktionen. Varje gång en fil hittas i källmappen, ökar den totala filräknaren med antalet filer i den aktuella katalogen.

3. Nästa steg är att skapa en målmapp i "dest_parent_folder" där alla filer kommer att flyttas till. Funktionen använder "os.makedirs" för att skapa mappen och "exist_ok=True" innebär att mappen kommer att skapas om den inte redan finns.

4. Efter att målmappen har skapats går funktionen igenom källmappen igen. För varje fil i källmappen hämtar funktionen filnamnet och filändelsen med hjälp av "os.path.splitext" och sparar filnamnet i variabeln "source_item". Funktionen avgör sedan vilken typ av fil det är baserat på filändelsen och anropas en annan funktion (move_zip_file, move_mf4_file eller move_else_file) för att flytta filen till rätt målkatalog.

5. Efter att en fil har flyttats ökar funktionen filräknaren med 1 och skriver ut ett meddelande som informerar användaren om vilken fil som har flyttats och hur många filer som återstår att flytta.

6. Slutligen anropar funktionen en annan funktion (remove_empty_folders) för att ta bort eventuella tomma mappar i källmappen.

Funktionen move_zip_file

```
def move_zip_file(source_item):
    """
    Funktionen move_zip_file flyttar en zip-fil från källmappen till en zip-mapp i målmappen.
    zip_folder är sökvägen till zip-mappen där filen ska flyttas till.
    os.makedirs används för att skapa zip-mappen om den inte redan finns.
    dest_item är sökvägen för den nya filen i zip-mappen.
    shutil.move används för att flytta filen från källmappen till zip-mappen.
    Ingen utskrift görs i denna funktion.
    """
    folder_name_zip = os.path.splitext(os.path.basename(volume_info[0]))[0].split('_')[0]
    zip_folder = os.path.join(r"C:\Users\fakhe\Desktop\testing\zip", folder_name_zip , 'zzip')
    os.makedirs(zip_folder, exist_ok=True)
    dest_item = os.path.join(zip_folder, os.path.basename(source_item))
    shutil.move(source_item, dest_item)
```

1. Funktionen "move_zip_file" tar ett argument "source_item", som är sökvägen till zip-filen som ska flyttas från källmappen till zip-mappen i målmappen.

2. Först definierar funktionen variabeln "folder_name_zip" genom att extrahera en del av namnet på zip-filen från variabeln "volume_info". Namnet på zip-mappen kommer att vara den här delen av filnamnet.

3. Sedan skapar funktionen en variabel "zip_folder" som är sökvägen till zip-mappen i målmappen. Detta görs genom att använda "os.path.join" för att kombinera sökvägarna till målmappen, den aktuella zip-mappen som ska skapas (med hjälp av variabeln "folder_name_zip") och en undermapp "zzip". "os.makedirs" används för att skapa zip-mappen om den inte redan finns.

4. Efter att zip-mappen har skapats skapar funktionen en variabel "dest_item", som är sökvägen för den nya filen i zip-mappen. Detta görs genom att använda "os.path.join" för att kombinera sökvägen till zip-mappen med filnamnet på zip-filen som ska flyttas (hämtat med "os.path.basename").

5. Till sist flyttar funktionen zip-filen från källmappen till zip-mappen i målmappen med hjälp av "shutil.move". Inga utskrifter görs i funktionen.

Funktionen move_mf4_file och move_else_file

```python
def move_mf4_file(source_item):
    """
    samma som förra men till mf4 och dat filerna
    """
    folder_name_mf4 = os.path.splitext(os.path.basename(volume_info[0]))[0].split('_')[0]

    mf4_folder = os.path.join(r"C:\Users\fakhe\Desktop\testing\data", folder_name_mf4 , 'data')
    os.makedirs(mf4_folder, exist_ok=True)
    dest_item = os.path.join(mf4_folder, os.path.basename(source_item))
    shutil.move(source_item, dest_item)


def move_else_file(source_item):
    """
    samma som förra men till alla filer som inte är mf4, dat, zip eller rar
    """
    else_folder = os.path.join(r"C:\Users\fakhe\Desktop\testing\else", volume_info[0] , 'else')
    os.makedirs(else_folder, exist_ok=True)
    dest_item = os.path.join(else_folder, os.path.basename(source_item))
    shutil.move(source_item, dest_item)
```

Funktionerna move_mf4_file och move_else_file utför samma sak som move_zip_file fast skickar filerna till separata mappar beroenden på filens typ.

Funktionen remove_empty_folders

```python
def remove_empty_folders(folder):
    """
    Tar bort tomma undermappar i en mapp.
    """
    if not os.path.exists(folder) or not os.path.isdir(folder):
        return

    for root, dirs, files in os.walk(folder, topdown=False):
        for dir_name in dirs:
            full_dir_path = os.path.join(root, dir_name)
            try:
                os.rmdir(full_dir_path)
            except OSError:
                pass

    try:
        os.rmdir(folder)
        print(f"Tagit Bort mappen: {folder}")
    except OSError:
        pass
```

Funktionen "remove_empty_folders" tar en sökväg till en mapp som argument.

1. Först kontrollerar funktionen om sökvägen existerar och om det är en mapp med hjälp av "os.path.exists" och "os.path.isdir". Om sökvägen inte existerar eller inte är en mapp returnerar funktionen.

2. Därefter använder funktionen "os.walk" för att iterera genom alla undermappar och filer i den angivna mappen. För varje undermapp i "dirs" sökvägen, kontrollerar funktionen om mappen är tom och tar bort den om det är fallet genom att använda "os.rmdir". Eftersom "os.rmdir" endast kan ta bort tomma mappar, så är det säkert att använda den här funktionen.

3. Efter att alla tomma undermappar har tagits bort, försöker funktionen ta bort den ursprungliga mappen med "os.rmdir". Om mappen inte kan tas bort ignoreras felet tyst och inga utskrifter görs. Om mappen tas bort skrivs en utskrift till konsolen som visar den raderade mappen.

Huvud koden:

```python
if __name__ == '__main__':
    # connected_drives är en mängd som innehåller enhetsbokstäverna för alla anslutna enheter.
    connected_drives = set()
    #En oändlig loop börjar som kontinuerligt söker efter nya anslutna enheter och borttagna enheter.

    while True:
        # new_drives är en lista som innehåller enhetsbokstäverna för nya anslutna enheter som inte redan finns i connected_drives.
        new_drives = [d for d in 'ABDEFGHIJKLMNOPQRSTUVXYZ' if d not in connected_drives and win32api.GetLogicalDrives() & (1 << ord(d) - 65)]
        # Om det finns nya enheter, läggs de till i mängden "connected_drives"
        if new_drives:
            connected_drives.update(new_drives)
        # disconnected_drives är en lista som innehåller enhetsbokstäverna för borttagna enheter från mängden connected_drives.
        disconnected_drives = connected_drives - {d for d in connected_drives if win32api.GetLogicalDrives() & (1 << ord(d) - 65)}
        # Om det finns borttagna enheter, skrivs ett meddelande ut med enheternas bokstäver och de tas bort från mängden "connected_drives"
        if disconnected_drives:
            print("Usb(s) frånkopplad:", ", ".join(disconnected_drives))
            connected_drives -= disconnected_drives

        time.sleep(1)
        # Loopar igenom alla nya enheter
        for new_drive in new_drives:
            try:
                # Hämtar information om enheten och skriver ut dess namn
                volume_info = win32api.GetVolumeInformation(new_drive + ':\\')
                print(f"Hittade enhet: {volume_info[0]}")
                if '_VPT_PLOPP' in volume_info[0] or '_ATA' in volume_info[0]:
                    # Sätter sökvägen till enheten och listar alla filer i sökvägen
                    source_folder = new_drive + ':\\'
                    items = os.listdir(source_folder)
```

```python
                    if items:
                        #dest_parent_folder = r'\\gbw9061109.got.volvocars.net\PROJ2\9413-SHR-VCC127500\MEP2\Hällered'
                        dest_parent_folder = volume_info[0]
                        drive_folder = os.path.join(dest_parent_folder)
    #                    move_files(drive_folder, source_folder)
                        move_files(source_folder, drive_folder)
                    else:
                        # Om enheten inte innehåller "_Data" i volymnamnet, skrivs ett meddelande ut om det.
                        print("Inga filer att Flytta på enheten: " + new_drive)
                else:
                    print(f"{volume_info[0]} innehåller inte '_VPT_PLOPP '.")

            except Exception as e:
                print()
```

Sammanfattningsvis söker koden efter anslutna enheter och flyttar filer från usb_stickor till en specifik destination om vissa söktermer matchar det önskade namnet.

1. Den här delen av koden söker efter anslutna enheter på en Windows-dator och flyttar filer från en specifik enhet till en annan destination. Koden använder win32api-biblioteket för att få information om enheter och volymer, och os-biblioteket för att hantera filer och mappar.

2. Koden börjar med att definiera en mängd "connected_drives" som innehåller enhetsbokstäverna för alla anslutna enheter. Sedan startar en oändlig loop som kontinuerligt söker efter nya anslutna enheter och borttagna enheter.

3. I varje iteration av loopen söker koden efter nya enheter genom att jämföra enhetsbokstäverna i en fördefinierad lista med enhetsbokstäverna i "connected_drives". De nya enheterna läggs till i mängden "connected_drives". Sedan söker koden efter borttagna enheter genom att jämföra "connected_drives" med enheterna som för närvarande är anslutna. De borttagna enheterna tas bort från "connected_drives".

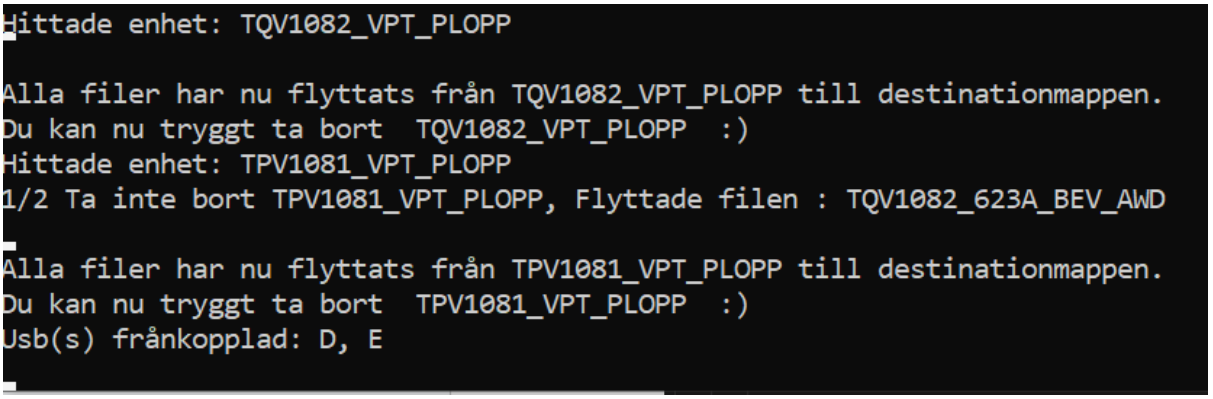
4. Om det finns nya enheter skriver koden ut ett meddelande om att en enhet har hittats och hämtar information om enheten. Om enhetens volymnamn innehåller vissa söktermer, till exempel "_VPT_PLOPP" eller "_ATA", flyttar koden filer från enheten till en specifik destination. Filerna flyttas bara om det finns filer på enheten. Annars skrivs ett meddelande ut som indikerar att det inte finns några filer att flytta. Om volymnamnet inte innehåller de sökta termerna, skrivs ett meddelande ut om det.

5. Koden fångar eventuella undantag som kan uppstå under körning och skriver ut dem. Detta hjälper till att identifiera eventuella problem som kan uppstå under körning.

Budget

Ingen budget lagts ner eftersom allting som behövs redan finns tillgängligt och det har inte uppstått något behov av att köpa något ytterligare. De nödvändiga resurserna för att genomföra projektet är tillgängliga inom organisationen och tillhandahålls av intressenterna.

Resultat



När en enhet ansluts till datorn visas ett meddelande med texten: "Hittade enhet: enhetens namn". Därefter kontrolleras antalet och typen av filer som finns på USB-stickan. Beroende på filernas typ flyttas de till den förinställda mappen.

Under flyttningen visas en varning och antalet flyttade filer räknas för att jämföra med antalet filer som fanns ursprungligen.

När alla filer har flyttats visas följande meddelande:
"Alla filer har nu flyttats från "enhetsnamnet" till destinationmappen.
Du kan nu tryggt ta bort “enhetsnamnet” :)".

Vid borttagning av USB-stickan visas ett meddelande som informerar vilken/vilka enheter som har kopplats bort.

Riskanalys

Risknivå och konsekvens:
Systemkrasch (hög risk, hög konsekvens),
Examensarbetet blir klart innan tidsplanen (Hög risk, låg konsekvens),
Nytt jobberbjudande leder till byte av arbetsplatsen (låg risk, hög konsekvens).

Förutsättningar, begränsningar och avgränsningar

Det finns några viktiga förutsättningar och begränsningar som är relevanta för examensarbetet. För det första är det viktigt att notera att systemet PLOPP redan är implementerat och används i test körningarna på Volvo Cars i Hällered.

Examensarbetet handlar om att förbättra systemet genom att automatisera flyttningen av filer till servern och att ta fram metadata per bil för att enklare kunna plotta relevant data.

En annan viktig förutsättning är att det finns en kontaktperson för PLOPP, Niklas Westring, som kommer att vara tillgänglig för att hjälpa till med att konstruera examensarbetet.

Det finns vissa tekniska begränsningar som måste beaktas, till exempel kapaciteten för servern som filerna kommer att flyttas till och möjligheterna att hantera stora mängder data på ett effektivt sätt.

Problem

Under projektet stötte jag på några problem som jag behövde lösa.
Det första problemet jag stötte på var att programmet flyttade vissa filer och mappar som var systemfiler eller cachade filer. Jag löste detta genom att lägga till kod för att flytta över de filerna och mapparna till en else mapp där allt oönskat hamnar.

Det andra problemet jag stötte på var när jag försökte flytta en fil som var öppen och i användning av ett annat program. Programmet kunde inte flytta filen och kraschade med ett felmeddelande. Jag löste detta genom att använda try-except-block för att hantera undantag och skriva ut felmeddelanden om inflyttningsprocessen misslyckades.

Det fjärde problemet jag stötte på var att koden inte hittade USB-stickorna som var anslutna till datorn. Eftersom USB-stickor inte alltid har samma enhetsbeteckning, kunde koden inte hitta enheterna med hjälp av enhetsbeteckning.

För att lösa problemet valde jag att identifiera enheterna baserat på deras beteckning istället. Genom att skapa en lista över enheter som börjar med bokstäverna D till Z, kunde jag hitta alla USB-stickor som var anslutna till datorn och loopa igenom dem för att flytta filer.

En annan lösning var att undvika att flytta data från C-enheten på datorn, eftersom den alltid är tillgänglig på datorn och det inte finns något behov av att göra en säkerhetskopia av den. Istället kollade koden efter andra enheter och flyttade filerna därifrån.

Totalt sett var dessa problem tidskrävande och medelsvåra att lösa, men det var viktigt att hantera dem för att säkerställa att programmet fungerade korrekt och pålitligt men genom att identifiera och lösa problemet kunde jag fortsätta med projektet och slutföra det på ett framgångsrikt sätt.

Diskussion

Under projektets gång har olika aspekter diskuterats och utvärderats för att säkerställa att programmet är så effektivt och användarvänligt som möjligt. Här följer en kort diskussion om några av de viktigaste aspekterna.

Prestanda:

Programmets prestanda har varit en viktig faktor att ta hänsyn till under utvecklingen. Det har varit viktigt att se till att programmet kan hantera stora filer och mappar utan att det påverkar systemets prestanda allt för mycket. För att uppnå detta har funktioner som att hoppa över vissa mappar och filer implementerats. Dessutom har programmet utvecklats för att vara så effektivt som möjligt när det gäller att jämföra filer och mappar för att undvika onödig filöverföring.

Användarvänlighet:

Programmet har utvecklats med användarvänlighet i åtanke. Det har varit viktigt att göra det så enkelt som möjligt för användaren att använda programmet utan att behöva ha för mycket teknisk kunskap. För att uppnå detta har gränssnittet utformats så enkelt som möjligt och alla meddelanden och felmeddelanden är tydligt beskrivna.

Säkerhet:

Säkerheten har också varit en viktig faktor att ta hänsyn till. Programmet har utvecklats så att det inte kan flytta filer från enheter som inte innehåller det önskade namnet. Dessutom har programmet utvecklats så att det endast kan köras av en auktoriserad användare som har rättigheter att flytta filer.

Sammanfattningsvis har utvecklingen av programmet varit en lärorik och utmanande uppgift. Genom att ta hänsyn till prestanda, användarvänlighet och säkerhet har programmet utvecklats för att vara så effektivt som möjligt. Med hjälp av detta program kan filöverföring från bil till server utföras på ett enkelt och säkert sätt.

Gantt Diagram

Gantt-diagram

Kod

```
# Koden är ett Python-skript som flyttar filer från en källmapp till en målmapp baserat på filtyp.
# Koden innehåller en funktion "move_files" som är huvudfunktionen och tar två argument, källmappen och målmappen.
# Källmappen och dess undermappar genomsöks rekursivt för att hitta filer som sedan flyttas till lämpliga målmappar baserat på deras filtyp.
# Om filen är en mapp, flyttas filerna i den mappen rekursivt till motsvarande målmapp baserat på filtyp.
# Funktionen tar också bort tomma mappar i källmappen efter att alla filer har flyttats.

# Koden använder moduler som os, shutil, filecmp, win32api och time för att hantera filer och mappar, jämföra filer, interagera med operativsystemet och hantera tid.
# Det finns också tre hjälpfunktioner, "move_zip_file", "move_mf4_file" och "move_else_file", som flyttar filer med .zip, .mf4 och .dat filtyp, respektive andra filtyper till lämpliga målmappar.
# Slutligen finns det en funktion "remove_empty_folders" som tar bort tomma mappar i en mapp.
# Koden söker också kontinuerligt efter nya anslutna enheter, och om en ny enhet hittas som innehåller en mapp med en viss namnstruktur
# körs huvudfunktionen för att flytta filer från den mappen till lämpliga målmappar.

# Koden är skriven och testad av Fakhreddin kabawe
# Om du behöver hjälp med koden, kontakta mig på [fakhreddin.kabawe@icloud.com] Eller på [+46721270123]
# en detaljerad rapport om koden finns på https://docs.google.com/document/d/1zs4iZ8eTRHo7vqXMQJBo3UMa03LaC13MpeIjZ2uMUOc/

# win32api är inte installerad på alla datorer av sig själv och "pip install pypwin32" behöver köras
# Koden flyttar bort filerna om man vill att den ska kopiera filerna då ska man byta "shutil.move" till "shutil.copy2"
# Och ta bort koden som ta bort mapparna.


import os               # Importerar funktioner för att interagera med operativsystemet
import shutil           # Importerar funktioner för att hantera filer och mappar
import win32api         # Importerar funktioner för att interagera med Windows-API:et
import time             # Importerar funktioner för att hantera tid och vänta mellan operationer
import subprocess

#Funktionen move_files flyttar filer från en källmapp till en målmapp baserat på filtyp.

'''
Denna funktion flyttar filer från en källmapp till en destinationmapp baserat på filtypen.
Funktionen räknar först antalet filer i källmappen, skapar sedan en destinationmapp och flyttar sedan filerna enligt deras filtyp.
För varje fil kontrolleras filtypen genom att kontrollera filändelsen med hjälp av os.path.splitext().
Om filen har en ".rar" eller ".zip" filändelse, flyttas den till en annan mapp med hjälp av en separat funktion move_zip_file().
Om filen har ".mf4" eller ".dat" filändelse flyttas den till en annan mapp med hjälp av move_mf4_file().
```

```
Alla andra filer flyttas till en annan mapp med hjälp av move_else_file().
För varje flyttad fil skrivs en beskrivning av filen till konsolen, t.ex. "Flyttade filen 1/10: filnamn.txt".
När alla filer har flyttats, tas tomma mappar i källmappen bort och ett meddelande skrivs ut till konsolen för att informera användaren om att flyttningen är klar och enheten kan tas bort.
Detta är en användbar funktion för att organisera och flytta filer från en mapp till en annan baserat på filtyp.
Funktionen kan användas för att flytta filer från en enhet till en annan eller för att organisera filer på en dator.
'''
def move_files(source_folder, dest_parent_folder):
    total_files = 0
    num_files = 0

    for root, dirs, files in os.walk(source_folder):
        total_files += len(files)

    dest_folder = os.path.join(dest_parent_folder)
    os.makedirs(dest_folder, exist_ok=True)

    for root, dirs, files in os.walk(source_folder):
        for item in files:
            source_item = os.path.join(root, item)
            extension = os.path.splitext(item)[1].lower()

            if extension in ('.rar', '.zip'):
                move_zip_file(source_item)

            elif extension in ('.mf4', '.dat'):
                move_mf4_file(source_item)

            else:
                move_else_file(source_item)

            num_files += 1
            if num_files != total_files:
                print(f"{num_files}/{total_files} Ta inte bort {volume_info[0]}, Flyttade filen : {item[:20]}")

    remove_empty_folders(source_folder)

    if num_files == total_files:
        print(f"\nAlla filer har nu flyttats från {volume_info[0]} till destinationmappen. \nDu kan nu tryggt ta bort {volume_info[0]}  :)")


def move_zip_file(source_item):
    """
    Funktionen move_zip_file flyttar en zip-fil från källmappen till en zip-mapp i målmappen.
    zip_folder är sökvägen till zip-mappen där filen ska flyttas till.
    os.makedirs används för att skapa zip-mappen om den inte redan finns.
    dest_item är sökvägen för den nya filen i zip-mappen.
    shutil.move används för att flytta filen från källmappen till zip-mappen.
    Ingen utskrift görs i denna funktion.
    """
    folder_name_zip = os.path.splitext(os.path.basename(volume_info[0]))[0].split('_')[0]
    zip_folder = os.path.join(r"C:\Users\fakhe\Desktop\testing\zip", folder_name_zip , 'zzip')
    os.makedirs(zip_folder, exist_ok=True)
```

```
    dest_item = os.path.join(zip_folder, os.path.basename(source_item))
    shutil.move(source_item, dest_item)


def move_mf4_file(source_item):
    """
    samma som förra men till mf4 och dat filerna
    """
    folder_name_mf4 = os.path.splitext(os.path.basename(volume_info[0]))[0].split('_')[0]

    mf4_folder = os.path.join(r"C:\Users\fakhe\Desktop\testing\data", folder_name_mf4 , 'data')
    os.makedirs(mf4_folder, exist_ok=True)
    dest_item = os.path.join(mf4_folder, os.path.basename(source_item))
    shutil.move(source_item, dest_item)


def move_else_file(source_item):
    """
    samma som förra men till alla filer som inte är mf4, dat, zip eller rar
    """
    else_folder = os.path.join(r"C:\Users\fakhe\Desktop\testing\else", volume_info[0] , 'else')
    os.makedirs(else_folder, exist_ok=True)
    dest_item = os.path.join(else_folder, os.path.basename(source_item))
    shutil.move(source_item, dest_item)

def remove_empty_folders(folder):
    """
    Tar bort tomma undermappar i en mapp.
    """
    if not os.path.exists(folder) or not os.path.isdir(folder):
        return

    for root, dirs, files in os.walk(folder, topdown=False):
        for dir_name in dirs:
            full_dir_path = os.path.join(root, dir_name)
            try:
                os.rmdir(full_dir_path)
            except OSError:
                pass

    try:
        os.rmdir(folder)
        print(f"Tagit Bort mappen: {folder}")
    except OSError:
        pass



if __name__ == '__main__':
    # connected_drives är en mängd som innehåller enhetsbokstäverna för alla anslutna enheter.
    connected_drives = set()
    #En oändlig loop börjar som kontinuerligt söker efter nya anslutna enheter och borttagna enheter.

    while True:
```

```
        # new_drives är en lista som innehåller enhetsbokstäverna för nya anslutna enheter som inte redan
finns i connected_drives.
        new_drives = [d for d in 'ABDEFGHIJKLMNOPQRSTUVXYZ' if d not in connected_drives and
win32api.GetLogicalDrives() & (1 << ord(d) - 65)]
        # Om det finns nya enheter, läggs de till i mängden "connected_drives"
        if new_drives:
            connected_drives.update(new_drives)
        # disconnected_drives är en lista som innehåller enhetsbokstäverna för borttagna enheter från
mängden connected_drives.
        disconnected_drives = connected_drives - {d for d in connected_drives if win32api.GetLogicalDrives()
& (1 << ord(d) - 65)}
        # Om det finns borttagna enheter, skrivs ett meddelande ut med enheternas bokstäver och de tas bort
från mängden "connected_drives"
        if disconnected_drives:
            print("Usb(s) frånkopplad:", ", ".join(disconnected_drives))
            connected_drives -= disconnected_drives

        time.sleep(1)
        # Loopar igenom alla nya enheter
        for new_drive in new_drives:
            try:
                # Hämtar information om enheten och skriver ut dess namn
                volume_info = win32api.GetVolumeInformation(new_drive + ':\\')
                print(f"Hittade enhet: {volume_info[0]}")
                if '_VPT_PLOPP' in volume_info[0] or '_ATA' in volume_info[0]:
                    # Sätter sökvägen till enheten och listar alla filer i sökvägen
                    source_folder = new_drive + ':\\'
                    items = os.listdir(source_folder)

                    if items:
                        #dest_parent_folder =
r'\\gbw9061109.got.volvocars.net\PROJ2\9413-SHR-VCC127500\MEP2\Hällered'
                        dest_parent_folder = volume_info[0]
                        drive_folder = os.path.join(dest_parent_folder)
        #                 move_files(drive_folder, source_folder)
                        move_files(source_folder, drive_folder)
                    else:
                        # Om enheten inte innehåller "_Data" i volymnamnet, skrivs ett meddelande ut om det.
                        print("Inga filer att Flytta på enheten: " + new_drive)
                else:
                    print(f"{volume_info[0]} innehåller inte '_VPT_PLOPP '.")

            except Exception as e:
                print()
```