

Automatisierungslösung für interne Projektmanagement-Software

Martin Klatt

Fachinformatiker Anwendungsentwicklung

Identnummer: 865044
Prüflingsnummer: 95014

BFW Eckert gGmbH

Herr Manfred Petsch
manfred.petsch@eckert-schulen.de
09402 502374

Inhaltsverzeichnis:

1	Einleitung	3
1.1	Projektumfeld	3
1.2	Umfelddetails	3
2	Initialisierung	3
3	Projektdefinition	4
3.1	Auftragsbeschreibung	4
3.2	Ist-Analyse	4
3.3	Erstellen eines Soll-Konzepts	5
3.4	Durchführungszeitraum	5
4	Planungsphase	5
4.1	Einzusetzende Ressourcen	5
4.2	Vergleich verfügbarer Software	6
4.3	Auswahl Framework	7
4.4	Konzeptionierung der Automation	7
5	Durchführungsphase	8
5.1	Realisierung des Algorithmus	8
5.2	White-Box-Test: Prototyp	11
5.3	Implementierung in das interne Netzwerk	11
5.4	Black-Box-Test	11
6	Abschluss	12
6.1	Übergabe des Projekts und Einweisung der Key User	12
6.2	Erstellen der Dokumentation und des Benutzerhandbuches	12
7	Fazit	12
7.1	Vergleich der Soll- und Ist-Zeit	12
7.2	Wirtschaftliche Nachbetrachtung	13
7.3	Zusammenfassung	14
8	Anhang	14
8.1	Quellcode	
8.2	Benutzerhandbuch	
8.3	Protokoll der durchgeführten Projektarbeit	

1 Einleitung

1.1 Projektumfeld

Das Umfeld besteht aus der Projektmanagementabteilung, die ein Teilbereich der IT-TA (TA = Technische Abteilung) des Unternehmens Max Bögl ist. Der Standort befindet sich in Sengenthal. Der Auftraggeber ist der Leiter der technischen Abteilung Dr. Christoph Pflug. Ich bin im 4. Semester der außerbetrieblichen Umschulung zum Fachinformatiker für Anwendungsentwicklung an den Eckert-Schulen in Regensburg und seit 18.08.2020 als Praktikant bei Max Bögl tätig. Der Geschäftsbereich IT-TA-Projektmanagement ist verantwortlich für die Implementierung, Pflege und Verbesserung der Projektmanagement-Software für Bauprojekte.

1.2 Umfelddetails

In der Abteilung IT-TA kommen unterschiedliche CAD-Systeme (Modellierungs-Software, „computer aided design“) zum Einsatz, die rechnerunterstütztes Konstruieren ermöglichen. Diese sind „Autodesk“, „Revit“, „Tekla“, „NX“ (Siemens), „Sketchup“ und noch ein paar kleinere Programme. Damit wird in den verschiedenen Bereichen (Stahlbau, Hochbau, Infrastruktur, Fertigteilewerke etc.) in 3D geplant.

Zu den Projektmanagement-Systemen gehört u.a. „iTWO“. Damit entsteht aus der Planung in 3D durch Hinzufügen der Faktoren Zeit und Kosten, sogenannte 5D-Modelle.

„Desite“ ist die Modellvisualisierungskomponente. Damit kann der Fortschritt des Projekts betrachtet werden. Mit Hilfe von Rückmeldungen wird der Status von Bauabschnitten oder Fertigteilen aus dem Werk aktualisiert.

Projektplattformen sind der zentrale Ort für alle Projektbeteiligten (intern und extern), in dem Informationen bereitgestellt und verwaltet werden. Dazu zählen Planverwaltung, Mängelmanagement, Qualitätssicherung, Bautagebuch oder Protokollierung.

„Microsoft Teamcenter“ ist eine „PLM-Lösung“ (Product Lifecycle Management) für einige Arbeitsbereiche wie Produktentwicklung, Planung oder Fertigung.

„BIM“ steht für „Building Information Modeling“ und wird mit „Bauwerksdatenmodellierung“ übersetzt. Es beschreibt eine Methode der vernetzten Planung, Ausführung und Bewirtschaftung von Gebäuden und anderen Bauwerken mithilfe von Computerprogrammen.

Das Programm „BIM-Publisher“ kümmert sich hierbei um die Versionsaktualisierungen der Projekte.

2 Initialisierung

Im Rahmen des Betriebspraktikums übergab mir der Abteilungsleiter Dr. Christoph Pflug in einem Meeting die Aufgabe, eine Projektlösung zur Automatisierung von Dateisystemen in der betriebsinternen Domäne durchzuführen. Die Bürokollegen Herr Philipp H. und Herr Wladimir D. waren ebenfalls anwesend.

3 Projektdefinition

3.1 Auftragsbeschreibung

Der Bedarf entstand durch ein offenes Ticket im internen Support-System. Der „BIM-Publisher“ veröffentlicht im Intranet täglich die aktuellsten Teklamodellversionen, die durch Multiuserbetrieb bearbeitet wurden. Vor dem Abruf muss jedoch die Dateistruktur innerhalb aller Projektordner stimmen. Da eine Veränderung des Sourcecodes des Programms nicht möglich ist und noch keine Automatisierung existiert, muss dies derzeit jeden Tag per Hand manuell für alle Projekte angepasst werden.

Ziel des internen Ein-Personen-Projektes, das in Teilzeit bearbeitet wird, ist die Ausarbeitung bzw. Durchführung einer geeigneten Lösung des Problems. Hierbei soll unter der Beachtung technischer und wirtschaftlicher Gesichtspunkte entschieden werden, welche Lösung für die Firma Bögl die passendste ist.

Dem Auftraggeber ist eine lange Lebenszeit des Programms wichtig. Ein Key-User (hauptsächliche Nutzer des Programms) äußerte den Wunsch, die die Software über das Microsoft Office-Programm „Excel“ einstellen zu können.

3.2 Ist-Analyse

Aktuell stehen dem Arbeitskreis drei Automatisierungsserver zur Verfügung. Hierbei wird die Remotezugriffsfunktion und die Aufgabenplanung der Windows-10-Enterprise-Rechner, sowie verschiedenste Batchskripte („bat“ ist ein Dateiformat zur einfachen Konfiguration von DOS-Kommandozeilenbefehlen) genutzt.

Mir steht ein Firmenlaptop von „Dell“ frei zur Benutzung. Dieser besitzt zwar keine SSD-Festplatte, bietet jedoch durch schnelle Prozessorkerne und viel Arbeitsspeicher ausreichend Bedienkomfort. Die Entscheidung über ein zuverlässiges Betriebssystem fiel auf Windows 10 Enterprise.

Da dieses Notebook nur einen 15 Zoll-Bildschirm besitzt, konnten im Laufe des Projekts noch zwei weitere Monitore per Dell-Dockingstation angeschlossen werden.

Headset, Tastatur, Computermouse und Mauspad stelle ich zur Verfügung und werden deshalb nicht in die Kalkulation miteinberechnet.

Für anfordernde Software kann das Intranetkiosk von „Baramundi“ zur schnellen Installation von Open-Source-Software verwendet werden. Aus Sicherheitsgründen können nur Mitarbeiter mit Administrationsrechten ausführbare Programmdateien de- und installieren. Mobile Applikationen, die keine Installation benötigen, fallen nicht unter diese Einschränkung. Wenn Programmlizenzen eingekauft werden sollen, kann das zuständige IT-Team kontaktiert werden.

Durch einen schnellen Internetanschluss und diverse vorhandene IT-Literatur kann bei Bedarf recherchiert werden.

3.3 Erstellen eines Soll-Konzept

Zur Lösung des beschriebenen Ausgangsproblems gibt es verschiedenste Lösungsansätze. Ziele, die vom Auftraggeber genannt wurden, sind die Beseitigung des Mangels des händischen Kopiervorgangs. Dieser verbraucht je nach Anzahl der Projekte eine tägliche Arbeitszeit von bis zu einer Stunde. Durch die Pflege der Excelliste muss jedes Projekt nur einmal eingetragen werden und wird so in die tägliche Automationsroutine aufgenommen.

Da sich die Struktur und Anzahl der Projektordner täglich ändern kann, muss die Algorithmierung dynamisch erfolgen. Ein z.B. fest eingetragener Dateiname im Quellcode ist ausgeschlossen.

3.4 Durchführungszeitraum

Als Bearbeitungszeitraum des Projekts ist der Zeitraum vom 03.11.2020 bis 11.01.2021 festgelegt. Ich bearbeite das Projekt in Teilzeit.

4 Planungsphase

4.1 Einzusetzende Ressourcen

Darunter fallen Zeit, Soft- und Hardware und die damit verbundenen Kosten. Das Projekt ist auf maximal 70 Stunden begrenzt. Die Erstellung der Dokumentation darf höchstens sieben Stunden erfordern.

Im Projektantrag wurde folgender grober Zeitaufwand kalkuliert:

Geplante Zeitdauer in Stunden	Tätigkeit
1	Detailanforderungen klären
1	Projektbudget kalkulieren
2,5	Zeitplanung und Festlegung von Meilensteinen
1	Vergleich von Frameworks und IDE's
0,5	Software hausintern anfordern
8	Konzeptionierung der Automation
8	Realisierung des Algorithmus
8	Durchführung: White-Box-Test des Prototyps
4	Anpassung des Codes durch neu erworbene Kenntnisse

8	Implementierung in das interne Netzwerk
1	Testteilnehmer organisieren
8	Durchführung: Black-Box-Test
4	Anpassung des Codes durch neu erworbene Kenntnisse
1	Benutzerhandbuch anfertigen
1	Einweisung der Mitarbeiter (Key-User)
5	Betreuung und Kontrolle des Systems im Testbetrieb über 2 Wochen
7	Projektdokumentation anlegen
2	Kostenaufstellung und Fazit erstellen

Gesamt: 70 Stunden

In der wirtschaftlichen Nachbetrachtung wird auf die Kosten und die tatsächlich verwendete Zeit eingegangen.

4.2 Vergleich verfügbarer Software

Da auf den Automationsservern Windows läuft, ist es naheliegend, Software zu kaufen oder zu erstellen, die auf dieses Betriebssystem ausgelegt sind. Nach kurzer Suche konnten einige Open Source Programme getestet werden. Leider besaßen jedoch z.B. „RealTimeSync“, „RoboMirror“ oder „Copy Handler“ nicht genügend Konfigurationsmöglichkeiten, um das Problem mit der sich veränderlichen Dateistruktur zu lösen.

Entwicklungsumgebungen (IDE = Integrated Development Environment) für Windows sind z.B. „Visual Studio 2017/19“, „Visual Studio Code“, „Power Shell IDE“, „NotePad++“ oder „SharpDevelop“. Da die Powershell IDE jedoch eine zu umfangreiche Einarbeitungszeit benötigt, wird sie hier nicht berücksichtigt. Es folgt eine Nutzwertanalyse um eine Entscheidung der zu vergleichenden Software zu erleichtern und nachvollziehbar zu gestalten.

Software		Visual Studio 17/19		Visual Studio Code		NotePad++		SharpDevelop	
Kriterien	Gewichtung	Erfüllung	Nutzwert	Erfüllung	Nutzwert	Erfüllung	Nutzwert	Erfüllung	Nutzwert
Preis	10 %	1	0,1	10	1	10	1	10	1
Umfang	20 %	10	2	8	1,6	5	1	4	0,8
Zuverlässigkeit	20 %	8	1,6	9	1,8	8	1,6	7	1,4
Kompatibilität	30 %	9	2,7	8	2,4	7	2,1	6	1,8
Bedienbarkeit	20 %	10	2	7	1,4	7	1,4	6	1,2
Summen	100 %		8,4		8,2		7,1		6,2

Dem Preis wird nur eine geringe Gewichtung zugeteilt, da Lizenzen für Visual Studio Pro 2017 ausreichend in der Abteilung vorhanden sind. Visual Studio Community darf in größeren Firmen nicht eingesetzt werden.

Visual Studio Code wäre eine gute Alternative, die anderen zwei Open Source IDE's jedoch eher weniger. NotePad++ wäre eine Allroundlösung, besitzt aber erst nach umständlicher manueller Erweiterung dieselben Debugger-Funktionen (Fehlererkennung im Code) wie VS (Visual Studio). SharpDevelop wird leider seit dem Jahr 2016 nicht mehr gepflegt und besitzt auch kein IntelliSense (Tool zur Automatischen Codevervollständigung), ähnlich wie Notepad++.

Nach der Nutzwertanalyse entscheide ich mich für Visual Studio 2017 Pro (kaum Unterschiede zur 2019-Version). Diese IDE ist durch den VS Installer und die NuGet-Anbindung (Integriertes Tool für Erweiterungen) eine bequeme und vielfältig nutzbare Entwicklungsumgebung. Benötigte Frameworks (Programmiergerüste) können rasch ausgewählt und heruntergeladen werden.

4.3 Auswahl Framework

Bevor die eigentliche Programmierung beginnt, ist es wichtig, dass man sich auf eine Programmiersprache und ein Framework festlegen. In der Ausbildung an der Eckertschule habe ich die objektorientierte Programmierung mit der Sprache „C#“ vertieft gelernt. Sie ist vergleichbar mit z.B. „Java“ oder „Python“ und sehr verbreitet.

Für Windows hat man im „.NET-Framework“ die Auswahl zwischen einer Konsolenanwendung und einem Programm mit grafischer Oberfläche. Wenn wir die auszuführende Datei mit der Windows Aufgabenplanung automatisieren wollen, ist ein „Console-Project“ völlig ausreichend.

4.4 Konzeptionierung der Automation

Um mit C# auf ein Exceldokument zuzugreifen, wird ein zusätzliches Tool (Erweiterung, Werkzeug) benötigt. Microsoft stellt hierfür die API (Schnittstelle) „Interop“ zur Verfügung.

Es gibt auch noch Alternativen wie OleDb oder freie Software von GitHub wie „Spire“, die ähnliche Funktionen bieten. Weil für OleDb sehr viel Informationsmaterial zur Verfügung steht und es als schneller als Interop gilt, entscheide ich mich für zunächst dafür.

Mit Interop oder OleDb kann man die Exceldatei einlesen und die Daten weiterverarbeiten. Excelarbeitsmappen bestehen nur aus XML-Dateien. Man könnte die Datei zur Verwaltung auch als einfache XML-oder JSON-Datei pflegen. Der Kundenwunsch lag jedoch bei Excel und seiner bequemen Bedienbarkeit. Zum Einlesen der Dateinamen kann man mit Schleifen die Tabelle durchiterieren und anschließend z.B. in Arrays oder Listen zwischenspeichern.

Wenn man die Daten eingelesen hat, muss man den Kopiervorgang in Windows ausführen lassen. Da ich in diesem Projekt kein Powershell nutze, kann ich als Alternative die Kommandozeile verwenden. Hierbei greifen wir zurück auf „Robocopy“. Das ist ein sehr mächtiges Windowstool zum Erstellen von Backups oder Überprüfen von Dateien. In

älteren DOS-Versionen hieß dies noch „XCopy“. Die durch VS Studio erstellte ausführbare Datei kann zum Schluss mit der Windowsaufgabenplanung nach Belieben automatisiert werden.

5 Durchführungphase

5.1 Realisierung des Algorithmus

Der vollständige Quellcode ist im Anhang zu finden. Es wird auf ein paar interessante Stellen im Algorithmus eingegangen.

```
using System;
using System.Collections.Generic;
using System.IO;
using System.Diagnostics;
using System.Runtime.InteropServices;
using Excel = Microsoft.Office.Interop.Excel;
```

Es werden hier keine externen Bibliotheken wie „Spire“ verwendet. Die hier benutzten sind alle von Microsoft selbst.

```
static void Main(string[] args)
{
    ReadExcel();
    ReadLog();
}
```

In der Hauptfunktion werden nur die zwei ausgelagerten Methoden verwendet. Wenn sich streng an die Standards der objektorientierten Programmierung gehalten werden würde, müsste man noch extra Klassen erstellen, wofür jedoch keine Zeit übrig blieb, da dies den Rahmen von ca. 70 Arbeitsstunden sprengt.

Mit Hilfe von Try-und Catch-Blöcken kann auf auftretende Fehlermeldungen reagiert werden:

```
try
{
    //Pfad neben der .exe ermitteln für die Excel-Datei
    string exeFile = System.Reflection.Assembly.GetExecutingAssembly().Location;
    string exePath = Path.GetDirectoryName(exeFile);
    string xlFilePath = Path.Combine(exePath, "WHITE_BOX_TEST.xlsx");
}
```

Hier sieht man den Dateinamen der Excelliste. Auf ihren Aufbau wird im Benutzerhandbuch eingegangen. Die Kommentierung des Programms ist mir persönlich wichtig. Es bietet eine Hilfe für Personen, die mit fremden Code arbeiten.

Die auszuführende Datei muss immer neben der .xlsx-Datei liegen. Der obige Codeabschnitt kann sie somit finden und ist nicht „hart“ festgelegt.

```
//Durch die Tabelle iterieren und Werte ausgeben.
Excel startet wegen Überschriften bei 1/2, nicht 0/0
for (int i = 2; i <= rowCount; i++)
{
```

Wie aus der Kommentierung hervorgeht, haben Arrays und Exceltabellen eine unterschiedliche Indexierung. Da wir nur ein Tabellenblatt benutzen, reicht eine For-Schleife aus.


```

var modelpfadVar = (xlRange.Cells[j, 2] as Excel.Range).Value;
string modelpfad = Convert.ToString(modelpfadVar);
list.Add(modelpfad);
string sourceLoc = modelpfad;

```

Die Modelpfade werden als String in einer Liste zwischengespeichert.

Es wird sich um lesbare Variablennamen bemüht. Auf Anfrage bekam ich erklärt, dass das Betriebsteam hierauf nicht allzu große Relevanz legt.

```

        Copy(sourceLoc, destLoc, list2);
        list.Clear();
        list2.Clear();
    }
    Muellabfuhr(xlApp, xlWorkbook, xlWorksheet, xlRange);
}
catch (IOException ex)
{
    Console.WriteLine(ex.Message);
    Console.ReadLine();
}

```

Das Freigeben von Speicherplatz und das Leeren der Liste ist wichtig, da für jede Datei ein Kopiervorgang stattfindet und ohne das „Clear“ es immer wieder dieselbe Datei kopieren würde.

```

}
private static void Copy(string sourceLoc, string destLoc, List<string> list2)
{
    //Pro Datei 1x Kopiervorgang durchführen
    list2.ForEach(delegate (string datei)
    {

```

Es kommt eine „ForEach“-Schleife zum Einsatz. Dafür werden hier Delegate eingesetzt. Außerdem erhält die „Copy“-Methode die Parameter aus der vorherigen Funktion.

```

        try
        {
            Process p = new Process();
            p.StartInfo.Arguments = string.Format("/C Robocopy /Z /LOG+:log_cj.txt {0} {1} {2} ",
sourceLoc, destLoc, datei);

```

Hier sieht man den Robocopy-Befehl. Der Switch „/Z“ sorgt dafür, dass es den Kopiervorgang bei Netzwerkabbruch unterbricht und „sicher“ zu Ende kopiert, wenn das Netzwerk wieder vorhanden ist. Der „/LOG“-Switch sorgt für das Speichern einer Logdatei namens „log_cj“.

Die Parameter 1, 2 und 3 in geschweiften Klammern stellen die zu kopierende Datei, das Quellverzeichnis sowie das Zielverzeichnis dar.

```

            p.StartInfo.FileName = "CMD.EXE";
            p.StartInfo.CreateNoWindow = true;
            p.StartInfo.UseShellExecute = false;
            p.Start();
            p.WaitForExit();
            //p.Dispose();
            //p.Kill();
            //GC.Collect();
            //GC.WaitForPendingFinalizers();

```

Nun werden die gewünschten Prozesse ausgeführt und anschließend der Speicher wieder freigegeben. Auskommentierte Programmzeilen waren bis zuletzt kritisch und wurden vor allem beim Testen ein- oder wieder ausgefügt.

Im Falle von unerwarteten Fehlern werden diese Meldungen für ein einfacheres Debugging in sog. „Ausnahmen“ abgefangen und aufgerufen.

Visual Studio Pro 2017 mit seinen eigenen Debuggingfunktionen erleichtert das Verfolgen des Programmablaufs durch das explizite Beobachten von z.B. Variablenwerten. Haltepunkte und Einstiegszeilen können festgesetzt werden sowie die Programmgeschwindigkeit überprüft werden. Da jedoch das gesamte Programm mit den vorhandenen Maschinen keine große Performancehürden darstellt, wird darauf nur zweitrangig geachtet.

```

        catch (Exception ex)
        {
            Console.WriteLine(ex);
            Console.ReadLine();
        }
    }
}

```

Diese „Müllabfuhr“-Funktion kann leicht abgeändert in zukünftigen C#-Programmen verwendet werden und sorgt dafür, dass Objekte wie Excelarbeitsblätter richtig geschlossen werden.

```

private static void Muellabfuhr(Excel.Application xlApp, Excel.Workbook xlWorkbook,
Excel.Worksheet xlWorksheet, Excel.Range xlRange)
{
    GC.Collect();
    GC.WaitForPendingFinalizers();
    Marshal.ReleaseComObject(xlRange);
    Marshal.ReleaseComObject(xlWorksheet);
    xlWorkbook.Close(Type.Missing, Type.Missing, Type.Missing);
    Marshal.ReleaseComObject(xlWorkbook);
    xlApp.Quit();
    Marshal.ReleaseComObject(xlApp);
    xlApp = null;
    GC.Collect();
    GC.WaitForPendingFinalizers();
}

```

Zum Schluss wird noch die erstellte Logdatei geöffnet. Dabei wird pro Kopiervorgang das Ergebnis mit der Dauer an das vorherige Kopierergebnis angehängt. Die Kommandozeile „cmd“ kann bei Bedarf aus-oder eingeblendet werden. Da je nach Projektdichte und verfügbaren Computerressourcen dieser Vorgang recht schnell geschieht, sieht man meistens nur ein kurzen „Aufblitzen“ der Commandshell.

```

private static void ReadLog()
{
    try
    {
        string exeFile = System.Reflection.Assembly.GetExecutingAssembly().Location;
        string exePath = Path.GetDirectoryName(exeFile);
        string logFilePath = Path.Combine(exePath, "log_cj.txt");
        Process pp = new Process();
        pp.EnableRaisingEvents = false;
        pp.StartInfo.FileName = logFilePath;
        pp.Start();
        pp.Dispose();
        //File.Open(txtFilePath)
        //string log = File.ReadAllText("log_cj.txt");
        //Console.WriteLine(log);
        //Console.WriteLine("Press any key to exit.");
        //Console.ReadKey();
        //File.WriteAllText("file.txt", String.Empty);
        GC.Collect();
        GC.WaitForPendingFinalizers();
    }
    catch (Exception ex)
    {
        Console.WriteLine(ex);
        Console.ReadLine();
    }
}

```

5.2 White-Box-Test: Prototyp

Nachdem im Algorithmus Meilensteine, wie das Einlesen von Exceldateien erreicht wurden, wird am eigenen lokalen Rechner mit Codeeinsicht getestet und Bugs behoben. Neben unabsichtlich eingebauten Rekursionen, die zum Absturz des Programms führten, kam es auch zu Fehlern, die durch Rechtschreibfehler im Code erzeugt wurden, und erst nach längerer Suche entdeckt wurden, da Logikfehler vermutet wurden.

Als Test wurde ein Ordner auf dem lokalen C-Laufwerk erstellt. Dieser beinhaltet die erstellte .exe-Datei „Sandbox“, die .xlsx-Datei „WHITE_BOX_TEST“ und einen weiteren Ordner „Test“. Der Aufbau der Exceldatei wird im Benutzerhandbuch erläutert.

Der Unterordner „Test“ erhält weitere Ordner: „Quellordner“ und „Zielordner“. Im erst genannten Verzeichnis befindet sich die .txt-Datei „Test1“. Der Zielordner ist leer.

Da in der Excel-Liste die absoluten Pfade eingetragen sind, kopiert es die Test1-Datei nach einem Doppelklick auf die ausführbare Datei in den Ordner „Zielordner“.

Erst mit dem Hinzufügen mehrerer Testdateien oder dem Verändern von Gegebenheiten fielen Bugs auf, die behoben werden mussten. Als Beispiel wurde z.B. nur „Test1“ in alle Verzeichnisse kopiert, aber nicht „Test2“ oder „Test3“. Dank des Debuggers konnte in diesem Fall der Fehler der nicht bereinigten generischen Liste (nach bestimmten Schleifendurchgängen) gefixt werden.

5.3 Implementierung in das interne Netzwerk

Beim Kopieren der Routine auf den Rechner „Automax01“ fiel auf, dass im Netzwerk keine Laufwerksbuchstaben verwendet werden können. Es wurde direkt mit „//filer01“ adressiert.

Microsoft Excel musste auf dem Zielrechner nachinstalliert werden. Außerdem funktionierte die Automatisierung im Windowstaskplaner erst, als man den korrekten Startpfad eingetragen hatte. Ansonsten würde das Programm zwar zur ausgewählten Uhrzeit starten, durch den default Pfadwert jedoch abstürzen.

5.4 Black-Box-Test

Als Hilfestellung im letzten Test wurde auf dem Automax01 eine README.txt-Datei erstellt, da noch kein Benutzerhandbuch existierte. In der Textdatei ist beschrieben, dass die Excelliste neben der .exe-Datei liegen muss, im Namen nicht geändert werden darf oder in der Aufgabenplanung der Startpfad eingetragen werden muss.

Da die vorherigen Tests viele Fehler aufgezeigt hatten und behoben wurden, verlief der letzte Black-Box-Test ohne große Zwischenfälle. Zwar musste der Test wegen der Corona-Pandemie vollständig digital stattfinden, dies war aber durch ausreichend Remote- und Chat-Tools kein Problem.

Behoben wurden u.a. das Problem der Dateinamen mit Leerzeichen im Namen.

Cisco „Jabber“ bietet praktische Videotelefonie- und Bildschirmübertragungsfunktionen. Ich befand mich in Schwandorf im mobilen Arbeiten, mein Testpartner Herr Engl vor Ort im Büro in Sengenthal.

6 Abschluss

6.1 Übergabe des Projekts und Einweisung der Key-User

Da der wichtigste Key-User Herr Engl schon in der Testphase beteiligt war, fiel die Einweisung und Übergabe recht kurz und unkompliziert aus. Die Projektdokumentation und das Benutzerhandbuch erhält er mit Abgabe der Dokumentation an die IHK.

Der geplante Punkt „Betreuung und Kontrolle im Testbetrieb über zwei Wochen“ fiel recht kurz aus, da es kaum Rückfragen gab.

6.2 Erstellen der Projektdokumentation und des Benutzerhandbuchs

Zum Abschluss des Projekts wurde von mir diese Dokumentation, das Benutzerhandbuch und eine kleine Powerpointpräsentation für die mündliche IHK-Prüfung erstellt.

7 Fazit

7.1 Vergleich der Soll- und Ist-Zeit

Geplante Zeitdauer in Stunden	Tatsächlich eingesetzte Zeit	Tätigkeit
1	2	Detailanforderungen klären
1	0	Projektbudget kalkulieren
2,5	2,5	Zeitplanung und Festlegung von Meilensteinen
1	2	Vergleich von Frameworks und IDE's
0,5	0,5	Software hausintern anfordern
8	8	Konzeptionierung der Automation
8	19,5	Realisierung des Algorithmus
8	4	Durchführung: White-Box-Test des Prototyps

4	5	Anpassung des Codes durch neu erworbene Kenntnisse
8	6	Implementierung in das interne Netzwerk
1	0,5	Testteilnehmer organisieren
8	4	Durchführung: Black-Box-Test
4	5	Anpassung des Codes durch neu erworbene Kenntnisse
1	1	Benutzerhandbuch anfertigen
1	1	Einweisung der Mitarbeiter (Key-User)
5	2	Betreuung und Kontrolle des Systems im Testbetrieb über 2 Wochen
7	7	Projektdokumentation anlegen
2	2	Kostenaufstellung und Fazit erstellen

Gesamt: 70 Stunden

Nach dem Vergleich fallen einige Dinge auf. Vor allem für die Realisierung des Algorithmus musste mehr Zeit investiert werden. Dies kam dadurch zu Stande, dass man sich mitten im Codierprozess gegen OleDB entschieden hat, da es zu Problemen bei der Datenmanipulation kam.

OleDB nutzt den SQL-Befehl „INCLUDE ALL“ für den Import der Excel-Daten. Da kein zufriedenstellender Workaround für die Iterierung gefunden wurde, wechselte man auf die Interop-Bibliothek. Die nötige Zeit konnte man am Ende der Durchführungsphase mit einer rascheren Test-Phase wieder einholen. Die geplante Projektbudgetkalkulierung entfiel, da die verwendeten Ressourcen alle in den kalkulierten Lohnkosten enthalten sind. Die tatsächliche Gesamtzeit betrug wie geplant ca. 70 Stunden.

7.2 Wirtschaftliche Nachbetrachtung

Nun überprüfe ich die Wirtschaftlichkeit des Projekts. Dazu muss ich alle verursachten Kosten auflisten. Zur Vereinfachung gehen wir von einem Praktikantenstundenlohn von 15,00 € (anstatt 0,00 €) aus, der die aufgezählten Gemeinkosten abdeckt.

70 Stunden = ca. 9 Mannstage à 8 Stunden: $70 * 15,00 \text{ €} = 1050,00 \text{ €}$

Miete, Strom, Wasser, Heizung: In den Lohnkosten enthalten

Lizenzkosten für Windows und Visual Studio Pro: In den Lohnkosten enthalten

Leasingkosten Hardware: In den Lohnkosten enthalten

Gesamtkosten: 1050,00 €

Ersparnis durch das durchgeführte Projekt:

Durch das von mir erstellte Programm können täglich ca. 30-45 Minuten Kopierarbeit eingespart werden. Vereinfacht wird von einem Facharbeiterstundenlohn von 30,00 € ausgegangen. Dadurch ergibt sich folgende Rechnung:

$0,5h * 30,00 € * 20 \text{ Tage}$

-> **300,00 € Ersparnis** an Arbeitszeit im Monat bzw. übrig für andere IT-Arbeit.

=> Nach spätestens **3 Monaten** Lebenszeit des Programms kann man von einer Art der Amortisierung sprechen.

7.3 Zusammenfassung

Zusammenfassend kann ich sagen, dass dieses Projekt erfolgreich umgesetzt werden konnte und die Projektziele damit erreicht wurden.

8 Anhang

8.1 Quellcode

```
using System;
using System.Collections.Generic;
using System.IO;
using System.Diagnostics;
using System.Runtime.InteropServices;
using Excel = Microsoft.Office.Interop.Excel;

namespace Sandbox
{
    class CJ
    {
        static void Main(string[] args)
        {
            ReadExcel();
            ReadLog();
            //File.WriteAllText("log_cj.txt", String.Empty); //Clear logfile nach erfolgreichem
                                                                Kopiervorgang
            //Console.ReadLine();
        }

        public static void ReadExcel()
        {
            try
            {
                //Pfad neben der .exe ermitteln für die Excel-Datei
                string exeFile = System.Reflection.Assembly.GetExecutingAssembly().Location;
                string exePath = Path.GetDirectoryName(exeFile);
                string xlFilePath = Path.Combine(exePath, "WHITE_BOX_TEST.xlsx");

                //COM Objekte erzeugen
            }
            catch { }
        }
    }
}
```

```

Excel.Application xlApp = new Excel.Application();
Excel.Workbook xlWorkbook = xlApp.Workbooks.Open(xlFilePath);
Excel.Worksheet xlWorksheet = xlWorkbook.Sheets[1];
Excel.Range xlRange = xlWorksheet.UsedRange;

//Länge der benutzen Zeilen (und Spalten) ermitteln
int rowCount = xlRange.Rows.Count;
//int colCount = xlRange.Columns.Count;

var list = new List<String>();

//Durch die Tabelle iterieren und Werte ausgeben. Excel startet wegen
Überschriften bei 1/2, nicht 0/0

for (int i = 2; i <= rowCount; i++)
{
    /*
    var modelNameVar = (xlRange.Cells[i, 1] as Excel.Range).Value;
    string modelName = Convert.ToString(modelNameVar);
    list.Add(modelName);
    */

    var modelpfadVar = (xlRange.Cells[i, 2] as Excel.Range).Value;
    string modelpfad = Convert.ToString(modelpfadVar);
    list.Add(modelpfad);
    string sourceLoc = modelpfad;

    var zielpfadVar = (xlRange.Cells[i, 3] as Excel.Range).Value;
    string zielpfad = Convert.ToString(zielpfadVar);
    list.Add(zielpfad);
    string destLoc = zielpfad;

    var dateienVar = (xlRange.Cells[i, 4] as Excel.Range).Value;
    string dateien = Convert.ToString(dateienVar);
    string[] datSplit = dateien.Split(',');
    List<string> list2 = new List<string>();
    list2.AddRange(datSplit);

    Copy(sourceLoc, destLoc, list2);
    list.Clear();
    list2.Clear();
}

Muellabfuhr(xlApp, xlWorkbook, xlWorksheet, xlRange);
}

catch (IOException ex)
{
    Console.WriteLine(ex.Message);
    Console.ReadLine();
}

}

private static void Copy(string sourceLoc, string destLoc, List<string> list2)
{
    //Pro Datei 1x Kopiervorgang durchführen
    list2.ForEach(delegate (string datei)
    {
        try
        {
            Process p = new Process();
            p.StartInfo.Arguments = string.Format("/C Robocopy /Z /LOG+:log_cj.txt {0} {1} {2} ", sourceLoc, destLoc, datei);

```

```

        p.StartInfo.FileName = "CMD.EXE";
        p.StartInfo.CreateNoWindow = true;
        p.StartInfo.UseShellExecute = false;
        p.Start();
        p.WaitForExit();
        //p.Dispose();
        //p.Kill();
        //GC.Collect();
        //GC.WaitForPendingFinalizers();
    }
    catch (Exception ex)
    {
        Console.WriteLine(ex);
        Console.ReadLine();
    }
}
});
}

```

```

private static void Muellabfuhr(Excel.Application xlApp, Excel.Workbook xlWorkbook,
Excel.Worksheet xlWorksheet, Excel.Range xlRange)
{
    GC.Collect();
    GC.WaitForPendingFinalizers();
    Marshal.ReleaseComObject(xlRange);
    Marshal.ReleaseComObject(xlWorksheet);
    xlWorkbook.Close(Type.Missing, Type.Missing, Type.Missing);
    Marshal.ReleaseComObject(xlWorkbook);
    xlApp.Quit();
    Marshal.ReleaseComObject(xlApp);
    xlApp = null;
    GC.Collect();
    GC.WaitForPendingFinalizers();
}

```

```

private static void ReadLog()
{
    try
    {
        string exeFile = System.Reflection.Assembly.GetExecutingAssembly().Location;
        string exePath = Path.GetDirectoryName(exeFile);
        string logFilePath = Path.Combine(exePath, "log_cj.txt");
        Process pp = new Process();
        pp.EnableRaisingEvents = false;
        pp.StartInfo.FileName = logFilePath;
        pp.Start();
        pp.Dispose();
        //File.Open(txtFilePath)
        //string log = File.ReadAllText("log_cj.txt");
        //Console.WriteLine(log);
        //Console.WriteLine("Press any key to exit.");
        //Console.ReadKey();
        //File.WriteAllText("file.txt", String.Empty);
        GC.Collect();
        GC.WaitForPendingFinalizers();
    }
    catch (Exception ex)
    {
        Console.WriteLine(ex);
        Console.ReadLine();
    }
}
}

```

} //Quellcode zu auch zu finden unter <https://github.com/yasafae/essayAutomation>

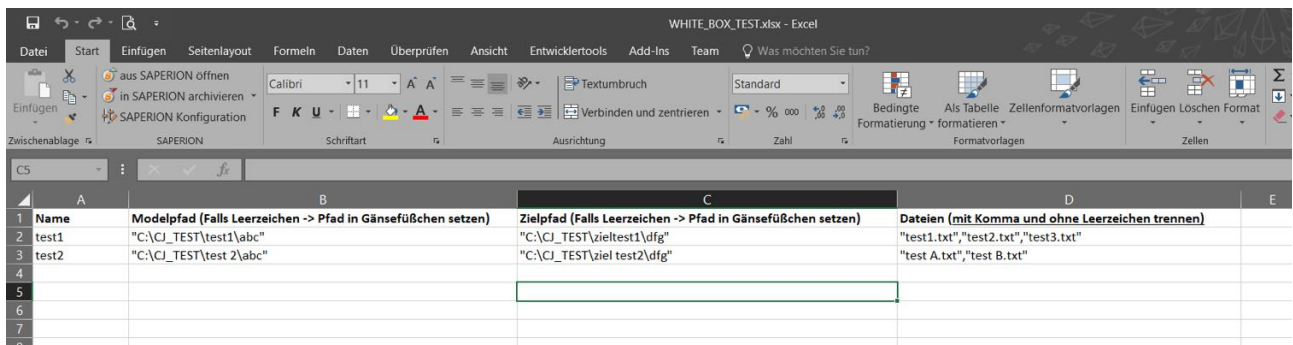
Benutzerhandbuch

Zum Bedienen des Programms für die Kopierautomatisierung gehen Sie per Remotedesktopzugriff auf den Rechner „AUTOMAX01“. Das Passwort erhalten Sie von dem Nutzer „tengl“ oder Ihrem System-Admin.

Auf dem Desktop befindet sich der Ordner „BIM_Routine“. Öffnen Sie diesen per Doppelklick.

Hierin befinden sich die .exe-Datei „BIM_Copy“, die .xlsx-Datei „Copy_Doc“ und die .txt-Datei „README“. Außerdem entsteht täglich eine weitere Textdatei namens „Copy_Log“. Im Regelfall arbeiten Sie nur mit der Exceldatei.

Pflegen Sie die gewünschten Kopiervorgänge nach vorgegebenem Muster unter der Berücksichtigung von Sonderfällen ein. Diese können durch Leerzeichen im Dateipfad oder bei der Eintragung von Kommata entstehen.



Name	Modellpfad (Falls Leerzeichen -> Pfad in Gänsefüßchen setzen)	Zielpfad (Falls Leerzeichen -> Pfad in Gänsefüßchen setzen)	Dateien (mit Komma und ohne Leerzeichen trennen)
test1	"C:\CJ_TEST\test1\abc"	"C:\CJ_TEST\zieltest1\dfg"	"test1.txt","test2.txt","test3.txt"
test2	"C:\CJ_TEST\test 2\abc"	"C:\CJ_TEST\ziel test2\dfg"	"test A.txt","test B.txt"

Wie in der README.txt steht, muss die Excel-Datei neben der .exe-Datei liegen und beide sollten nicht unbenannt werden.

Der Inhalt der Logdatei sollte nach erfolgreichem Kopiervorgang ungefähr so aussehen:

```
-----
ROBOCOPY      ::      Robustes Dateikopieren fr Windows
-----

Gestartet: Donnerstag, 7. Januar 2021 11:07:32
Quelle : C:\CJ_TEST\test1\abc\
Ziel : C:\CJ_TEST\zieltest1\dfg\

Dateien : test1.txt

Optionen: /DCOPY:DA /COPY:DAT /Z /R:1000000 /W:30
-----

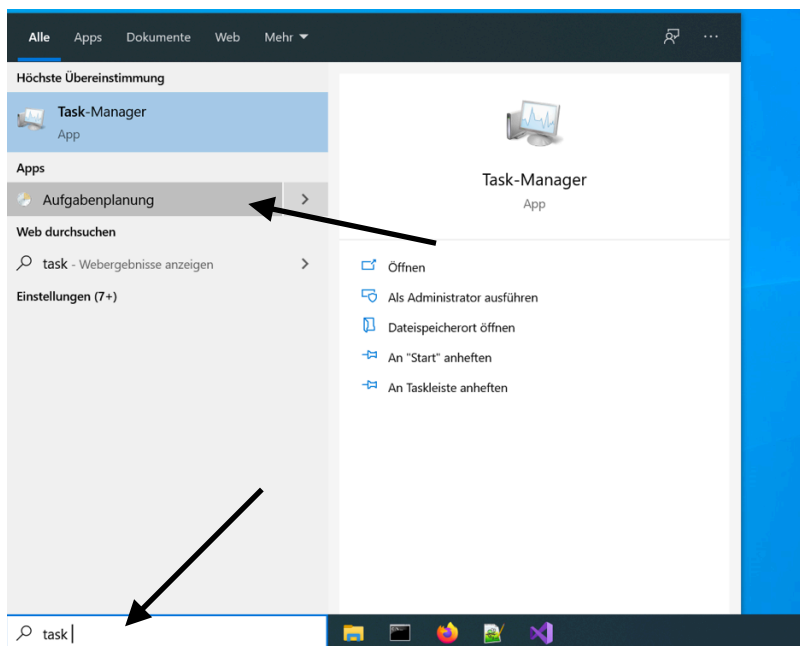
          1  C:\CJ_TEST\test1\abc\
Neue Datei      0      test1.txt
100%

-----

Insgesamt  KopiertÜbersprungenKeine Übereinstimmung  FEHLER  Extras
Verzeich.:      1      0      1      0      0      0
Dateien:      1      1      0      0      0      0
Bytes:      0      0      0      0      0      0
Zeiten:  0:00:00  0:00:00      0:00:00  0:00:00
Beendet: Donnerstag, 7. Januar 2021 11:07:32
```

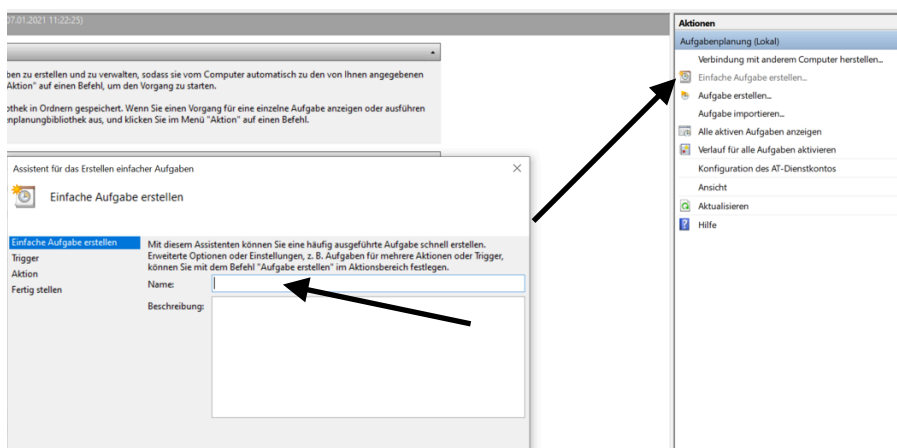
Falls Fehlermeldungen auftreten und die Routine nicht funktioniert: Alle eingetragenen Pfade und Dateinamen auf Fehler überprüfen, die Logdatei hilft hier in der Regel.

Wenn der Task in der Windowsaufgabenplanung neu eingetragen werden muss, geht man wie folgt vor:



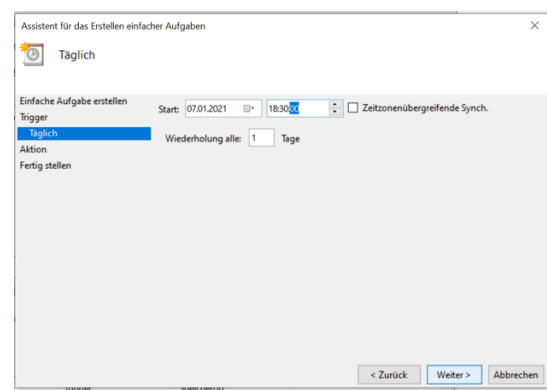
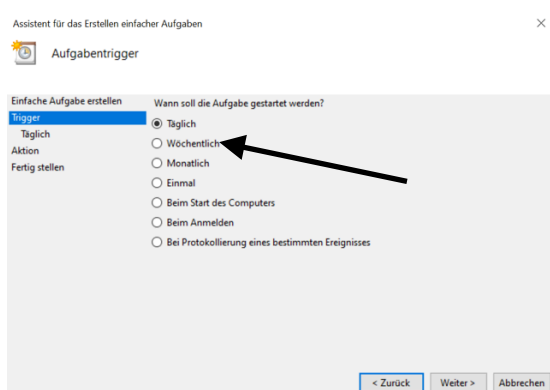
Wenn man in die Suchleiste „task“ einträgt, wird automatisch die Aufgabenplanung vorgeschlagen. Auf diese klicken.

Nicht zu verwechseln mit dem Task-Manager, den wir hier nicht benötigen.

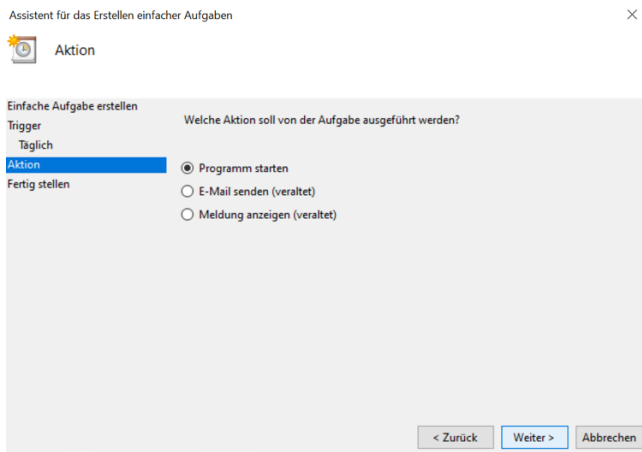


Nun öffnet sich die Aufgabenplanung. Nicht erschrecken, diese erscheint erstmal recht umfangreich.

Wir benötigen nur die Aktion „Einfache Aufgabe erstellen“, zu finden oben rechts.

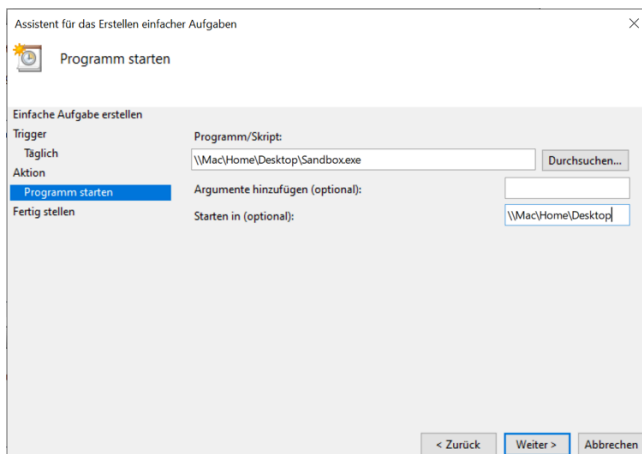


Nachdem Sie einen Namen eingetragen haben und auf „Weiter“ klicken, stellen Sie „täglich“ als Trigger ein. Als Uhrzeit muss ein Zeitpunkt **nach** dem Start des BIM-Publishers gewählt werden.



Als nächstes werden Sie gefragt, was zu diesem Zeitpunkt passieren soll.

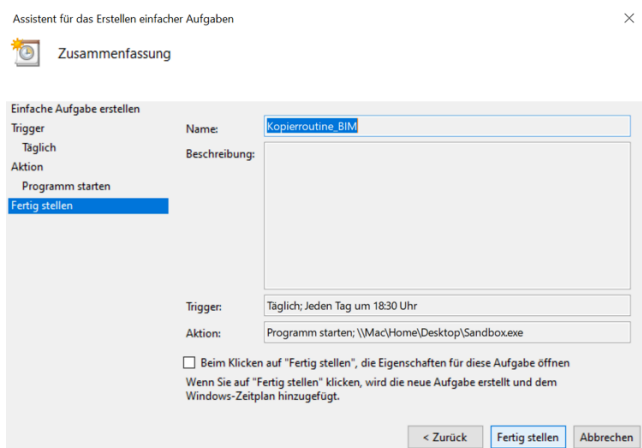
Wählen Sie „Programm starten“ aus.



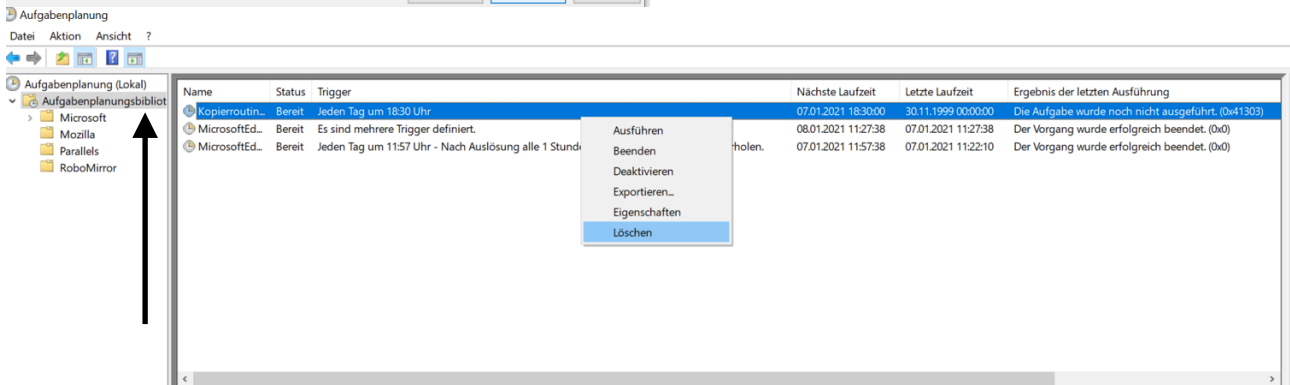
Jetzt müssen Sie den Pfad des Programms eintragen. Nutzen Sie dazu den „Durchsuchen“-Button.

Auf dem Automax01-Desktop ist der Ordner „BIM_Routine“ zu finden und die .exe-Datei darin heißt „BIM_Copy“.

Außerdem ist es wichtig, den Pfad zum Desktop unter „Starten in“ einzutragen.



Zuletzt erhalten Sie eine Zusammenfassung und schließen mit „Fertig stellen“ die Erstellung der einfachen Aufgabe ab.



Die Routine können Sie unter „Aufgabenplanungsbibliothek“ (oben links) einsehen und per Rechtsklick auch wieder löschen oder bearbeiten.

Anschrift Prüfling:

Klatt Martin
Weinbergstr. 45
92421 Schwandorf

Anschrift Ausbildungsbetrieb:

BFW Eckert gGmbH
Doktor-Robert-Eckert-Straße
93128 Regenstauf

Protokoll der durchgeführten Projektarbeit

Prüf-Nr.:

95014

Ausbildungsberuf:

Fachinformatiker Fachrichtung: Anwendungsentwicklung

1. Arbeitszeit:

1.1 Die vom Prüfungsteilnehmer kalkulierte Zeit entspricht der betrieblichen Kalkulation.

☒ ja ☐ nein

Wenn nein: Sie ist um _____ % höher _____ % niedriger

1.2 Das Projekt wurde vom Prüfungsteilnehmer in der kalkulierten Zeit komplett fertiggestellt (einschließlich eventueller Nacharbeit):

☒ ja ☐ nein

Wenn nein: Um _____ Stunden früher fertig geworden,
_____ Stunden länger gebraucht.

2. Ausführung:

2.1 Wurde das Projekt entsprechend dem eingereichten Konzept ausgeführt?

☒ ja ☐ nein

Wenn nein: Welche Änderungen ergaben sich?

bitte weiter auf der Rückseite!

Prüfungsteilnehmer:

Martin Klatt

BFW Eckert Regenstauf

Name / Vorname

Firma

2.2 Wurde das Projekt selbständig und ohne fremde Hilfe ausgeführt?

ja



nein

Wenn nein: Begründung und Umfang der Hilfestellung.

2.3 Das Projekt konnte ohne Nacharbeit in einem einwandfreien Zustand übergeben werden.

ja



nein

Wenn nein: Begründung und Umfang der Nacharbeit.

3. Dokumentation:**3.1** Die Dokumentation wurde vom Prüfungsteilnehmer selbständig und ohne fremde Hilfe erstellt.

ja



nein

Wenn nein: Welche Hilfestellung wurde gegeben?

3.2 Die Dokumentation entspricht den betrieblichen Anforderungen.

ja



nein

Wenn nein: Worin besteht die Abweichung?

07.01.2021

