HTL - Perg
Höhere Abteilung für Informatik

Diplomarbeit

SlideAssembler

Projektteam: Peitl Stefan
Kunnert Philipp

Projektbetreuer: Dipl.-Ing. Michael Stumpfl

In Zusammenarbeit mit softaware GmbH

Betreuer: Herr Daniel Sklenitzka

Bearbeitungszeitraum: 01.7.2024 – 4.04.2025

Eidesstattliche Erklärung

Hiermit versichern wir, die vorliegende Arbeit selbständig, ohne fremde Hilfe und ohne Benutzung anderer als der von uns angegebenen Quellen angefertigt zu haben. Alle Stellen, die wörtlich oder sinngemäß aus fremden Quellen direkt oder indirekt übernommen wurden, sind als solche gekennzeichnet.

Gendererklärung

Im Rahmen dieser Diplomarbeit wird aus Gründen der besseren Lesbarkeit auf die gleichzeitige Verwendung der weiblichen sowie der männlichen Form verzichtet. Alle Personenbezeichnungen beziehen sich gleichermaßen auf alle Geschlechter. Diese Vereinfachungen dienen ausschließlich der Lesbarkeit und sollen in keiner Weise Diskriminierung oder Benachteiligung bestimmter Geschlechter implizieren.

Danksagung

Wir möchten uns bei sämtlichen Personen der HTL Perg und der Softaware bedanken, die bei der Entstehung dieser Arbeit geholfen haben.

Dabei möchten wir uns besonders bei unserem Diplomarbeitsbetreuer Dipl.-Ing. Michael Stumpfl bedanken, der uns immer mit Rat und Tat beistand.

Weiters bedanken wir uns herzlich bei der Softaware für die Unterstützung, insbesondere bei unserem Betreuer Daniel Sklenitzka, der uns während des praktischen Teils stets unterstützte.

Abstract

SlideAssembler is a .NET library designed to facilitate the automated creation of frequently required PowerPoint presentations. A template serves as the foundation, offering placeholders and formatting options to dynamically insert data.

The library operates with a straightforward structure, allowing both the data and the template to be passed directly. Thanks to its well-thought-out Fluent API concept, SlideAssembler provides a wide range of powerful methods to effortlessly replace placeholders, dynamically generate charts from data, insert images, and flexibly adjust complex objects like tables.

A standout feature is its automatic adherence to formatting rules: numbers, dates, or other format-sensitive content are automatically rendered in the desired style. This ensures that the presentation is not only factually accurate but also visually appealing and consistent.

In no time at all, a complete, tailored PowerPoint presentation containing all required information is generated. This automated approach not only saves time but also ensures the creation of consistent, professionally designed reports—perfect for regular updates, corporate reports, or other presentations with recurring requirements.

Zusammenfassung

SlideAssembler ist eine .NET-Bibliothek, die die automatisierte Erstellung regelmäßig benötigter PowerPoint-Präsentationen erleichtert. Dabei dient eine Vorlage als Basis, die sowohl Platzhalter als auch Formatierungsoptionen bereitstellt, um Daten dynamisch einzufügen.

Die Bibliothek arbeitet mit einer einfachen Struktur, bei der sowohl die Daten als auch die Vorlage direkt übergeben werden. Dank des durchdachten Fluent-API-Konzepts stellt SlideAssembler eine Vielzahl von leistungsstarken Methoden bereit, mit denen sich Platzhalter mühelos ersetzen, Diagramme dynamisch aus Daten generieren, Bilder einfügen und komplexe Objekte wie Tabellen flexibel anpassen lassen.

Ein besonderes Highlight ist die automatische Berücksichtigung von Formatierungsregeln: Zahlen, Datumsangaben oder andere formatierungsrelevante Inhalte werden automatisch in das gewünschte Erscheinungsbild gebracht. So bleibt die Präsentation nicht nur inhaltlich korrekt, sondern auch optisch ansprechend und einheitlich.

Innerhalb kürzester Zeit entsteht so eine vollständige, maßgeschneiderte PowerPoint-Präsentation, die alle erforderlichen Informationen enthält. Dieser automatisierte Ansatz spart nicht nur Zeit, sondern garantiert auch die Erstellung konsistenter und professionell gestalteter Berichte – ideal für regelmäßige Updates, Unternehmensberichte oder andere Präsentationen mit wiederkehrenden Anforderungen.

Inhaltsverzeichnis

1 Einleitung

1.1 Auftraggeber

Der Auftraggeber dieses Projekts ist die Softaware GmbH, ein innovatives Softwareunternehmen, das 2012 in Asten gegründet wurde. Softaware entwickelt maßgeschneiderte Web- und App-Anwendungen sowie verschiedenste Lösungen in den Bereichen Cloud-Computing und künstliche Intelligenz,um individuelle und zukunftsorientierte Lösungen für verschiedenste Kunden zu realisieren.

1.2 Problemstellung

Eine der immer größer werdenden Herausforderungen, mit denen kundennahe Unternehmen konfrontiert sind, ist das regelmäßige Erstellen individualisierter Präsentationen, Kundenberichte oder Studien.

Doch auch bei Produktpräsentationen extern bei Kunden sind standardisierte PowerPoint-Präsentationen längst nicht mehr ansprechend.

Das Problem dabei ist, dass es diese Unternehmen Zeit, Mühe und auch Geld kostet, bestehende Powerpoint-Präsentationen manuell zu ändern oder neu zu erstellen, was nervig fehleranfällig und unproduktiv ist.

Schreibfehler, Denk- oder Rechenfehler order auch dass sich jemand in der falschen Zeile verirrt sind alles Szenarien, die bis jetzt nie ausgeschlossen werden konnten .

Eine extra Software zur schnelleren Bearbeitung von mehreren Power-Point Präsentationen gleichzeitig würde für die betroffen Unternehmen wieder Schulungsaufwand für das Personal und somit auch hohe Kosten bedeuten, da diese auch jemand bedienen muss.

1.3 Ziel des Projekts

Die Softaware GmbH bekam einen Auftrag, in dem es darum ging, genau diese Marktlücke zu behandeln, und als wir die Größe dieses Problems gesehen haben, wussten wir: Wir müssen etwas entwickeln, was sämtlichen Unternehmen weder Schulungsaufwand noch viel Geld kostet, aber trotzdem ihre Präsentationen flexibel macht.

Gemeinsam mit der Softaware GmbH setzten wir uns also das Ziel, mit SlideAssembler eine C# Bibliothek zu schaffen, welche mit geringstem Aufwand in die unternehmenseigene Software eingebunden werden kann und die es per Knopfdruck erlaubt, sämtliche Präsentationen mit nur einer Vorlage generieren zu können, mittels eines Platzhaltersystems.

Der Umfang von SlideAssembler sollte also die Möglichkeit beinhalten:

- Texte inklusive Formatierungen,
- Graphen,
- Bilder,
- Höhen und Breitenparameter von Balken und sonstigen Figuren,
- etwaige andere Modifizierungen an einem Objekt,

in Powerpoint-Präsentationen einzusetzen und/oder einstellen zu können.

2 Grundlagen und Methoden

2.1 C#

Die 2002 erstmals von Microsoft veröffentlichte Programmiersprache C# ist eine moderne objektorientierte Sprache, die im Rahmen der .NET-Plattform (im nachfolgendem Abschnitt 2.2 beschrieben) entwickelt wurde. Seitdem hat sie sich zu einer der beliebtesten und am weitest verbreiteten Programmiersprachen der Welt entwickelt, da sie viele Vorteile anderer vereint. C# kombiniert die Effizienz und Leistungsfähigkeit von C++ mit der Einfachheit und Sicherheit von Java.

Die Sprache zeichnet sich nicht nur durch ihre Flexibilität aus, denn auch ihre Anwendungszwecke sind sehr vielseitig. Sie eignet sich für die Entwicklung von Windows-Anwendungen, Webentwicklung, Spielentwicklung, mobiler App-Entwicklung als auch für die Entwicklung Cloud-basierter Dienste. Aufgrund dessen haben wir uns für diese Programmiersprache entschieden, da die meisten unternehmenseigenen Software auf C# basieren und es so möglich ist, unsere Bibliothek einzubinden [1].

2.2 .NET

Die von Microsoft geschaffene Plattform „.NET“ ist eine umfassende Entwicklungsplattform, die erstmals am 13. Februar 2002 mit der Veröffentlichung von „Visual Studio 2002“ eingeführt wurde. Von Beginn an unterstützte sie mehrere Programmiersprachen, darunter C# 1.0, F# 1.0, VB.NET 1.0 sowie „Managed C++“, das später durch C++/CLI ersetzt wurde [2].

Im Gegensatz zu heute war .NET in seinen Anfängen ausschließlich für Windows-Systeme konzipiert. Dies änderte sich erst mit der Einführung von .NET Core 1.0, das erstmals eine plattformübergreifende Entwicklung ermöglichte. Damit war es möglich, Anwendungen nicht nur für Windows, sondern auch für Linux- und macOS-Systeme zu entwickeln. Allerdings existierte das klassische .NET Framework für Windows und .NET Core bis zur Version .NET Core 3 zunächst parallel, wodurch Entwickler je nach Anwendungsfall zwischen den beiden Varianten wählen mussten [3].

Mit der Veröffentlichung von .NET 5.0 am 10. November 2020 wurde diese Trennung schließlich aufgehoben. Seitdem existiert mit .NET eine einheitliche Plattform, die sowohl die Vorteile des ursprünglichen .NET Frameworks als auch die plattformübergreifenden Möglichkeiten von .NET Core kombiniert. Entwickler können nun mit einer gemeinsamen Codebasis Anwendungen für verschiedene Betriebssysteme erstellen und von einer Vielzahl an Tools und Bibliotheken profitieren, die .NET bereitstellt [3].

2.3 Bibliotheken

Bei einer Bibliothek in der Softwareentwicklung handelt es sich um eine Sammlung von vorgefertigten Funktionen, Klassen, Interfaces etc., die spezifische Aufgaben effizient lösen und somit den Entwicklern unter die Arme greifen.

In C# werden Bibliotheken in Form von DLLs (Dynamic Link Libraries) bereitgestellt. Diese können dann als NuGet-Packages veröffentlicht werden [4].

2.4 NuGet

NuGet ist ein Paketverwaltungssystem für .Net-Anwendungen und wird für die vereinfachte Verwaltung und Bereitstellung von Bibliotheken und Abhängigkeiten innerhalb von .Net-Projekten verwendet. NuGet erleichtert Entwicklern die Installation und das Einbinden sowie das Aktualisieren externer C#-Bibliotheken ohne großen Konfigurationsaufwand [5].

2.4.1 NuGet Packages

Wenn man also von NuGet-Packages redet, so meint man die installierbaren Bibliotheken über NuGet. Eine Bibliothek kann also in ein Archiv oder ein sogenanntes „NuGet Package“ mit dem „.nupkg“-Format komprimiert werden und dann einfach in ein anderes Projekt eingebunden werden.

Diese Pakete können Folgendes beinhalten:

- **DLLs** — Kompilierter Code welcher in .Net-Projekten genützt wird.
- **Konfigurationsdateien** – Für die Bibliothek relevante XML- oder Json-Dateien
- **Skripte und Ressourcen** – zusätzliche Dateien wie zum Beispiel Bilder oder Stylesheets
- **Abhängigkeiten** – andere NuGet Packages die in diesem enthalten sind und so mitinstalliert werden müssen.
- **Metadaten** – Informationen über das Paket selbst wie Version, Autor oder eine kurze Beschreibung

Wir haben NuGet im Rahmen des Projekts genutzt, um unsere Bibliothek von außen mittels eines einzigem NuGet-Pakets einfach für Unternehmen zugänglich und installierbar zu machen.

2.5 Fluent-API-Konzept

Fluent API ist ein Designkonzept in der Softwareentwicklung, das darauf abzielt, Code möglichst lesbar und intuitiv zu gestalten. Sie weist folgende Merkmale auf [6]:

- **Method Chaining:** Dabei werden verschiedene Methoden in einer Kette aufgerufen. Jede Methode gibt entweder das Hauptobjekt oder ein verwandtes Objekt zurück, sodass Methoden direkt auf dem Ergebnis der vorherigen Methode aufgerufen werden können.
- **Selbsterklärende Methodennamen:** Die Methoden einer Fluent API sollten möglichst sprechende Namen haben, um eine einfache Verwendung zu gewährleisten.
- **Frühe Validierung:** Argumente sollten in den verschiedenen Methoden stets auf `null` oder andere ungültige Werte überprüft werden [7] .
- **Eindeutiger Abschluss:** Fluent APIs verwenden eine Abschlussoperation, die das finale Objekt zurückgibt.

Im folgenden Code-Listing ist ein Beispielaufruf der Fluent API in SlideAssembler zu sehen, der zwei Operationen auf die Präsentation anwendet:

Listing 1: Beispielaufruf der Fluent API

```
presentation
    .Apply(new FillPlaceholders(data))
    .Apply(new FillChart("ChartName", series))
    .Save(outputStream);
```

Das Fluent-API-Konzept wurde gewählt, da es eine schnelle und einfache Anwendung verschiedener Operationen auf die Präsentation ermöglicht.

2.6 Git

Git ist ein verteiltes Versionskontrollsystem, das Softwareentwicklern hilft, Änderungen am Programm zu verfolgen, zu verwalten oder gegebenenfalls zu verwerfen. Es hilft Teams effizient parallel zusammenzuarbeiten, indem es ermöglicht, verschiedene Versionen des Codes zu erstellen, diese zu vergleichen oder sie auch zusammenzuführen. Durch einen kleinen Kommentar oder die sogenannte „Commit-Message“ wird jede Änderung am Code leicht für jeden verständlich [8].

2.6.1 GitHub

GitHub hingegen ist eine webbasierte Plattform, die auf Git aufbaut und zusätzliche Funktionen für die Zusammenarbeit und Projektverwaltung unter Entwicklerteams bietet. GitHub dient als zentraler Ort, an dem der Code gespeichert, geteilt und diskutiert werden kann.

GitHub bietet Funktionen wie:

- **Forks erstellen:** Kopien anderer Repositories erstellen
- **Pull Requests:** Anfragen auf einen Merge den andere User bestätigen müssen
- **Copilot:** KI-gestützter Assistent für Programmieraufgaben

und vieles mehr [9].

Wir haben GitHub auch als unsere zentrale Codeverwaltungsstelle genommen und außerdem benutzten wir die GitHub-Actions um neue Versionen automatisiert zu testen.

2.6.2 GitHub Actions CI/CD

GitHub-Workflows, auch bekannt als GitHub-Actions, sind ein integraler Bestandteil der **Continuous Integration** (CI) und **Continuous Deployment** (CD) Prozesse auf GitHub.

Diese Workflows können verschiedenste Aufgaben im Softwareentwicklungsprozess automatisieren, wie zum Biespiel:

1. **CI:** Automatisiert den Build-Prozess und das Testen von Code bei jedem Commit oder Pull-Request. Dies hilft, Fehler frühzeitig zu erkennen und die Codequalität zu verbessern [10]
2. **CD:** Automatisiert den Prozess der Softwarebereitstellung. Nach erfolgreichen Tests kann der Code automatisch in verschiedene Umgebungen ausgerollt werden.

Diese Workflows können nach verschiedensten Ereignissen konfiguriert werden wie zum Beispiel nach einem Push oder nach einer gewissen Zeit. Sie können aber genauso auch manuell gestarten werden [11].

2.7 Notion

Notion ist eine vielseitige Produktivitäts- und Kollaborationsplattform, die 2016 von Ivan Zhao und Simon Last gegründet wurde. Sie hat sich schnell zu einem beliebten Tool für Einzelpersonen, Teams und Unternehmen entwickelt, da eine einfache und benutzerfreundliche Plattform gegeben ist, die denjenigen Teams die Planung und Dokumentation ihrer Arbeit vereinfacht. Notion kombiniert Funktionen wie Textverarbeitung, Tabellenkalkulation, Datenbanken, Wikis und Aufgabenmanagement in einer integrierten Plattform [12]. Notion wurde im Rahmen dieser Diplomarbeit verwendet, um einen Überblick über das Projekt zu behalten und um Fehler in der Bibliothek oder Besprechungen mit dem Auftraggeber1.1 zu Protokollieren.

2.8 Powerpoint

Bei PowerPoint handelt es sich um eine Präsentationssoftware, welche von Microsoft entwickelt wurde. Benutzer können damit ansprechende Folien mit verschiedenen Elementen gestalten. Seit 2007 verwendet PowerPoint das Office-Open-XML-Format, was es SlideAssmbler ermöglicht, aus Templates fertige Präsentationen zu erstellen.

2.8.1 Office Open XML Format

Das Office Open XML Format wurde 2007 von Microsoft eingeführt und löste die alten binär-Formate von Office Dateien ab. Im Grunde ist eine OOXML-Datei (alle Office-Dateien mit x am Ende, z.B .pptx) nichts anderes als ein ZIP-Container, in dem sich mehrere XML-Dateien befinden. Die verschiedenen XML-Dateien beschreiben dabei die Struktur, den Inhalt sowie die Formatierung des Dokumentes. Eine XML-Datei kann dabei folgende Bestandteile haben [13]:

- **Parts:** Dabei handelt es sich um die einzelnen Inhaltskomponenten, z.B Text, Bilder etc.
- **Items:** Items sind Metadaten, die beschreiben, wie die einzelnen Parts in der Präsentation dargestellt werden sollen. Dabei gibt es zwei Arten von Items:
  - Relationship Items: Beschreibt, wie die verschiedenen Parts zusammenhängen.
  - Content Type Item: Beschreibt, wie die Parts dargestellt werden müssen.

Struktur

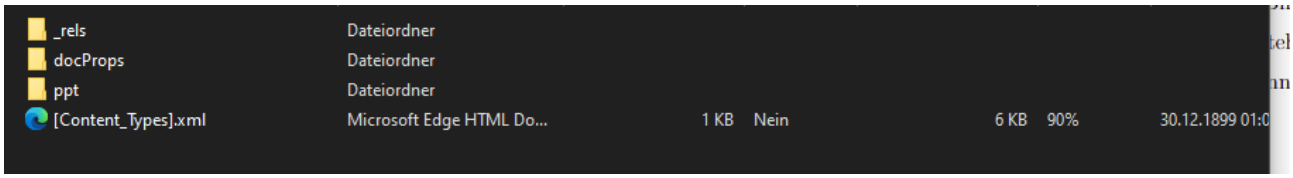

Abbildung 1: Struktur des ZIP-Contaniers einer Beispiel-PowerPoint

Im Wurzelverzeichnis eines ZIP-Containers befindet sich eine XML-Datei namens Content_Types.xml und folgende drei Unterverzeichnisse: _rels, docProps und ppt.

- **Content_Types.xml:** Diese Datei beinhaltet eine Beschreibung der Inhalte des ZIP-Archives bzw.wo welche XML-Dateien gespeichert werden.
- **_rels:** In diesem Verzeichnis werden alle Abhängigkeiten zwischen den verschiedenen Parts gespeichert. Dabei hat jeder Part seine eigene Datei mit der Endung .rel.
- **docProps:** Dieses Verzeichnis enthält folgende zwei Dateien: app.xml und core.xml. Diese Dateien speichern Metadaten wie Autor, Speicherdatum etc.
- **ppt:** Dieses Verzeichnis beinhaltet alle Dokumentinhalte, in diesem Fall die Inhalte der PowerPoint, z.B die einzelnen Folien.

2.9 Verwendete Softwarebibliotheken

2.9.1 Open XML SDK

Das Open XML SDK ist eine von Microsoft entwickelte Softwarebibliothek, die das Arbeiten mit dem OpenXML-Schema erleichtert. Sie stellt folgende Funktionen bereit [14]:

- **Bearbeitung von Dokumenten**
- **Generierung von Dokumenten**
- **Leistungsoptimierung**
- **Strukturverwaltung**

Listing 2: Beispiel: Open XML SDK

```
1  // Get all the text in a slide.
2  public static string[] GetAllTextInSlide(string presentationFile, int slideIndex)
3  {
4      // Open the presentation as read-only.
5      using (PresentationDocument presentationDocument =
           PresentationDocument.Open(presentationFile, false))
6      {
7          // Pass the presentation and the slide index
8          // to the next GetAllTextInSlide method, and
9          // then return the array of strings it returns.
10         return GetAllTextInSlide(presentationDocument, slideIndex);
11     }
12 }
```

Der oben angeführte Code zeigt, wie der gesamte Text einer PowerPoint-Präsentation ausgelesen werden kann. Dazu wird der Dateipfad der Präsentation sowie der Index der gewünschten Folie benötigt. Diese Parameter werden an die Methode `GetAllTextInSlide` übergeben, welche ein Array von Zeichenketten zurückgibt.

2.9.2 ShapeCrawler

ShapeCrawler ist eine Open-Source-.NET-Bibliothek zur Manipulation von PowerPoint-Präsentationen. Die Bibliothek basiert auf einer Objektstruktur, die das Open-XML-SDK 2.9.1 nutzt. Mit ShapeCrawler lassen sich Texte, Bilder, Diagramme und Tabellen einfach bearbeiten. Eine Einschränkung von ShapeCrawler besteht jedoch darin, dass nicht alle Elemente einer PowerPoint-Präsentation modifiziert werden können – beispielsweise ist es nicht möglich, SmartArt-Grafiken zu bearbeiten.

Funktionsweise

ShapeCrawler stellt ein einfaches Objektmodell für PowerPoint-Präsentationen zur Verfügung und bietet verschiedene Attribute. Dabei wird über das Open-XML-SDK auf die entsprechenden XML-Attribute von PresentationML zugegriffen. Die Klasse `Presentation` dient als Root-Klasse und benötigt als Konstruktorparameter entweder einen Stream oder einen String mit dem Dateinamen (bzw. dem Pfad zur Datei), um auf die vorhandene PowerPoint-Präsentation zuzugreifen. `Presentation` verfügt über eine Sammlung von Slides, die die einzelnen Folien repräsentieren. Jede Folie enthält verschiedene Shapes. Eine Shape ist ein Element auf einer Folie und kann z. B. ein Diagramm, eine Tabelle oder ein Textfeld sein. Jede Shape besitzt spezifische Attribute und Methoden.

Im folgenden Abschnitt werden einige Beispiele der wichtigsten Funktionen von ShapeCrawler gezeigt, die im Rahmen der Implementierung dieser Arbeit verwendet wurden.

Beispiele

Folgendes Code-Listing zeigt, wie es mit ShapeCrawler möglich ist, verschiedene Textoperationen auf einer Shape auszuführen.

Listing 3: Beispiel für Textmanipulation mit ShapeCrawler

```
    // open presentation and get first slide
var pres = new Presentation("helloWorld.pptx");
var slide = pres.Slides.First();

// get text holder auto shape
var shape = slide.Shapes.First();

// update text of shape
shape.TextBox.Text = "A new shape text";

// change text for a certain paragraph
var paragraph = shape.TextBox.Paragraphs[1];
paragraph.Text = "A new text for second paragraph";

// get font name and size
var paragraphPortion = shape.TextBox.Paragraphs.First().Portions.First();
Console.WriteLine($"Font name: {paragraphPortion.Font.LatinName}");
Console.WriteLine($"Font size: {paragraphPortion.Font.Size}");

// set bold font
paragraphPortion.Font.IsBold = true;

// get font ARGB color
var fontColor = paragraphPortion.Font.Color.Hex;

// update font color
paragraphPortion.Font.Color.Update("FF0000");

pres.Save();
```

Zunächst wird die PowerPoint-Datei geladen, indem der Dateiname in den Konstruktor der `Presentation`-Klasse übergeben wird. Anschließend wird die erste Folie ausgelesen und von dieser eine Shape ermittelt, deren Text, Schriftart und Farbe mithilfe der verschiedenen Attribute ausgelesen und geändert werden.

Es ist mit ShapeCrawler auch möglich, verschiedene Operationen an Diagrammen vorzunehmen. Im folgenden Listing wird beschrieben, wie eine Datenserie in einem Diagramm bearbeitet bzw. gelöscht werden kann.

Listing 4: Beispiel für Charts mit ShapeCrawler

```
var pres = new Presentation("test.pptx");
var chart = pres.Slides[0].Shapes.GetByName<IChart>("Chart 1");
var point = chart.SeriesList[0].Points[0];
point.Value = 10;
chart.SeriesCollection.RemoveAt(0);
```

Das Diagramm mit dem Namen "Chart 1"wird mithilfe der Funktion `.GetByName` gefunden. Dabei stellt der generische Typ sicher, dass nur Shapes, die das `IChart`-Interface implementieren, berücksichtigt werden. Das Diagramm verfügt über eine `SeriesList`, welche die verschiedenen Datenserien speichert. Die `SeriesCollection` besitzt außerdem eine `RemoveAt`-Funktion, mit der eine Datenserie aus dem Diagramm entfernt werden kann.

Zusätzlich besitzen Diagramme weitere Eigenschaften, die im folgenden Listing kurz beschrieben werden.

Listing 5: Beispiel für Chart Properties

```
var pres = new Presentation("helloWorld.pptx");
var slide = pres.Slides.First();

// get chart
var chart = (IChart)slide.Shapes.First(sp => sp is IChart);

// print chart title
if (chart.HasTitle)
{
    Console.WriteLine(chart.Title);
}

if (chart.Type == ChartType.BarChart)
{
    Console.WriteLine("Chart type is BarChart");
}

// update category
chart.Categories[0].Name = "Price";
```

In diesem Beispiel wird das erste Diagramm aus der ersten Folie geladen. Danach wird mit `HasTitle` überprüft, ob das Diagramm einen Titel besitzt, und falls ja, wird dieser über `.Title` ausgelesen. Zudem kann mit `chart.Type` die Art des Diagramms bestimmt werden. Darüber hinaus ermöglicht `Categories` den Zugriff auf die verschiedenen Kategorien eines Diagramms.

Neben der Modifikation bestehender Shapes ist es auch möglich, neue Shapes in eine Folie einzufügen.

Listing 6: Beispiel für das Hinzufügen von Shapes

```
1 var pres = new Presentation();
2 var shapes = pres.Slides[0].Shapes;
3
4 shapes.AddRectangle(x:50, y:60, width:100, height:70);
```

Um eine neue Shape zu erstellen, müssen wie zuvor die Präsentation und die Folie geladen werden. Anschließend werden alle Shapes der Folie geladen, und mit der Funktion `shapes.AddRectangle` wird ein neues Rechteck in die Folie eingefügt. Als Eingabeparameter werden die x- und y-Koordinaten sowie die Breite und Höhe des gewünschten Rechtecks benötigt [15].

Alternativen

Es gibt andere Bibliotheken mit ähnlicher Funktionalität. Diese bieten jedoch entweder nicht die gewünschten Features, sind komplexer oder kostenpflichtig. Eine weitere Alternative wäre der direkte Zugriff auf die XML-Dateien zur Manipulation der Präsentationen. Dies würde jedoch einen erheblichen Mehraufwand für das Projekt bedeuten, der nicht notwendig ist. Des Weiteren ist ein direkter Zugriff fehleranfälliger bzw. funktioniert nicht in allen Fällen zuverlässig.

2.9.3 Syncfusion Presentationrenderer

Syncfusion PresentationRenderer ist eine kommerzielle .NET-Bibliothek zur Verarbeitung und Konvertierung von Powerpoint-Präsentationen. Sie ist Teil der Syncfusion Essential Presentation Suite und bietet umfangreiche Funktionen für die Arbeit mit Powerpoint-Dateien innerhalb einer C#-Anwendung.

Diese Bibliothek bietet eine Vielzahl an Funktionen wie das Konvertieren von Folien in „PDF“-Files oder „PNG“-Files. Aber auch kann direkt über den Code auf die Präsentation wie in der Bibliothek „Shapecrawler“2.9.2 zugegriffen werden. Bei diesem Projekt wurde aber trotzdem Shapecrawler aufgrund der einfacheren Anwendung verwendet.

Eine Konvertierung von einer Powerpoint zu einer „PDF“-Datei kann mit Syncfusion PresentationRenderer ziemlich einfach wie folgt realisiert werden:

Listing 7: Konvertierung von PowerPoint nach PDF

```
using Syncfusion.Presentation;
using Syncfusion.PresentationRenderer;

public void ConvertPptToPdf(string inputPath, string outputPath)
{
    using (IPresentation presentation = Presentation.Open(inputPath))
    {
        presentation.RenderAsPdf(outputPath);
    }
}
```

Wie bereits erwähnt ist dies eine kommerzielle Bibliothek, wodurch sie normalerweise bis zu 695$ im Monat kosten kann. Für dieses Open-Source-Projekt haben wir allerdings eine Community-Lizenz erhalten und dürfen sie so kostenfrei nutzen.

Diese Bibliothek wurde im Rahmen dieses Projekts genutzt, um die verschiedenen Folien einer Präsentation in „.png“-Bilddateien zu verwandeln und sie anschließend mit verifizierten Bildern dieser Folien zu vergleichen. Siehe 4.0.2

Alternativen

Es gibt viele Alternativen zu dieser Bibliothek, die es auch ermöglichen, die Folien in „.png“-Dateien umzuwandeln oder vieles mehr, jedoch unterscheiden sich alle in ihrer Handhabung und Einfachheit. Syncfusion ist unkompliziert zu bedienen und kann nur mit wenigen Zeilen Code das gewünschte Ergebnis herbeiführen, weshalb wir uns für diese Bibliothek entschieden haben.

2.9.4 MSVerifyTests

`MSVerifyTests` ist eine Open-Source-`.Net`-Bibliothek, die eine nahtlose Integration des `Verify-Frameworks` in MSTest-Projekte ermöglicht. Diese Bibliothek vereinfacht den Prozess des Snapshot-Testings in MSTest-Umgebungen erheblich [16].

Die Bibliothek basiert auf dem `Verify-Framework` und erweitert dessen Funktionalität speziell für MSTest. Mit `MSVerifyTests` können Entwickler komplexe Objekte und Strukturen einfach überprüfen, indem sie Snapshots von Testergebnissen erstellen und diese anschließend automatisch vergleichen lassen [17].

VerifyBase-Klasse

Ein zentrales Element von `MSVerifyTests` ist die `VerifyBase`-Klasse. Diese Klasse dient als Grundlage für Testklassen, die `Verify` in Verbindung mit MSTest nutzen möchten. Durch die Vererbung von `VerifyBase` erhalten Entwickler automatisch Zugriff auf die `Verify`-Methode in ihren Tests, ohne zusätzliche Attribute oder Konfigurationen hinzufügen zu müssen. Die `VerifyBase`-Klasse kümmert sich um die Initialisierung der `Verify`-Umgebung, das Konfigurationsmanagement und die Integration in den MSTest-Lebenszyklus.

Alternativen

Es gibt andere Snapshot-Testing-Bibliotheken wie zum Beispiel „Snapshooter“ [18], die ähnliche Funktionalitäten bieten. Diese sind jedoch entweder nicht speziell für MSTest optimiert oder bieten nicht die gleiche nahtlose Integration. Eine Alternative wäre die manuelle Implementierung von Snapshot-Tests ohne spezialisierte Bibliothek, was jedoch einen erheblichen Mehraufwand für das Projekt bedeutet hätte und so nicht in frage kam.

2.10 Regular Expressions

Eine *Regular Expression* (kurz *Regex*) ist ein Muster, das verwendet wird, um bestimmte Textstrukturen zu suchen, zu validieren oder zu ersetzen. Regular Expressions, oder im deutschen Sprachraum auch Reguläre Suchausdrücke genannt, spielen eine zentrale Rolle in der Softwareentwicklung, da sie eine effiziente Möglichkeit bieten, Zeichenketten zu durchsuchen oder zu manipulieren. Sie werden unter anderem in der Datenvalidierung, Textverarbeitung, Suchfunktionen und bei der Analyse großer Datenmengen eingesetzt.

Regular Expressions in C#

In C# stellt der Namespace `System.Text.RegularExpressions` die benötigten Klassen und Methoden für die Arbeit mit Regular Expressions bereit. Die Klasse `Regex` ermöglicht das Erstellen und Anwenden regulärer Ausdrücke.

Listing 8: Beispiel für eine Regular Expression

```
using System;
using System.Text.RegularExpressions;

class Program
{
    static void Main()
    {
        Regex myRegex = new Regex("^[0-9]+$"); // Nur Zahlen erlaubt
        string input = "12345";

        if (myRegex.IsMatch(input))
        {
            Console.WriteLine("Eingabe ist gueltig.");
        }
        else
        {
            Console.WriteLine("Eingabe enth lt ungueltige Zeichen.");
        }
    }
}
```

- `^` – Anfang der Zeichenkette
- `[0-9]+` – Mindestens eine Ziffer (0–9)
- `$` – Ende der Zeichenkette

Dieses Beispiel überprüft, ob eine Zeichenkette nur aus Zahlen besteht. Falls ja, wird eine entsprechende Meldung ausgegeben [19].

Im Zuge der Implementierung des Projekts wird eine Regular Expression verwendet um die definierten Platzhalter zu erkennen und auszutauschen.

2.11 Reflection

Mit Reflection ist es einem Programm möglich, seine eigene Struktur während der Laufzeit zu analysieren oder zu modifizieren. Außerdem erlaubt Reflection das dynamische Instanziieren von Objekten zur Laufzeit. In .NET werden verschiedene Metadaten in den Assemblies gespeichert und können über den Namespace `System.Reflection` ausgelesen werden. Zu diesen Metainformationen gehören unter anderem:

- Typinformationen (z. B. Klassen- und Schnittstellennamen)
- Informationen über Felder und Eigenschaften
- Methodeninformationen (Methodennamen, Parameter, Rückgabewerte)
- Konstruktorinformationen
- Assembly-Informationen (z. B. Versionsnummer, Abhängigkeiten)

Listing 9: Beispiel für Reflection in C#

```
using System;
using System.Reflection;

public class Person
{
    public string Name { get; set; }
    public int Age { get; set; }

    public void SayHello()
    {
        Console.WriteLine("Hello!");
    }
}

class Program
{
    static void Main()
    {
        // Typ abrufen
        Type personType = typeof(Person);

        // Klassenname ausgeben
        Console.WriteLine($"Klassenname: {personType.Name}");

        // Eigenschaften auslesen
        Console.WriteLine("\nEigenschaften:");
        foreach (PropertyInfo property in personType.GetProperties())
        {
            Console.WriteLine($"- {property.Name} ({property.PropertyType.Name})");
        }

        // Methoden auslesen
        Console.WriteLine("\nMethoden:");
        foreach (MethodInfo method in personType.GetMethods(BindingFlags.Public |
            BindingFlags.Instance))
        {
            Console.WriteLine($"- {method.Name}");
        }
    }
}
```

Erklärung des Codes:

- `Type personType = typeof(Person);` – Ermittelt den Typ der Klasse `Person`.
- `personType.Name` – Gibt den Namen der Klasse aus.
- `personType.GetProperties()` – Listet alle Eigenschaften (Properties) der Klasse `Person` auf.
- `personType.GetMethods(BindingFlags.Public | BindingFlags.Instance)` – Listet alle öffentlichen Methoden der Klasse auf.

Im obigen Beispiel werden die verschiedenen Attribute und Methoden der Klasse `Person` zur Laufzeit ausgelesen [20]. Im Projekt wurde Reflection benutzt um die Daten und Platzhalter richtig zuordnen zu können.

3 Implementierung

3.1 Architektur

Die Architektur von SlideAssembler basiert auf dem im Kapitel 2.5 beschriebenen `Fluent API-Konzept`. Dadurch können die verschiedenen Operationen flexibel auf das Template angewendet werden.

Hauptkomponenten

SlideAssembler besteht aus folgenden Hauptkomponenten, die die wichtigsten Methoden zur Verwaltung und Manipulation von PowerPoint-Dateien sowie Schnittstellen für die einzelnen Operationen bereitstellen:

- Template
- Presentation
- IPresentationContext
- NamedShapeOperation

Die Funktionsweise der oben gelisteten Hauptkomponenten wird in den folgenden Kapiteln genauer beschrieben.

Datenfluss

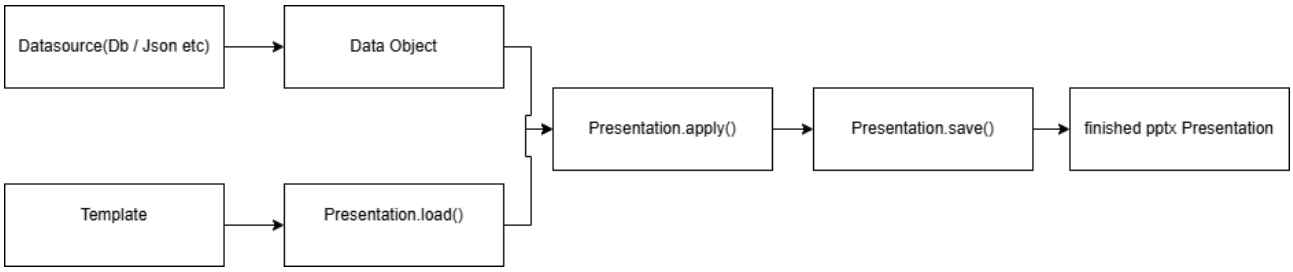

Abbildung 2: Datenflussdiagramm von SlideAssembler

Der Datenfluss in SlideAssembler gliedert sich wie folgt: Zunächst wird ein Template über die `load`-Methode in der `Presentation`-Klasse (siehe Kapitel 3.1.3) geladen. Anschließend ist es möglich, mit `Presentation.Apply(new Operation())` verschiedene Operationen sequenziell auszuführen. Dabei benötigen die Operationen ein einfaches Objekt, dessen Attribute die gleichen Namen wie die entsprechenden Platzhalter im Template haben müssen.

Es wurde bewusst ein simples C#-Objekt gewählt, da es so möglich ist, dieses aus verschiedenen Datenquellen wie z. B. einer Datenbank oder JSON-Dateien zu befüllen. Dadurch bleibt die Bibliothek leicht in bestehenden Programmcode integrierbar.

Als Ergebnis der Operation entsteht eine fertige PowerPoint-Präsentation, in der alle Platzhalter mit den entsprechenden Daten befüllt wurden. Diese kann anschließend mit `Presentation.Save()` über einen Stream gespeichert werden.

3.1.1 Aufbau eines Templates

Ein Template bildet die Grundlage für SlideAssembler, um eine Präsentation zu erstellen. Dabei besteht ein Template aus mehreren Platzhaltern, die auf Charts, Textelemente oder beliebige Objekte angewendet werden können. Des Weiteren verfügen Platzhalter über eine optionale Formatierungsoption.

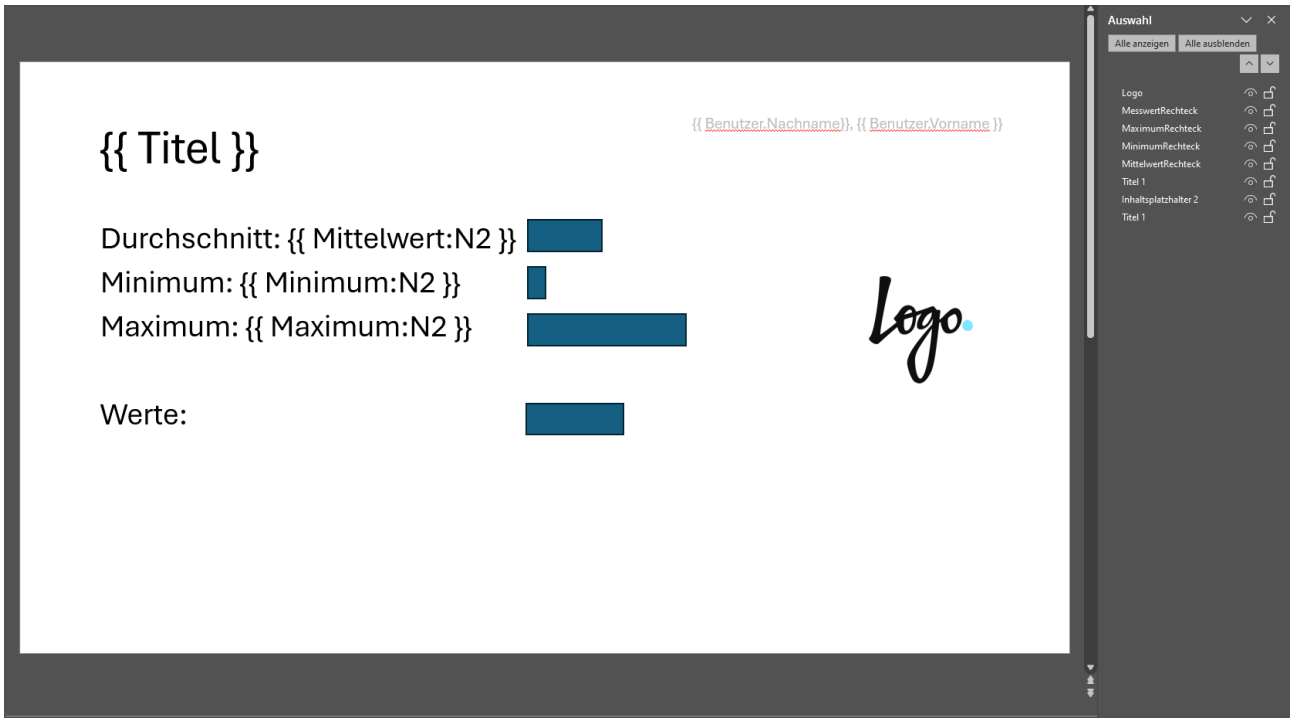

Abbildung 3: Beispiel für ein einfaches Template

Abbildung 3 zeigt ein einfaches Template. Platzhalter werden immer folgendermaßen definiert: `{{data:h2}}`. Dabei steht ‘data‘ für die Datenquelle, während ‘:h2‘ eine optionale Formatierungsoption darstellt.

Es ist außerdem möglich, Formen und Bilder als Platzhalter zu definieren. In diesen Fällen wird lediglich ein Name benötigt. Auch Charts können manipuliert werden, allerdings müssen diese vorher bereits existieren und Werte enthalten.

3.1.2 IPresentationOperation

Die Klasse IPresentationOperation ist ein Interface, das von jeder Funktion von SlideAssembler implementiert wird. Es verpflichtet jede dieser Funktion eine „Apply“ Methode zu besitzen, sodass die Ausführung einheitlich bleibt.

An die „Apply“ Methode wird immer die PresentationContext-Klasse übergeben welche im Abschnitt 3.1.4 beschrieben wird.

Listing 10: IPresentationOperation Interface

```
1       namespace SlideAssembler
2   {
3       public interface IPresentationOperation
4       {
5           void Apply(PresentationContext context);
6       }
7   }
```

3.1.3 Presentation

Die Klasse `Presentation` ist eine Wrapper-Klasse für `ShapeCrawler`, die zusätzliche Überprüfungen und Funktionen bereitstellt. `Presentation` bietet folgende drei Hauptfunktionen:

- **Load**

Listing 11: Load-Funktion in der Presentation-Klasse

```
1    public static Presentation Load(Stream stream, bool throwOnError = true)
2   {
3       if (!IsPowerPointStream(stream))
4       {
5           throw new FileLoadException("The file given is not a PowerPoint
                presentation!");
6       }
7
8       return new Presentation(
9           new PresentationContext(
10              new ShapeCrawlerPresentation(stream),
11              throwOnError));
12  }
```

Die Methode `Load` lädt eine PowerPoint-Präsentation aus einem beliebigen `Stream`. Dabei wird überprüft, ob es sich tatsächlich um eine PowerPoint-Datei handelt, indem die

Methode `IsPowerPointStream` verwendet wird. Zusätzlich kann der Benutzer über den Parameter `throwOnError` steuern, ob die Erstellung der Präsentation bei einem Fehler (z. B. ein nicht vorhandener Platzhalter) abgebrochen wird. Die Methode gibt ein neues `Presentation`-Objekt zurück, auf dem weitere Operationen ausgeführt werden können.

- **Save**

Listing 12: Save-Funktion in der Presentation-Klasse

```
1  public void Save(Stream stream)
2  {
3      if (stream == null)
4      {
5          throw new ArgumentNullException(nameof(stream), "Stream can't be null.");
6      }
7
8      this.context.Presentation.SaveAs(stream);
9  }
```

Die Methode `Save` speichert die PowerPoint-Präsentation in einen `Stream`. Falls der übergebene `Stream` `null` ist, wird eine `ArgumentNullException` ausgelöst.

- **Apply**

Listing 13: Apply-Funktion in der Presentation-Klasse

```
1  public Presentation Apply(params IPresentationOperation[] operations)
2  {
3      foreach (var operation in operations)
4      {
5          operation.Apply(this.context);
6      }
7
8      return this;
9  }
```

Die Methode `Apply` ermöglicht es, verschiedene Manipulationen an der Präsentation vorzunehmen. Dazu müssen die gewünschten Operationen das Interface `IPresentationOperation` implementieren. Es ist außerdem möglich, mehrere Operationen zu übergeben; diese werden dann innerhalb der `foreach`-Schleife nacheinander auf die PowerPoint-Datei angewendet. Die Methode gibt das aktuelle `Presentation`-Objekt zurück, wodurch das Fluent API-Konzept unterstützt wird.

3.1.4 PresentationContext

Listing 14: Apply-Funktion in der Presentation-Klasse

```
1  public record PresentationContext(
2      ShapeCrawlerPresentation Presentation,
3      bool ThrowOnError = false);
```

`PresentationContext` ist eine Klasse, die zwei Attribute beschreibt:

- Eine `ShapeCrawlerPresentation`, die für spätere Operationen genutzt wird.
- `ThrowOnError`, das für die Fehlerbehandlung zuständig ist (bereits in Kapitel 3.1.3 beschrieben).

3.1.5 NamedShapeOperation

Bei `NamedShapeOperation` handelt es sich um eine abstrakte Basisklasse für Funktionen, die mit verschiedenen Shapes (Formen) wie z. B. Rechtecken oder anderen Elementen arbeiten. Die Klasse ermöglicht es, gezielt auf Shapes mit bestimmten Namen zuzugreifen und diese zu modifizieren.

Listing 15: NamedShapeOperation-Klasse

```
1
2  namespace SlideAssembler.Operations
3  {
4      public abstract class NamedShapeOperation<TShape>(string name) : IPresentationOperation
5          where TShape : IShape
6      {
7          public virtual void Apply(PresentationContext context)
8          {
9              foreach (var slide in context.Presentation.Slides)
10             {
11                 var shape = slide.Shapes.OfType<TShape>().FirstOrDefault(
12                 c => c.Name == name);
13
14                 if (shape is not null)
15                 {
16                     this.Apply(context, shape);
17                     return;
18                 }
19             }
20
21             if (context.ThrowOnError)
22             {
23                 throw new InvalidDataException($"'{name}' not found.");
24             }
25         }
26         protected abstract void Apply(PresentationContext context, TShape shape);
27     }
28 }
```

Die Klasse basiert auf einer generischen Struktur (`TShape`), die es erlaubt, spezifische Shape-Typen (z. B. Textboxen oder Diagramme) zu definieren. Des Weiteren wird das Interface `IPresentationOperation` (3.1.2) implementiert, um die `Apply`-Funktion bereitzustellen.

Die Methode `Apply` erhält als Parameter ein Objekt der Klasse `PresentationContext` (3.1.4). Anschließend wird jede Folie der übergebenen Präsentation durchlaufen und geprüft, ob eines der enthaltenen Shapes mit dem im Konstruktor übergebenen Namen übereinstimmt. Falls eine Übereinstimmung gefunden wird, wird die abstrakte `Apply`-Methode aufgerufen, die in einer Unterklasse implementiert werden muss und dafür gedacht ist, die Shape zu manipulieren.

3.2 Funktionen

3.2.1 FillPlaceholders

Die Klasse `FillPlaceholders` ist dafür zuständig, Textplatzhalter durch die entsprechenden Daten zu ersetzen. Dazu implementiert sie das in Kapitel 3.1.2 beschriebene Interface `IPresentationOperation`.

In folgendem Listing ist der Algorithmus, welcher das im Kapitel Formatierungsproblem in FillPlaceholders 6.1.2 beschriebene Problem löst.

Listing 16: Algorithmus für richtige Formatierung

```
1       public void Apply(PresentationContext context)
2       {
3           foreach (var slide in context.Presentation.Slides)
4           {
5               foreach (var textFrame in slide.TextFrames())
6               {
7                   foreach (var paragraph in textFrame.Paragraphs)
8                   {
9                       // if the paragraph doesn't contain a placeholder, move on
10                      if (!PlaceholderRegex().IsMatch(paragraph.Text))
11                      {
12                          continue;
13                      }
14
15                      // Otherwise, we need to combine all portions that have the same
                            formatting
16                      // because sometimes Powerpoint splits the text into multiple portions,
17                      // e.g. "*This* is a {{placeholder}}" might lead to the following
                            portions:
18
19                      // *This*
20                      // is a
21                      // {{
22                      // placeholder
23                      // }}
24
25                      var combinablePortions = new List<IParagraphPortion>();
26                      var enumerator = paragraph.Portions.GetEnumerator();
27
28                      var hasMore = enumerator.MoveNext();
29
30                      while (hasMore)
31                      {
32                          // Start with the current portion...
33
34                          combinablePortions.Add(enumerator.Current);
35
36                          // ...and collect all subsequent portions
37                          // as long as they have the same formatting:
38
39                          hasMore = enumerator.MoveNext();
40                          while (hasMore)
41                          {
```

```
42                              if (HasEqualFormatting(enumerator.Current,
                                    combinablePortions.First()))
43                              {
44                                  combinablePortions.Add(enumerator.Current);
45                                  hasMore = enumerator.MoveNext();
46                              }
47                              else
48                              {
49                                  // We found a portion with different formatting.
50                                  // Try to combine all portions we collected so far
51                                  // and replace placeholders in the combined text.
52
53                                  ReplaceCombinedText(combinablePortions);
54
55                                  // Then start again to collect portions with equal
                                        formatting.
56                                  combinablePortions.Clear();
57                                  break;
58                              }
59                          }
60                      }
61
62                      // We reached the end of the paragraph.
63                      // Combine all remaining portions and replace one final time.
64                      ReplaceCombinedText(combinablePortions);
65                  }
66              }
67          }
68
69          bool HasEqualFormatting(IParagraphPortion portion1, IParagraphPortion portion2)
70          {
71              return Compare(p => p.Font?.Color?.Type) &&
72                     Compare(p => p.Font?.Color?.Hex) &&
73                     Compare(p => p.Font?.Size) &&
74                     Compare(p => p.Font?.OffsetEffect) &&
75                     Compare(p => p.Font?.IsItalic) &&
76                     Compare(p => p.Font?.EastAsianName) &&
77                     Compare(p => p.Font?.LatinName) &&
78                     Compare(p => p.Font?.IsBold) &&
79                     Compare(p => p.Font?.Underline) &&
80                     Compare(p => p.TextHighlightColor.Hex);
81
82              bool Compare<T>(Func<IParagraphPortion, T> propertyAccessor)
83              {
84                  var value1 = SafeGetValue(portion1);
85                  var value2 = SafeGetValue(portion2);
86
87                  return value1 is null
88                      ? value2 is null
89                      : value1.Equals(value2);
90
91                  T? SafeGetValue(IParagraphPortion portion)
92                  {
93                      try
94                      {
95                          return propertyAccessor(portion);
96                      }
97                      catch
98                      {
99                          return default;
100                     }
101                 }
102             }
103         }
104
105         void ReplaceCombinedText(List<IParagraphPortion> combinablePortions)
106         {
107             var combinedText = string.Join(string.Empty, combinablePortions.Select(p =>
                    p.Text));
108             var newText = ReplacePlaceholders(combinedText, context.ThrowOnError);
109             if (newText != combinedText)
110             {
111                 combinablePortions.First().Text = newText;
112                 foreach (var portion in combinablePortions.Skip(1))
113                 {
114                     portion.Text = string.Empty;
115                 }
116             };
117         }
118     }
```

Zuerst werden alle Paragraphen in der Präsentation durchlaufen. Enthält dabei einer dieser Paragraphen eine Portion werden alle Portions mit der gleichem Formatierung mittles der Funktion `HasEqualFormatting` gesucht. Wenn es keine Portion mehr mit dem selben Format gibt wird diese zu einer zusammengesetzt und der Text wird mittels `ReplaceCombinedText` ersetzt daraufhin die nächste Portion wird gesucht und die Schritte wiederholt. Wenn das Ende des Paragraphen erreicht wurde werden alle übrig geblinen Portions kombiniert und der es wird noch einmal `ReplaceCombinedText` aufgerufen.

Zur Suche nach Platzhaltern wird eine Regular Expression verwendet, die alle Platzhalter sowie deren Formatierungsoptionen extrahiert:

Listing 17: Regular Expression zur Suche nach Platzhaltern

```
1 [GeneratedRegex(@"{{(.*?)(:(.*?))?}}", RegexOptions.None, 1000)]
2 private static partial Regex PlaceholderRegex();
```

Erklärung des Regex Musters

- `{{` – Sucht exakt nach zwei aufeinanderfolgenden geschweiften Klammern `{` (kein Escape nötig, da `{` hier kein Sonderzeichen ist).
- `(.*?)` – Eine Gruppe `()` für einen beliebigen Inhalt zwischen `{{` und einem möglichen `:`. Das `.*?` bedeutet: „Finde beliebige Zeichen (`.`), so wenige wie möglich (`*?`, lazy match)“.
- `(:` – Der Doppelpunkt `:` ist optional (da die gesamte Gruppe mit `()?` umschlossen ist). Falls er vorhanden ist, signalisiert er einen zusätzlichen Wert nach dem Platzhalternamen.
- `(.*?)` – Falls ein Doppelpunkt existiert, wird diese Gruppe verwendet, um den Wert nach dem Doppelpunkt zu erfassen.
- `)?` – Macht die vorherige Gruppe optional – das bedeutet, dass der gesamte `:(.*?)`-Teil entweder existieren kann oder nicht.
- `}}` – Sucht exakt nach zwei aufeinanderfolgenden geschweiften Klammern `}`, um das Platzhaltermuster abzuschließen.

Der Regex wird in der Methode `ReplacePlaceholders` genutzt, die dafür verantwortlich ist, die Platzhalter mit den gewünschten Daten zu ersetzen.

Listing 18: Funktion ReplacePlaceholders

```
public string ReplacePlaceholders(string text, bool throwOnError)
{
    var matches = PlaceholderRegex().Matches(text);

    foreach (Match match in matches)
    {
        var placeholder = match.Groups[1].Value;
        var format = match.Groups[3].Value;

        // Wert aus dem Datenobjekt abrufen
        var value = GetDataValue(placeholder, throwOnError);

        if (value != null)
        {
            // Falls eine Formatierung angegeben ist, anwenden
            string formattedValue;
            if (!string.IsNullOrEmpty(format) && value is IFormattable formattable)
            {
                formattedValue = formattable.ToString(format.Trim(), null);
            }
            else
            {
                formattedValue = value?.ToString() ?? string.Empty;
            }

            // Platzhalter mit dem formatierten Wert ersetzen
            text = text.Replace(match.Value, formattedValue);
        }
    }

    return text;
}
```

Die Methode erhält als Parameter einen `string text`, der ein Textsegment einer PowerPoint-Folie darstellt. Dieser wird mithilfe der Regular Expression auf Platzhalter überprüft. Falls keine Übereinstimmungen gefunden werden, bleibt der Text unverändert. Andernfalls wird der entsprechende Wert aus dem Datenobjekt mithilfe von Reflection ermittelt.

Wurde ein passender Wert gefunden, wird geprüft, ob eine Formatierungsoption angegeben ist, und diese gegebenenfalls angewandt. Schließlich wird der Platzhalter durch den formatierten Wert ersetzt, und der bearbeitete Text wird zurückgegeben.

3.2.2 FillChart

Die Klasse `FillChart` ermöglicht es, Werte in einem Diagramm zu aktualisieren. Sie erbt von der im Kapitel 3.1.5 beschriebenen abstrakten Klasse `NamedShapeOperation`, durch die die Suche nach einem Diagramm innerhalb einer Präsentation abgewickelt wird. Das Diagramm wird dann über das Interface `IChart` repräsentiert.

Listing 19: Konstruktor der FillChart-Klasse

```
1       public partial class FillChart(string name, params Series[] seriesList)
```

Der Konstruktor erwartet zunächst nicht nur den Namen des Diagrammes, sondern auch eine Liste von `Series`-Objekten. Das zuvor erwähnte `IChart`-Interface enthält eine Liste dieser `Series`-Objekte, welche jeweils eine Datenreihe in einem „Chart“ darstellen. Ein `Series`-Objekt besteht aus einem Namen und einer Menge von Werten, die die einzelnen Datenpunkte der Serie dann enthalten, wie im nachfolgenden Listing gezeigt.

Listing 20: Series-Klasse

```
1       public class Series
2       {
3           public string Name { get; set; }
4           public double[] Values { get; set; }
5       }
```

Im folgenden Listing ist zu sehen, wie in der `Apply`-Methode über alle `Series` der übergebenen Liste iteriert wird und man schaut, ob es eine gleichnamige Serie innerhalb des Diagramms gibt. Für jede gefundene `Series` wird dann eine `For`-Schleife genutzt, um die Datenpunkte in das Diagramm zu übertragen, wobei nur vorhandene Datenpunkte überschrieben werden.

Listing 21: Apply-Methode der FillChart-Klasse

```
1   protected override void Apply(PresentationContext context, IChart chart)
2   {
3       foreach (var series in seriesList)
4       {
5           var chartSeries = chart.SeriesList.FirstOrDefault(s => s.Name == series.Name);
6
7           if (chartSeries is null)
8           {
9               if (context.ThrowOnError)
10              {
11                  throw new InvalidDataException($"Series '{series.Name} in chart '{name}'
                        not found.");
12              }
13              else
14              {
15                  continue;
16              }
17          }
18
19          for (int pointIndex = 0; pointIndex < series.Values.Length; pointIndex++)
20          {
21              if (pointIndex < chartSeries.Points.Count)
22              {
23                  chartSeries.Points[pointIndex].Value = (double)series.Values[pointIndex];
24              }
25          }
26      }
27  }
```

3.2.3 ReplaceImage

Die Klasse `ReplaceImage` dient dazu, Bilder in einer PowerPoint-Präsentation zu ersetzen, was es zum Beispiel ermöglicht, individuelle Logos einzufügen.

Listing 22: Apply-Methode der ReplaceImage-Klasse

```
1  public void Apply(PresentationContext context)
2  {
3      foreach (var slide in context.Presentation.Slides)
4      {
5          foreach (var shape in slide.Shapes)
6          {
7              if (shape is IPicture)
8              {
9                  UpdateImage((IPicture)shape, context.ThrowOnError);
10             }
11         }
12     }
13 }
```

Hingegen vieler anderer Operationen ist bei `ReplaceImage` die eigentliche Änderung in der Präsentation in eine andere Methode ausgelagert. Wie oben zu sehen ist, wird lediglich immer ein Bild gesucht und anschließend an die Methode `UpdateImage` übergeben.

Listing 23: UpdateImage-Methode der ReplaceImage-Klasse

```
1  private void UpdateImage(IPicture image, bool throwOnError = false)
2  {
3      var property = data.GetType().GetProperty(image.Name);
4
5      if (property != null && typeof(Stream).IsAssignableFrom(property.PropertyType))
6      {
7      Stream stream = (Stream)property.GetValue(data);
8      image.Image.Update(stream);
9      }
10     else if (throwOnError)
11     {
12     throw new InvalidDataException("Property is null or not a stream");
13     }
14 }
```

Die `UpdateImage`-Methode ist die eigentliche Methode, welche die Bilder ersetzt. Sie sieht nach, ob in den übergebenen Daten „data“ ein `Property` mit dem selben Namen wie der hinterlegte `image.Name` im `IPicture` Bild der PowerPoint vorhanden ist. Ist ein passendes Bild in den Daten vorhanden, so wird das aktuell vorhandene Bild im `IPicture`-Objekt überschrieben wie in Zeile 8 zu sehen ist.

3.2.4 SetHeight

Mit der `SetHeight`-Operation ist es möglich, die Höhen-Eigenschaft verschiedener Shapes zu modifizieren.

Listing 24: SetHeight-Klasse

```
1  public class SetHeight : NamedShapeOperation<IShape>
2  {
3      private readonly string name;
4      private readonly decimal height;
5
6      public SetHeight(string name, decimal height) : base(name)
7      {
8          if (height < 0.0m)
9          {
10             throw new ArgumentOutOfRangeException("Height has to be >= 0", nameof(height));
11         }
12
13         this.name = name;
14         this.height = height;
15     }
16
17     protected override void Apply(PresentationContext context, IShape shape)
18     {
19         shape.Height = height;
20         if (shape == null && context.ThrowOnError)
21         {
22             throw new InvalidDataException($"Shape '{name}' not found.");
23         }
24     }
25 }
```

`SetHeight` erbt von der in Kapitel 3.1.5 beschriebenen abstrakten Klasse `NamedShapeOperation`. Dadurch entfällt die Suche nach der gewünschten Shape, und es muss lediglich die Höhe modifiziert werden. Dies geschieht durch die Überschreibung der `Apply`-Methode. Im Konstruktor der Klasse werden der Name der Shape sowie die gewünschte Höhe übergeben.

3.2.5 SetWidth

Die `SetWidth`-Operation ist sehr ähnlich zur im Abschnitt 3.2.4 beschriebenen „SetHeight-Operation“. Sie bietet die Möglichkeit, die Breite eines Objektes zu modifizieren. Die Klasse ist prinzipiell gleich aufgebaut wie die `SetHeight` Klasse, jedoch unterscheiden sie sich in der auf der nächsten Seite angeführten „Apply“-Methode:

Listing 25: Apply-Methode der SetWidth-Klasse

```
1       public void Apply(PresentationContext context)
2       {
3           var shapeFound = false;
4
5           foreach (var slide in context.Presentation.Slides)
6           {
7               var shape = slide.Shapes.FirstOrDefault(s => s.Name == name);
8               if (shape is not null)
9               {
10                  shape.Width = width;
11                  shapeFound = true;
12              }
13          }
14
15          if (!shapeFound && context.ThrowOnError)
16          {
17              throw new InvalidDataException($"Shape '{name}' not found.");
18          }
19      }
```

Im obenstehenden Code ist ersichtlich, wie hier die „shape.width“ geändert wird, was den Unterschied zur `SetHeight` Klasse macht.

3.2.6 ModifyObject

`ModifyObject` bietet die Funktionalität, alle verfügbaren Attribute einer einzelnen `Shape` anzupassen.

Listing 26: ModifyObject-Klasse

```
1   public class ModifyObject<TShape>(string name, Action<TShape> action)
2       : NamedShapeOperation<TShape>(name)
3       where TShape : IShape
4   {
5       protected override void Apply(PresentationContext context, TShape shape)
6       {
7           action(shape);
8       }
9   }
```

Dazu erbt `ModifyObject` von `NamedShapeOperation` (3.1.5), und die `Apply`-Methode wird überschrieben. In dieser Methode wird durch den Aufruf von `action(shape)` die Modifikation des Objekts durchgeführt.

Listing 27: Beispielaufruf für ModifyObject

```
1   Presentation.Load(template, true)
2       .Apply(new ModifyObject<IShape>("TestObject", o =>
3       {
4           o.TextBox.Text = "text1";
5           o.Width = 82;
6           o.Height = 40;
7       }));
```

Wie im Beispielaufruf gezeigt, ist es mittels einer Lambda-Funktion möglich, auf die verschiedenen Attribute einer Shape zuzugreifen und diese zu ändern.

4 Test und Validierung

4.0.1 Testfälle

Alle im Abschnitt 3.2 beschriebenen Funktionen wurden ausgiebig getestet und durch die im Abschnitt **??** beschriebene „Github-Action“ automatisiert nach jedem `Push` oder nach jedem `Pull-Request` erneut auf die Konsistenz ihrer Funktionalität geprüft.

Es wurden insgesamt 22 Testmethoden mit insgesamt 36 Testfällen erstellt, die in der nachstehenden Aufzählung, welche immer die Referenz zur getesteten Funktion in der Überschrift hat, mit je einer kurzen Beschreibung ersichtlich sind:

1. **FillChart** 3.2.2

    a) **`ThrowOnErrorIsFalse_ShouldNotThrowException:`** Testet die Funktionalität der `throwOnError` Implementierung, wie im Absatz 3.1.3 beschrieben, sodass kein Fehler geworfen werden sollte.

    b) **`ThrowOnErrorIsTrue_ShouldThrowException:`** Dieser Test ist ähnlich zum `ThrowOnErrorIsFalse_ShouldNotThrowException`-Test, mit dem Unterschied, dass nun die „InvalidDataException“ erwartet wird.

    c) **`NullReferencesTest_singleSeries_RegardlessOfThrowOnError:`** Hier wird getestet, ob trotzdem ein Fehler auftritt, wenn eine `Series` mit `null`-werten übergeben wird mit je einmal `throwOnError` auf `true` und einmal auf `false`. Dies sollte nicht so sein, da in mehreren Präsentationen der Fall eintreten kann, dass in dieser keine Werte vorhanden sind und das Programm aber trotzdem weiterlaufen soll.

    d) **`NullReferencesTest_MultipleSeries_RegardlessOfThrowOnError:`** Bei diesem Test wird exakt dasselbe geprüft wie beim Test zuvor, jedoch diesmal nicht mit einer `Series` mit `null` Werten sondern mit mehreren.

e) **`NullReferencesTest_NoSeriesName_ThrowOnErrorIsTrue:`** Dieser Test ähnelt den letzten beiden Tests, wobei diesmal nicht die `Series` fehlt, sondern in der `Series`-Liste die einzelnen `Series` keine Namen (`null`) haben. Hierbei sollte ebenfalls wieder kein Fehler geworfen werden, da es durchaus Wertereihen geben kann, die keine Namen haben und trotz eines „throwOnError“ nicht zu Problemen führen sollen.

f) **`NullReferencesTest_NoChartName_ThrowOnErrorIsTrue:`** Wie leicht zu erkennen ist, handelt es sich hierbei um denselben Fall wie im Test zuvor, aber nun ohne Namen für den ganzen Chart.

2. **FillPlaceHolders** 3.2.1

a) **`TestDoubleFormat_ReplacePlaceholders:`** Hier wird getestet, ob beim Ersetzen der Platzhalter die „Kultureinstellungen“ auch richtig übernommen werden. Getestet wird es mit einem '.' oder einem ',' je nach der Einstellung der jeweiligen `DataRow`.

b) **`MissingDataTest_ReplacePlaceholders:`** In diesem Test wird geprüft, ob diejenigen Platzhalter für die keine Daten vorhanden sind, trotzdem als Platzhalter bestehen bleiben, wobei `throwOnError`, wie im Absatz 3.1.3 beschrieben, `false` ist.

c) **`MissingPlaceholdersTest:`** Ob Daten, die keinen zugehörigen Platzhalter haben, einen Fehler werfen? Das wird in dieser `TestMethod` getestet, wobei es über drei `Datarows` getestet wird: einmal mit Platzhaltern, einmal ohne und noch einmal mit nur einem Teil der Platzhalter.

d) **`IgnoreMissingData_GetDataValue:`** Dieser Test ähnelt dem `MissingDataTest_ReplacePlaceholders` jedoch mit einem entscheidenden Unterschied. Denn er ersetzt keine Platzhalter direkt, sondern prüft nur die Methode, die für die Suche der Zugehörigen `values` zu den Platzhaltern zuständig ist.

e) **`NullObjectTest:`** Hier wird generell überprüft, ob der richtige Fehler geworfen wird, sollten `null`-werte als Daten übergeben werden.

f) **`NotAPowerpointTest:`** Bei diesem Test wird in die `load` Methode ein Stream übergeben welcher keine PowerPoint ist um das Verhalten bei falschen Parameter übergaben zu Testen.

3. **ModifyObjectUnitTests** 3.2.6 Da diese Funktion lediglich die `modify`-Funktion der Bibliothek „Shapecrawler“ (Abschnitt 2.9.2) benützt, wurde hier nur die korrekte Implementierung von `throwOnError` getestet, wobei die nachstehenden Testnamen den Test selbst erklären.

   a) **`ThrowOnError_IsFalse_ShouldNotThrowException`**

   b) **`ThrowOnError_IsTrue_ShouldThrowException`**

4. **SetWidthAndSetHeightUnitTests** 3.2.53.2.4

   a) **`WidthNotValidTest:`** Dieser Test überprüft die `SetWidth`-Klasse über zwei `Datarows` ungültige Werte (sämtliche negativen Werte), sodass der richtige Fehler `ArgumentOutOfRangeException` geworfen wird.

   b) **`WidthValidTest:`** Im gegensatz zu `WidthNotValidTest` wird hier genau um alle gültigen Zahlen und dass diese auch wirklich akzeptiert werden. Es werden drei `Datarows` durchlaufen, wobei 0 sowie eine normale positive reelle Zahl wie 12 als auch eine Komma-Zahl wie 1,5 getestet wird.

   c) **`HeightNotValidTest:`** Hier wird dasselbe getestet wie im `WidthNotValidTest` lediglich für die Höhe, also die Funktion `SetHeight`.

   d) **`HeightValidTest:`** Ebenfalls wird hier wie im `WidthValidTest` vorgegangen, doch für die `SetWidth`-Klasse.

5. **UnitTestsReplaceImage** 3.2.3

   a) **`ReplaceImage_WithMissingData_ThrowsException:`** In dieser `Testmethod` wird wieder die richtige Implementierung von `throwOnError` getestet, wobei hier der Fehler `InvalidDataException` geworfen wird.

   b) **`ReplaceImage_WithMissingData_IgnoresException_WhenThrowOnErrorIsFalse:`** Ebenfalls wie im vorherigem Test wird auch hier wieder auf die korrekte Verwendung von `throwOnError` geachtet wobei nun kein Fehler kommen sollte.
   `ReplaceImage_InvalidStream_ThrowsException:` Hier wird noch getestet, ob sich die Funktion auch richtig verhält, sollte eine Übereinstimmung der Namen in den Daten auftauchen, der `value` allerdings kein `Imagestream` sein.

6. **Verification-Test**
   Da diese Tests sehr essentiell für die Entwicklung dieses Projekts waren, wurden sie detaillierter in einem extra Abschnitt 4.0.2 beschrieben.

4.0.2 Verification-Test

Die `Verification`-Tests dienen dazu, sicherzustellen, dass keine ursprünglich funktionierenden Funktionen aus Versehen verloren gehen, wenn ein neues Feature implementiert wird. Diese Art von Tests wird gerne auch „Snapshot-Tests“ oder „Golden Master Tests“ genannt und sind besonders effektiv bei Projekten die visuelle Ausgaben haben.

Ablauf der Tests

Die implementierten Tests nutzen das Konzept der visuellen Regression, um sich ändernde Ausgaben zu erkennen. Dabei werden die nachfolgenden Schritte durchlaufen:

1. **Erzeugung der Präsentation:** Basierend auf einer Vorlage, die die gewünschten Objekte die getestet werden sollen, enthält, wird eine PowerPoint-Präsentation mit `Slideassemlber` nach dem im Abschnitt 2.5 beschriebenem `Fluent API-Konzept` erstellt.

Listing 28: Beispielgenerierung einer Powerpoint der TestAllFeatures-Methode

```
Series[] seriesList = [new Series("Datenreihe 1", values),
new Series("Datenreihe 2", values.Select(v => v * 2).ToArray())];
            SlideAssembler.SlideAssembler.Load(template)
                          .Apply(new FillPlaceholders(data)).Apply(new
                              FillChart("MesswertDiagramm", new Series("Werte",
                              values)))
                          .Apply(new SetWidth("MittelwertRechteck",
                              (Decimal)data.Mittelwert))
                          .Apply(new SetWidth("MaximumRechteck",
                              (Decimal)data.Maximum))
                          .Apply(new SetWidth("MinimumRechteck",
                              (Decimal)data.Minimum))
                          .Apply(new FillChart("LineChart", seriesList))
                          .Apply(new ModifyObject("MittelwertRechteck", o =>
                          {
                              o.TextFrame.Text = data.Mittelwert.ToString("N2");
                              o.Width = (Decimal)data.Mittelwert;
                              o.Height = 10;
                          })).Apply(new ReplaceImage(data))
                          .Save(stream);
```

2. **Rendering der Folien:** Anschließend wird für jede Folie eine `PNG`-Bilddatei gerendert und in das angegebene Verzeichnis, hier 4.0.2, temporär zur Laufzeit des Tests gespeichert.

Listing 29: Rendering der Folien zu PNG-Bilddatein

```
using var presentation = Syncfusion.Presentation.Presentation.Open(stream);
       presentation.PresentationRenderer = new PresentationRenderer();
       var slides =
           presentation.RenderAsImages(Syncfusion.Presentation.ExportImageFormat.Png);
```

3. **Vergleich mit Referenzbildern:** Die nun generierten `PNG`-Bilder werden anschließend mit den bereits verifizierten Bildern, wie in 4.0.2 beschrieben, verglichen.

Listing 30: Verifizieren der Folien

```
1  for (int i = 0; i < slides.Length; i++)
2  {
3          await Verify(slides[i], "png")
4              .UseDirectory(verificationDirectory)
5              .UseFileName("slide-" + (i + 1));
6  }
```

In dem oben angeführten Code ist der Aufbau eines `Verify`-Aufrufs gut ersichtlich. Dieser basiert ebenso auf dem im Abschnitt 2.5 beschriebenen Fluent API-Konzept

4. **Validierung:** Sollten Unterschiede zu den Referenzbildern vorliegen, so wird durch diese potenziellen Fehler oder unbeabsichtigten Änderungen ein Fehler geworfen. Dies wird allerdings automatisch durch die `Verify`-Methode der im Abschnitt 2.9.4 beschriebenen `VerifyBase`-Klasse durchgeführt.

verificationfiles-Ordner

Der Ordner „verfication files“ ist ebenfalls im Testprojekt enthalten und ist ein fester Bestandteil der `Verify`-Tests. Dieser Ordner muss am Anfang der `VerifyTests`-Klasse angegeben werden, wie in folgendem Code gezeigt:

Listing 31: Einbindung verificationfiles-Ordner

```
1  [TestClass]
2  public class VerifyTests : VerifyBase
3  {
4      private const string verificationDirectory = "verificationfiles"; //directory, where
           the template and all verification slide snapshots are located. (directory will be
           created automatically in the first run)
```

Wie auch im nebenstehenden Kommentar schon herauszulesen ist, ist dies der Ordner, von dem die Vorlagen und die bereits verifizierten Bilddateien bezogen werden zum „Snapshot“-testen.

Die verifizierten Bilddateien sind an der Dateinamenserweiterung „.verified“ zu erkennen. Beim ersten Durchlauf der Tests muss der Entwickler diese Dateien selbst verifizieren und sie als „.verified“ markieren. Die „PNG“-Bilddateien bekommt er von den `VerifyTests` erstmalig mit dem Suffix „.received“ und er muss jede Folie einmal verifizieren, bevor ein Verify-Test komplett durchlaufen werden kann.

Implementierung

In diesem Projekt wurden zwei verschiedene `Verification`-Tests implementiert:

- **`TestDifferentFormatting:`**

  In diesem Test wird das Ersetzen der Platzhalter mit verschiedenen Datenformaten getestet.

Listing 32: TestDifferentFormatting-Methode

```
1  [TestMethod]
2  public async Task TestDifferentFormatting()
3  {
4      CultureInfo.CurrentCulture = CultureInfo.CurrentUICulture = new
           CultureInfo("de-AT");
5
6      using var stream = new MemoryStream();
7      using var template = File.OpenRead(Path.Combine(verificationDirectory,
           "Template_different_formating.pptx"));
8
9      var data = new
10     {
11         Titel = "TestPresentation",
12         Projekt = "Slideassembler",
13         Datum = "31.10.2024",
14         Data1 = new
15         {
16             Headline = "Data 1",
17             Date = "1.1.2001",
18             Value = 20,
19             Time = "17:30"
20         },
21         Data2 = new
22         {
23             Headline = "Data 2",
24             Date = "2.1.2001",
25             Value = 19,
26             Time = "18:00"
27         }
28     };
29
30     Presentation.Load(template)
31                 .Apply(new FillPlaceholders(data))
32                 .Save(stream);
33
34     stream.Position = 0;
35
36     await VerifyAllSlidesAsync(stream);
37 }
```

Wie oben im Code zu sehen ist, ist der Aufbau ziemlich einfach. Zunächst wird die „CultureInfo.CurrentCulture“ gesetzt, sodass die Datums-, Währungs- und Zeitformate einheitlich bleiben, egal wo dieser Test ausgeführt wird. Anschließend wird ein Objekt „Data“ mit Testdaten befüllt und eine Präsentation erstellt.

Beim Erstellen der Präsentation wird lediglich die im Abschnitt 3.2.1 beschriebene Funktion `FillPlacehodlers` verwendet, da dort die Formatierungsmöglichkeiten in der Vorlage, wie auch im zuvor erwähnten Abschnitt beschrieben, bestehen.

- **`TestAllFeatures:`**

  Hier werden alle im Absatz 3.2 beschriebenen Funktionen auf ihre korrekte Ausgabe überprüft.

Listing 33: TestAllFeatures-Methode

```
1  [TestMethod]
2  public async Task TestAllFeatures()
3  {
4      CultureInfo.CurrentCulture = CultureInfo.CurrentUICulture = new
           CultureInfo("de-AT");
5
6      using var stream = new MemoryStream();
7
8      using var template = File.OpenRead(Path.Combine(verificationDirectory,
           "Template_all_features.pptx"));
9      {
10         var values = new[] { 82d, 88d, 64d, 79d, 31d };
11
12         var data = new
13         {
14             Titel = $"Messwerte vom {new DateTime(2024, 8, 7):d}",
15             Benutzer = new
16             {
17                 Vorname = "Max",
18                 Nachname = "Mustermann"
19             },
20             Minimum = values.Min(),
21             Mittelwert = values.Average(),
22             Maximum = values.Max(),
23             Werte = values
24         };
25
26         var seriesList = new[]
27         {
28             new Series("Datenreihe 1", values),
29             new Series("Datenreihe 2", values.Select(v => v * 2).ToArray())
30         };
```

Auch in dieser Testmethode wird gleich am Anfang wie in `TestDifferentFormatting` die Variable `cultureInfo.CurrentCulture` gesetzt, um Formatierungsfehler unabhängig vom Testrechner oder den aktuellen Ortseinstellungen zu vermeiden. Anschließend wird noch ein „stream“ für die Ausgabe erstellt und das Template von den im Absatz 4.0.2 beschriebenen „verificationfiles“ geladen.

Schließlich, vor der Erstellung der Präsentation, werden die Testdaten wie auch in der vorherigen Methode erstellt, lediglich mit mehr Differenz zwischen den Daten, da nun auch `Series` für die Diagramme zur Verfügung stehen wie im `data`-Objekt die `Werte = values`.

Die Erstellung der Präsentation ist im vorherigen Code-listing 28 einzusehen und wurde somit hier nicht mehr in das oben ersichtliche Code-listing übernommen.

4.0.3 Github-Action

Die grundlegende Funktion von „Github-Actions“ wurde bereits im Abschnitt 2.6.2 erklärt. Die in diesem Projekt erstellte „Github-Action“ ist zuständig für die automatisierte Testung des Codes, um unbeabsichtigte menschliche Fehler während der laufenden Implementierung frühzeitig zu erkennen, um so größere Schäden zu vermeiden.

Die „Github-Action“ ist in drei Hauptteile unterteilt, die nachfolgend beschrieben werden:

1. **Definierte Umgebungsvariablen**

Listing 34: Github-Action: Definierte Umgebungsvariablen

```
1       env:
2           SyncfusionLicenseKey: ${{ secrets.SYNCFUSIONLICENSEKEY }}
```

„Github-Actions“ erlauben es, Umgebungsvariablen zu definieren, die innerhalb eines Workflows verwendet werden können. In diesem Fall wird die Variable `SyncfusionLicenseKey` definiert. Diese enthält einen Lizenzschlüssel, der für die Nutzung von Syncfusion, wie im Absatz 2.9.3 beschrieben, erforderlich ist.

2. **Trigger der Action**

Listing 35: Github-Action: Trigger der Github-Action

```
1       on:
2           push:
3               branches: [ "main" ]
4           pull_request:
5               branches: [ "main" ]
6           workflow_dispatch:
```

Die „Action“ wird durch verschiedene vordefinierte Ereignisse ausgelöst. Sie startet automatisch, wenn Änderungen auf den `main`-Branch „gepusht“ oder ein „Pull-Request“ gegen diesen Branch ausgeführt wird. Zusätzlich kann sie manuell über das „GitHub-Interface“ mit `workflow_dispatch` gestartet werden, sollte die manuelle Validierung über Github einmal erforderlich sein. Diese Konfiguration stellt sicher, dass jede relevante Codeänderung getestet wird, bevor sie in den `main`-Branch integriert wird.

3. **Job-Definition**

Listing 36: Github-Action: Job-Definition der Github-Action

```
jobs:
    build:
        runs-on: windows-latest
        steps:
            - uses: actions/checkout@v4
```

Der Workflow besteht aus einem einzelnen Job namens `build`, der auf einem temporär erstellten Windows-Container ausgeführt wird. Der erste Schritt in diesem Job nutzt die GitHub Action `actions/checkout`, um den aktuellsten Codestand aus dem Repository nach Ausführen einer der zuvor beschriebenen Aktionen wie „push“ oder „pull“, zu klonen. Dies stellt sicher, dass der neueste Stand des Codes für die weiteren Schritte verfügbar ist.

4. **Einrichtung und Build-Prozess**

Listing 37: Github-Action: Einrichtung und Build-Prozess

```
      - name: Setup .NET
        uses: actions/setup-dotnet@v4
        with:
            dotnet-version: 8.0.x
      - name: Restore dependencies
        run: dotnet restore
      - name: Build
        run: dotnet build --no-restore -c Release
```

Um das Projekt erfolgreich zu kompilieren, muss eine „.NET“-Umgebung eingerichtet werden. Dazu wird die `setup-dotnet` Action verwendet, die die erforderliche Version der „.NET“-Umgebung bereitstellt (hier verwendeten wir die „.NET“-Version 8.0.x, wobei das x auf die aktuellste Version schließt). Anschließend werden die Abhängigkeiten des Projekts mithilfe von `dotnet restore` geladen woraufhin gleich danach mit `dotnet build` der eigentliche „Build“ erfolgt.

5. **Testausführung**

Listing 38: Github-Action: Testausführung

```
      - name: Test
        run: dotnet test --no-build -c Release --verbosity normal
```

Nach dem erfolgreichen „Build“-Prozess folgt die Testphase. Hierbei werden alle definierten „Unit“-Tests, die im Abschnitt 4.0.1 aufgelistet sind, durchgeführt. Der Parameter `-no-build` verhindert, dass das Projekt erneut kompiliert wird, da dies bereits im vorherigen Schritt 4 erfolgt ist.

6. Artefakt-Uploads

Listing 39: Github-Action: Artefakt-Uploads

```
            - name: Upload verification files
              if: success() || failure()
              uses: actions/upload-artifact@v4.3.6
              with:
                  name: verification-files
                  path: "D:\\a\\slide-assembler\\....\\*.png"
            - name: Upload nuget package
              uses: actions/upload-artifact@v4.3.6
              with:
                  name: nuget-package
                  path: "D:\\a\\slide-assembler\\....\\*.nupkg"
```

Der letzte Schritt des Workflows besteht aus dem Upload von Artefakten, also einer Art von Dateipaketen mit Dateien, welche während des Workflows erstellt wurden. Hierbei wurden einerseits alle Dateien des im Abschnitt 4.0.2 beschriebenen Ordners „Verificationfiles“ hochgeladen, um im Nachhinein Einsicht auf die Nicht-Übereinstimmungen zwischen den erstellten und den verifizierten Bildern zu haben. Außerdem wird das NuGet-Paket vom Projektstand noch mitgespeichert. Dieser Schritt stellt letztendlich sicher, dass alle relevanten Dateien nach dem „Build“- und Testprozess verfügbar bleiben, unabhängig davon, ob der Workflow erfolgreich war oder fehlgeschlagen ist.

5 Ergebnis

Das zentrale Ergebnis dieser Diplomarbeit ist die Entwicklung der Bibliothek SlideAssembler, die alle in Kapitel 3.2 beschriebenen Funktionen umfasst. Diese Bibliothek ermöglicht es, innerhalb kürzester Zeit und mit minimalem Aufwand aus einer vordefinierten Vorlage eine vollständig generierte PowerPoint-Präsentation zu erstellen. Dank der intuitiven Fluent API können Entwickler die Präsentationen flexibel und effizient befüllen, ohne sich tiefgehend mit der zugrunde liegenden PowerPoint-Struktur auseinandersetzen zu müssen.

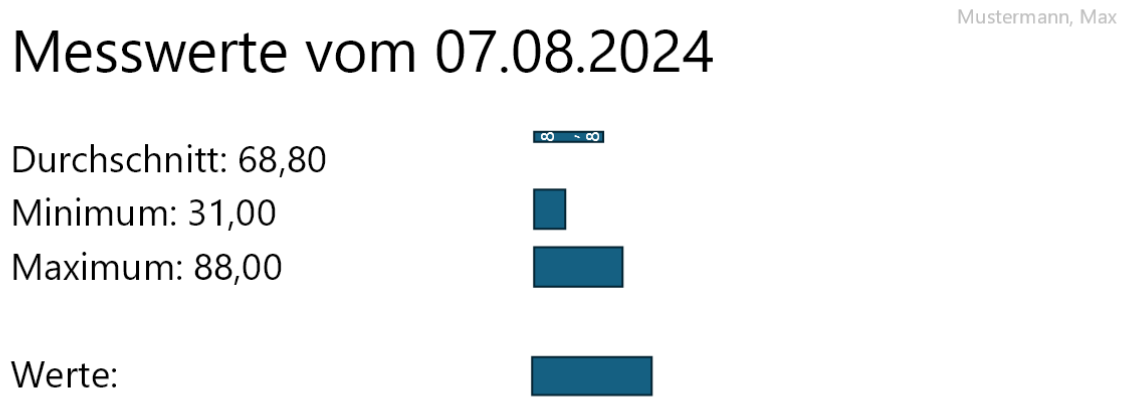

Abbildung 4: Generierte Präsentation aus Tempalte

In der obigen Abbildung ist ein Beispiel für eine fertig generierte Präsentation, welche auf dem Template in Abbildung 3 basiert

Um die Nutzung und Verbreitung der Bibliothek zu vereinfachen, wurde SlideAssembler als NuGet-Paket bereitgestellt. Dadurch lässt es sich mit wenigen Klicks in verschiedenste .NET-Projekte integrieren und sofort produktiv einsetzen.

Ein weiterer wichtiger Meilenstein in diesem Projekt war die Entscheidung, SlideAssembler als Open-Source-Software zu veröffentlichen. Diese Entscheidung fiel während der Implementierungs-

phase, um eine aktive Weiterentwicklung und Verbesserung durch die Entwickler-Community zu ermöglichen. Durch den offenen Quellcode können neue Features ergänzt, bestehende Funktionen optimiert und potenzielle Fehler schneller behoben werden. Dies fördert nicht nur die langfristige Wartbarkeit und Skalierbarkeit der Bibliothek, sondern bietet auch anderen Entwicklern die Möglichkeit, von den gewonnenen Erkenntnissen und Lösungen zu profitieren.

Mit SlideAssembler steht somit eine leistungsstarke und leicht erweiterbare Lösung zur Verfügung, die das automatisierte Erstellen von PowerPoint-Präsentationen revolutioniert und sowohl für den Einsatz in Unternehmen als auch in individuellen Projekten einen erheblichen Mehrwert bietet.

6 Resümee

Durch das im Zuge des Projektunterrichts erworbene Wissen stellten diverse Planungsaufgaben wie das Definieren von Meilensteinen, die Aufgabenverteilung und die Zeitplanung keine großen Herausforderungen dar, was ein effizientes und schnelles Arbeiten ermöglichte.

Während der Implementierung der Diplomarbeit wurde der Entwicklungsstand durch regelmäßige Meetings überprüft sowie verschiedene Probleme besprochen und anschließend gelöst. Zudem konnte viel neues Wissen in folgenden Bereichen erlangt werden:

- C#
- Office-OpenXml 2.8.1
- CI/CD
- GitHub Actions

Zusammenfassend kann gesagt werden, dass die Diplomarbeit erfolgreich abgeschlossen wurde und durch die Zusammenarbeit mit der Software GmbH ein umfangreicher Einblick in die Berufswelt als Softwareentwickler möglich war.

6.1 Herausforderungen und Probleme

Während der Implementierung der Diplomarbeit kam es gelegentlich zu Problemen und verschiedenen Herausforderungen, welche in diesem Kapitel beschrieben werden.

6.1.1 Einschränkungen von ShapeCrawler

Dynamische Charts

Ein Problem, das sich während der Implementierung der Diplomarbeit herauskristallisierte, ist, dass ShapeCrawler über einige essenzielle Funktionen nicht verfügt. Eine nicht vorhandene Funktion ist das dynamische Erweitern von Charts. Charts müssen bereits alle Datenpunkte enthalten, damit sie korrekt mit den richtigen Werten aus dem `data`-Objekt befüllt werden können. Sind zu wenige Platzhalter in einer Serie im Chart vorhanden, so werden Daten

ausgelassen; sind zu viele vorhanden, bleiben die überschüssigen Einträge unverändert. Daher muss der Benutzer stets sicherstellen, dass die Serien genau so viele Datenpunkte enthalten wie das übergebene Serien-Objekt (3.2.2).

SmartArt

Eine weitere Einschränkung besteht darin, dass es mit ShapeCrawler nicht möglich ist, SmartArt-Grafiken zu manipulieren, da diese von der Bibliothek nicht unterstützt werden. Allerdings ist zu beachten, dass diese einen sehr speziellen Anwendungsfall darstellen und durch gewöhnliche Formen ersetzt werden können. Zudem können SmartArt-Grafiken zwar in Templates verwendet werden, jedoch nicht als Platzhalter.

6.1.2 Formatierungsproblem in FillPlaceHolders

Ein Problem, das bei der Nutzung von `FillPlaceHolders` auftrat, war die fehlerhafte Behandlung unterschiedlicher Formatierungen (z. B. verschiedene Schriftarten) innerhalb eines Textabschnitts. Beim Ersetzen eines Platzhalters durch Daten wurden diese unterschiedlichen Formatierungen nicht berücksichtigt. Dies liegt daran, dass PowerPoint Texte innerhalb eines Paragraphs in mehrere *Portions* aufteilt, insbesondere wenn unterschiedliche Formatierungen vorhanden sind. Das bedeutet, dass ein Platzhalter wie `{Name}` durch Formatierungsbrüche in mehrere *Portions* zerlegt sein kann. Beispiel:

Dies ist ein `{Platzhalter}`

→ könnte intern als folgende *Portions* gespeichert sein:

Dies

ist ein

{{

Platzhalter

}}

Das Problem ist, dass ein einfacher Textersatz nicht funktioniert, da die Platzhalter über mehrere *Portions* verteilt sind. Würde man nur einzelne *Portions* ersetzen, könnte der Platzhalter unvollständig ersetzt oder die Formatierung verloren gehen. Als Lösung wurde ein eigener Algorithmus implementiert, welcher im Kapitel 3.2.1 beschrieben wird.

7 Ausblick

Obwohl die Diplomarbeit bereits abgeschlossen ist, gibt es weiterhin Verbesserungspotenzial sowie neue Features, die in Zukunft für SlideAssembler entwickelt werden könnten. Aufgrund zeitlicher Beschränkungen konnten diese bislang nicht umgesetzt werden.

7.0.1 Dynamische Charts

Eine bedeutende Erweiterung für SlideAssembler wäre die Implementierung einer dynamischen Erstellung von Diagrammen. Dies würde die Benutzerfreundlichkeit der Bibliothek erheblich steigern. Allerdings müsste hierfür entweder eine völlig neue Bibliothek anstelle von ShapeCrawler eingesetzt oder ein direkter Zugriff auf das XML-Schema der PowerPoint-Datei realisiert werden. Beide Ansätze wären jedoch mit erheblichem Entwicklungsaufwand verbunden.

7.0.2 SmartArt

Ein weiteres Feature wäre die Unterstützung von SmartArt-Grafiken, da diese von ShapeCrawler derzeit nicht verarbeitet werden können. Allerdings ist der praktische Nutzen dieser Funktionalität eher gering, da diese durch herkömmliche Objekte relativ einfach ersetzt werden können.

7.0.3 Integration von Audio- und Videodateien

Eine zusätzliche Erweiterung wäre die Unterstützung von Audio- und Videodateien in Präsentationen. Dazu könnte ein Platzhalter-Video in das Template eingefügt und korrekt benannt werden. Die gewünschte Mediendatei könnte dann in der finalen Präsentation per Stream bereitgestellt werden. Diese Operation wäre vergleichbar mit der bestehenden `ReplaceImage`-Funktion und würde daher nur einen moderaten Implementierungsaufwand erfordern. Zudem unterstützt ShapeCrawler bereits die Manipulation von Audio- und Videodateien, was die Integration weiter erleichtern würde.

Glossar

CD Continuous Deployment

CI Continuous Integration

CI/CD Continuous Integration / Continuous Deployment, kombiniert

Fluent API Fluent Application Programming Interface

Git Versionskontrollsystem

NuGet Paketmanager für .NET

Reflection Laufzeitanalyse und Modifikation von Code in .NET

Regex Regular Expressions

Literaturverzeichnis

[1] Bernhard Baltes-Götz, „Einführung in das Programmieren mit C 6," 2017. Online verfügbar: https://www.uni-trier.de/fileadmin/urt/doku/csharp/v60/csharp6.pdf

[2] Stefan Schultz (9,9%), „.NET-Framework," 2025. Online verfügbar: https://de.wikipedia.org/wiki/.Net-Framework

[3] Mshobohm (51,6%), „.NET (Plattform)," 2025. Online verfügbar: https://de.wikipedia.org/wiki/.NET_%28Plattform%29

[4] . Shaddim(22, „Programmbibliothek," 2025. Online verfügbar: https://de.wikipedia.org/wiki/Programmbibliothek

[5] . JonDouglas, alexbuckgit, „Eine Einführung in NuGet," *learn.microsoft.com*, 2023. Online verfügbar: https://learn.microsoft.com/de-de/nuget/what-is-nuget

[6] Thimo Buchheister, „Fluent APIs," 2025. Online verfügbar: https://www.thimobuchheister.com/2025-01-22-create-a-fluent-api-client/

[7] Philipp Cordes (50,1%), „Fluent Interface," 2021. Online verfügbar: https://de.wikipedia.org/wiki/Fluent_Interface

[8] , „Was ist die Versionskontrolle von Git?" *about.gitlab.com*, 2025. Online verfügbar: https://about.gitlab.com/de-de/topics/version-control/what-is-git-version-control/

[9] PaulAsimov (9,2%), „GitHub," 2025. Online verfügbar: https://de.wikipedia.org/wiki/GitHub

[10] L. Mooney, „Continuous Integration with GitHub Actions," *endjin*, 2022. Online verfügbar: https://endjin.com/blog/2022/09/continuous-integration-with-github-actions

[11] -, „About continuous deployment with GitHub Actions," 2025. Online verfügbar: https://docs.github.com/en/actions/about-github-actions/about-continuous-deployment-with-github-actions

[12] digitales-institut, „Die Grundlagen von Notion," 2023. Online verfügbar: https://digitales-institut.de/was-ist-notion/

[13] . Existhigh(17, „Office Open XML," 2025. Online verfügbar: https://de.wikipedia.org/wiki/Office_Open_XML

[14] . o365devx, mikeebowen, „Welcome to the Open XML SDK for Office," *learn.microsoft.com*. Online verfügbar: https://learn.microsoft.com/en-us/office/open-xml/open-xml-sdk

[15] A. Shakhabov, „ShapeCrawler," *github.com/ShapeCrawler*. Online verfügbar: https://github.com/ShapeCrawler/ShapeCrawler

[16] K. Abuhakmeh, „Snapshot Testing in .NET with Verify," *Jetbrains Blog*, 2024. Online verfügbar: https://blog.jetbrains.com/dotnet/2024/07/11/snapshot-testing-in-net-with-verify/

[17] -, „Verify," 2025. Online verfügbar: https://github.com/VerifyTests/Verify

[18] Swiss Life, „A snapshot test utility to simplify your tests," 2021. Online verfügbar: https://swisslife-oss.github.io/snapshooter/

[19] . adegeo, BillWagner, „.NET regular expressions," *learn.microsoft.com*. Online verfügbar: https://learn.microsoft.com/en-us/dotnet/standard/base-types/regular-expressions

[20] g. dotnet bot, „Reflektion in .NET," *learn.microsoft.com*. Online verfügbar: https://learn.microsoft.com/de-de/dotnet/fundamentals/reflection/reflection

Abbildungsverzeichnis

Quellcodeverzeichnis

Anhang

A Projektteam

Dieses Projekt wurde von den folgenden zwei Schülern mit Hilfe ihres Betreuers mit einer klaren Aufteilung der Rollen und Aufgaben entwickelt:

A.1 Schüler

- **Peitl Stefan** - Verantwortlich für die Entwicklung von Tests, CI/CD sowie auch mithilfe bei der Implementierung der einzelnen Funktionen für die Library
- **Kunnert Philipp** - Hauptverantwortlich für die Implementierung der Bibliothek sowie der Schnittstelle zu Powerpoint

A.2 Betreuer

Der Betreuer dieser Diplomarbeit war Dipl.-Ing. Michael Stumpfl, welcher uns mit seinem technischem Know-how in der Softwarentwicklung unterstützte.

B Meilensteine

- **13.06.2024** Kickoff
- **20.06.2024** Aufsetzen der Entwicklungsumgebung / Zeitaufzeichnung / Git / etc.
- **30.06.2024** Schnittstelle zur PowerPoint
- **13.07.2024** Manipulation von Powerpoints/Templates
- **20.08.2024** Generierung von Powerpoints aus Daten (in Templates)
- **10.09.2024** Formatierung der Daten in die PP-Präsentation vollständig implementiert
- **29.09.2024** Tests vollständig abgewickelt
- **24.12.2024** Inhaltsverzeichnis der schriftlichen Ausarbeitung steht
- **31.03.2025** Dokumentation des Projektes ist vollständig

C Dateien

C.1 Logo

Abbildung 5: Logo von Slideassembler

C.2 Plakat

Abbildung 6: Diplomarbeitsplakat von Slideassembler