Institut für Softwaretechnik

UNIVERSITÄT KOBLENZ · LANDAU
Fachbereich 4: Informatik

Foils

Wartung und Weiterentwicklung eines Systems zur Präsentation von Lehrinhalten mittels eines Tablet PC Systems

Studienarbeit
im Studiengang Computervisualistik

vorgelegt von:
Eckhard Großmann
Eckhard.Grossmann@uni-koblenz.de

Betreuer: Prof. Dr. Jürgen Ebert, Institut für Softwaretechnik, Fachbereich 4
Dr. Volker Riediger, Institut für Softwaretechnik, Fachbereich 4

Koblenz, im Februar 2009

Erklärung

Ich versichere, dass ich die vorliegende Arbeit selbständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel benutzt habe.

Mit der Einstellung dieser Arbeit in die Bibliothek bin ich einverstanden.

Der Veröffentlichung dieser Arbeit im Internet stimme ich zu.

Koblenz, den . . . . . .

Danksagung

An dieser Stelle will ich meinen Betreuern Prof. Dr Jürgen Ebert und Dr. Volker Riediger herzlich danken, dass sie mir die Chance gegeben haben eine so tolle und interessante Arbeit durchzuführen. Sie standen mir immer mit Rat zur Seite und haben mich während der Studienarbeit sehr gut betreut.

Ich will auch Elisa Ziegler danken. Sie hat mich unermütlich auf Schwächen in meinen Texten hingewiesen. Selbst wenn eine Korrektur die ganze Nacht gedauert hat, hat es sie nicht davon abgehalten mir zu helfen. Außerdem hatte sie immer ein offenes Ohr für meine fachlichen Probleme, die mich gerade beschäftigten.

Einen Dank möchte ich auch an meinen Freund Christian Fuchs aussprechen, der mir öfters bei Problemen mit .Net weiterhelfen konnte.

Ein letzter Dank geht an meine Eltern, die in der letzten Phase dieser Arbeit mir geholfen haben die Texte noch verständlicher zu gestalten.

Inhaltsverzeichnis

1. Einleitung **1**

1.1. Foils - Ein kurzer Einblick 1

1.1.1. Möglichkeiten im Programm Foils 2

1.2. Grobe Zeitphasen 3

1.3. Übersicht 4

2. C# und .Net **5**

2.1. C# 5

2.1.1. Ein paar Unterschiede zu Java 5

2.2. .Net 6

2.2.1. Windows Presentation Foundation 6

2.3. Speicherverwaltung in C# 7

2.3.1. Destruktoren 7

2.3.2. Garbage Collector 8

2.3.3. Freigabemethoden und das `IDisposable`-Interface 10

2.3.4. Die **`using`**-Anweisung 12

2.3.5. Zusammenfassung und Schlussfolgerung 12

3. Nähere Betrachtung von Foils **13**

3.1. Ein Blick auf die alte Version von Foils 13

3.1.1. Die Folien und die Zeichenfläche 13

3.1.2. Kommando-Objekt 14

3.1.3. Der Undo-Stack 15

3.1.4. InkChangeOperation 16

3.2. Ein Blick auf die neue Version von Foils 17

3.2.1. Eine neue Zeichenfläche aus der WPF 17

3.2.2. Die `SlideCanvas`-Klasse 20

3.3. Zusammenfassung 22

4. Analyse und Korrektur des Speicherverbrauchs **23**

4.1. `Replay()`-Funktion 23

4.2. Messungen und deren Auswertung 25

4.2.1. Originalversion Foils 26

4.2.2. Auswirkungen eines Garbage Collector-Aufrufs 28
4.2.3. Optimierung des GC-Aufrufs 29
4.2.4. InkChangeOperation - eine genauere Betrachtung 29
4.2.5. Demonstration der Auswirkungen ohne Skalierung 30
4.2.6. Neue Implementation 31
4.3. Schlussfolgerung aus den Messergebnissen 32

5. Genaue Fehler und Lösungen **35**
5.1. Navigation über das Drop-Down-Menü 35
5.2. Das Verhalten der Präsentationsansichten 35
5.2.1. Einfügen einer `Slide` 36
5.2.2. Darzustellende Folien in den Präsentationsansichten 36
5.3. Die Lasso-Funktion 38
5.4. Merken der zuletzt bearbeiteten Folie 39
5.5. Öffnen aus einer bestehenden Präsentation 41
5.6. Fenstersprünge bei der Benutzung von Undo/Redo 41
5.7. Gemeinsames Maximieren und Minimieren aller Fenster 42
5.8. Der Taskleisten-Eintrag 42
5.9. Fehlende Überlagerungen beim Export von Bildern 42
5.10. Neuer Export-Dialog 44
5.10.1. Maskenoptionen 45
5.11. Skalierung der Zeichenfläche 46
5.11.1. Skalierungsmöglichkeiten 47
5.11.2. Vorüberlegungen zum `ScalingModule` 47
5.11.3. Einbindung des `ScalingModule` in Foils 48

6. Umsetzung der Change-Requests **49**
6.1. Kleinere Änderungen an Foils 49
6.2. Bilder in Foils 49
6.2.1. Darstellung eines Bildes mittels der `InkCanvas` 50
6.2.2. Größenverhältnisse beim Einfügen eines Bildes 50
6.2.3. Darstellungmöglichkeiten der `Image`-Klasse 51
6.2.4. Seitenverhältnisse bei einem eingefügten Bild 51
6.3. Neues Design des Bearbeitungsfensters 52
6.3.1. Die Werkzeugleiste 53
6.3.2. Die Bilderauswahlleiste 56
6.3.3. Ungenutzter Bereich im Bearbeitungsfenster 56
6.3.4. Doppelte Belegung von Buttons 56
6.3.5. Die Scrollbuttons und die Demofunktion 57
6.3.6. Verwendung von klareren Symbolen 57
6.3.7. Anordnung der Symbole in der Werkzeugleiste 57

6.4. Die Navigationsleiste 57
6.4.1. Umsetzung der Navigationsleiste 58
6.5. Der Wizarddialog 59
6.5.1. Der neue Wizard-Dialog 59

7. Umsetzung der Extension-Requests **63**
7.1. Programmverknüpfung 63
7.2. Bilder während des Betriebs importieren 63
7.3. Autosave von Präsentationen 64
7.4. Präsentationverlust beim Speichern 65
7.5. Benutzerdefinierte Einstellungen 65
7.5.1. Umsetzung durch die Registry 66
7.5.2. Einstellungsdialog 70
7.6. Gelöschte Folien wiederherstellen 73
7.7. Erweiterungen der Navigationsleiste 74
7.8. Einbindung mehrerer Dateiformate 74
7.9. Zusammenfassung 74

8. Ausblick **75**
8.1. Vorschaufolien in der Navigationsleiste 75
8.2. PDF in Foils 75
8.3. Einfügen von Quellcode 75
8.4. Exportierung einer Präsentation 76
8.5. Seitenverhältnis von Bildern 76
8.6. Bilder zuschneiden 76
8.7. Fortsetzen und Zusammenführen mehrere Präsentationen 77
8.8. Zusammenfassung 77

9. Zusammenfassung **79**

A. Änderungsanforderungen **81**
A.1. Bugreport 81
A.2. Change-Requests 83
A.3. Extension-Requests 85
A.4. Abbildungen 87

B. Mängelliste **89**

C. Fragenkatalog **91**
C.1. Fragen zum Erscheinungsbild 91
C.2. Generelle Fragen 91
C.3. Fragen zu bestehenden Mängeln 92

C.4. Fragen zur Softwareentwicklung 92

D. Diagramme zum Speicher- und Zeitverbrauch **93**

E. Baustelle Foils **99**
E.1. Wegweiser in der Projektstruktur 99
E.2. Die Replay-Funktion 100

F. Inhalt der CD **101**

Abbildungsverzeichnis **104**

Tabellenverzeichnis **105**

Listingverzeichnis **107**

1. Einleitung

Diese Studienarbeit baut auf der Arbeit von Tim Steffens [Ste05] auf. Bei seiner Studienarbeit handelt es sich um ein System zur einfachen Präsentation handschriftlicher Lehrinhalte mittels eines Tablet PCs und eines Beamers. Im Wesentlichen wird das Beschreiben von Folien und deren gleichzeitige Projektion mit einem Overheadprojektor ersetzt. Das Programm, welches aus der Studienarbeit Tim Steffens hervorgegangen ist, enthält Mängel in der Programmierung und im Entwurf aus softwaretechnischer und -ergonomischer Sicht. Diese Mängel reichen von ungünstig gewählten Schaltflächen über ein immer langsamer werdendes System bis hin zu Abstürzen während des laufenden Betriebs. Meine Studienarbeit soll dieses System genauer analysieren, bestehende Fehler korrigieren und gleichzeitig das gesamte System nach neuen Anforderungen umgestalten.

Die Betreuer sind in diesem Fall Herr Prof. Dr. Jürgen Ebert und Herr Dr. Volker Riediger.

1.1. Foils - Ein kurzer Einblick

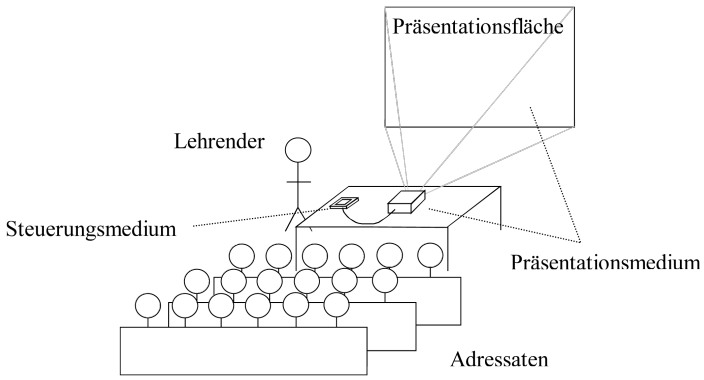

Abbildung 1.1.: Anwendungsszenario für Foils.

Was ist Foils? Foils ist das Programm, welches aus der Studienarbeit [Ste05] von Tim Steffens hervorgegangen ist. Sein Programm wurde für die Lehre entwickelt. Es zielt auf Lehrpersonal ab, das intensiv den Overheadprojektor als Präsentationsmittel einsetzt und mit Hilfe von Folien und Stiften seine Lehrinhalte vorträgt.

Tim Steffens Idee war, ein Programm zu schaffen, dass dem Lehrpersonal entgegenkommt und das Halten einer Vorlesung vereinfacht. Die Abb. 1.1 stellt das typische Anwendungsszenario dar und ist eine original Skizze aus seiner Studienarbeit. Da heutzutage in fast jedem Vorlesungsraum ein Beamer vorhanden ist und Computer in sehr portablen Varianten existieren, sollte der Overheadprojektor durch einen Tablet PC und einem Beamer erstetzt werden.

Ein Tablet PC ist ein Laptop mit einem Bildschirm, auf dem mit einem Digitizerstift geschrieben werden kann. Ermöglicht wird die Technik [Wac09], indem der Bildschirm durch ein elektromagnetisches Feld per Induktion Energie in den Digitizerstift einspeist. Der Stift ermittelt seine momentan ausgeübte Druckstärke und emittiert ein kleines elektromagnetisches Feld an der Stiftspitze. Dieses Feld kodiert die momentane Druckstärke des Stiftes. Seine Position wird vom Bildschirm ermittelt.

Mit dem Einsatz des Programms mit dem Tablet PC sind folgende Absichten verbunden:

- Kein Folien- oder Stifteverbrauch.
- Kein gleißendes Licht, dass den Dozenten bei der Vorlesung iritiert und evtl. Augenschäden verursacht.
- Keine Digitalisierung der Folien notwenig.

1.1.1. Möglichkeiten im Programm Foils

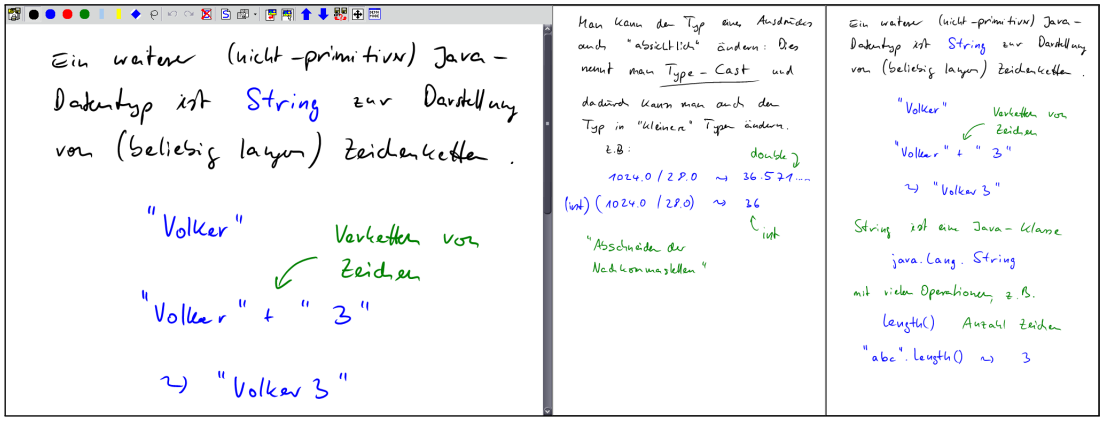

Abbildung 1.2.: Erste Version von Foils. Kontrollfenster (*links*) und das Präsentationsfenster mit den Folien (*rechts*).

Durch Foils ist es möglich eine Präsentation zu halten. Das Programm präsentiert sich mit zwei Fenstern, die in der Abb. 1.2 zu sehen sind. Das linke Fenster ist das Kontrollfenster für den Vortragenden, welches ihm viele Bearbeitungsmöglichkeiten

bietet. Das rechte Fenster ist das Präsentationsfenster für das Auditorium. Es zeigt die aktuelle und die zuletzt bearbeitete Folie.

Im Bearbeitungsfenster stehen dem Vortragenden sechs verschiedene vordefinierte Stifte zur Verfügung. Vier davon sind Stifte mit den Farben schwarz, dunkelblau, rot und dunkelgrün. Die zwei übrigen Stifte sind als Marker verwendbar und bieten die Farben hellblau und gelb. Beim Halten einer Präsentation stehen auch gebräuchliche Aktionen wie Löschen, Radieren Verschieben oder Skalieren von Strichen zur Verfügung. Außerdem kann Gebrauch von Bildmaterial gemacht werden und dieses in die Präsentation eingebunden werden. Auch hier existieren Bearbeitungsmöglichkeiten wie Verschieben, Skalieren und Löschen.

Damit die Präsentation gelingt, muss der Beamer den erweiterten Desktop des Computers anzeigen. Beim Start des Programms maximiert sich dann das Präsentationsfenster auf dem erweiterten Desktop und zeigt die am Anfang leeren Folien an.

1.2. Grobe Zeitphasen

In der ersten Phase sind durch umfangreiche Fragen an die Stakeholder die neuen Anforderungen erstellt worden. Die Stakeholder waren bei dieser Arbeit die Betreuer der Studienarbeit. Die gestellten Fragen, die sich im Anhang C auf Seite 91 befinden, befassten sich thematisch mit den Funktionen, dem Aufbau der Benutzungsoberfläche, Anordnungen von Schaltflächen und dem erwarteten softwaretechnisches Design des Programms. Außerdem existierte im Vorfeld bereits eine Mängelliste, die von den Betreuern erstellt wurde und im Anhang B auf Seite 89 abgedruckt ist. Aus der Befragung sowie der Mängelliste sind die Änderungsanforderungen entstanden, die sich im Anhang A auf Seite 81 befinden. Sie wurden nach drei Kategorien - Bug-Report, Change-Requests und Extension-Requests - unterteilt und nach ihren Prioritäten bewertet und sortiert.

Die nächste Phase ist für das Einarbeiten in das Programm und die Beschaffung von Hintergrundwissen verwendet worden. In diesem Zusammenhang sind Informationen über die Speicherverwaltung und die Funktionsweise der Tablet PC SDK 1.7 zusammengetragen worden.

In der letzten und gleichzeitig längsten Phase wurden die Änderungsanforderungen umgesetzt. Dabei ist zuerst der Bug-Report, dann der Change-Requests und zum Schluss der Extension-Requests bearbeitet worden.

1.3. Übersicht

Zuerst werden die verwendete Programmiersprache und ihre Klassenbibliotheken im Kapitel 2 auf der nächsten Seite genauer beleuchtet. Danach werden im Kaptiel 3 auf Seite 13 die verwendeten Klassen in Foils näher betrachtet um einige Fehlerursachen in den darauf folgenden Kapiteln besser zu verstehen. Nach diesen Kapiteln sollte das Hintergrundwissen für das nähere Verständnis von C#, .Net und Foils vorhanden sein.

Die nachfolgenden vier Kapitel beschäftigen sich mit der Bearbeitung der Anforderungen. Das Kapitel 4 auf Seite 23 greift sich die ersten zwei Anforderungen A.1.1 und A.1.2 heraus und behandelt sie gesondert, weil ihre Bearbeitung sehr umfangreich ist. Kapitel 5 auf Seite 35 behandelt die Anfordeungen des Bug-Report im Anhang A.1 auf Seite 81. Kaptiel 6 auf Seite 49 behandelt die Anforderungen aus dem Change-Requests im Anhang A.2 auf Seite 83. Kapitel 7 auf Seite 63 schließt mit den Anforderungen aus dem Extension-Requests im Anhang A.3 auf Seite 85 ab.

Es wird im Kapitel 8 auf Seite 75 noch ein Ausblick gegeben, was es noch an Potential für evtl. nachfolgende Arbeiten geben könnte. Abschließend wird im Kapitel 9 auf Seite 79 zusammengefasst, welches die Ziele waren und was erreicht wurde.

2. C# und .Net

Die Programmiersprache C# und das .Net Framework 3.5 wurden im bestehenden System von Tim Steffens verwendet und beibehalten. An dieser Stelle soll ein kleiner Einblick gegeben werden, was C# und .Net eigentlich sind und wie die Speicherverwaltung in C# funktioniert.

2.1. C#

C# ist eine von Microsoft entwickelte Programmiersprache [Sha08]. Nach der Kompilierung liegt Byte-Code vor, ein so genannter Zwischencode, der Common Intermediate Language genannt wird. Dieser Zwischencode kann von der Common Language Runtime (CLR) ausgeführt werden. Die CLR wurde auch von Microsoft entwickelt und bietet weitere Steuerungsmöglichkeiten des Programms, wie z.B. das Just-In-Time-Debuggen, automatisierte Speicherverwaltung und Reflection. Alle Programme, die für die CLR geschrieben sind, werden deswegen auch *managed code* genannt.

2.1.1. Ein paar Unterschiede zu Java

```
1   private double x;
2
3   private double X
4   {
5       get { return x; }
6       protected set { x = value; }
7   }
8
9   public Vector2D(double x, double y)
10  {
11      X = x;
12      ...
13  }
```

Listing 2.1: Beispiel für eine Eigenschaft in C#.

C# ähnelt der Sprache Java, aber unterscheidet sich in einigen Punkten. So wird ein **`package`** **`namespace`** genannt. Das Schlüsselwort **`super`** heißt in der Microsoftwelt **`base`**. Wie in Listing 2.1 zu sehen, gibt es zudem noch Eigenschaften in C#.

Eigenschaften sind nichts anderes als ein Strukturmittel in C#. Sie bündeln die sonst üblichen Getter- und Setter-Methoden zu einem Sprachmittel. Es ist auch möglich ihnen unterschiedliche Sichtbarkeiten zu geben oder einen `set`- oder `get`-Block auszulassen. Bei ihrer Verwendung können sie wie normale Variablen aussehen. Hier sei aber ein Wort der Vorsicht ausgesprochen. Bei der Verwendung von Eigenschaften kann es sehr leicht vorkommen, dass vergessen wird, dass beim Setzen oder der Wertermittlung eine Methode im Hintergrund ausgeführt wird.

2.2. .Net

.Net bezeichnet ein aus Klassenbibliotheken und der Common Language Runtime zusammengestelltes Framework [Hub08]. Es erleichtert durch umfangreiche Klassenbibliotheken das Schreiben von Programmen. .Net gibt es in verschiedenen Versionen. In dieser Arbeit wurde das .Net Framework 3.5 mit dem Service Pack 1 verwendet, das das .Net Framework 2.0 in der Service Pack Version 1.0 umfasst. Es stellt die Basisklassen und anderen Bibliotheken wie die Windows Forms zur Programmierung von GUI-Anwendungen zur Verfügung, sowie die CLR in der Version 2.0. Anzumerken ist, dass viele Klassen aus der Windows Forms Bibliothek nicht völlig in *managed code* geschrieben sind, sondern nur als Wrapperklassen zur Verfügung gestellt werden und so den Zugriff auf die GDI kapseln. GDI ist die Graphische Schnittstelle in Windows, mit der Zeichenoperationen durchgeführt werden können.

Auf dieser Basis baut das .Net Framework 3.0 mit Service Pack 1.0 auf. Es beinhaltet vier große Klassenbibliotheken, die da heißen:

- Windows Presentation Foundation (WPF),
- Windows Communication Foundation (WCF),
- Windows Workflow Foundation (WF) und
- Windows CardSpace (WCS).

Sie sind in *managed code* geschrieben. Für diese Arbeit ist aber nur die WPF von Bedeutung.

2.2.1. Windows Presentation Foundation

Die WPF ist für die Implementation von GUI-Anwendungen gedacht. Sie ist über den Namespace *System.Windows* zu erreichen. Sie soll die alten Windows Forms ablösen, die seit .Net 1.0 im .Net Framework vorhanden sind.

Eine Neuerung ist, dass Oberflächen der WPF durch DirectX - einer Graphikschnittstelle von Microsoft - beschleunigt werden. Alle Größen- und Positionsangaben von graphischen Elementen besitzen damit doppelte Fließkommagenauigkeit statt Pixelgenauigkeit. Die Folge ist, dass mit der WPF erstellte Oberflächen Vektorgrafiken und dadurch frei skalierbar sind und dass dargestellte Elemente immer richtig angeordnet werden können.

Eine weitere Neuerung ist die Art wie Oberflächen erstellt werden können. Es ist nicht mehr notwendig eine Oberfläche in Programmcode zu erstellen, sondern es kann für solche Aufgaben die eXtensible Application Markup Language (XAML) verwendet werden. XAML ist ein XML Dialekt, der eine bessere Trennung von Programmlogik und Benutzungsoberfläche ermöglicht. Dabei werden Quellcode-Dateien, die in C# geschrieben sind, in den gewohnten Quellcode-Dateien abgelegt, die die Endung "*.cs" tragen. Der Code für die Benutzungsoberfläche liegt in zusätzlichen Dateien, die die Endung "*.xaml" verwenden.

2.3. Speicherverwaltung in C#

In dieser Arbeit ist es notwendig die Speicherverwaltung [Sha08] näher zu betrachten, um zu verstehen, wann genau Objekte ihre Gültigkeit verlieren und der von ihnen beanspruchte Speicher wieder freigegeben wird.

Da sämtlicher in C# geschriebener Code *managed code* ist, wird eine Garbage-Collection zur Verwaltung des Speichers verwendet. Beim Speicher wird zwischen zwei Arten unterschieden: dem Stack und dem verwalteten Heap.

Der Stack speichert nur Referenzen und primitive Datentypen ab. Der Speicher, der verbraucht wird, wird nach dem Verlassen einer Methode automatisch freigegeben. Große Objekte werden aber nicht auf dem Stack abgelegt sondern auf dem Heap.

Der verwaltete Heap speichert alle Objekte, auf die vom Stack aus referenziert wird. Der Heap ist wiederum in zwei verschiedene Bereiche eingeteilt. Ein Bereich für normale und ein anderer für große Objekte. In der Quelle [Hof09] wird 22 Kb als ungefähre Mindestgröße für große Objekte angegeben. Es wird außerdem mit einem Generationenmodell gearbeitet, d.h. es gibt drei Stufen, null, eins und zwei, wobei zu der Generation null junge Objekte und zur zweiten Langzeitobjekte gehören .

2.3.1. Destruktoren

Trotz der Tatsache, dass C# eine Garbage-Collection besitzt, hat C# auch Destruktoren, die wie Destruktoren in C++ deklariert werden. Dabei besitzt der Destruktor keine Sichtbarkeit und kann nicht mit variierenden Parameterlisten überladen werden. Er

besitzt nur die leere Parameterliste. Im Gegensatz zu C++ kann in C# der Destruktor einer Klasse nicht explizit vom Programmierer aufgerufen werden. Das bedeutet, dass der Speicher eines Objekts nicht explizit vom Programmierer freigegeben werden kann, dies kann ausschließlich vom Garbage Collector (GC) erledigt werden.

```
1 ~MyClass() { ... }
```

Listing 2.2: Ein C#-Destruktor

```
1 protected override void Finalize()
2 {
3    try
4    {
5       // Code aus dem Destruktor
6    } finally { base.Finalize() }
7 }
```

Listing 2.3: Die compilerseitige Übersetzung des Destruktors in die Methode `Finalize()`.

Der beispielhafte Destruktor der Klasse `MyClass` aus Listing 2.2 wird automatisch vom Compiler als eine Methode übersetzt, die `Object.Finalize()` [Mic09b] überschreibt. Dabei wird der Code vom Destruktor in die `Finalize()`-Methode, win in Listing 2.3 zu sehen ist, gekapselt.

Im **`finally`**-Block wird der Code **`base`**`.Finalize()` ausgeführt. Dadurch wird sichergestellt, dass immer der Destruktor der geerbten Klasse aufgerufen wird.

Aber auch die Methode `Finalize()` kann nicht vom Programmierer aufgerufen werden. Durch diese Einschränkung sollen folgende Fälle verhindert werden:

- Dass ein Objekt vergessen wird gelöscht zu werden.
- Dass ein Objekt gelöscht wird, auf das noch eine aktive Referenz an einer anderen Stelle im Code existiert.
- Dass ein Objekt mehrmals zerstört wird.

Da der Programmierer keine Handhabe bei der Speicherverwaltung hat, ist ein genauerer Blick auf den Garbage Collector erforderlich.

2.3.2. Garbage Collector

Der Garbage Collector (GC) erfüllt folgende Bedingungen:

- Er zerstört jedes Objekt, das nicht mehr referenziert wird, und ruft - falls vorhanden - dessen Destruktor auf.
- Er zerstört jedes referenzierte Objekt bei Programmende.
- Er zerstört jedes Objekt nur einmal.

Der GC läuft in einem eigenen Thread. Er kann erst aktiv werden, wenn das Ende einer Methode erreicht wird. Es ist nicht genau festgelegt, wann er aktiv wird.

Wenn der GC seine Arbeit beginnt, werden alle Threads des Programms angehalten. Dies ist notwendig, da er Objekte verschieben und dann entsprechende Verweise neu setzen können muss. Bei seiner Arbeit geht er folgendermaßen vor:

- Im ersten Schritt wird eine Tabelle mit allen erreichbaren Objekten erstellt. Für diesen Aufbau werden alle Verweise auf Objekte verfolgt und Objekte, die erreicht werden können, in diese Tabelle eingetragen. Dabei werden zyklische Verweisketten automatisch erkannt. Objekte, die nicht in dieser Liste auftauchen, können zerstört und ihr Speicher freigegeben werden.
- Im zweiten Schritt wird für alle Objekte, die nicht erreicht werden können, überprüft, ob sie einen Destruktor besitzen. Falls sie einen besitzen, werden sie in eine Warteschlange eingereiht, die so genannte *Finalizer*-Warteschlange.
- Im nächsten Schritt werden von allen Objekten, die keinen Destruktor besitzen, die Speicherbereiche freigegeben, der Speicher defragmentiert und alle Referenzen aktualisiert.
- Nach diesen Schritten wird auch anderen Threads wieder die Möglichkeit gegeben Prozessorzeit zu beanspruchen.
- In einem separaten Thread wird jetzt die *Finalizer*-Warteschlange abgearbeitet.

Aus diesem Prinzip folgt, dass es länger dauert Objekte mit Destruktor freizugeben als solche ohne Destruktor. Außerdem ist nicht garantiert, in welcher Reihenfolge die Objekte in die *Finalizer*-Warteschlange abgelegt und tatsächlich freigegeben werden.

Der Programmierer kann den GC, wenn es nötig ist, explizit anstoßen. Dafür wird die statische Methode `GC.Collect()` verwendet. Damit nur von null bis zu einer bestimmte Generation aufgeräumt wird, kann die Methode `GC.Collect(Int32)` verwendet werden.

Da es ungewiss ist, wann der GC die *Finalizer*-Warteschlange abarbeitet, gibt es einen weiteren Methodenaufruf `GC.WaitForPendingFinalizers()`. Mit diesem Aufruf wird garantiert, dass zuerst die Warteschlange komplett bearbeitet wird, bevor andere Threads wieder aktiv werden.

Durch den Garbage Collector ist es außerdem möglich die tatsächliche Größe des derzeitig verbrauchten Speichers zu ermitteln. Die Methode `long`

`GetTotalMemory(bool)` gibt dabei die Größe des verbrauchten Speichers in Bytes zurück. Mit der übergebenen Boolean-Flag, kann eine Garbage Collection vor der Speicherermittlung durchgeführt werden. Anzumerken ist aber, dass die Methode nur den verwalteten Speicher zurückgeben kann, der von der CLR angefordert wird [JS03]. Werden Klassen verwendet, die nur einen Wrapper zur Verfügung stellen und im Hintergrund auf Code außerhalb der CLR zugreifen, kann nicht der genaue Speicher in Erfahrung gebracht werden.

2.3.3. Freigabemethoden und das `IDisposable`-Interface

Um von einem Objekt verwaltete Ressourcen freizugeben, wie zum Beispiel einen Socket oder einen Dateihandle, bevor sie durch den GC zerstört werden, gibt es eine weitere Möglichkeit und zwar die Verwendung einer Freigabemethode. Dabei sollte aber bei jeglichem Methodenaufruf eines solchen Objekts nach der Freigabe darauf geachtet werden, ob dieses Objekt bereits seine Ressourcen freigegeben hat oder nicht. Falls dies geschehen ist, sollte eine Ausnahme geworfen werden.

Eine Deklaration für eine Freigabemethode stellt das Interface `IDisposable` in Form der Methode `Dispose()` zur Verfügung. Wird sie implementiert, ist im Destruktor nur noch der Aufruf von `Dispose()` erforderlich. Hierbei ist zu bedenken, dass die `Dispose()`-Methode nur einmal aufgerufen werden darf. Dies kann durch eine Boolean-Flag gewährleistet werden, die beim Betreten abgefragt wird. Somit ist sichergestellt, dass die Ressourcen nicht abermals freigegeben werden. Außerdem kann der Destruktoraufruf durch den GC mittels einer statischen Methoden namens `GC.SuppressFinalize(Object)` unterdrückt werden.

```
1  class Example : IDisposable
2  {
3     private bool disposed = false;
4
5     ...
6
7     ~Example()
8     {
9        Dispose();
10    }
11
12    public virtual void Dispose()
13    {
14       if (!this.disposed)
15       {
16          try {
```

```
17              // Hier knappe Ressourcen freigeben
18          }
19          finally {
20              this.disposed = true;
21              System.GC.SupressFinalize(this);
22          }
23      }
24  }
25
26  public void SomeBehavior() // Beispielmethode
27  {
28      checkIfDisposed();
29      ...
30  }
31
32  private void checkIfDisposed()
33  {
34      if (disposed)
35      {
36          throw new ObjectDisposedException("Example: Objekt wurde
                freigegeben");
37      }
38  }
39
40  ...
41
42 }
```

Listing 2.4: Programmbeispiel für einen Destruktor mit dem `IDisposable`-Interface

Bei der Implementierung ist darauf zu achten, dass auf jeden Fall `GC.SuppressFinalize(Object)` aufgerufen wird. Ein Aufruf der Methode kann garantiert werden, indem sie in die **`try`** `{ ... }` **`finally`** `{ ... }` Anweisung, wie in Listing 2.4 demonstriert, geschrieben wird. Der **`finally`**-Block wird unter allen Umständen ausgeführt, wodurch sichergestellt ist, dass `GC.SuppressFinalize(Object)` ausgeführt wird.

Bei der Verwendung eines Objekts kann es zu Ausnahmen kommen, die das Aufrufen der `Dispose()`-Methode des Objekts verhindern. Solche Situationen müssen behandelt werden, indem die Ressourcen im **`finally`**-Block freigegeben werden. Alle diese Umstände erschweren es ausnahmesicher zu programmieren und nicht den Überblick über den Programmcode zu verlieren. Außerdem existiert das Objekt der Freigabe der Ressourcen im Gültigkeitsbereich.

Mit wachsendem Programmcode kann diese Art des Ressourcenmanagments sehr unübersichtlich werden und es kann zu Unachtsamkeit von Seiten des Programmierers führen, wodurch wieder Fehler hervorgerufen werden. Um dies zu umgehen sollte die **using**-Anweisung benutzt werden.

2.3.4. Die `using`-Anweisung

```
1 using ( <Typ> <Variable> = <Wert> )
2 {
3    <Anweisungsblock>
4 }
```

Listing 2.5: Schematisches Beispiel für die Verwendung der **using**-Anweisung

Die **using**-Anweisung, deren Benutzung in Listing 2.5 zu sehen ist, stellt einen Mechanismus bereit, der es ermöglicht, die Lebenszeit von Ressourcen gezielt zu steuern. Dabei wird das zu verwaltende Objekt direkt in der **using**-Anweisung deklariert und initialisiert. Das zu verwaltende Objekt existiert im Anweisungsblock genau so lange, bis der Block verlassen wird.

Variablen, die mittels **using** verwaltet werden, müssen das erwähnte Interface `IDisposable` implementieren.

2.3.5. Zusammenfassung und Schlussfolgerung

Da nicht genau bekannt ist, wann der GC seine Arbeit aufnimmt, sollte der GC regelmäßig angestoßen werden, um den Speicher aufzuräumen. Destruktoren sollten nach Möglichkeit vermieden werden, weil nicht mit Sicherheit gesagt werden kann, wann ein Objekt in der *Finalizer*-Warteschlange bearbeitet und aus dem Speicher entfernt wird. Falls es sich nicht verhindern lässt einen Destruktor zu implementieren, sollte das `IDisposable`-Interface verwendet und die `Dispose()`-Methode im Destruktor aufgerufen werden. Diese bietet die zusätzliche Möglichkeit, nach Ende der Lebenszeit eines Objekts dessen Ressource freizugeben.

Hierbei kann die **using**-Anweisung sehr viel Arbeit abnehmen. Sie ist allerdings bei komplizierteren Programmstrukturen nicht mehr anwendbar.

Im folgenden Kapitel 3 werden die verwendeten Klassen in Foils genauer beleuchtet, damit ihre Funktionsweise bei späteren Betrachtungen geklärt ist.

3. Nähere Betrachtung von Foils

In diesem Kapitel werden mehrere tiefe Einblicke in den Aufbau von Foils getätigt. Im Laufe der Arbeit wurden einige Klassen aus dem Projekt herausgenommen und neue hinzugefügt. Für die spätere Analyse im Kapitel 4 ist es wichtig die schon bestehenden Implementation genauer unter die Lupe zu nehmen.

Nachdem die bestehende Implementation vorgestellt wurde, werden die neuen integrierten Klassen vorgestellt.

3.1. Ein Blick auf die alte Version von Foils

Die Bestandteile von Foils, die näher betrachtet werden sind: Der Strich, die Folie, die Zeichenfläche, das Kommando-Objekt und der Undo-Stack.

3.1.1. Die Folien und die Zeichenfläche

Ein Strich entsteht, wenn ein Stift auf einem Tablet aufgesetzt und wieder abgesetzt wird.

Ein Strich ist ein `Stroke`-Objekt und wird mit mehreren `Stroke`-Objekten in einem `Strokes`-Objekt zusammengefasst. Das `Strokes`-Objekt gehört zu einem `Ink`-Objekt, welches von einem `InkOverlay`-Objekt automatisch verwaltet wird. Diese Objekte werden von der Microsoft Tablet PC SDK bereitgestellt. Diese SDK ist *closed-source*.

Eine Folie umfasst Texte, Zeichnungen und Grafiken, die auf einer begrenzten Fläche dargestellt und bearbeitet werden können.

Eine Folie wird durch die Klasse `Slide` abgebildet. Eine `Slide`-Klasse umfasst ein `Ink`-Objekt und eine `ArrayList` von `GraphicContainer`-Objekten. Die `GraphicContainer`-Klasse dient der Darstellung eines Bildes in der Präsentation. Um mehrere Folien zusammenzufassen, existiert die Klasse `SlideSequence`, die einzelne Folien - sogenannte Objekte der Klasse `Slide` - verwaltet.

Auf einer Zeichenfläche ist es möglich, zu schreiben, die gezeichnete Schrift zu speichern und darzustellen.

Die Zeichenfläche wird von der Klasse `ControlView` eingebunden und heißt `InkEditingPanel`. Sie beinhaltet ein Objekt der Klasse `InkOverlay`. Durch ein Objekt der Klasse `InkOverlay` ist es möglich, Schrift zu sammeln und darzustellen. Das Sammeln und Darstellen der Schrift geschieht hierbei automatisch. `ControlView` verwaltet die darzustellende Folie und ermöglicht es auch Graphiken abzubilden.

Folien werden in einer normalisierten Form vorrätig gehalten. D.h., dass das enthaltene `Ink`-Objekt auf eine virtuelle Foliengröße normalisiert ist. Somit ist es möglich, bei der Nutzung mit einem anderen Tablet PC oder einem PC mit angeschlossenem Tablet dieses Programm weiter- und auch alte Präsentation wiederzuverwenden. Wird nun eine bestimmte Folie dargestellt und bearbeitet, wäre es naheliegend, dass die Folie von der Klasse `ControlView` direkt dargestellt wird. Dabei wird das enthaltende `Ink`-Objekt vom `InkEditingPanel`-Objekt verwaltet und bearbeitet. Tatsächlich wird aber nicht das `Ink`-Objekt verwaltet, sondern nur eine skalierte Kopie dessen.

3.1.1.1. Probleme mit der `Ink`-Klasse

Für die `Ink`-Klasse und die enthaltene `Strokes`-Klasse existieren `Add()`-Funktionen. Leider fügen sie nur `Stroke`-Objekte hinzu, die schon im `Ink`-Objekt enthalten sind. Andernfalls tritt ein Ausnahmefehler auf. Falls ein enthaltendes `Stroke`-Objekt aus einem `Ink`-Objekt in eine Variable abgespeichert und dann aus dem `Ink`-Objekt gelöscht wird, ist es komplett gelöscht. D.h. dass das abgespeicherte `Stroke`-Objekt - auch wenn auf es noch referenziert wird - nicht mehr existiert. Damit ist auch ein erneutes Einfügen nicht mehr möglich. Die Klassen der Tablet PC SDK sind nicht direkt von der CLR ausführbar, sondern bieten ihre Funktionalität nur über Wrapperklassen an [JS03]. Alle Klasse der Tablet PC SDK kümmern sich deshalb selber um ihre Speicherverwaltung. Aus dem Grund wird direkt beim Löschen eines `Stroke`-Objekts aus einem `Ink`-Objekt dessen Speicher freigegeben.

Die einzige Möglichkeit ein `Stroke`-Objekt zwischenzuspeichern ist somit dessen Objektzustände auszulesen. Um ihn wiederherzustellen muss mit der auf ein `Ink`-Objekt definierten Methode `CreateStroke()` ein neues `Stroke`-Objekt erstellt werden

3.1.2. Kommando-Objekt

Ein Kommando-Objekt kapselt ein Kommando in ein Objekt ein und ermöglicht damit die Speicherung des Kommandos. Dieses Vorgehen wird *Command-Pattern* [GHJV94] genannt. Um das Kommando auszuführen wird eine `Execute()`-Funktion bereitgestellt.

`Operation` ist ein Interface, das nach dem Command-Pattern implementiert wurde. Die `Do()`-Funktion entspricht der `Execute()`-Funktion. Von `Operation` gibt es aber

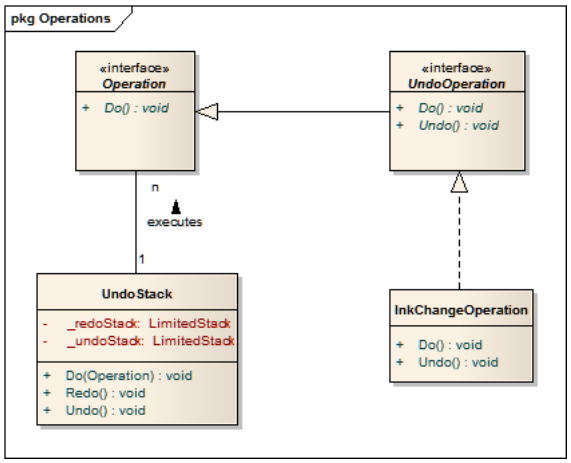

Abbildung 3.1.: Ein Klassendiagramm des Undo-Stacks und den verwendeten Kommando-Objekten.

auch eine Spezialisierung: `UndoableOperation`. Diese deklariert zusätzlich die Funktion `Undo()` und ermöglicht so bei einer konkreten Implementation, dass die getätigte Operation rückgängig gemacht werden kann.

Von den Interfaces `Operation` und `UndoableOperation` existieren Implementationen, die dem Benutzer ermöglichen, Operationen durchzuführen und fast alle ausgeführten Operationen wieder rückgängig zu machen. So kann ein getätigter Strich wieder entfernt, aber eine gelöschte Folie nicht wieder hergestellt werden, weil die dazugehörige Klasse nur das Interface `Operation` implementiert.

3.1.3. Der Undo-Stack

Ein Undo-Stack ermöglicht es, ausgeführte Kommando-Objekte zu speichern und gegebenenfalls rückgängig zu machen. In Foils ist schon ein Undo-Stack implementiert, der `Operations` genannt worden ist.

`Operations` umfasst zwei Stacks, die als `doneStack` - für ausgeführte - und `undoneStack` - für rückgängig gemachte Operationen - bezeichnet werden. Auf dem Undo-Stack können die Funktionen `Do()`, `Redo()` und `Undo()` ausgeführt werden. Die `Do()`-Funktion bekommt zusätzlich ein `Operation`-Objekt übergeben, das direkt ausgeführt wird.

Wird nun ein `Operation`-Objekt an die `Do()`-Funktion von `Operations` übergeben, so wird von dem übergebenen Objekt die `Do()`-Funktion ausgeführt. Ist das Objekt sogar vom Interface `UndoableOperation` abgeleitet, so wird es auf einen `doneStack` abgelegt und der `undoneStack` komplett gelöscht. Wird die Funktion `Undo()` aufgerufen, wird vom auf dem `doneStack` oben liegenden Objekt die `Undo()`-Funktion aufgerufen und das Objekt auf den `undoneStack` abgelegt. Liegt kein Objekt auf dem

Stack, geschieht nichts. Die `Redo()`-Funktion verhält sich zur `Undo()`-Funktion entgegengesetzt.

Die Größe des implementierten Undo-Stacks wird bei seiner Erstellung festgelegt.

Leider muss hier bemerkt werden, dass bei der Implementation des Undo-Stack und des `Operation`-Interface die Bezeichnungen unglücklich gewählt wurden. So vermittelt der Name `Operations` nicht auf den ersten Blick, worum es sich bei dieser Klasse handelt. Der Name `UndoableOperation` bedeutet sogar, dass die Operation nicht machbar ist. Ebenfalls führt die Nähe der beiden Namen `Operation` und `Operations` zu häufigen Missverständnissen. Diese Schwachpunkte sind behoben, indem `Operations` jetzt `UndoStack` und `UndoableOperation` `UndoOperation` heißt. Im weiteren Verlauf der Arbeit werden nur noch die neuen Bezeichnungen verwendet. Das Klassendiagramm in Abb. 3.1 illustiert nochmal die Beziehung zwischen den Klassen `UndoStack` und `InkChangeOperation` und den Inferfaces `Operation` und `UndoOperation`.

3.1.4. InkChangeOperation

Der Klasse `InkChangeOperation` kommt später im Abschnitt 4.2.4 eine besondere Bedeutung zu. `InkChangeOperation` ist vom Interface `UndoOperation` abgeleitet. Ein `InkChangeOperation`-Objekt speichert Änderungen am aktuellen `Ink`-Objekt ab, um sie durchführen und/oder rückgängig machen zu können.

Bei der Erstellung eines `InkChangeOperation`-Objekts, wird das `Ink`-Objekt der Zeichenfläche und das `Ink`-Objekt der derzeitig im Bearbeitungsfenster dargestellten `Slide` kopiert. Die `Do()`-Funktion, die im weiteren Verlauf vom Undo-Stack aufgerufen wird, führt eine `UpdateInk()`-Funktion auf dem aktuell dargestellten `Slide`-Objekt aus. Ihr wird die derzeitige Größe der Bearbeitungsansicht übergeben sowie das neue Ink-Objekt. Die Funktion bewirkt, dass das übergebene `Ink`-Objekt geklont, auf die interne Größe, die die `Slide` verwendet, skaliert wird, und das schon vorhandene `Ink`-Objekt ersetzt. Nach der `UpdateInk()`-Methode, sofern diese Funktion nicht zum ersten Mal ausgeführt wird, werden alle Beobachter der gerade bearbeiteten Folie benachrichtigt. Die Benachrichtigung hat ein Neuzeichnen der Zeichenfläche zur Folge.

Die `Undo()`-Funktion führt die gleichen Operationen aus jedoch mit dem alten `Ink`-Objekt. Außerdem wird das `Ink`-Objekt des `InkOverlay`-Objekts mit einer skalierten Kopie des alten `Ink`-Objekts überschrieben.

Der Grund für das Kopieren des `Ink`-Objekts der Zeichenfläche ist, dass es den neu gezeichneten Strich beinhaltet und das `Ink`-Objekt der Folie noch kein Update erhalten hat. Aus diesem Sachverhalt ist gut zu erkennen, dass das `Ink`-Objekt der

aktuellen Folie und der Zeichenfläche nicht identisch sind. Somit wird ein Vorher- und Nachher-Bild festgehalten. Diese simpel und logisch erscheinende Methode birgt ihre Tücken, die später noch im Abschnitt 4.2.4 genauer betrachtet werden .

3.2. Ein Blick auf die neue Version von Foils

In der neuen Version von Foils sind einige Klassen ausgetauscht worden, weil neuere und bessere Implementationen zur Verfügung standen. Sie betrachten die `InkCanvas` und die aus ihr heraus entstandene `SlideCanvas`-Klasse.

3.2.1. Eine neue Zeichenfläche aus der WPF

In der WPF gibt es das `InkCanvas`-Objekt. Es stellt eine Zeichenfläche bereit, auf die Striche gezeichnet werden können. Striche werden von einem `Stroke`-Objekt abgebildet. Alle `Stroke`-Objekte werden in einem `StrokeCollection`-Objekt abgelegt, das an das `InkCanvas`-Objekt angebunden ist und von diesem verwaltet wird.

Daraus folgt, dass das `InkCanvas`-Objekt an die bereits bekannte `InkOverlay`-Klasse aus der Tablet PC SDK 1.7 angelehnt ist. Der große Unterschied zwischen den beiden Klassen ist, dass das `InkCanvas`-Objekt zum größten Teil in *managed code* geschrieben wurde, der von der CLR verwaltet werden kann.

Es existiert ein Beispiel-Programm [Mic09a] der `InkCanvas`, das die Fähigkeiten des Objekts gut demonstriert. Bei dieser Implementation sind auch Undo- und Redo-Operationen möglich. Bei genauerer Betrachtung des Quellcodes wird deutlich, dass die Undo- und Redo-Operationen mittels `InkCanvas` wesentlich leichter implementierbar sind. Auch muss an keiner Stelle eine Invalidierung eines Bereichs auf der Zeichenfläche erfolgen, damit sie sich neuzeichnet, wie es bei einem `InkOverlay`-Objekt der Fall ist. Außerdem ist die Lasso-Funktion bei der `InkCanvas` besser implementiert. Sie zeichnet nur noch dann einen neuen Lasso-Punkt, wenn dieser Punkt in einem bestimmten Abstand vom zuletzt gesetzten Lasso-Punkt entfernt ist. Bei der Implementation des `InkOverlay`-Objekts ist dies nicht der Fall. Bei ihr wird nach einem bestimmten Zeitintervall ein neuer Lasso-Punkt gesetzt.

Während dieser Studienarbeit erfolgte ein Wechsel von der verwendeten `InkOverlay`- zur `InkCanvas`-Klasse. Vor der Nutzung der neuen Zeichenfläche muss geklärt werden, ob alle vorherigen Funktionen auch weiterhin unterstützt werden können. Diese sind:

- Skalieren der Zeichenfläche
- Laden und korrektes Darstellen der alten Präsentationen

- Speichern einer Präsentation
- Nutzung von Undo- und Redo-Operationen
- Auswahl von Strichen per Lasso
- Export der Präsentation als Bilder
- Einbinden von Bildern auf der Zeichenfläche

Das Skalieren der Zeichenfläche ist notwendig, um Striche in der richtigen Größe darstellen zu können.

Es ist möglich, die Zeichenfläche zu skalieren. Hierzu muss die angebundene Transformationsmatrix, die über die Eigenschaft `LayoutTransform` erreichbar ist, mit einer neuen Matrix überschrieben werden, die die benötigte Skalierung durchführt. Die Matrix in der `LayoutTransform`-Eigenschaft bewirkt, dass die Eigenschaften `Width` und `Height` vor der Darstellung durch sie zuerst berechnet werden. Alle untergeordneten Elemente werden ebenfalls zuerst durch die angebundene Matrix transformiert, bevor sie dargestellt werden.

Das Laden alter Präsentation und deren korrekte Darstellung ist nur für evtl. Wiederverwendungen von alten Präsentationen notwendig. Dabei ist das Laden von serialisierten `Ink`-Objekten kein Problem.

Nach dem erfolgreichen Laden einer alten Präsentation ist die korrekte Darstellung dieser Präsentation noch nicht gegeben. Dies liegt an der Tatsache, dass bei der Tablet PC SDK nur kreisförmige oder rechteckige Stiftspitzen existieren. Dies ist bei der Implementation in der WPF anders. Hier gibt es elliptische und eckige Stiftspitzen. Aus diesem Grund ist es erforderlich, dass vor dem Darstellen alle `Stroke`-Objekte einer `StrokeCollection` durchwandert werden und die Höhe und Breite einer Stiftspitze auf das Maximum der Werte gesetzt wird.

Nach dieser Korrektur ist die Darstellung der Folien äquivalent zur der der `InkOverlay`-Klasse.

Das Speichern der Präsentation ist möglich. Dafür muss die Methode `void Save(Stream)` auf einem `Stroke`-Objekt aufgerufen werden. Sie schreibt in das übergebene `Stream`-Objekt ihre Daten. Der Inhalt des `Stream`-Objekts muss nur noch in eine Stringdarstellung zur Basis 64 umgewandelt werden, damit die Speicherung in ein XML-Format möglich ist.

Das Undo bzw. Redo von Operationen müssen mit der `InkCanvas`-Klasse durchführbar sein. Wie Anfang des Abschnitts 3.2.1 angedeutet, ist die Implementation von `Operation`-Klassen mittels einfacher Operationen auf einer `StrokeCollection` möglich. `StrokeCollection` bietet die Möglichkeit, `Stroke`- oder `StrokeCollection`-Objekte zum Löschen oder Einfügen zu verwenden.

Die Auswahl von Strichen per Lasso kann zur Verfügung gestellt werden. Dafür muss die `EditingMode`-Eigenschaft richtig gesetzt werden. Die `EditingMode`-Eigenschaft erlaubt den Zugriff auf einen Wert der Enumeration `InkCanvasEditingMode`. Es stehen mehrere Wahlmöglichkeiten zur Auswahl, aber im Rahmen diese Studienarbeit sind die Modi `Ink`, `Select`, `EraseByPoint` und `EraseByStroke` interessant.

Wenn der Modus `Ink` über die `EditingMode`-Eigenschaft gesetzt wird, kann auf der `InkCanvas` mit einem Stift geschrieben werden. Der Modus `Select` ermöglicht es, mit einem Lasso alle Elemente innerhalb der `InkCanvas` auszuwählen, zu verschieben oder zu skalieren. Der Modus `EraseByPoint` bietet die Möglichkeit, mit einem voreingestellten quadratischen Stift zu radieren. Der letzte Modus `EraseByStroke` radiert nicht, sondern löscht einen Strich komplett, sobald er vom Stift berührt wird.

Die Auswahl von Strichen ist durch den Modus `Select` gegeben. Sie wird aktiviert, wenn die `EditingMode`-Eigenschaft des `InkCanvas`-Objekts den Wert `Select` übergeben bekommt.

Die Exportierung von Folien zu Bilder ist realisierbar. Dafür bietet die `StrokeCollection` mit der Methode `public void Draw(DrawingContext)` eine einfache Variante, um Striche auf eine Zeichenfläche zu übertragen.

Das Einbinden von Bildern auf der Zeichenfläche ist umsetzbar. Bilder können als untergeordnete Elemente der `InkCanvas` eingebunden werden. Dabei ist es sogar möglich, mittels der Lasso-Funktion diese Elemente auszuwählen, zu verschieben oder zu skalieren. Mehr Informationen zu Bilder und deren Einbindung in ein `InkCanvas`-Objekt sind im Abschnitt 6.2 auf Seite 49 zu finden.

3.2.1.1. Beurteilung der `InkCanvas`

Die `InkCanvas` bietet alle Funktionen, die für Foils erforderlich sind. Sie implementiert einige Funktionen besser als die `InkOverlay`-Klasse. Da sie und die enthaltene `StrokeCollection` auch von der CLR verwaltet werden können, wird der Speicher im Gegensatz zum `InkOverlay`-Objekt innerhalb der CLR verwaltet. Außerdem muss nicht mehr befürchtet werden, dass beim Löschen eines `Stroke`-Objekts dieser nicht mehr im Speicher vorhanden ist.

3.2.1.2. Umstellung auf die `InkCanvas`-Klasse

Es wurde wegen der `InkCanvas`-Klasse das Programm von den Windows Forms Bibliotheken komplett auf die WPF umgeschrieben. Das Resultat ist im Abschnitt 6.3 auf Seite 52 zu sehen. Es wurde außerdem der Speicherverbrauch und die Abspielgeschwindigkeit einer Präsentation gemessen mit Hilfe der Replay-Funktion, die im Abschnitt 4.1 auf Seite 23 eingeführt wird. Das Ergebnis der Messung ist im Abschnitt 4.2.6 auf Seite 31 dargestellt und wird dort genauer erläutert.

3.2.2. Die `SlideCanvas`-Klasse

Die graphischen Oberflächen der WPF sind nach dem *Composite Pattern* [GHJV94, Hub08] aufgebaut und geben eine strikte Baumstruktur für alle logischen Elemente vor. Die Elemente werden in einen *Logical Tree* [GHJV94] abgelegt. Es ist also nicht möglich, dass ein `UIElement` an mehreren Knoten im *Logical Tree* als Kind angehangen wird. Um Objekte als Kinder an andere `UIElement`-Objekte anzufügen, gibt es zwei Varianten. Die erste ist durch die `Child`-Eigenschaft gegeben. Sie lässt nur das Anfügen eines Kindes zu. Die zweite Variante lässt es zu mehrere Kinder anzufügen. Die entsprechende Eigenschaft heißt `Children` und stellt ein `UIElementCollection`-Objekt zur Verfügung, an das Kindelemente einzufügt werden können.

Ein Objekt der `Image`-Klasse wird durch das Hinzufügen an die `Children`-Eigenschaft in den *Logical Tree* eingeordnet. Dadurch kann ein `Image`-Objekt nicht gleichzeitig zwei `InkCanvas`-Objekten zugeordnet werden. Deswegen müssen Bilder durch eine `GraphicContainer`-Klasse gekapselt werden.

Die Klasse `GraphicContainer` hält neben dem Bild selbst Informationen darüber bereit, wie groß und an welcher Stelle das Bild auf der Zeichenfläche der `InkCanvas` erscheinen soll. Die Klasse `Image` ist in Wirklichkeit aber nicht das Bild, sondern ein Objekt, das einen Bildkontext darstellt. Der Bildkontext wird durch die abstrakte Klasse `ImageSource` beschrieben. Eine Implementation ist die Klasse `BitmapImage`. Ein Objekt dieser Klasse wird in der Klasse `GraphicContainer` verwendet.

Durch die Kapselung der Bilder in `GraphicContainer`-Objekte muss eine `InkCanvas` nur die `GraphicContainer`-Objekte aus der `GraphicCollection` der darzustellenden Slide auslesen. Aus dem `GraphicContainer`-Objekt wird das `BitmapImage`-Objekt für ein neues `Image`-Objekt verwendet, dieses positioniert und seine Größe angepasst.

Damit ein `InkCanvas`-Objekt auch Bilder einer `Slide` darstellt, wurde eine neue Klasse namens `SlideCanvas` geschrieben. Die Klasse ist nach dem *Model-View-Controller-Pattern* [Bur09] implementiert. Wie in Abb. 3.2 zu sehen ist, ist das `SlideCanvas`-Objekt *Controller*. Es besitzt Zugriff auf ein `InkCanvas`-Objekt, das als *View* verwendet

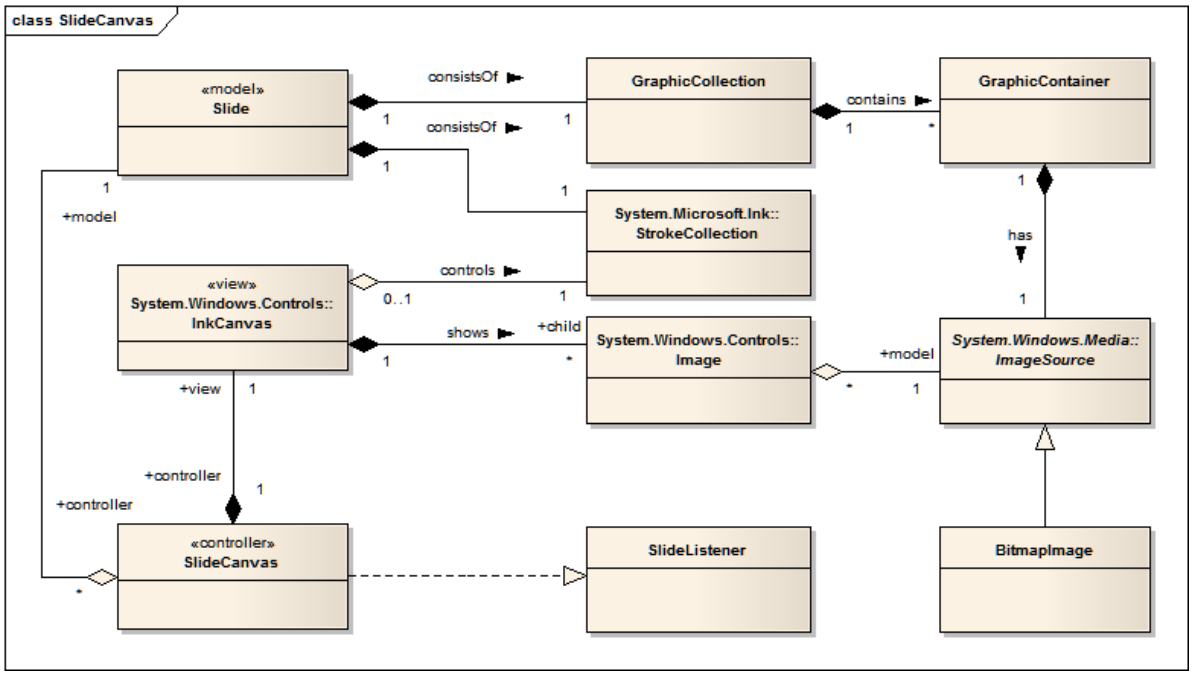

Abbildung 3.2.: Klassendiagramm der aktuellen internen `Slide`-Zusammensetzung und deren Darstellung durch ein `SlideCanvas`-Objekt.

wird, und ein `Slide`-Objekt, das das darzustellende *Model* ist. Das *Model* besteht aus einem `StrokeCollection`- und einem `GraphicCollection`-Objekt. Über den *Controller* erhält die *View* die darzustellende `StrokeCollection` des *Models*.

Die `InkCanvas`-Klasse kann die `StrokeCollection` einbinden und darstellen. Änderungen an einem `StrokeCollection`-Objekt werden über das bereits vorhandene `StrokesChanged`-Event automatisch an alle Abonnenten des Events gemeldet. Da ein `InkCanvas`-Objekt beim Anfügen eines `StrokeCollection`-Objekts das Event abonniert, erfährt es auch alle Änderungen an den Strichen und übernimmt diese.

Die `GraphicCollection` kann nicht so unproblematisch dargestellt werden wie die `StrokeCollection` der `InkCanvas`-Klasse. Die in der `GraphicCollection` enthaltenen `GraphicContainer`-Objekte müssen einzeln zur *View* hinzugefügt werden. Es werden aber nicht die `GraphicContainer`-Objekte hinzugefügt, sondern aus dem enthaltenen `ImageSource`-Objekt wird ein `Image`-Objekt erstellt. Dieses wird zur `Children`-Eigenschaft der *View* hinzugefügt. Zum Laden von Bildern, die durch die abstrakte Klasse `ImageSource` abgebildet werden, wird die Implementation `BitmapImage` von `ImageSource` verwendet.

Die `SlideCanvas`-Klasse implementiert das Interface `SlideListener`, mit dem Änderungen an `Slide`-Objekten abgefangen werden können. Änderungen an einer `Slide` schließen nur Änderungen an `GraphicContainer`-Objekten ein.

3.3. Zusammenfassung

In diesem Kapitel wurden einige Klassen und Vorgänge in Foils vorgestellt. Sie sind für den weiteren Verlauf dieser Arbeit wichtig und kommen im folgenden Kapitel zum Tragen. Im folgenden Kapitel wird ein genauerer Blick auf das Speicherproblem von Foils geworfen und behoben.

4. Analyse und Korrektur des Speicherverbrauchs

Alle Änderungsanforderungen, die in diesem Abschnitt erwähnt werden, sind im Anhang A zu finden. Dort ist auch eine Legend für die verwendeten Schlüsselbegriffe enthalten. Die wichtigsten Änderungsanforderungen sind A.1.1 und A.1.2 und werden in diesem Kapitel behandelt.

A.1.1 ✓ (+) **B**: Nach längerem Betrieb steigt der Speicherverbrauch massiv an, bis kein Speicher mehr zur Verfügung steht. **L**: Die Ursache hierfür muss analysiert und behoben werden.

A.1.2 ✓ (+) **B**: Die vorhandene Undo-Funktion ist wahrscheinlich die Ursache des zu hohen Speicherverbrauchs. **U**: Es werden evtl. zu viele Undoschritte gespeichert. **L**: Es muss untersucht werden, ob die Undo-Funktion für den hohen Speicherverbrauch verantwortlich ist. Ist dies der Fall, muss die Anzahl der gespeicherten Undoschritte reduziert oder die Undo-Funktion komplett aus dem Programm entfernt werden.

Um die aufgeführten Probleme zu lösen, müssen mehrere Teilaufgaben bearbeitet werden:

- Ermitteln des Speicherverbrauchs.
- Analysieren des Speicherverbrauchs.
- Beheben des zu hohen Speicherverbrauchs.

Diese Aufgaben werden nacheinander durchgeführt, wobei dies iterativ geschehen muss, damit der Erfolg dieser Maßnahmen kontrolliert und gegebenenfalls neue oder nicht entdeckte Auswirkungen gefunden werden können. Außerdem müssen alle Messungen mit dem vorhandenen Tablet PC durchgeführt werden, damit die gleichen Bedingungen wie bei einer realen Präsentation mit dem Tablet PC gegeben sind.

4.1. `Replay()`-Funktion

Um den genauen Speicherverbrauch feststellen zu können, wäre es notwendig, immer wieder eine Präsentation von Hand durchzuführen, wobei es sich bei jeder Ausführung zumindest um eine vergleichbare Präsentation handeln müsste. Solch eine

Durchführung wäre zu zeitintensiv. Außerdem ist es unmöglich, eine immer vergleichbare Präsentation zu erstellen. Deswegen ist eine `Replay()`-Funktion erstellt worde.

Diese `Replay()`-Funktion soll eine bestehende Präsentation laden und diese dann von der ersten bis zur letzten Folie Strich für Strich abspielen und dabei existierende Strukturen des Programms nutzen. Somit ist es möglich das Verhalten des Programms genau zu studieren und an einem immer gleich ablaufenden Beispiel zu wiederholen.

Um solch eine Wiederholung zu ermöglichen, lädt die `Replay()`-Funktion eine bereits bestehende Präsentation. Alle notwendigen Daten werden, bevor das Abspielen beginnt, in entsprechende Repräsentationen überführt, sodass kein unnötiger Zeitverbrauch während der Ausführung entsteht. Dies wiederum bedeutet, dass alle `Stroke`-Objekte zunächst skaliert und dann ausgelesen werden müssen. Das Auslesen kann leider nicht umgangen werden, weil die *closed-source* API von Microsoft keine einfachen Funktionen bietet, `Stroke`-Objekte von einem `Ink`-Objekte in ein anderes `Ink`-Objekt einzufügen.

Damit eine abgespielte Präsentation sich ähnlich verhält wie eine reale Präsentation, wird ein `Stroke` nach dem anderen in das `Ink`-Objekt der Zeichenfläche eingefügt. Die visuelle Rückmeldung, die normalerweise beim Schreiben entsteht, erfolgt bei diesem Vorgang nicht. Um eine visuelle Rückmeldung zu erlangen, müssen die entsprechenden Striche explizit mit einer Zeichenfunktion auf die Zeichenfläche gezeichnet werden. Bei den restlichen Operationen wird auf vorhandene Funktionen und Kommando-Objekte zurückgegriffen. Die verwendeten Kommando-Objekte sind ausschließlich Objekte der Klassen `InkChangeOperation` und `InsertSlideOperation`[1]. Um die korrekte Ausführung der Kommando-Objekte zu gewährleisten, wurden sie an den implementierten Undo-Stack übergeben und dadurch ausgeführt.

Laut [Str09] enthält .Net keine Klasse, die genaue Zeitmessungen im Milli- bis Nanosekundenbereich zulässt. Aus dem Grund wurde die Klasse `HighPerfTimer`, die ebenfalls in der Quelle enthalten ist, für die Messungen verwendet. Sie enthält die benötigte Funktionalität.

Die Ausführungszeit jeder `InkChangeOperation` wurde gemessen und entsprechend der Speicherbedarf des gesamten Programms bestimmt. Dabei wurde zum einen der virtuelle Speicher und der tatsächliche verwendete Programmspeicher gemessen. Die Ergebnisse sind direkt in eine CSV (Comma Separated Values)-Datei geschrieben und in Form von Diagrammen ausgewertet worden.

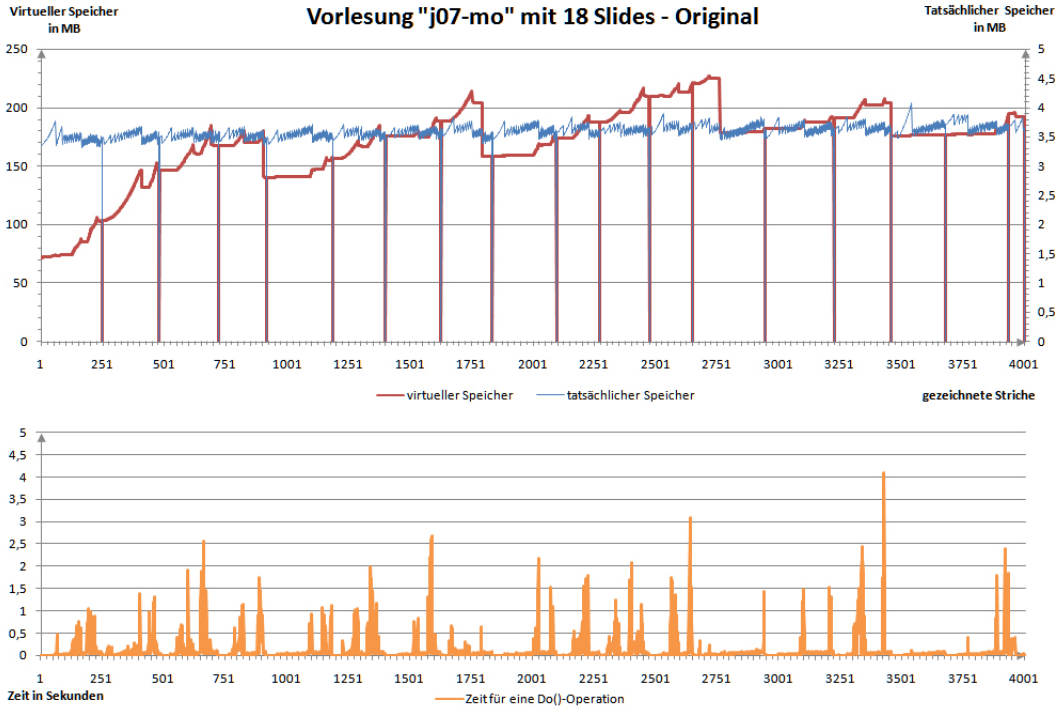

Abbildung 4.1.: Diagramm zum Zeitbedarf und Speicherverbrauch der unveränderten Foils Version.

4.2. Messungen und deren Auswertung

Die Messungen wurden mit insgesamt vier Präsentationen, mit jeweils unterschiedlicher Anzahl an Folien durchgeführt. Die verwendeten Präsentationen sind während Vorlesungen in den letzten Jahren entstanden. Die Ergebnisse der Messungen sind aufgearbeitet worden, sodass sie als Diagramme zur Verfügung stehen. Eine der erstellten Graphiken, z.B. Abb. 4.1, beinhaltet hierbei je zwei Diagramme. Das obere Diagramm zeigt den Speicherverbrauch, das untere Diagramm die Dauer einer ausgeführten Operation. Die Skalierungen können dabei variieren. Im oberen Diagramm sind zwei Datensätze untergebracht, die zueinander unterschiedlich skaliert sind. Die x-Achse entspricht in beiden Diagrammen getätigten `Do()`-Operationen. `Do()`-Operation ist in diesem Fall jede Operation, die beim Ausführen der `Do()`-Funktion des Undo-Stacks, anfällt.

Auffallend ist, dass im Speicherdiagramm immer wieder Einbrüche vorhanden sind. Bei diesen Einbrüchen handelt es sich aber nicht um Speichereinbrüche. Sie signalisieren lediglich den Beginn einer neuen Folie. Der anfängliche virtuelle Speicherverbrauch einer Präsentation (rot) ist normalerweise wesentlich niedriger als hier zu sehen und liegt bei ca. 25 MB. Die geladenen Foliensätze, die zum Abspielen benötigt

[1] `InsertSlideOperation` erstellt eine neue `Slide` und fügt sie hinter die gerade verwendete `Slide` ein.

werden, kommen als zusätzlich verbrauchter Speicher hinzu und verursachen so die Diskrepanz. Die Diskrepanz beschleunigt das Auftreten des zu behandelnden Phänomens, das durch zu hohen Speicherverbrauch entsteht. Dadurch wird mit steigender Laufzeit das Programm zunehmend verlangsamt.

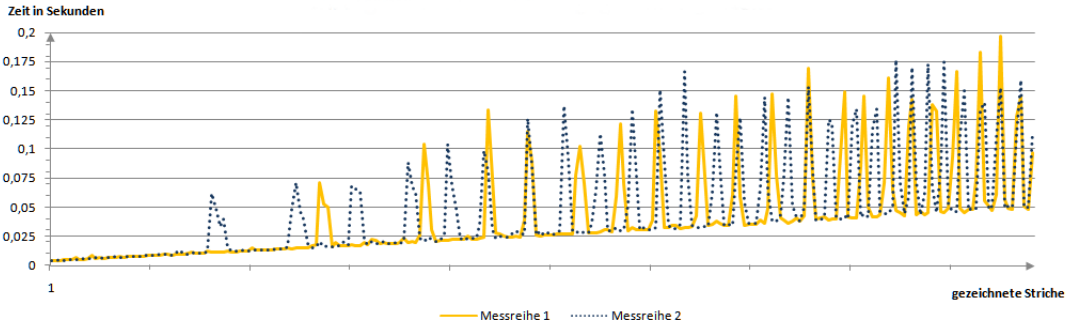

Abbildung 4.2.: Gegenüberstellung zweier Messreihen der gleichen Präsentation.

Wie schon erwähnt wurde, ist die Tablet PC SDK größtenteils nicht in *managed code* geschrieben [JS03]. Deshalb kann mit der Methode `long GetTotalMemory(bool)` des GC nicht der größte Teil des tatsächlichen Speichers (blau dünn) ermittelt werden. Aus dem Grund sollte auch nicht der tatsächliche Speicher als Anhaltspunkt für die weiteren Betrachtungen genommen werden. Der tatsächliche Speicher ist dennoch aufgeführt. Bei der neuen Umsetzung von Foils ist der tatsächliche Speicher wieder als Referenz verwendbar, weil die WPF größtenteils in *managed code* geschrieben ist.

Im Diagramm, in dem die gemessenen Zeiten dargestellt werden, sind viele Peaks zu sehen. Um Fehler handelt es sich dann, wenn die Peaks im Toleranzbereich von $\pm 0,3$ Sekunden zu ihren benachbarten Messungen liegen. Die einzelnen Fehler, die im Diagramm auftauchen, sind allein mittels des Diagramms selbst nicht zu erklären. Es ist aber wahrscheinlich, dass es sich hier um Messfehler handelt, die durch die Teilung der Prozessorzeit mit anderen Prozessen entstanden sind. Leider konnte nicht zurückverfolgt werden, welcher Prozess der Verursacher ist. Diese Fehler tauchen auch nicht immer an der gleichen Stelle auf, was in der Abb. 4.2 zu sehen ist. Peaks umfassen mehrere Operationen und nicht eine einzelne Operation.

Es befinden sich weitere Diagramme im Anhang D auf Seite 93, die mit Hilfe anderer Präsentationen erstellt wurden.

4.2.1. Originalversion Foils

Bei der Betrachtung der ersten Messergebnisse in Abb. 4.1 ist auffällig, dass eine `Do()`-Operation teilweise bis zu vier Sekunden dauern kann. Außerdem steigt der

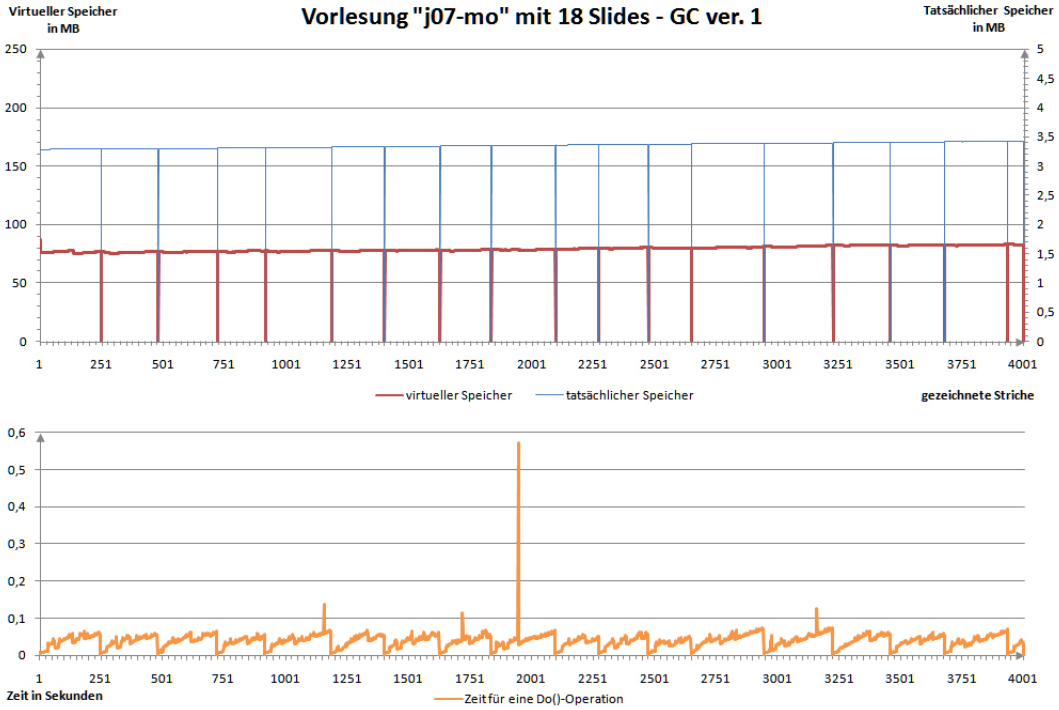

Abbildung 4.3.: Messergebnisse mit einem GC-Aufruf nach einer ausgeführten Operation.

virtuelle Speicher von ca. 70 MB innerhalb von etwa nur 750 Strichen auf ca. 180 MB an - dies entspricht ca. drei Folien. Allein diese Messergebnisse zeigen, dass Foils ungeheure Speicherbedarf hat. Auch ist festzustellen, dass mit jeder Speicherallokation im virtuellen Speicher der Zeitbedarf für eine Operation erheblich ansteigt. Vergleiche auch Abb. D.1 auf Seite 94. Dies führt zu der Vermutung, dass bei geringerer Speicherallokation im virtuellen Speicher, auch weniger Zeit beansprucht wird und das Programm schneller laufen würde.

Bei einer genaueren Betrachtung des Programmcodes ist festzustellen, dass die `Ink`-Objekte bei jeder Operation dreimal geklont werden, was entsprechende Speicherallokationen bedingt. Besonders durch die Größe des Undo-Stacks - standardmäßig eingestellt sind 50 Undoschritte - und fehlender regelmäßiger Aufräumarbeiten durch den GC, kann brachliegender Speicher nicht genutzt werden. Aus diesem Grund muss neuer Speicher beim Betriebssystem angefordert werden, was zusätzlich Zeit beansprucht.

Aufgrund dieses Ergebnisses ist eine zweite Messreihe erstellt worden, bei der der GC explizit am Ende der `Do()`-Funktion des Undo-Stacks aufgerufen wird.

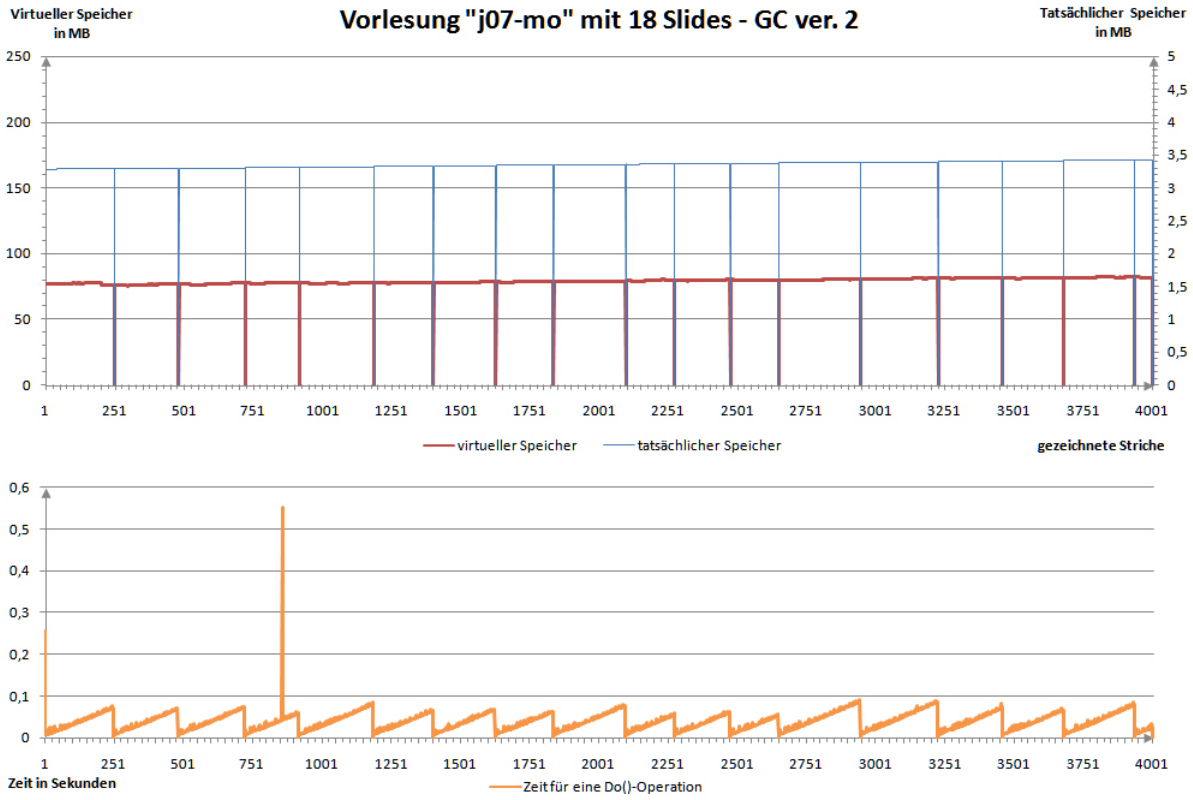

Abbildung 4.4.: Messergebnisse für den korrekten GC-Aufruf.

4.2.2. Auswirkungen eines Garbage Collector-Aufrufs

Nach dem Einfügen dieser Zeile Code, sieht das Speicherbild in Abb. 4.3 gleichbleibend aus. Jetzt ist auch zu erkennen, dass sich ein Sägezahnmuster beim Zeitdiagramm herauskristallisiert. Der einzelne große Ausreißer, der im Diagramm auftaucht, ist wahrscheinlich ein Prozess außerhalb des Programms, der für sich Prozessorzeit beansprucht hat.

Das Sägezahnmuster sieht noch sehr unregelmäßig aus. Diese Unregelmäßigkeit ist auf den nicht korrekten Aufruf des GCs zurückzuführen. Beim normalen Aufruf des GCs wird die *Finalizer*-Warteschlange nicht direkt, sondern parallel zur Programmausführung in einem Thread der *Finalizer*-Warteschlange abgearbeitet. So kann es vorkommen, dass der zuständige Thread zu ungünstigen Zeitpunkten aufgerufen wird.

Um diese Vermutung zu bestätigen, ist bei der dritten Messreihe die `GC.WaitForPendingFinalizers()`-Funktion des GC hinzugefügt worden.

4.2.3. Optimierung des GC-Aufrufs

Der Verlauf des Zeitdiagramms in Abb. 4.4 ist nun wesentlich ruhiger. Das Sägezahnmuster ist jetzt sehr deutlich zu sehen. Die einzelnen großen Messfehler bestehen immer noch, können aber ignoriert werden. Das jetzt deutlich zu sehende Muster zeigt, dass mit der wachsenden Anzahl von Strichen auch die Zeitdauer ansteigt. Beim Beginn einer neuen Folie ist die aufzuwendende Zeit für eine Operation kaum existent und wächst dann wieder an. Dieses Wachstum ist linear.

Die zu sehende Abhängigkeit von Strichen und Zeit sollte normalerweise nicht vorhanden sein. Da dieses Verhalten mit den durchgeführten Operationen zu tun hat, lohnt es sich die durchgeführte `Operation` und `InkChangeOperation` genauer zu betrachten.

4.2.4. InkChangeOperation - eine genauere Betrachtung

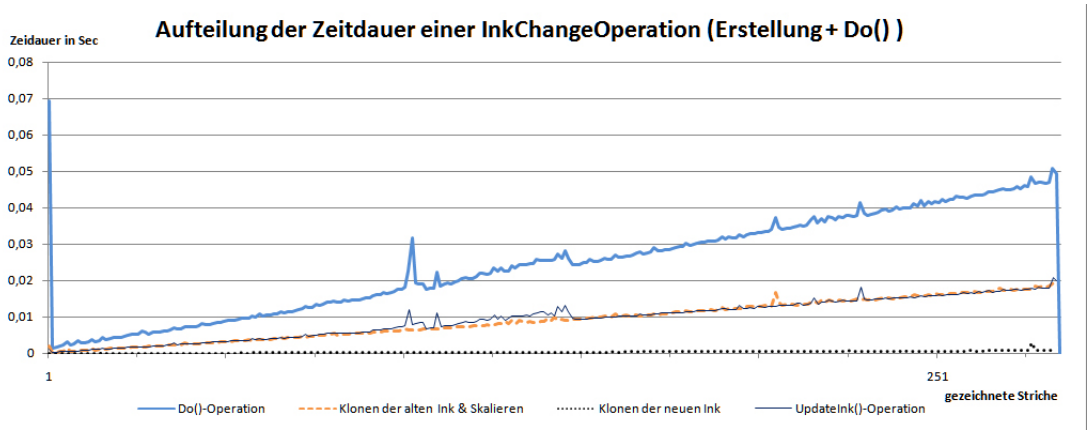

Abbildung 4.5.: Diagramm über die Zeitaufteilung mehrerer `InkChangeOperation`-Objekte.

Da das Speicherproblem nun nicht mehr im Mittelpunkt steht, sondern welche Operationen bei der Ausführung wie viel Prozent der Gesamtzeit beanspruchen, wurden die Ausführungszeiten der markantesten Codezeilen gestoppt. Für diesen Test wurde eine Folie mit vielen kleinen Strichen erstellt und abgespeichert. Diese Folie wurde dann mit Hilfe der `Replay()`-Funktion abgespielt. Hierfür wurde die Version mit den zwei GC-Aufrufen verwendet, da bei der entsprechenden Messreihe das sauberste Ergebnis für die Gesamtzeit zu sehen war.

Die markantesten Operationen bei der Ausführung sind wie erwähnt, das Klonen des aktuellen `Ink`-Objekts der Zeichenfläche, das Klonen und gleichzeitige Skalieren des `Ink`-Objekts der aktuellen Folie sowie die `UpdateInk()`-Funktion. `UpdateInk()` skaliert und klont wiederum das übergebene `Ink`-Objekt.

Beim Diagramm in Abb. 4.5 repräsentiert die oberste Linie (blau) die Gesamtzeit, alle anderen Linien stehen für Teilzeiten der Gesamtzeit. Nicht enthalten ist die restliche, nicht von den Teilzeiten abgedeckte Zeit. Die nicht enthaltene Zeitdauer fällt so gering aus, dass sie vernachlässigt werden kann.

Bei der Betrachtung des Diagramms fällt auf, dass die Kurven für das Klonen und Skalieren der alten `Ink`-Objekte (orange gestrichelt) sowie für das Ausführen der `UpdateInk()`-Funktion (dunkelblau dünn) sehr exakt aufeinander liegen. Der Grund dafür ist, dass nahezu die gleichen Operationen ausgeführt werden. Der Unterschied ist, dass die Methode das `Ink`-Objekt als neues `Ink`-Objekt des `InkOverlay`-Objekts setzt. Beide Operationen zusammengenommen ergeben ca. 80% der Zeit.

Die noch nicht erwähnte Kurve (schwarz gepunktet), die kaum Laufzeit in Anspruch nimmt, ist das reine Klonen des aktuellen `Ink`-Objekts der Zeichenfläche. Anhand dessen ist zu sehen, dass das Klonen nicht viel Zeit in Anspruch nimmt. Das Skalieren ist dagegen um einiges aufwändiger.

Der Grund des linearen Anstieges der Kurven ist somit ebenfalls leicht zu beantworten. Beim Skalieren und Klonen eines `Ink`-Objekts werden alle Elemente durchgegangen, was zur Folge hat, dass die Ausführungszeit von der Anzahl der enthaltenen `Stroke`-Objekte abhängig ist. Da über die Zeit immer mehr `Stroke`-Objekte erstellt werden, wächst die zum Zeichnen eines kompletten `Ink`-Objekts benötigte Zeits.

Das Problem der linear anwachsenden Zeit ist das ständige Skalieren der `Ink`-Objekte. Da der Grund des Skalierens auf der normalisierten Darstellungsform der `Slide`-Objekte beruht, ist es offensichtlich, dass hier Verbesserungsbedarf besteht. Außerdem ist das Speichern von redundanten Informationen - durch das Klonen von `Ink`-Objekten - keine geeignete Lösung. Auch die fehlende Verbundenheit der Zeichenfläche mit dem eigentlichen `Ink`-Objekt der aktuellen `Slide` ist eine Fehlkonstruktion.

4.2.5. Demonstration der Auswirkungen ohne Skalierung

Damit die Auswirkungen der Skalierung noch genauer zu sehen sind, wurde eine weitere Messreihe durchgeführt. Der Programmcode wurde so verändert, dass keine Skalierung mehr stattfindet.

Das Sägezahnmuster im Diagramm in Abb. 4.6 ist nicht mehr zu sehen. Bei starker Vergrößerung ist das Muster vorhanden, da noch `Ink`-Objekte geklont werden, was nur mit einem Aufwand von O(n) geschehen kann.

Leider ist diese Variante aus zwei Gründen ungünstig. Zum einen ist der Umstand nicht optimal, dass die `Ink`-Objekte immerzu geklont werden und somit - zwar nur noch in einem kleineren Rahmen - Speicher verbrauchen. Zum anderen müssen die

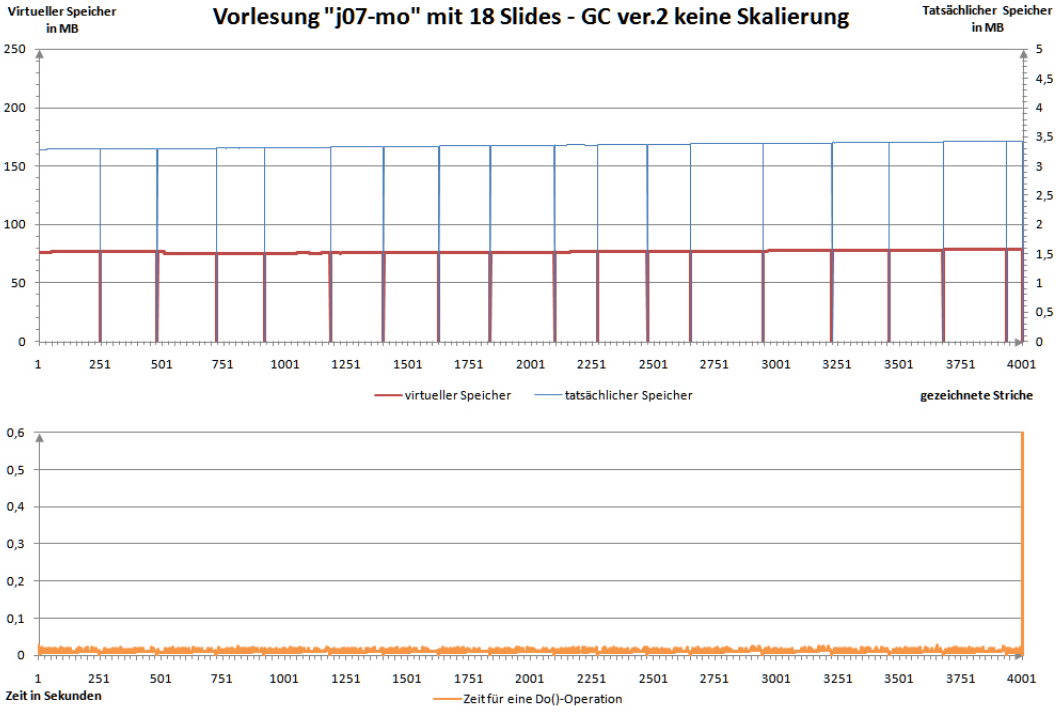

Abbildung 4.6.: Messergebnisse ohne Skalierung und mit korrektem GC-Aufruf.

Folien und somit die `Ink`-Objekte immer noch skaliert werden, damit sie richtig dargestellt werden können.

4.2.6. Neue Implementation

Wie in Abschnitt 3.2.1 auf Seite 17 erwähnt, wurde die `InkOverlay`-Klasse durch die `InkCanvas`-Klasse ausgetauscht. Dadurch ist es möglich ein `Stroke`-Objekt abzuspeichern statt zweier `Ink`-Objekte, die jetzt als `StrokeCollection` bezeichnet werden. Aufgrund der neue Implementation stimmen die ermittelten Messwerte nicht mehr und müssen neu ermittelt werden. Die Abb. 4.7 illustriert die neuen Messwerte.

Der Speicherverbrauch des virtuellen Speichers beginnt bei etwa 50 MB und steigt über den Verlauf der Messung bis auf ungefähr 100 MB an. Insgesamt werden um die 20 MB mehr als bei den alten Programmversionen verbraucht. Der Zeitbedarf für eine `Do()`-Operation ist sehr gering, konstant und liegt insgesamt unter einer Zehntel Sekunde.

Diese Implementation ist für die Durchführung einer Präsentation gut geeignet: Weder hoher Speicherverbrauch noch hoher Zeitbedarf zum Zeichnen eines neuen Strichs fallen nun an. Die neue Variante benötigt evtl. dennoch Verbesserungen, da der Speicherverbrauch, wenn auch nur unwesentlich, höher ist als bei der alten Version.

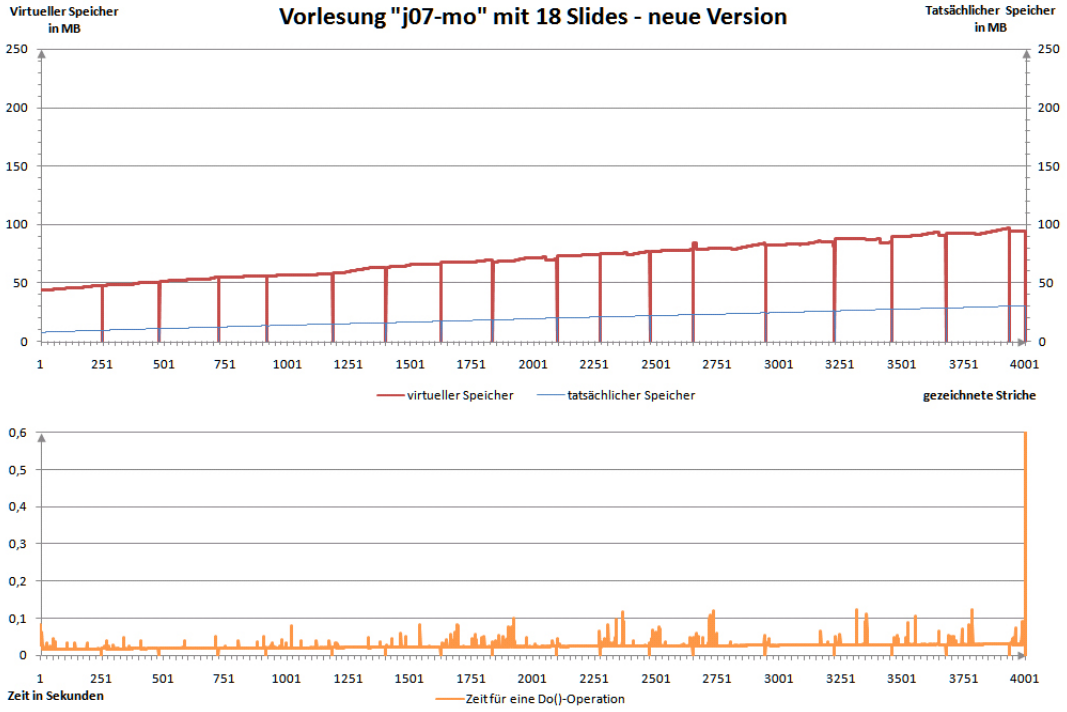

Abbildung 4.7.: Messergebnisse der neuen Implementierung von Foils.

	Original	GC ver.1	GC ver.2	GC ver.2 ohne Skal.	Neue Implementation
j07-mo	573,77	214,38	169,66	54,11	161,43
modelle	125,05	59,03	41,55	12,28	26,97
infb-0531	93,27	46,26	32,27	9,61	22,98
vwa-iw1	540,47	187,81	147,92	44,14	121,16

Tabelle 4.1.: Ausführungszeiten für verschiedene Versionen des Programms Foils und Präsentationen.

Zum Schluss ist in Übersichtstabelle 4.1 jede verwendete Präsentation in jeder Version mit ihren Ausführungszeiten aufgelistet.

4.3. Schlussfolgerung aus den Messergebnissen

Die anfänglich recht einfach und logisch erscheinende Idee - ein Vorher- und ein Nachherbild abzulegen, s. Abschnitt 3.1.4 - hat sich hiermit als Speicherfresser und Geschwindigkeitsbremse erwiesen. Mit zwei kleinen Zeilen Code kann zwar dieses Problem optimiert werden, doch das Resultat ist noch immer nicht optimal, weil ein Zeitaufwand von $O(n)$ vorliegt und unnötig Speicher verbraucht wird. Die Lösung des Problems ist dabei recht einfach: Anstatt zwei Kopien der Folien zu speichern, ist nur deren Differenz interessant, also zum Beispiel ein Strich.

Somit liegt das Problem zum einen an mangelnden Aufrufen des Garbage Collectors und an der zu Grunde liegenden Implementierung, die auf die Speicherverwaltung keine Rücksicht nimmt. Die Ursache dessen liegt wahrscheinlich bei der Tablet PC SDK, die nur eine unübersichtliche Dokumentation anbietet. Aus diesem Grund werden auch die Vorher- und Nachherkopien der `Ink`-Objekte angelegt, weil zum Hinzufügen bzw. Speichern von einzelnen `Stroke`-Objekten keine Hilfestellung angeboten wird.

Nachdem nun die Ursache des Speicherverbrauchs ermittelt und beseitigt ist, müssen die weiteren Anforderungen abgearbeitet werden. Die nachfolgenden drei Kapitel beschäftigen sich jeweils mit einem Teil der Anforderungen. Die Anforderungen sind in drei Teile aufgeteilt:

- Bug-Report,
- Change-Requests und
- Extension-Requests.

In den Kapiteln wird genauer auf das jeweilige Problem eingegangen und die umgesetzte Lösung vorgestellt.

5. Genaue Fehler und Lösungen

Alle Anforderungen, die besprochen werden, befinden sich zusätzlich im Anhang A.

In Abschnitt 4 wurden die Anforderungen A.1.1 und A.1.2 besprochen und deren Lösung vorgestellt. Nun werden die Anforderungen A.1.3 bis A.1.10 und A.1.12 bis A.1.14 behandelt. Die Anforderung A.1.11 wird im Abschnitt 6.3 besprochen. Es wird im Abschnitt 5.10 vorgegriffen und die Anforderung A.2.9 bearbeitet.

Bei der Behandlung der einzelnen Anforderungen wird die Ursache jedes Bugs beleuchtet und evtl. die Lösung näher erläutert. Der entsprechende Auszug der Anforderung aus dem Bugreport wird zuvor aufgelistet.

5.1. Navigation über das Drop-Down-Menü

A.1.3 ✓ (+) **B**: Bei wachsender Anzahl an Folien, wird die Navigation über das Drop-Down-Menü sehr langsam. **U**: Die Vorschaubilder der Folien werden stets neu generiert. **L**: Die Vorschaubilder sollen nur einmalig generiert werden oder durch eine Navigationsleiste abgelöst werden.

Die Vorschaubilder werden von dem Objekt `SlideSelectionItem` gezeichnet. Die Generierung der Vorschaubilder erfolgt bei einem Paint-Event. Dieses Event wird ausgelöst, wenn der Mauszeiger über die einzelnen Elemente fährt. Es wird sogar zweimal ausgelöst: Einmal, wenn der Mauszeiger gerade auf das `SlideSelectionMenuItem` gelangt ist, und ein zweites Mal, wenn er es wieder verlässt.

Da das DropDown-Menü in der jetzigen Version von der Navigationsleiste ersetzt ist, wurde diese Anforderung nicht weiter umgesetzt. In der aktuellen Version werden zwar auch Vorschaubilder für die Folien in der Navigationsleiste generiert, dies wird aber von dem verwendeten `InkCanvas`-Objekt sehr effizient gelöst. Aus dem Grund werden auch für diese Variante keine Vorschaubilder explizit generiert.

5.2. Das Verhalten der Präsentationsansichten

A.1.4 ✓ (+) **B**: Wenn eine Folie *F*, wobei *F* nicht die letzte Folie sei, im Bearbeitungsfenster ausgewählt ist und eine neue Folie *N* hinzugefügt wird, wechseln alle

Fenster auf *N* und *N* wird vor *F* eingefügt. **L**: *N* soll hinter *F* eingeordnet werden und die Vorgängerfolie *F* links und *N* rechts im Präsentationsfenster dargestellt werden.

Dieser Bug besteht aus zwei Problemen. Das erste ist, dass die neue `Slide` vor einer bestimmten `Slide` eingeordnet wird. Das zweite Problem ist, dass nicht die gewünschten `Slide`-Objekte in den Präsentationsansichten angezeigt werden.

5.2.1. Einfügen einer `Slide`

Das Objekt `SlideSequence` verwaltet alle `Slide`-Objekte. Bei ihm wurde nur die Möglichkeit eingeräumt, dass eine `Slide` vor einer bestimmten `Slide` eingeordnet oder ans Ende der `SlideSequence` angehangen werden kann. Das Kommando-Objekt wurde im Abschnitt 3.1.2 auf Seite 14 bereits ausführlich beschrieben. Die `InsertSlideOperation`-Klasse ermöglicht das Einfügen einer `Slide` an einer bestimmten Position. Aufgrund der Implementation der `SlideSequence` wurde bei der `InsertSlideOperation` die gleiche Operation zum Einfügen verwendet. Dadurch ist das beobachtete Verhalten zu erklären.

Um dieses Verhalten zu korrigieren wurde das Einfügen einer `Slide` hinter einer bestimmten `Slide` in der Klasse `SlideSequence` zur Verfügung gestellt und entsprechende Einträge in `InsertSlideOperation` geändert.

5.2.2. Darzustellende Folien in den Präsentationsansichten

Das nächste Problem umfasst die Präsentationsansichten links (P1) und rechts (P2) im Präsentationsfenster und das Bearbeitungsfenster mit der Bearbeitungsansicht (C).

P1 und P2 werden von dem Präsentationsfenster `PresentationForm`, welches das Interface `SlideNumberChangeListener` implementiert, kontrolliert. Das Interface deklariert die Methoden `void SlideAdded(Slide)`, `void SlideRemoved(Slide)` und `void AllSlidesRemoved()` und ist nach dem *Observer-Pattern* [GHJV94] aufgebaut. Diese Methoden informieren die Observer gezielt, welche `Slide` eingefügt oder gelöscht wurde. `void AllSlidesRemoved()` wird nur aufgerufen, wenn eine Präsentation geladen wird. Mit Hilfe des Interfaces kontrolliert `PresentationForm` die untergeordneten Präsentationsansichten. Die `SlideSequence` ist dabei das beobachtete Subjekt.

Bei der Implementierung wurden zwei Regeln festgelegt.

1. Stellt eine der Ansichten P1 oder P2 eine der letzten beiden `Slide`-Objekte dar und wird eine `Slide` am Ende eingefügt, so zeigt P2 im Folgenden die von P1 zuvor dargestellte `Slide` und P2 die neue `Slide`.

2. Wird keine `Slide` am Ende angefügt, sondern an einer beliebigen anderen Stelle, wechseln beide Ansichten P1 und P2 auf die gerade eingefügte `Slide`.

Diese Regeln orientieren sich nicht an der zu beobachtenden Bearbeitungsansicht und decken nicht den Fall ab, dass C evtl. die `Slide` wechselt, ohne dass eine neue `Slide` eingefügt wird.

Um den Fall abzudecken, ist eine abstrakte Klasse `ViewedSlide` erstellt worden. Sie sorgt dafür, dass für eine Ansicht die Nummer der Folie und die Folie selber bereit gehalten werden. Sie implementiert die Eigenschaften `SlideNumber`, `Slide` und `RelativeSlideNumber`. `SlideNumber` gibt die darzustellende Folienummer zurück und `Slide` die Referenz zur darzustellenden Folie. Diese beiden Eigenschaften werden synchron gehalten, d.h., dass die `SlideNumber` der `Slide` entspricht und umgekehrt. Die Eigenschaft `RelativeSlideNumber` stellt die Folienummer relativ zur Kontrollansicht bereit. Relativ bedeutet in diesem Kontext von der Position der Kontrollansicht und der darzustellenden Präsentationsansicht abhängig. Existiert keine Kontrollansicht wird der aktuelle Wert von `SlideNumber` zurückgegeben. Außerdem wurde das *Observer-Pattern* implementiert. So können sich interessierte Ansichten bei der entsprechenden `ViewedSlide` registrieren und werden bei Änderungen benachrichtigt. Für die Klasse `ViewedSlide` gibt es zwei Implementationen: `PresentedSlide` und `ControledSlide`.

`ControledSlide` implementiert die drei Eigenschaften aus `ViewedSlide`. Dabei wurde ein weiterer Aspekt genauer behandelt: Es kann vorkommen, dass der Benutzer auf eine Folie wechseln und dabei immer noch die Präsentationsansichten auf ihren aktuellen Positionen lassen möchte. Aus dem Grund führt die `ControledSlide`-Klasse eine etwas andere Logik ein. Sie speichert generell zwei Foliennummern ab statt einer. Eine Foliennummer wird für die Eigenschaft `SlideNumber` verwendet. Die zweite wird beim Aufruf der Eigenschaft `RelativeSlideNumber` zurückgegeben. Beide Foliennummern werden auch synchron gehalten, d.h. beim Ändern der `SlideNumber`-Eigenschaft wird auch der Wert von `RelativeSlideNumber` geändert. `RelativeSlideNumber` wird in der Klasse `PresentedSlide` für die genaue Bestimmung der darzustellenden `Slide` verwendet. Damit ein Wechsel zu einer anderen `Slide` möglich ist, ohne dass sich alle von dieser `ControledSlide` abhängigen `PresentedSlide` ändern, ist eine Methode `setSlideNumberOnlyForObservers(int)` hinzugefügt worden. Sie bewirkt, dass sich nur die Eigenschaft `SlideNumber` ändert und nicht `RelativeSlideNumber`.

`PresentedSlide` implementiert ihr eigenes Verhalten. Zusätzlich zu geerbeten Eigenschaften gibt es die `ViewedSlide`-Eigenschaft. Sie speichert eine `ViewedSlide` ab und registriert sich als Listener bei der `ViewedSlide`. `SlideNumber` speichert ebenfalls die anzuzeigende Foliennummer ab. Dabei bleibt anzumerken, dass intern auch ein ungültiger Wert für die Eigenschaft `SlideNumber` abgespeichert werden kann.

Ein ungültiger Wert liegt vor, falls keine Folie mit der entsprechenden Nummer in der `SlideSequence` verfügbar ist. Ist der Wert ungültig, wird über die `RelativeSlideNumber`-Eigenschaft der gespeicherten `ViewedSlide` versucht, die Foliennummer relativ zur Kontrollansicht abzufragen. Falls auch bei diesem Versuch eine ungültige Foliennummber als Ergebnis angeboten wird, wird eine leere Folie dargestellt. Die Eigenschaft `RelativeSlideNumber` erfragt den Wert der `RelativeSlideNumber`-Eigenschaft des gespeicherten `ViewedSlide`-Objekts. Falls eine ungültige Foliennummer existiert, wird das ermittelte Ergebnis minus eins zurückgegeben, sonst nur das Ergebnis. Das Subtrahieren soll verhindern, dass die Slide nochmals dargestellt wird, wenn sie bereits durch eine `PresentedSlide` dargestellt wird. Bei einer Null-Referenz auf das gespeicherte `ViewedSlide`-Objekt wird die interne Foliennummer als Ergebnis geliefert.

Da diese Klassen oft genutzt werden, sind Instanzen dieser Klasse in den Metadaten der aktuellen Präsentation abgelegt. Die Instanzen werden so erstellt, dass es ein `ControledSlide`-Objekt gibt und zwei `PresentedSlide`-Objekte. Dabei verweist ein `PresentedSlide`-Objekt auf das `ControledSlide`-Objekt und stellt somit die Folie für die rechte Präsentationsansicht bereit. Das andere `PresentedSlide`-Objekt verweist auf das `PresentedSlide`-Objekt für die rechte Präsentationsansicht.

Damit C, P1 und P2 sich entsprechend der `ViewedSlide`-Objekte verhalten, müssen sie nur die entsprechende `ViewedSlide` einbinden. Durch das *Observer-Pattern* kann eine Änderung des darzustellenden `Slide`-Objekts abgefangen werden und entsprechend die `Slide` gewechselt werden.

5.3. Die Lasso-Funktion

A.1.5 ✓ (N) **B**: Die Lasso-Funktion wird bei längerer Benutzung langsam. **U**: Zu oft wird die aktuelle Position abgetastet und werden die dabei ermittelten Punkte neugezeichnet. **L**: Die Anzahl der Abtastungen muss reduziert und so selten wie möglich neugezeichnet werden.

Die Lasso-Funktion ist eine fest implementierte Funktion des `InkOverlay`-Objekts. Sie steht zur Verfügung, indem die Eigenschaft `EditingMode` auf den festdefinierten Wert `Select` gesetzt wird. Es handelt sich dabei um die `InkOverlayEditingMode` Enumeration, die verschiedene Bearbeitungsmöglichkeiten bietet. Sie ist zu vergleichen mit dem bekannten `InkCanvasEditingMode`, enthält aber nur die Optionen `Select`, `Ink` und `Delete` Das Problem dieser Funktion ist, dass die Kontrollpunkte immer wieder neu gezeichnet werden. Dies kann bei übermäßig langem Selektieren mit dem Lasso festgestellt werden. Nach einiger Zeit ist zu beobachten, dass die Kon-

trollpunkte zu flimmern beginnen. Besonders stark tritt dieser Effekt bei sich überlagernden Punkten zu Tage.

Um dieses Problem zu lösen, müssten die vom `InkOverlay`-Objekt mitgelieferten Funktionen komplett ersetzt werden. Dies beinhaltet das Malen aller Kontrollpunkte, die Selektion der Striche nach dem Absetzen des Stifts und evtl. alle Funktionalitäten wie Verschieben, Skalieren, etc., die auf den ausgewählten Strichen durchgeführt werden können.

Durch die Verwendung der `InkCanvas`-Klasse ist dieses Problem nicht mehr aufgetreten. Die Implementation der Lasso-Funktion der `InkCanvas`-Klasse ist schneller als die der `InkOverlay`-Klasse. Ein Punkt wird erst dann gezeichnet, wenn er sich in einem bestimmten Abstand zum vorherigen, gesetzten Punkt befindet. Dadurch müssen nicht mehr so viele Kontrollpunkte gezeichnet werden und die Lasso-Funktion ist effizienter.

5.4. Merken der zuletzt bearbeiteten Folie

A.1.6 ✓ (N) **B**: Wenn eine Präsentation geöffnet ist, existiert keine einfache Möglichkeit direkt an die Stelle der letzten Bearbeitung zurückzukehren. **L**: Dies soll durch die Speicherung des derzeitigen Bearbeitungspunkts und der Folienpositionen im Präsentationsfenster gelöst werden.

```
<?xml version="1.0" encoding="utf-8"?>
<Presentation>
  <MetaData>
    <DocumentName />
    <VirtualInkOverlaySize X="500" Y="750" />
  </MetaData>
  <SlideSequence>
    <Slide>
      <Ink> ...
      </Ink>
    </Slide>
    <Slide>
      <Ink> ...
      </Ink>
      <GraphicContainers>
        <GraphicContainer FileName="D:\EigeneDateien\Presentations\Test\
            Images\AM4.jpg">
          <Position X="0.067961165048543687" Y="0.077058823529411763" />
          <Size X="0.86407766990291268" Y="0.76880294117647063" />
        </GraphicContainer>
```

```
20      </GraphicContainers>
21    </Slide>
22  </SlideSequence>
23 </Presentation>
```

Listing 5.1: XML-Gerüst einer alten Präsentationsdatei von Foils

Beim Laden einer Präsentation wird eine Datei im XML-Format geladen. Diese XML-Datei ist wie in Listing 5.1 aufgebaut.

Damit die letzten Einstellungen der Bearbeitungsansicht und der zwei Präsentationsansichten auch abgespeichert werden, müssen weitere Daten zur XML-Datei hinzugefügt werden. Der `MetaData`-Knoten eignet sich am Besten für diese Aufgabe. Hier wir der Unterknoten `LastViewedSlide` hinzugefügt. Innerhalb des Knotens `LastViewedSlide` gibt es nur noch Knoten mit dem Namen `View`. Der Knoten `View` wird als Klasse `ViewedSlide` abgebildet. Ein `View`-Knoten hat zwei erforderliche und eine mögliches Attribut. Die zwei erforderlichen sind einmal `Type` und `Index`. Das mögliche Attibute lautet `Viewport`.

Das Attribut `Type` kann entweder die Werte „ControlView”, die Bearbeitungsansicht, oder „PresentationView”, eine Präsentationsansicht, enthalten. Wenn das Attribut auf „ControlView” gesetzt ist, wird der „View”-Knoten von einem `ControledSlide`-Objekt abgebildet, sonst von einem `PresentedSlide`-Objekt.

```
1  <?xml version="1.0" encoding="utf-8"?>
2  <Presentation>
3    <MetaData>
4      <DocumentName />
5      <VirtualInkOverlaySize X="500" Y="750" />
6      <LastViewedSlide>
7        <View Type="ControlView" SlideNumber="1" />
8        <View Type="PresentationView" SlideNumber="4" Viewport="Left" />
9        <View Type="PresentationView" SlideNumber="-1" Viewport="Right" />
10     </LastViewedSlide>
11       ...
12   </MetaData>
13     ...
14 </Presentation>
```

Listing 5.2: XML-Gerüst einer Präsentationsdatei von Foils.

Das Attribut `SlideNumber` hat als Wert den darzustellende Folienindex, der von null aus gezählt wird. Ist der Wert ungültig, also nicht in `SlideSequence` enthalten, wird er auf den Wert minus eins in dem entsprechenden `ViewedSlide`-Objekt gesetzt. Das

bedeutet, dass eine Folie relativ zur Folie in der Bearbeitungsansicht dargestellt werden soll. Der Wert des Attributs wird der Eigenschaft `SlideNumber` zugewiesen.

Das Attribut `Viewport` bezeichnet die Position im Präsentationsfenster. Es gibt nur zwei Werte „Left" und „Right", für die links und für die rechts dargestellte Folie. Sie wird nur wirksam, wenn das Attribut `Type` auf „PresentationView" gesetzt ist.

Ein Beispiel ist im Listing 5.2 zu sehen. In der abstrakten Klasse `ViewedSlide` sind Speicher und Ladefunktionen vorgesehen, die XML-Daten generieren und wieder lesen können.

5.5. Öffnen aus einer bestehenden Präsentation

A.1.7 ✓ (N) **B**: Das Öffnen einer Präsentation aus einer bestehenden Präsentation heraus führt zu einem Fehler und ist nur auf Umwegen möglich. **L**: Dieser Fehler soll korrigiert werden.

Beim Öffnen aus einer bestehenden Präsentation passiert Folgendes: Die neue Präsentation wird aus der XML-Datei geladen. Zuerst werden alle Folien in der `SlideSequence` angefügt. Es wird eine `Slide` nach der anderen erstellt und mittels der vorhandenen Methode `void AddSlide(Slide)` zur `SlideSequence` hinzugefügt. Nach dem Hinzufügen wird noch die Methode `void NotifySlideAdded()` ausgeführt. Wie in Abschnitt 5.2.1 erwähnt, erhalten so alle Observer die neue `Slide` und wechseln die Ansicht. Einer der Observer ist die Klasse `ControlView`.

Dies hat zur Folge, dass jede `Slide` beim Einfügen einmal komplett gezeichnet wird, bevor die nächste `Slide` eingefügt werden kann. Besonders bei großen XML-Dateien führt dies zu langen Ladezeiten. Leider konnte bis jetzt noch kein Absturz beim Öffnen einer weiteren Präsentation reproduziert werden, was der eigentliche Hintergrund dieser Anforderung ist.

Die Lade-Funktion von Foils wurde dahingehend verbessert, dass sie jetzt alle `Slide`-Objekte hinzufügt, ohne ein Notify-Event auszulösen, und erst nach dem kompletten Laden die letzte geladene `Slide` als darzustellende `Slide` im `ControledSlide`-Objekt setzt. Außerdem registriert sich `ControlView` nicht mehr als Observer von `SlideSequence` sondern von `ControledSlide`.

5.6. Fenstersprünge bei der Benutzung von Undo/Redo

A.1.8 ✓ (N) **B**: Wird die Undo-Funktion verwendet, springt der Fensterausschnitt des Bearbeitungsfensters ganz nach oben. **L**: Die Ausschnittsposition soll auch nach Anwendung der Undo-Funktion unverändert sein.

Durch die Umstellung von der `InkOverlay`- auf die `InkCanvas`-Klasse, tritt dieses Verhalten nicht mehr auf.

5.7. Gemeinsames Maximieren und Minimieren aller Fenster

A.1.9 ✓ (N) **B**: Wenn das Programm minimiert ist und maximiert wird, wird das Präsentationsfenster nicht ebenfalls maximiert. **L**: Das Präsentationsfenster soll sich entsprechend dem Bearbeitungsfenster maximieren oder minimieren. Falls das Bearbeitungsfenster nur normal dargestellt wird und nicht maximiert ist, soll das Präsentationsfenster maximiert sein.

Um beide Fenster gleichzeitig zu maximieren, wenn nur eines maximiert wird, müssen die Ereignisse `Activated` und `Deactivated` vom Kontrollfenster abgefangen werden. Das Event `Activated` tritt auf, wenn das Fenster im Vordergrund steht. `Deactivated` tritt auf, wenn das Fenster nicht mehr im Vordergrund ist.

Mit den zwei Events `Activated` und `Deactivated` wurden zwei Fälle implementiert:

Im Fall, dass das Ereignis `Activated` auftritt und das Fenster nicht minimiert ist, muss das Präsentationsfenster maximiert werden.

Im Fall, dass das Ereignis `Deactivated` ausgelöst wird und das Fenster minimiert ist, muss auch das Präsentationsfenster minimiert werden.

5.8. Der Taskleisten-Eintrag

A.1.10 ✓ (N) **B**: In der Taskleiste befinden sich zwei Taskleisteneinträge des Programms. **L**: Es soll nur der Taskleisteneintrag des Bearbeitungsfensters in der Taskleiste vorhanden sein.

Der zusätzliche Taskleisteneintrage ist deaktiviert worden durch das Setzen der `ShownInTaskbar`-Eigenschaft auf false.

5.9. Fehlende Überlagerungen beim Export von Bildern

A.1.12 ✓ (N) **B**: Beim Exportieren der Folien als Bilder werden Überlagerungen von Markern als schwarze Balken dargestellt. **L**: Die Exportierfunktion soll verbessert werden, sodass diese Überlagerungen korrekt abgespeichert werden.

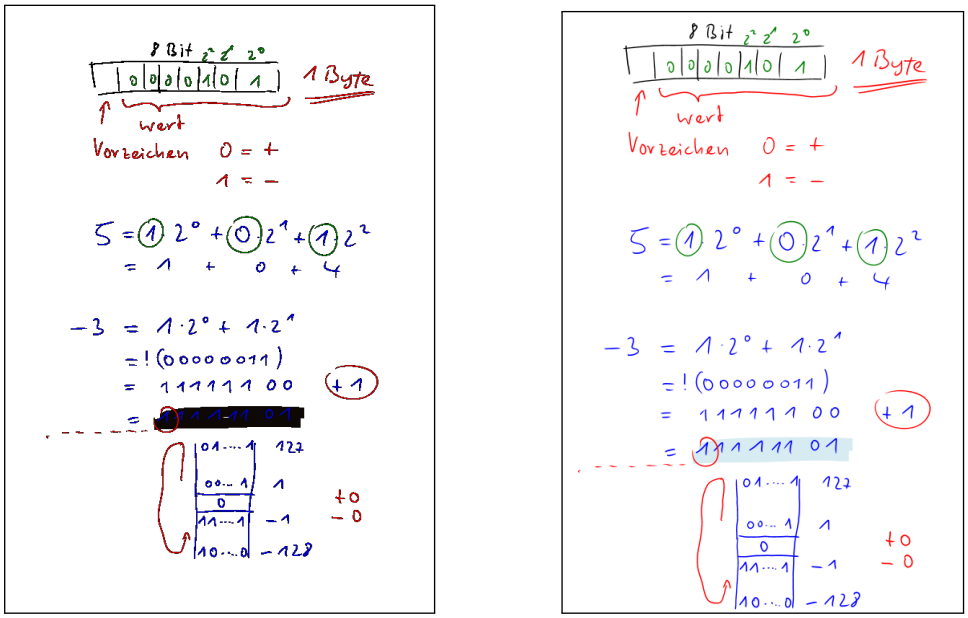

Abbildung 5.1.: Eine exportierte Folie mit Transparenzen. *Links*: Variante aus der Original Fassung von Foils; *rechts*: eigentliches Aussehen der Folie.

In Abb. 5.1 ist eine Gegenüberstellung von zwei Folien zu sehen, die dasselbe darstellen. Die linke Folie ist mit der alten Version exportiert worden. Die rechte Folie zeigt hingegen wie sie aussehen sollte. Die zusätzlichen Abstände zum Rand - in der linken Darstellung - sind bei der ursprünglichen Exportierfunktion voreingestellt gewesen, weshalb die Originalfolie (*rechts*) diese Ränder nicht besitzt.

Listing 5.3: Zu importierende Funktion aus der DLL-Datei: gdi32.dll.

```
1 [DllImport("gdi32")]
2 public static extern bool BitBlt(int hDC, int x, int y, int nWidth, int
      nHeight, int hSrcDC, int xSrc, int ySrc, int dwRop);
```

Das Fehlen von Transparenzen ist ein bekannter Bug in der Tablet PC SDK 1.7 [Mic09c]. Er ist auf die intern verwendeten Funktionen der Windows Forms Bibliothek zurückzuführen. Er kann gelöst werden, indem eine externe Funktion eingebunden wird, s. Listing 5.3. Dafür wird die Bibliotheksdatei „gdi32.dll" importiert.

Da nicht mehr die Tablet PC SDK 1.7 verwendet wird sondern die Implementationen in der WPF, die im Abschnit 3.2.1 auf Seite 17 vorgestellt wurde, treten die Fehler aus der Windows Forms Bibliothek nicht mehr auf. Deshalb wird der genaue Workaround an dieser Stelle ausgelassen.

5.10. Neuer Export-Dialog

A.1.13 ✓ (N) **B**: Die Benennung der Folien beim Exportieren ist ungünstig gelöst. **L**: Beim Exportieren soll es möglich sein, den Namen des zu exportierenden Folienpakets und den Start der Nummerierung festzulegen.

A.2.9 ✓ (N) **C**: Beim Exportieren der Folien als Bilder, werden alle Bilder in einem Verzeichnis mit den Präsentationsdateien abgespeichert. **L**: Alle Bilder sollen in einem eigenen Ordner abgespeichert werden.

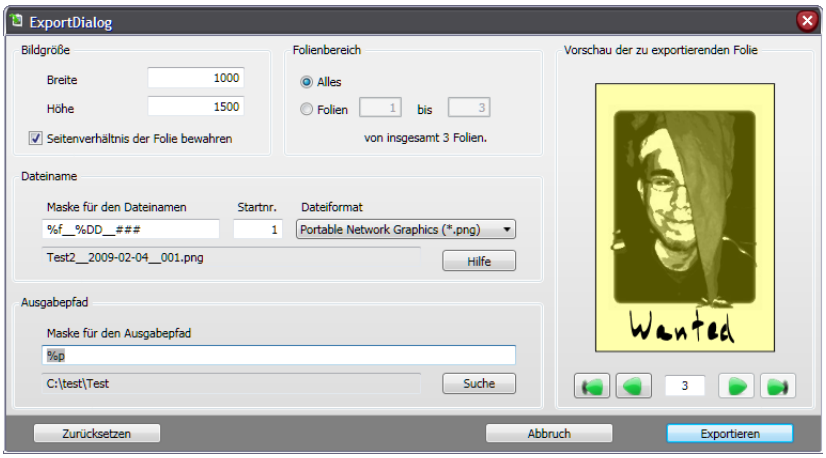

Abbildung 5.2.: Der Exportierdialog von Foils um aus Folien Bilder zu generieren.

Für die Umsetzung wurde ein Export-Dialog angelegt, s. Abb. 5.2. Das Dialog-Fenster ist in mehrere Bereiche eingeteilt. Die Bereiche sind *Bildgröße*, *Folienbereich*, *Dateiname*, *Ausgabepfad* und *Vorschau der zu exportierenden Folie*.

Der Bereich *Bildgröße* legt die Bildgröße der Ausgabebilder der zu exportierenden Folien fest. Zusätzlich ist es mittels einer Checkbox möglich, das Seitenverhältnis der Folien zu bewahren.

Der Bereich *Folienbereich* legt das Intervall an Folien fest, die exportiert werden sollen. Dabei gibt es zwei Optionen. Die erste bewirkt, dass alle Folien exportiert werden. Die zweite lässt eine genauere Eingrenzung des Intervalls zu.

Der Bereich unter *Bildgröße* und *Folienbereich* heißt *Dateiname*. In diesem Bereich kann eine Maske für den Dateinamen, die Startnummer, von der aus gezählt werden soll, und das Bildausgabeformat angegeben werden. Die Maskenoptionen sind im Abschnitt 5.10.1 genauer erläutert.

Der Bereich *Ausgabepfad* gibt an, in welches Verzeichnis die Bilder exportiert werden sollen. Mit dem Button „Suche” kann ein Dialogfeld geöffnet werden, in dem das gewünschte Verzeichnis ausgewählt werden kann.

Auf der rechten Seite befindet sich der letzte Bereich: *Vorschau der zu exportierenden Folie*. In diesem Bereich ist ein Vorschaubild zu sehen, das beim Öffnen des Exportierdialogs die erste zu exportierende Folie zeigt. Alle Folien innerhalb des ausgewählten Folienbereichs können angesehen werden. Hierzu werden die Navigationsbuttons unter dem Vorschaubild verwendet oder es wird direkt die gewünschte Foliennummer eingegeben.Das eingestellte Seitenverhältnis wird korrekt angezeigt.

Schließlich gibt es noch weitere drei Buttons namens „Zurücksetzen", „Abbruch" und „Exportieren" die ihrem Namen entsprechende Aktionen ausführen.

5.10.1. Maskenoptionen

Dem Benutzer wird durch die Verwendung von Masken viel Spielraum im Benennen der Folien gelassen.

Maskenoption	Beschreibung	Beispiel
%DD	Datum	2008-02-29
%TT	Uhrzeit	23-12
%Y	Jahr	2008
%M	Monat	02
%D	Tag	27
%h	Stunde	23
%m	Minute	12
%s	Sekunde	55
%f	Dokumentendateiname	Bsp
%t	Präsentationsname	Maskenbeispiele

Tabelle 5.1.: Generelle Maskenoptionen.

Maskenoption	Beschreibung	Beispiel
%p oder %P	Präsentationsordner	C:\Präsentation
%i oder %I	Bilderordner relativ zur Präsentation	Image
%~	Eigene Dateien Ordner	C:\Eigene Dateien

Tabelle 5.2.: Spezielle Muster für Ordner.

Maskenoption	Beschreibung	Beispiel
#	Platzhalter für eine Laufnummer.	aus „###" und 12 → „012"

Tabelle 5.3.: Spezielle Muster für Bilddateinamen zu exportierender Folien.

Eine Maske kann verschiedene Muster enthalten, s. die Tabellen 5.1, 5.2 und 5.3, dabei werden als Beispielwerte das Datum 27.02.2008, die Uhrzeit 23:12:55, der Dateiname „Bsp.foils" und der Titel der Präsentation „Maskenbeispiele" verwendet.

Bei den Maskenoptionen aus Tabelle 5.1 handelt es sich um allgemeine Optionen, die in jeder Maske verwendet werden können. Maskenoptionen aus Tabelle 5.2 können nur auf Ordnermasken angewandt werden. Die speziellen Optionen aus der Tabelle 5.3 sind nur auf die Dateinamensmasken im Exportierdialog anwendbar.

Zu der Maskenoption „#" muss noch gesagt werden, dass bei der Abwesenheit einer Raute automatisch die fortlaufende Nummer ans Ende der Zeichenkette angehangen wird. Falls mehre Gruppen von „#"-Zeichen in einer Zeichenketten enthalten sind, wird nur das erste Vorkommen für die Laufnummer verwendet, die restlichen Vorkommen werden aus der Zeichenkette entfernt.

5.11. Skalierung der Zeichenfläche

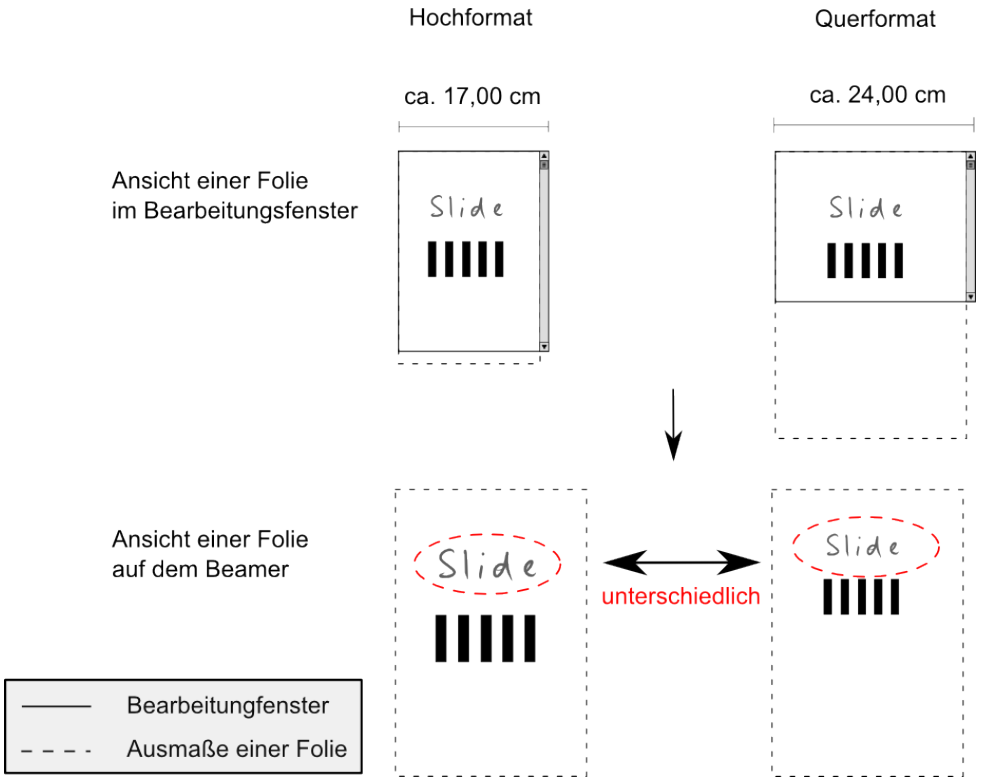

Abbildung 5.3.: Skalierungsverhalten bei unterschiedlicher Benutzung des Tablet PC's. Oben ist jeweils ein Bearbeitungsfenster zu sehen, unten das dargestellte Resultat.

Die Zeichenfläche skaliert sich entsprechend der Fensterbreite. Somit können ungewollte Effekte auftreten, wie in Abb. 5.3 zu sehen.

Für die Dozenten bedeutet dies, dass sich die Schriftgröße von Präsentation zu Präsentation unterscheiden kann. Dabei kommt es auf den gewählten Tablet PC an und

ob der Tablet PC im Hochformat oder Querformat verwendet wird, s. Abb. 5.3. Deswegen ist folgende Anforderung dem Bugreport A.1 hinzugefügt worden:

A.1.14 Neu

✓ (+)**B**: Beim Schreiben im Bearbeitungsfenster ist nicht gewährleistet, dass die Größe der Schreibfläche bei jeder Haltungsweise des Tablet PCs gleich ist. **U**: Die Zeichenfläche skaliert sich entsprechend der Breite des Bearbeitungsfensters. **L**: Eine Folie soll immer auf das Minimum aus der Breite und Höhe der aktuellen Bilschimgröße skaliert werden.

5.11.1. Skalierungsmöglichkeiten

Dem Benutzer soll nicht nur die Skalierungsvariante der Anforderung zur Verfügung stehen sondern auch alternative Skalierungsvarianten. Durch die Anforderung ist eine Skalierungsvariante spezifiziert, die *Standardskalierung* genannt wird. Die nächste Skalierungsvariante ist bereits vorhanden und bewirkt, dass die Folie in das Bearbeitungsfenster der Breite nach eingepasst wird. Sie wird mit *Ins Fenster einpassen* bezeichnet. Die letzte Variante soll dem Benutzer ermöglichen die Skalierung selber festzulegen und wird *Benutzerdefiniert* genannt.

Somit ergeben sich folgende Varianten:

1. Standardskalierung,
2. Ins Fenster einpassen und
3. Benutzerdefiniert.

Damit alle Komponenten, die die Skalierung benötigen, nicht selbst den gewählten Skalierungsmodus kennen und den Skalierungsfaktor errechnen müssen, ist ein neues Objekt namens `ScalingModule` erstellt worden.

5.11.2. Vorüberlegungen zum `ScalingModule`

Die Aufgabe des `ScalingModule`-Objekts ist es, den benötigten Skalierungsfaktor zu ermitteln. Für die erste Variante wird aus der Breite der virtuellen Größe der `Slide` und der Breite der gesamten Größe des Bildschirms im Hochformat der Skalierungfaktor berechnet. Es wird immer die Breite der Scrollbar abgezogen, sodass nicht ein Teil der Slide dahinter verschwindet. Die Formel dafür ist recht einfach und lautet:

$$\frac{\textit{Breite des Bildschirms in px} - \textit{Breite einer Scrollbar in px}}{\textit{Breite der Slide in px}} = \textit{Skalierungsfaktor}$$

Für die zweite Variante wird der Skalierungsfaktor ähnlich berechnet. Die Breite des Bildschirms wird dabei durch die Zielbreite ersetzt. Für diese Variante ist es wichtig, dass die Zielbreite übergeben wird.

Bei der letzten Variante errechnet sich der Skalierungsfaktor aus dem Skalierungsfaktor der Standardskalierung und dem angegebenen Skalierungsfaktor.

Der Skalierungsmodus kann beim `ScalingModule` über die Eigenschaft `ScalingMode` abgefragt werden. `ScalingMode` verwaltet dafür einen Wert der Enumeration `ScalingMode`. Die möglichen Modi entsprechen den festgelegten Möglichkeiten und sind: `Standard`, `FitToScreen` und `Userdefined`. Der Skalierungsfaktor kann über die Methode `double getScalingFactor(Size, Size)` abgefragt werden. Die Deklaration dieser Methode bietet die Möglichkeit, dass noch Änderungen an der Klasse `ScalingModule` vorgenommen werden können. Der benutzerdefinierte Skalierungsfaktor kann über die Eigenschaft `ScalingFactor` abgefragt und festgelegt werden.

5.11.3. Einbindung des `ScalingModule` in Foils

Da das `ScalingModule` zentral erreichbar sein muss, wird es als Eigenschaft des `Presentation`-Objekts zur Verfügung gestellt. `Presentation` ist über das *Singleton-Pattern* [GHJV94] immer erreichbar. Dies ermöglicht einen globalen Zugriff auf das `ScalingModule`-Objekt.

Über die öffentliche Methode `double getScalingFactor(Size)` des `ScalingModule` ist es möglich, den korrekten Skalierungsfaktor zu erlangen.

6. Umsetzung der Change-Requests

In diesem Abschnitt werden die Change-Requests bearbeitet. Jeder Unterabschnitt beginnt mit den zu behandelnden Anforderungen, gefolgt von der genaueren Umsetzung dieser. Angaben zur Legende der Anforderungen sind im Anhang A zu finden.

Anforderung A.2.9 wurde schon im Abschnitt 5.10 auf Seite 44 abgehandelt. Die Anforderung A.2.14 wird aus Zeitgründen nicht umgesetzt. Hingegen braucht die Anforderung A.2.20 nicht umgesetzt werden, weil die Radier-Funktion der inzwischen verwendeten `InkCanvas`-Klasse einwandfrei funktioniert.

6.1. Kleinere Änderungen an Foils

A.2.1 ✓ (+) **U**: Der alte Name “Studienarbeit Tim Steffens” beschreibt das Programm nicht passend. **L**: Der neue Name des Programms muss „Foils“ lauten.

A.2.2 ✓ (+) **C**: Die derzeitige Dateiendung lautet „xml” und kann dazu führen, dass xml-Dateien geöffnet werden, die nicht zu Foils gehören. **L**: Das Dateiformat muss in „foils” umbenannt werden.

Das Programm heißt jetzt Foils und abzuspeichernde Präsentationsdateien erhalten die Dateiendung „*.foils”.

6.2. Bilder in Foils

A.2.3 ✓ (+) **U**: Beim Einfügen eines Bildes in eine Folie wird dieses meistens zu klein skaliert oder es erscheint eine Fehlermeldung, die besagt, dass das Bild auf dieser Bildschirmseite nicht geeinfügt werden kann. **L**: Das Bild muss direkt auf die bestmögliche Größe skaliert werden, so dass es die Präsentationsfläche so gut wie möglich ausfüllt und die Seitenverhältnisse gewahrt bleiben. Außerdem soll die Fehlermeldung verschwinden.

A.2.4 ✓ (+) **B**: Beim Ändern der Größe eines Bildes bleibt das Seitenverhältnis nicht gewahrt. **L**: Es muss eine Option geben, die den Verlust des Seitenverhältnisses verhindert.

Änderung: Das Seitenverhältnis wird standardmäßig gewahrt. Dies kann nicht abgeschaltet werden.

Durch die in Abschnitt 3.2.1 beschriebene Umstellung von der `InkOverlay`- auf die `InkCanvas`-Klasse aus der WPF, wird die `Image`-Klasse der WPF für die Darstellung eines Bildes in Foils verwendet.

6.2.1. Darstellung eines Bildes mittels der `InkCanvas`

Die `Image`-Klasse wird zur Darstellung von Bildquellen verwendet und ist von der Basisklasse `UIElement` abgeleitet. `UIElement` ist die Basisklasse aller Elemente, die in einer GUI-Anwendung der WPF dargestellt werden. Somit können in einer GUI-Anwendung auch Bilder angezeigt werden.

Um `UIElement`-Objekte von einem `InkCanvas`-Objekt darstellen zu lassen, müssen sie zu der `Children`-Eigenschaft der `InkCanvas` hinzugefügt werden. Diese Eigenschaft bewirkt, dass alle eingefügten Elemente vom `InkCanvas`-Objekt dargestellt werden. Die `Children`-Eigenschaft ist eine `UIElementCollection` und stellt für das Anfügen von `UIElement`-Objekten die Methode `Add(UIElement)` bereit. Damit ist das Anfügen und Darstellen eines `UIElement` und auch eines `Image`-Objekts unkompliziert.

`UIElement`-Objekte, die von einer `InkCanvas` dargestellt werden, können genauso wie die `Stroke`-Objekte über den Auswahlmodus `Select` bearbeitet werden. Bearbeiten heißt hier: Auswählen, Verschieben oder Ändern der Größe.

Um einen Bildkontext darzustellen, muss ein `ImageSource`-Objekt über die `Source`-Eigenschaft des `Image`-Objekts gesetzt werden. `ImageSource` ist eine abstrakte Klasse für einen Bildkontext. Eine Implementation der `ImageSource`-Klasse ist die `BitmapImage`-Klasse. Sie ermöglicht es die Formate Bitmap (BMP), Joint Photographics (JPEG), Portable Network Graphics (PNG), Tagged Image File Format (TIFF), Microsoft Windows Media Photo (WMP), Graphics Interchange Format (GIF) und Icons zu laden. Es wird ein Enumerationtyp erstellt, damit diese unterstützten Formate aufgezählt werden können. Er heißt `SupportedBitmapExtensions` und hat als Werte: BMP , GIF, JPEG, PNG, TIFF und WMP. BMP entspricht dem Wert eins.

6.2.2. Größenverhältnisse beim Einfügen eines Bildes

Bei der jetzigen Version von Foils ist die Skalierung beim Einfügen der Bilder folgendermaßen geregelt: Ein Bild wird immer per Drag'n'Drop in eine Folie eingefügt. Die Positionskoordinate ist immer die linke obere Ecke des Bildes. Die Position des Bildes

ist dabei die Y-Position der Maus und in X-Richtung 2,5% der Breite einer Folie. Das Bild wird immer mit 95% der Breite einer Folie in die Präsentation eingefügt.

Dadurch wird zum rechten und linken Rand immer etwas Platz gelassen und es ist immer eindeutig festgelegt, wo der obere Rand des Bildes ist.

6.2.3. Darstellungmöglichkeiten der `Image`-Klasse

Ein `Image`-Objekt besitzt die Eigenschaft `Stretch`. Sie legt fest, wie ein Element dargestellt werden soll. Ihr können nur Elemente des Enumerationtyps `Stretch` zugewiesen werden. Die einzelnen Elemente sind `None`, `Fill`, `Uniform` und `UniformToFill`.

`None` bewirkt, dass das enthaltene Bild immer mit seiner tatsächlichen Pixelgröße angezeigt wird. `Fill` verursacht, dass das darzustellende Bild genau in das `Image`-Objekt eingepasst wird. Hierbei kann es zu Verformungen kommen. `Uniform` versucht das darzustellende Bild so gut wie möglich innerhalb der Grenzen des `Image`-Objekts einzupassen, ohne dass das Seitenverhältnis des darzustellenden Bildes verloren geht. `UniformToFill` ähnelt `Uniform`, doch es strebt nicht an innerhalb der Grenzen zu bleiben, sondern versucht die Fläche des `Image`-Objekts unter Wahrung des Seitenverhältnisses so gut wie möglich auszufüllen.

6.2.4. Seitenverhältnisse bei einem eingefügten Bild

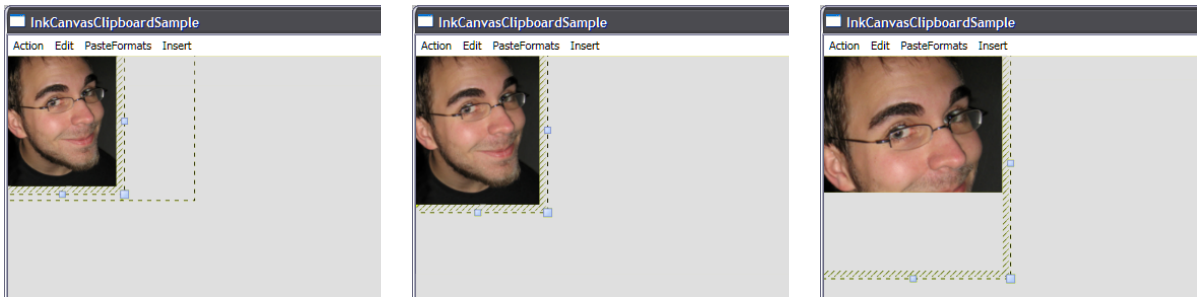

Abbildung 6.1.: Gezogener Rahmen zum Vergrößern des Bildes (*links*). Skalierte Bilder mit gesetzter `Stretch`-Property `Uniform` (*Mitte*) und `UniformToFill` (*rechts*) .

Für Foils kommen nur die Optionen `Uniform` und `UniformToFill` in Frage, weil sie das Seitenverhältnis wahren. Leider hat sich herausgestellt, dass `UniformToFill` zu Darstellungsproblemen führt. In der Abb. 6.1 sind diese Darstellungsprobleme illustriert. In der linken Darstellung deutet der gestrichelte Rahmen an wie der Benutzer das Bild in etwa skaliert haben möchte. Die mittlere Darstellung stellt die gesetzte Option `Uniform` dar. Die rechte Darstellung verdeutlicht das Problem mit der gesetzten

Option `UniformToFill`. Es ist zu sehen, dass der Bildkontext genau an den Grenzen des `Image`-Objekts abgeschnitten wird. Die Skalierung des Bildes ist zwar korrekt, aber das Ergebnis leider beschnitten.

Für Foils kommt deshalb nur die Option `Uniform` in Frage. D.h: Bilder, die in eine Folie eingefügt werden, behalten ihr Seitenverhältnis immer bei, da ihre `Stretch`-Eigenschaft auf `Uniform` voreingestellt ist.

6.3. Neues Design des Bearbeitungsfensters

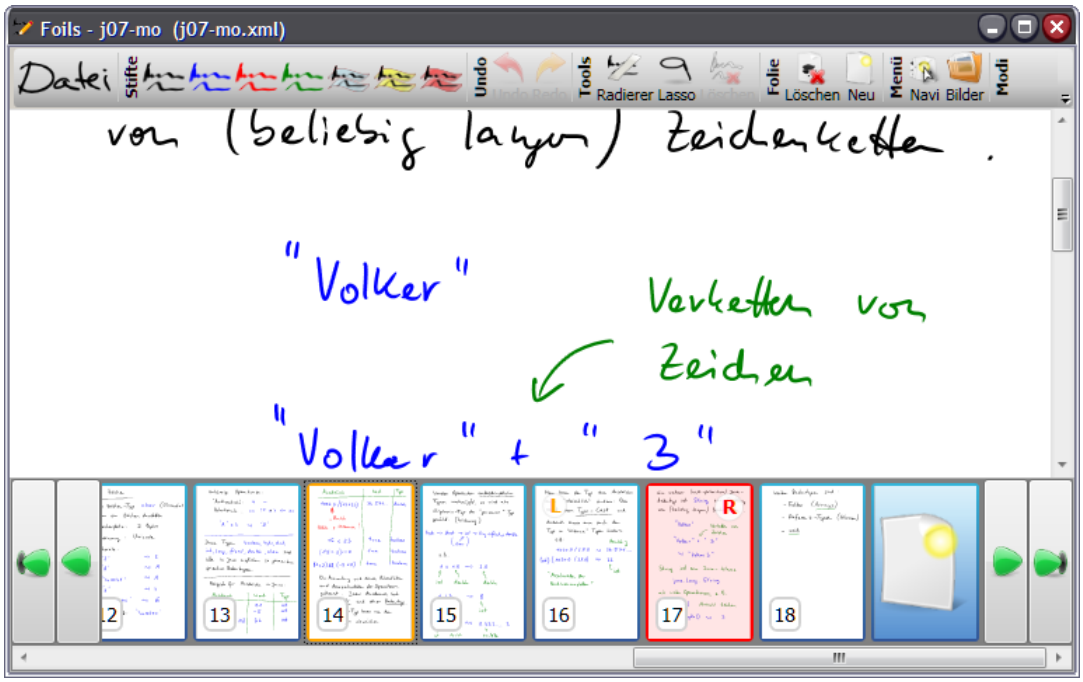

Abbildung 6.2.: Das neue Design nach dem Starten des Programms.

A.1.11 ✓ (N) **B**: Beim Maximieren des Bearbeitungsfensters und Wechsel in den Vollbildmodus entsteht eine ungenutzte Region im Bearbeitungsfenster. **L**: Diese freie Fläche soll beseitigt werden.

A.2.5 ✓ (+) **C**: Derzeitig wird der Knopf für die Löschung eines Bildes von mehreren Funktionen überlagert und verwirrt so den Benutzer. **L**: Die sich überlagernden Funktionen müssen auf mehrere Knöpfe aufgeteilt werden.

A.2.6 ✓ (+) **C**: Die Funktionen, die es ermöglichen, auf der derzeitigen Folien nach oben oder unten zu navigieren, werden nicht genutzt **L**: Diese Funktionen müssen entfernt werden.

A.2.8 ✓ (+) **C**: Die Demofunktion hat sich beim Arbeiten mit dem Programm nicht als nützlich erwiesen. **L**: Sie muss entfernt werden.

A.2.10 ✓ (N) **C**: Verwendete Symbole in ”Foils” sind nicht eindeutig bzw. ändern ihre Bedeutung. **L**: Alle Symbole sollen sich an schon bekannten Symbolen orientieren oder die dahinter stehende Bedeutung schnell und einfach vermitteln. Die Symbole sollen außerdem klar voneinander unterscheidbar sein.

A.2.11 ✓ (N) **C**: Oft liegen Symbole beieinander, die thematisch nicht zusammengehören. **L**: Es sollen alle Symbole nach ihren Hauptgebieten aufgeteilt werden. Diese Hauptgebiete sollen sein: Dateioperationen, Stifte, Undo-Operationen, Bearbeitunsoperationen, Operationen auf Folien, Navigation, Menüs und weitere Modi.

Das Design des Bearbeitungsfensters wurde neu entworfen und umgesetzt. Der Layout-Designer Expression Blend 2 von Microsoft wurde dafür zur Hilfe genommen.

Das neue Design des Bearbeitungsfensters ist in Abb. 6.2 illustriert. Im oberen Bereich befindet sich die Werkzeugleiste. Der mittlere Breich wird von der aktuellen Folie belegt, die bearbeitet werden kann. Den unteren Bereich nimmt die Navigationsleiste ein. Die Navigationsleiste wird im Abschnitt 6.4 näher erläutert.

6.3.1. Die Werkzeugleiste

Abbildung 6.3.: Werkzeugleiste mit (*oben*) und ohne Beschriftungstext (*unten*).

Die Werkzeugleiste in der Abb. 6.3 ist unterteilt in die Bereiche:

- Dateioperationen,
- Stifte,
- Undo-Operationen,
- Folienoperationen,
- Navigation,
- Menüs und
- sonstige Modi.

Der Navigationsbereich ist bei aktivierter Navigationsleiste nicht zu sehen. Es kann vorkommen, dass nicht alle Buttons Platz in der Werkzeugleiste finden. Für den Fall

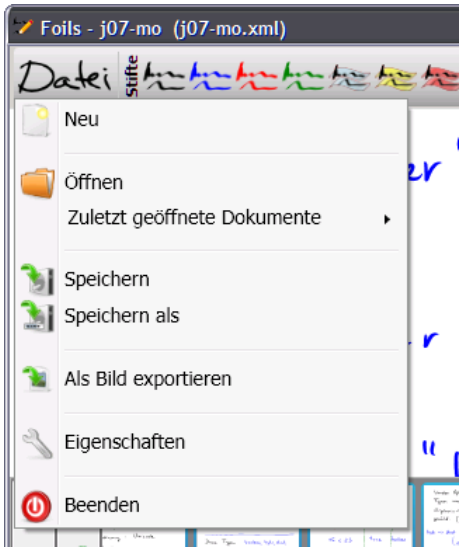

Abbildung 6.4.: Das Dateimenü von Foils.

werden einige Buttons nicht angezeigt, sind aber über ein DropDown-Menü verfügbar. Ein kleiner Button am rechten Seitenrand der Werkzeugleiste öffnet das DropDown-Menü.

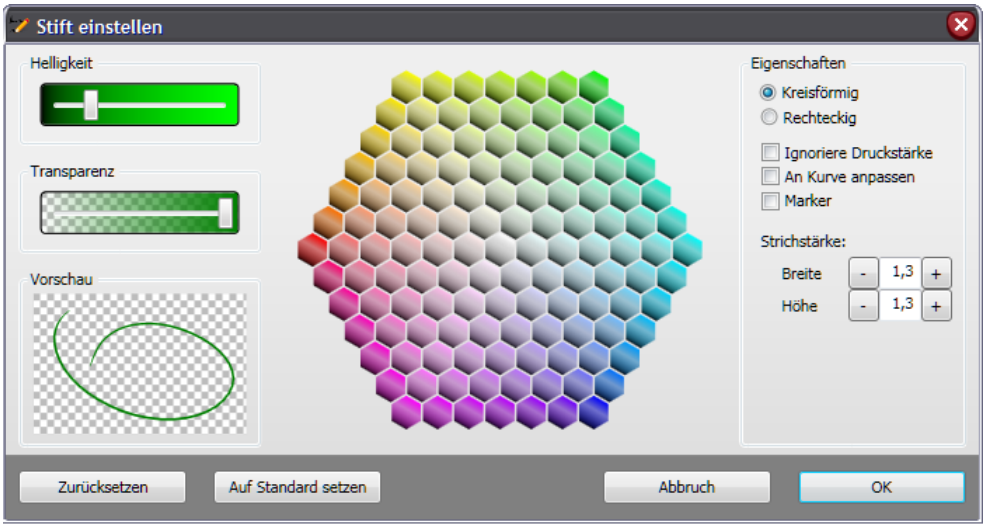

Abbildung 6.5.: Einstellungsdialog für einen Stift.

Die Dateioperationen sind über einen Button erreichbar, der ein DropDown-Menü öffnet [Wil09]. Das Menü ist in der Abb. 6.4 zu erkennen. Es ist nach Kategorien aufgeteil, die durch Trennstriche kenntlich gemacht sind. Sie sind:

- neue Präsentation erstellen,
- Präsentation öffnen,
- Präsentation speichern,

- Präsentation als Bild exportieren,
- Eigenschaften von Foils und
- Programm beenden.

Im Bereich Stifte, kann ein Stift aus sieben verschiedenen Stiften ausgewählt werden. Jeder einzelne Stift ist veränderbar. Ein Stift kann geändert werden, indem mit einem Rechtsklick auf ihn der Einstellungsdialog geöffnet wird, der in der Abb. 6.5 abgebildet ist. In diesem Dialog kann jeder Aspekt des Stifts verändert werden.

Der Bereich der Undo-Operationen, ermöglicht es alle getätigten Operationen auf der Zeichenfläche wieder rückgängig zu machen oder, falls sie schon zurückgesetzt worden sind, zu wiederholen.

Der Bereich der Bearbeitungswerkzeuge bietet drei Funktionen. Mit dem Radierer lassen sich Striche wegradieren. Die Lasso-Funktion lässt es zu, mehrere Striche oder Bilder auszuwählen. Falls eine Auswahl getroffen wurde, kann sie verschoben, skaliert oder mit dem letzten Button im Bereich der Bearbeitungswerkzeuge gelöscht werden.

Der Bereich Folien bietet Operationen die Folien betreffen. So gibt es die Funktion die derzeitige Folie zu löschen oder eine neue Folien nach der derzeitigen einzufügen.

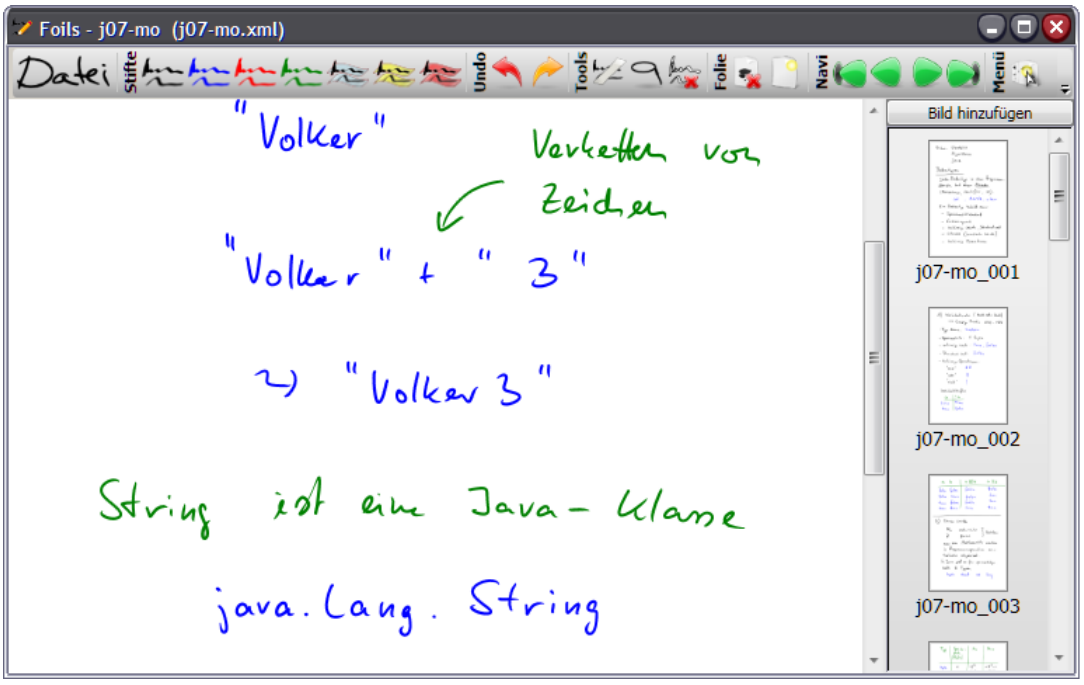

Abbildung 6.6.: Aktiviertes Bilderauswahlmenü auf der rechten Seite, durch dessen Hilfe Bilder in eine Präsentation eingefügt werden können.

Wie in Abb. 6.6 zu sehen ist, kann die Navigationsleiste auch deaktiviert werden. Falls die Navigationsleiste nicht mehr aktiv ist, wird die Werkzeugleiste um den Navi-

gationsbereich erweitert. Der Navigationsbereich bietet einfache Funktionalitäten, wie ans Ende der Präsentation oder zur vorherigen Folie zu springen.

Der darauf folgende Bereich ist den Menüs gewidmet und ermöglicht es, die Navigationsleiste oder die Bilderauswahlleiste anzeigen zu lassen oder zu deaktivieren. Für die Navigationsleiste ist dafür der Button „Navi" vorgesehen und für die Bilderauswahlleiste der Button „Bilder".

Der letzte Bereich gibt dem Benutzer die Möglichkeit, in den Vollbildmodus zu wechseln oder die Beschriftung der Buttons in der Werkzeugleiste ein- oder auszublenden. Der Vollbildmodus maximiert das Bearbeitungsfenster und lässt die Titelleiste sowie die Taskleiste verschwinden. Dies lässt die Nutzung der kompletten Schreibfläsche des Tablet PCs zu. Durch die Option die Beschrifung ein- und auszublenden, rücken alle Buttons bei der Deaktivierung der Schrift enger zusammen. Dies spart Platz in der Werkzeugleiste und ermöglicht es schneller auf Funktionen zuzugreifen, die sonst nur über das DropDown-Menü erreichbar gewesen wären.

6.3.2. Die Bilderauswahlleiste

Es ist möglich während der Präsentation Bilder in eine Präsentation einzufügen. Dafür gibt es die Bilderauswahlleiste. Sie kann eingeblendet werden, indem im Bereich Menü der Button „Bilder" ausgewählt wird. In der Abb. 6.6 ist ein Beispiel für diese Bilderauswahlleiste zu sehen. Das Einfügen eines Bildes ist nur durch Drag'n'Drop möglich. Es reicht das Bild auszuwählen, die linke Maustaste gedrückt zu halten und über der Zeichenfläche wieder loszulassen. Das implementierte Verhalten ist in Abschnitt 6.2.2 näher erläutert.

6.3.3. Ungenutzter Bereich im Bearbeitungsfenster

Durch die neue Umsetzung des Bearbeitungsfensters wurde darauf geachtet, dass alle Elemente den zur Verfügung stehenden Platz komplett ausfüllen.

6.3.4. Doppelte Belegung von Buttons

In der alten Version von Foils existierte ein Button für die Aktionen: *aktuelle Folie löschen*, *ausgewählte Striche löschen* und *ausgewähltes Bild löschen*. Die Aktionen *aktuelle Folie löschen* und *ausgewählte Striche löschen* sind auf zwei Buttons aufgeteilt, s Abb. 6.3. Weil Striche und Bilder jetzt mit der selben Operation ausgewählt werden können, werden Striche und Bilder auch über den selben Button gelöscht.

6.3.5. Die Scrollbuttons und die Demofunktion

Die Buttons, die es ermöglichen, die aktuelle Folie rauf und runter zu scrollen, sind entfernt worden. Ebenso wurde der Button zum Umschalten auf den Demomodus entfernt.

6.3.6. Verwendung von klareren Symbolen

Es wurde versucht die Symbole verständlich zu gestalten und der Anforderung A.2.10 Genüge zu leisten. Hierzu wurde darauf geachtet, dass die Symbole in einem einheitlichen Stil gehalten sind. Dafür wurden Symbole von der Seite GNOME Art [Gno] aus dem „Icon-Theme-Pack Dropline Neu!" [Her09] verwendet. Da in dem Icon-Theme-Pack die Icons als skalierbare Bilder zur Verfügung standen, war es möglich schnell und einfach neue Symbole zu kreieren.

6.3.7. Anordnung der Symbole in der Werkzeugleiste

Das Menü ist nach der Anforderung A.2.11 umstrukturiert, sodass alle Symbole thematisch nahe zusammen liegen. In der Abb. 6.3 auf Seite 53 ist die Umstrukturierung zu sehen. Der Bereich für die Navigation ist nicht zu sehen. Dieser fehlende Bereich ist in Abb. 6.6 auf Seite 55 zu sehen.

6.4. Die Navigationsleiste

In der alten Programmversion war die Navigation durch die Folien der aktuellen Präsentation über ein Drop-Down-Menü möglich, bei dem für jede Ansicht die gewünschte Folie ausgewählt werden konnte. Die nachstehenden Anforderungen befassen sich ausschließlich mit der Navigationsleiste.

A.2.12 ✓ (N) **C**: Das Drop-Down-Menü ist zu unhandlich und beansprucht zu viel Zeit. **L**: Das Drop-Down-Menü soll durch eine Navigationsleiste ersetzt werden.

A.2.13 ✓ (N) Die Folien-Navigationsleiste soll der Abb. A.1 auf Seite 87 nachempfunden werden und sich standardmäßig am unteren Rand befinden.

A.2.15 ✓ (N) Die Navigationsleiste soll des Weiteren eine Scrollbar besitzen, sowie vier Buttons für das Wechseln zum Anfang bzw. Ende der Präsentation, zu der vorherigen und der nächsten Folie.

A.2.18 ✓ (-) In der Navigationsleiste wird als letzte Folie immer eine leere Folie angezeigt: Durch Doppelklick auf sie wird eine neue Folie am Ende der Präsentation erstellt.

6.4.1. Umsetzung der Navigationsleiste

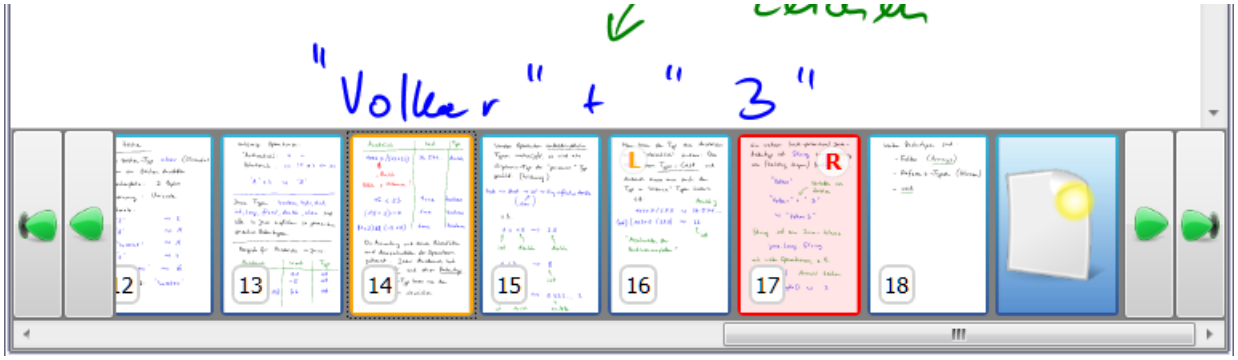

Abbildung 6.7.: Erste Umsetzung der `NavigationBar`.

Für die Umsetzung wurde die Skizze in Abb. A.1 auf Seite 87 herangezogen. Eine Klasse `NavigationBar` ist entworfen worden, die die Darstellung der Navigationsleiste übernimmt.

Die Klasse `NavigationBar` beinhaltet vier `Button`-Objekte, die für die schnelle Navigation ans Ende oder an den Anfang der Präsentation, sowie um eine Folie vor- oder zurückzugehen, zuständig sind.

Die Auflistung der Folien zwischen den Buttons wurde mit einer `ListBox` umgesetzt. Sie bietet die Möglichkeit, Objekte anzufügen, die dann automatisch in einer vertikalen Listenform dargestellt werden. Für die Listenelemente werden `SlideCanvas`-Objekte verwendet. Dadurch werden Änderungen an den einzelnen Folien direkt in der Navigationsleiste anzeigt. Durch ein *Observer-Pattern* [GHJV94] auf das `SlideSequence`-Objekt werden Änderungen an der `SlideSequence` registriert. Dadurch können die Änderungen direkt in der `NavigationBar` übernommen werden. Eine Änderung bedeutet hierbei, welche `Slide` an welcher Position gelöscht oder hinzugefügt worden ist. Als letztes Element wird generell ein Symbol eingefügt. Durch einen Doppelklick auf das Symbol wird eine neue, leere Folie ans Ende der Präsentation angefügt.

Die Umsetzung der Navigationleiste ist in Abb. 6.7 zu sehen. Die letzte Folie, die zum Einfügen einer neuen Folie gedacht ist, ist in der Abbildung blau unterlegt. Jede Folie besitzt ihre eigene Foliennummer und kann so eindeutig identifiziert werden. Eine rot umrahmte Folie bedeutet, dass sie im Bearbeitungsfenster dargestellt wird. Ein gelber Rahmen um eine Folie besagt, dass die Folie gerade angewählt ist. Es sind Aktionen auf der Navigationsleiste definiert, sodass sich mit einem Rechtsklick auf eine Folie diese auf eine bestimmte Präsentationsansicht schalten lässt. Auch ist es möglich, eine neue Folie vor oder hinter einer anderen Folie einzufügen. Das Löschen einer Folie ist auch vorgesehen, wie es in der Abb. 6.8 im Kontextmenü zu sehen ist.

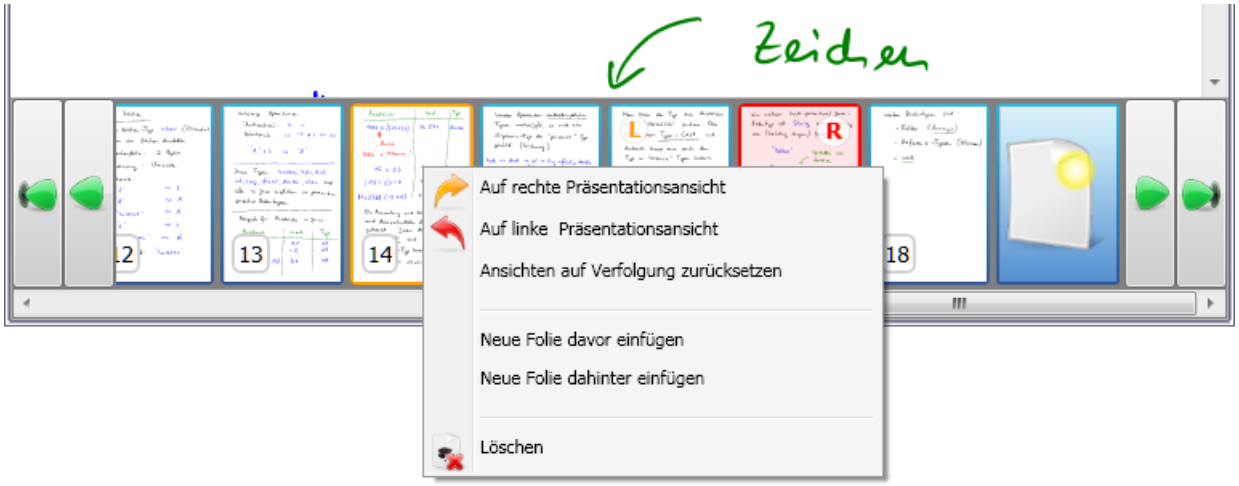

Abbildung 6.8.: Das Kontextmenü mit weiteren Operationen, die durch die Navigationsleiste ausgeführt werden können.

6.5. Der Wizarddialog

Die Anforderungen A.2.16, A.2.17 und A.2.19 befassen sich mit dem Wizarddialog. Der Wizard soll beim Starten des Programms erscheinen und dem Benutzer helfen, eine Präsentation zu erstellen oder erstellte Präsentationen zu öffnen. Die zu erfüllenden Anforderungen sind folgende:

A.2.16 ✓ (N) **C**: Der derzeitige Wizard-Dialog ist umständlich zu bedienen und bietet nicht viele - eigentlich selbstverständliche - Möglichkeiten. **L**: Beim Start von Foils soll ein neuer Wizard-Dialog erscheinen, der den Benutzer beim Erstellen und Öffnen von Folien unterstützt und kurze Bedienschritte ermöglicht.

A.2.17 ✓ (N) Der Wizard-Dialog soll bediener- sowie einsteigerfreundlich sein. Einsteigerfreundlich bedeutet, dass jede Auswahlmöglichkeit einen treffenden Namen und eine verständliche nähere Erklärung besitzt.

A.2.19 ✓ (-) Der Wizard-Dialog bietet folgende Wahlmöglichkeiten: leere Präsentation, Liste zuletzt bearbeiteter Präsentationen und direkt zum Arbeitsplatz (Bearbeitungsfenster) wechseln.

6.5.1. Der neue Wizard-Dialog

Beim Starten des Programms Foils wird direkt der Wizard-Dialog dargeboten und im Hintergrund das Bearbeitungs- und Präsentationsfenster gestartet und angezeigt. Der Wizard-Dialog ist in der Abb. 6.9 dargestellt. Der Wizard-Dialog wird als modaler Dialog in den Vordergrund gerückt. Das Bearbeitungsfenster wird mit einer neuen

Abbildung 6.9.: Der neue Wizarddialog mit Beispieldaten.

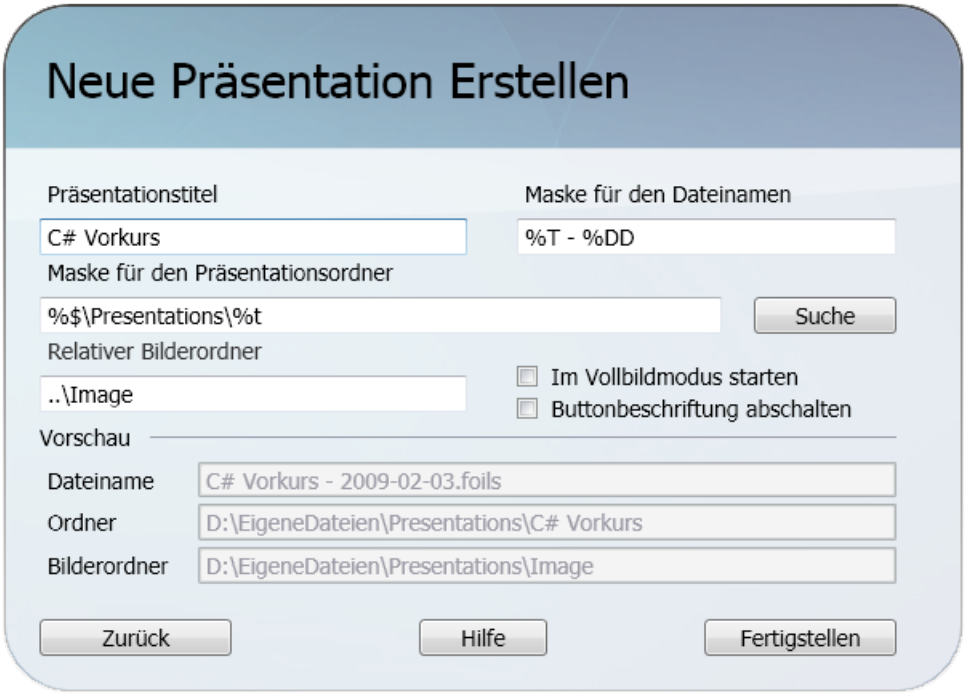

Abbildung 6.10.: Dialog zum Erstellen einer Präsentation.

leeren Präsentation gestartet, mit der direkt eine Vorlesung oder ein Vortrag gehalten werden kann.

Die Auswahlmöglichkeiten im Wizard-Dialog sind: eine leere Präsentation starten, eine Präsentation öffnen, eine der zuletzt geöffneten Präsentationen laden oder das Programm schließen.

Über den Link „Neue leere Präsentation beginnen" schließt sich der Dialog und der Benutzer kann direkt mit der leeren Präsentation starten, die sich schon im Bearbeitungsfenster befindet. Eine leere Präsentation ist noch nicht gespeichert und besitzt keinen Namen. Um sie zu speichern und ihr somit einen Namen zu geben, muss über „Datei" im Bearbeitungsfenster (vorher muss der Wizard-Dialog geschlossen werden) ⟶ „Speichern" oder „Speichern als" ausgewählt werden.

Die Abb. 6.10 stellt den Wizard-Dialog dar, um eine Präsentation zu erstellen. Erreicht werden kann dieser Dialog über den Link „Neue Präsentation erstellen ..." im Start-Dialog. Der Dialog bietet die Möglichkeit den Titel der Präsentation, eine Maske für den Dateinamen, den Präsentationsordner und einen relativen Bilderordner anzugeben, sowie direkt in den Vollbildmodus zu wechseln und die Beschriftung der Buttons abzuschalten. Die Maskenoptionen sind bereits genauer im Abschnitt 5.10.1 auf Seite 45 beschrieben worden. Über den Button „Zurück" gelangt der Benutzer wieder zum Wizard-Dialog. Der Button „Fertigstellen" bewirkt, dass die Präsentation erstellt wird.

Der Link „Präsentation öffnen ..." öffnet einen Dialog, der es erlaubt, eine Präsentation zu suchen und zu öffnen.

Alternativ zu „Präsentation öffnen ..." können die zehn zuletzt geöffneten Präsentationen geladen werden. Die zuletzt geöffneten Präsentationen werden von oben nach unten in der Reihenfolge ihrer letzten Verwendung dargeboten und können durch einen Doppelklick geöffnet werden.

Der Button „Programm schließen" schließt den Wizarddialog und das Programm. Falls der Wizarddialog beim Starten nicht erscheinen soll, kann über die Checkbox „Beim Start anzeigen" dieses Verhalten abgestellt werden und der Benutzer wird beim nächsten Start direkt mit einer leeren Präsentation beginnen. Falls der Wizarddialog wieder beim Starten angezeigt werden soll, kann er über den Einstellungsdialog ⟶ Tab „Erstellen" und dort durch die Aktivierung der Option „Den Startup-Wizard beim Start anzeigen" beim Start angezeigt werden.

Alle Möglichkeiten sind mit klaren Namen umschrieben. Falls diese Namen nicht ausreichen, wird der Benutzer durch Tooltip mit weiteren Beschreibungen unterstützt, die für jede durchführbare Aktion existieren.

7. Umsetzung der Extension-Requests

In diesem Abschnitt werden die Extension-Requests behandelt. Jeder Unterabschnitt beginnt mit den zu behandelnden Anforderungen, gefolgt von einer genauen Beschreibung der Umsetzung.

Die Anforderungen A.3.6, A.3.8, A.3.15, A.3.16, A.3.17, A.3.19, A.3.20, A.3.23, A.3.25, A.3.26 und A.3.27 sind aus Zeitmangel nicht umgesetzt worden.

7.1. Programmverknüpfung

A.3.1 ✓ (+) **U**: Es ist nicht möglich, eine Datei über den Datei-Explorer direkt zu öffnen. **L**: Die Dateiendung „foils“ muss mit dem Programm direkt assoziiert und beim Öffnen die Präsentationdatei geladen werden.

Die `main`-Methode des Programms ist so angepasst worden, dass dem Programm ein Dateipfad übergeben werden kann und dadurch eine Präsentation geladen wird. Bei dieser Art des Ladens, wird der Startup-Dialog nicht angezeigt, sondern direkt die gewünschte Präsentation.

Damit auch eine Programmverknüpfung verfügbar ist, wird diese während der Installation in Windows registriert.

7.2. Bilder während des Betriebs importieren

A.3.2 ✓ (+) **U**: Es kann kein Bild während einer laufenden Präsentation im System gesucht und zur Präsentation hinzugefügt werden. **L**: Das Suchen und Hinzufügen eines Bildes während der laufenden Präsentation muss ermöglicht werden. Das neue Bild wird in den Bilderordner kopiert.

A.3.7 ✓ (N) Bilder sollen im Auswahlmodus eine Vorschau besitzen.

Damit auch Bilder während einer laufenden Präsentation zum Bilderordner hinzugefügt werden können, gibt es einen Button namens „Bild hinzufügen” im oberen Bereich der Bilderauswahlleiste, s. Abb. 6.6 auf Seite 55. Dieser Button namens „Bild

hinzufügen" öffnet einen Dialog, mit dem es möglich ist nach einer oder sogar mehreren Bilddateien zu suchen und diese zum Bilderordner hinzuzufügen. Die Dateien werden direkt in den Bilderordner der Präsentation eingefügt und können dann über die aktualisierte Bilderauswahlleiste eingefügt werden.

Die Bilderauswahlleiste ist so abgeändert worden, dass alle im Bilderordner der Präsentation vorhandenen Bilder mit einem Vorschaubild angezeigt werden.

Damit auch Bilder hinzugefügt werden können, die der Benutzer während einer Präsentation ins Verzeichnis kopiert hat, wurde die Klasse `FileSystemWatcher` verwendet. Durch ein Objekt dieser Klasse ist es möglich einen bestimmten Ordner zu überwachen. Änderungen an Dateien in diesem Ordner können über die Events `Created`, `Deleted` oder `Renamed` des `FileSystemWatcher`-Objekts abgefangen werden. Diese Events sind so implementiert worden, dass jede Änderung im beobachteten Ordner registriert wird, und die Bilder in der Bilderauswahlleiste aktualisiert werden. So werden auch die Bilder, die über den vorher erwähnten Auswahldialog ins Bilderverzeichnis kopiert werden, automatisch in die Bilderleiste aufgenommen.

7.3. Autosave von Präsentationen

A.3.3 ✓ (+) **U**: Es existiert keine Autosave-Funktion. **L**: Die Autosave-Funktion muss nachgerüstet werden, sodass in einem bestimmten Zeitintervall und evtl. mit jedem Hinzufügen einer Folie gespeichert wird.

Die Autosave-Funktionalität wird in einem separaten Thread ausgeführt. Das Speicherintervall ist voreingestellt auf fünf Minuten, kann aber geändert werden. Falls das Speicherintervall auf 0 Minuten oder eine negative Minutenzahl gesetzt wurde, ist die Autosave-Funktion deaktiviert. Die Präsentation wird außerdem immer automatisch gespeichert, wenn eine neue Folie zur Präsentation hinzugefügt wird. Zum Speichern wird die vorhandenen Speichermethode von Foils verwendet. Beim Speichern wird nicht die Präsentation selber überschrieben, sondern nur eine Datei mit der Endung „*.foils.autosave" angelegt bzw. überschrieben.

Um zu garantieren, dass die zu speichernden Daten nicht verändert werden, wird dafür auf dem `Dispatcher`-Objekt des Hauptprogramms die `Invoke(Delegate, params Object)`-Methode aufgerufen. Sie bekommt ein `Delegate`-Objekt und mehrere `Object`-Objekte übergeben. Ein `Delegate`-Objekt verweist auf eine auszuführende Methode. Das Schlüsselwort `params` bewirkt, dass das nachfolgende `Object`-Array nicht als Array aufgefasst wird, sondern als variable Parameterliste interpretiert wird. Dadurch wird die übergebene Methode im Thread des Hauptprogramms ausgeführt und verhindert so, dass die zu speichernden Daten währenddessen verändert werden.

7.4. Präsentationverlust beim Speichern

A.3.4 ✓ (+) **U**: Es ist vorgekommen, dass das Programm während des Speicherns abgestürzt ist. Die zu überschreibende Präsentationsdatei war nicht mehr existent. **L**: Beim Speichern muss zuerst ein Backup der bisherigen Präsentation erstellt werden. Danach erst darf die aktuelle Präsentation gespeichert werden.

Damit kein Verlust der Präsentation beim Speichern auftreten kann, wird die Speichermethode von Foils so angepasst, dass die existierende Präsentation zuerst kopiert wird und die zusätzliche Dateiendung „*.bak" erhält. So wird aus der Datei „Beispiel.foils" „Beispiel.foils.bak". Dann erst wird die aktuelle Präsentation gespeichert und somit die alte überschrieben. Dieses Verhalten wird auch bei der Autosave-Funktion angewendet, sodass aus „Beispiel.foils.autosave" „Beispiel.foils.autosave.bak" wird. Die Backup-Datei der Autosave-Funktion wird jedoch nach erfolgreichem Speichern gelöscht.

7.5. Benutzerdefinierte Einstellungen

In der ersten Version von Foils war es nicht möglich Einstellungen benutzerdefiniert abzuspeichern, geschweige denn innerhalb des Programms Einstellungen zu ändern. Alle Einstellungen wurden in einer XML-Datei namens „settings.xml" in einem Unterorder „Settings" relativ zum Programmpfad abgespeichert.

A.3.5 ✓ (N) Die Autosave-Funktion soll individuell einstellbar sein.

A.3.9 ✓ (N) Folgende Einstellungen sollen getroffen werden können, die beim Starten automatisch angewendet werden: Fullscreen & Textmodus.

A.3.10 ✓ (N) **U**: Die Farbe der einzelnen Stifte kann nur in der xml-Datei geändert werden. **L**: Bei den Stiften sollen die Farben personalisierbar sein und automatisch gespeichert werden, dabei soll es aber immer noch die Möglichkeit geben, auf Standardeinstellungen zurückzugreifen.

A.3.11 ✓ (N) Das Ein- und Ausblenden der Buttonbeschriftung, genannt Textmodus, soll personalisierbar sein.

A.3.18 ✓ (-) Die im Wizard-Dialog getroffenen Entscheidungen können gespeichert und beim nächsten Programmstart automatisch angewendet werden.

A.3.21 ✓ (-) Einstellungen, die personalisiert werden können, sollen über einen Einstellungsdialog erreichbar sein und verändert werden können.

A.3.22 ✓ (-) Gespeicherte Einstellungen können für jeden Benutzer einzeln abgespeichert werden.

7.5.1. Umsetzung durch die Registry

Für die Realisierung wurde die Registry von Windows verwendet. In .Net wird die Registry in einer Klasse namens `Registry` gekapselt. Sie besitzt statische Referenzen, mit denen auf die verschiedenen Hauptschlüssel zugegriffen werden kann. Bei der Umsetzung sind nur die beiden Hauptschlüssel `HKEY_LOCAL_MACHINE` und `HKEY_CURRENT_USER` interessant. Diese bieten die Unterteilung der Einstellungen in Standardeinstellungen und Benutzereinstellungen.

Standardeinstellungen werden unter `HKEY_LOCAL_MACHINE` abgespeichert, denn der Schlüssel ist für jeden Benutzer zugänglich. Die Benutzereinstellungen werden unter `HKEY_CURRENT_USER` abgelegt. Der Schlüssel ist nur für den jeweiligen Benutzer zugänglich und eignet sich aus dem Grund sehr gut für die Benutzereinstellungen.

Ein Hauptschlüssel wird durch ein `RegistryKey`-Objekt abgebildet. Mit ihm ist es möglich, weitere Unterschlüssel anzulegen oder in ihm enthaltene Werte anzulegen bzw. auszulesen. Für Foils wurde der Schlüsselpfad „Software/Uni-Koblenz/Foils/Settings" als Standard festgelegt. In diesem Schlüssel werden alle Einstellungen abgespeichert. Der Schlüsselpfad ist für beide Hauptschlüssel identisch.

Wenn das Programm Foils installiert wird, werden zu Anfang nur im Hauptschlüssel `HKEY_LOCAL_MACHINE` alle Grundeinstellungen eingetragen. Startet ein Benutzer das Programm erstmals, sind keine Einstellungen in der Registry unter dem Hauptschlüssel `HKEY_CURRENT_USER` vorhanden. Die fehlenden Einträge werden aus den Einträgen von `HKEY_LOCAL_MACHINE` ausgelesen und übertragen.

Zum Speichern der Werte werden Stringdarstellungen benutzt, damit die Lesbarkeit der enthaltenen Daten gewährleistet ist. Es werden verschiedene Datentypen abgespeichert: `Boolean`, `Integer`, `Double`, `String`, `Size`, `ScalingMode`, `SupportedBitmapExtensions` und `DrawingAttributes`.

7.5.1.1. Speicherung von einfachen Datentypen

`Boolean` wird mit den Stringrepräsentationen „True" und „False" abgespeichert. `Integer` und `Double` werden auch in eine übliche Stringdarstellung umgewandelt. Bei dem Typ `Double` bleibt anzumerken, dass als Trennzeichen zwischen Vor- und Nachkommastellen ein Punkt verwendet wird. `String` wird als eine normale Zeichenkette abgespeichert. `Size` ist ein Objekt, dass eine Breite und eine Höhe jeweils vom Typ `double` speichert. `Size` wird in einer Zeichenkette so dargestellt, dass zuerst die Breite aufgeführt wird und dann, durch ein Semikolon getrennt, die Höhe folgt. Die beiden Typen `ScalingMode` und `SupportedBitmapExtensions` sind Enumeration-Typen. `ScalingMode` wurde im Abschnitt 5.11.2 auf Seite 48 eingeführt und genauer erklärt. Ebenso `SupportedBitmapExtensions` im Abschnitt 6.2.1 auf Seite 50. Durch

die Klasse `Enum` können Enumeration-Werte in Stringrepräsentationen umgewandelt und auch wieder zurücktransformiert werden. Eine Stringrepräsentation entspricht jeweils dem Namen des repräsentierten Enumeration-Werte.

7.5.1.2. Speicherung von `DrawingAttributes`

Name	Standardwert	Werttyp	Beschreibung
AskBeforeDeletingASlide	„True"	Boolean	Bestimmt, ob beim Löschen einer Folie dies bestätigt werden soll.
AutosaveInterval	„5"	Integer	Legt das Speicherintervall der Autosavefunktion in Minuten fest. 0 oder ein negativer Wert bedeutet, dass die Funktion deaktiviert ist.
ButtonTextVisible	„True"	Boolean	Zeigt bei „True" den Buttonbeschriftungstext an.
FullscreenMode	„False"	Boolean	Beschreibt, ob beim Starten in den Vollbildmodus gewechselt werden soll.
GraphicsSelectionPanelFontSize	„14"	Integer	Gibt den verwendeten Schriftgrad der Bilderauswahlleiste in Pixeln festlegt.
GraphicsSelectionPanelSize	„100; 80"	Size	Bestimmt die Größe der Bilderauswahlleiste in logischen Einheiten.
GraphicsSelectionPanelVisible	„False"	Boolean	Entscheidet die Sichtbarkeit der Bilderauswahlleiste beim Starten bestimmt.
MaxUndoSteps	„50"	Integer	Gibt die Anzahl möglicher Undoschritte.
NavigationBarVisible	„True"	Boolean	Besagt, ob die Navigationsleiste beim Starten sichtbar ist.
SlideThumbnailFontSize	„14"	Integer	Legt den Schriftgrad einer Vorschaufolie in Pixeln fest.
SlideThumbnailSize	„70; 105"	Size	Definiert die Größe einer Vorschaufolie in logischen Einheiten.

Tabelle 7.1.: Allgemeine Einstellungen für das Arbeiten mit Foils.

Ein `DrawingAttributes` definiert mehrere Attribute: `Color`, `Width`, `Height`, `IsHighlighter`, `FitToCurve`, `IgnorePressure` und `StylusTip`. Die ersten drei Attribute sollten geläufig sein. `Color`bestimmt die Farbe des Stiftes. `Width` und `Height` speichern die Breite und Höhe der Stiftspitze als einen Wert vom Typ `double` ab. `IsHighlighter`, `FitToCurve` und `IgnorePressure` speichern einen Boolean-Wert. `IsHighlighter` legt fest, ob der Stift ein Marker ist. Mehrere Striche des

Name	Standardwert	Werttyp	Beschreibung
ScalingFactor	„1"	Double	Legt den Skalierungsfaktor zum Skalieren der Folie im Bearbeitungsfenster bei eingestelltem Skalierungsmodus `Userdefined` fest.
ScalingMode	„Standard"	Enumeration `ScalingMode`	Bestimmt die Skalierungsart, s. Abschnitt 5.11.2 auf Seite 48.

Tabelle 7.2.: Skalierungseinstellungen.

Name	Standardwert	Werttyp	Beschreibung
ImageExportFormat	„PNG"	Enumeration `Supported-Bitmap-Extensions`	Besagt, welches Format beim Export verwendet werden soll, s. Abschnitt 6.2.1 auf Seite 50.
ImageExportKeepsAspectRatio	„True"	Boolean	Beschreibt, ob das Folienseiten-verhältnis gewahrt werden soll.
ImageExportNameMask	„%f_%DD_###"	String	Definiert die Maske für den Dateinamen beim Export.
ImageExportPathMask	„%p"	String	Bestimmt, wie die Maske für den Ausgabeordner beim Export lauten soll.
ImageExportSize	„1000;1500"	Size	Legt fest, wie groß das Ausgabebild sein soll.

Tabelle 7.3.: Exporteinstellungen

selben Markers überlagern sich nicht, sondern ergeben eine einheitlich Fläche in der Farbe des Markers. Das Attribut `FitToCurve` bewirkt, dass Striche, bevor sie als `Stroke`-Objekte in eine `StrokeCollection` eingefügt werden, an eine Kurve angenähert werden. `IgnorePressure` legt fest, ob die Druckstärke beim Schreiben ignoriert werden soll. Hinter dem Attribut `StylusTip` verbirgt sich ein Enumerationtyp mit den möglichen Werten `Ellipse` und `Rectangle`. Mittels `StylusTip` kann das Erscheinungsbild der Stiftspitze verändert werden.

Um eine Stringrepräsentation zu erhalten wird zuerst `Color`, mit den vier Komponenten Alpha-, Rot-, Grün- und Blaukanal, in eine hexadezimale Darstellung umgewandelt. Dieser Darstellung geht eine Raute voraus, die den Anfang der Zeichenkette bildet. Durch jeweils ein Semikolon getrennt werden die restlichen Attribute, in der Reihenfolge wie sie erwähnt wurden, hinter `Color` angehängt. Die einzige Besonderheit ist, dass die Eigenschaft `StylusTip` durch einen Boolean-Wert abgespeichert wird. Der Boolean-Wert ist „True", wenn `StylusTip Ellipse`

Name	Standardwert	Werttyp	Beschreibung
DefaultPresentationNameMask	„%t_%DD”	String	Definiert, wie die Maske für den Präsentationsnamen sein soll.
DefaultPresentationPathMask	„%~\\ Präsentationen\\%tT”	String	Legt fest, wie die Maske für den Präsentationsordner sein soll.
DefaultRelativeImagePath	„Images”	String	Bestimmt den relativen Pfad zum Bilderordner.
DefaultVirtualSlideSize	„500; 750”	Size	Legt fest, wie groß die virtuelle Foliengröße sein soll.
StartUpDialogVisible	„True”	Boolean	Entscheidet, ob der Startup-Dialog beim Start angezeigt werden soll.

Tabelle 7.4.: Erstellungseinstellungen

Name	Standardwert	Werttyp	Beschreibung
Pen1	„Black; 1.3; 1.3; False;False;False;False”	DrawingAttributes	Legt die Zeicheneigenschaften für den ersten Stift fest.

Tabelle 7.5.: Stifteinstellungen des ersten Stifts. Alle restlichen sechs Stifte sind analog definiert.

enthält. Zum Beispiel wäre für einen schwarzen Stift mit der Breite und Höhe 1.0 und allen anderen Werten auf „True” die entsprechende Stringrepräsentation „`#FF000000;1.0;1.0;True;True;True;True`”. Es ist außerdem möglich statt einen Hexadezimalwert den englischen Namen einer Farbe zu verwenden. Im vorherigen Beispiel würde der Name „`Black`” zum gleichen Ergebnis führen.

7.5.1.3. Abgespeicherte Werte

Die Einstellungen wurden nach Kategorien unterteilt und zwar: Allgemeine Einstellungen zu Foils und der Skalierung und Standardeinstellungen des Export-Dialogs, der Erstellungsparameter und der Stifte. Es folgt eine Aufzählung aller Einträge aus den Tabellen 7.1, 7.2, 7.3, 7.4 und 7.5, die sich unter dem Schlüssel `HKEY_LOCAL_MACHINE` befinden und durch das Setup eingetragen werden.

Die Einstellungen werden direkt beim Starten des Programms geladen und zur Verfügung gestellt. Dadurch können der Textmodus, der Vollbildmodus, das Erscheinen des Startupdialogs und auch die Standardstiftfarben übernommen werden.

7.5.2. Einstellungsdialog

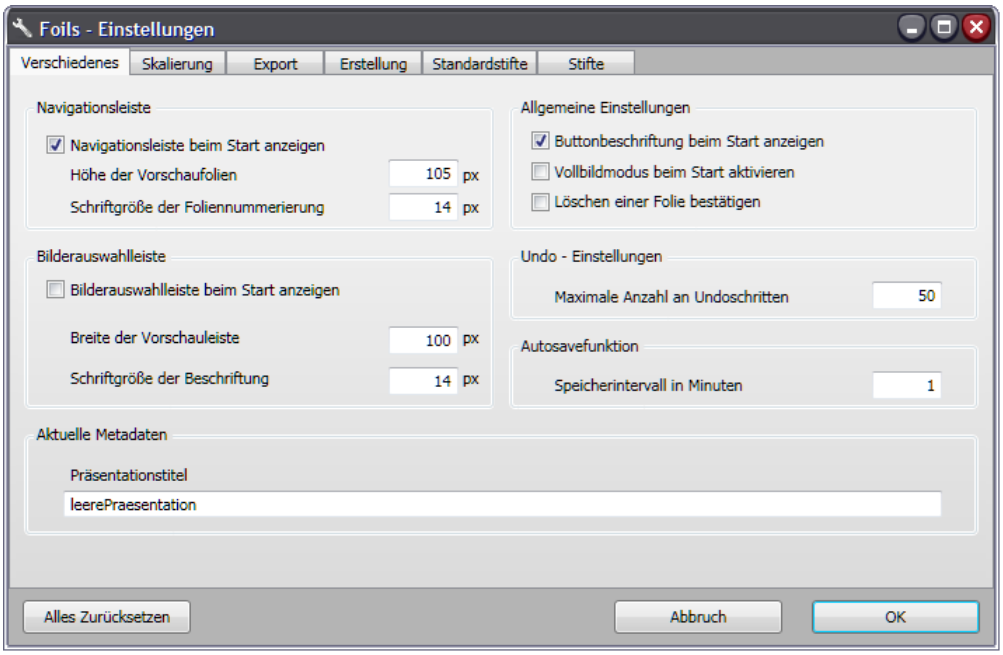

Abbildung 7.1.: Einstellungsdialog mit dem gewähltem Tab „Verschiedenes".

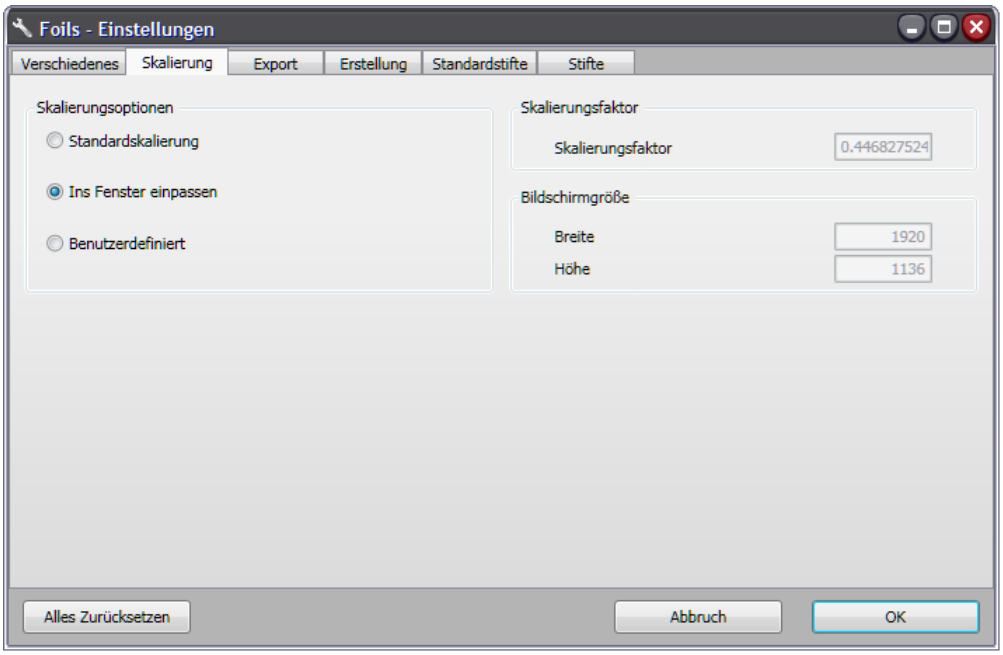

Abbildung 7.2.: Einstellungsdialog mit dem gewähltem Tab „Skalierung".

Der Einstellungsdialog ist ein wichtiges Element in Foils. Über ihn können alle möglichen Einstellungen eingesehen, verändert und wieder auf ihren Standardzustand

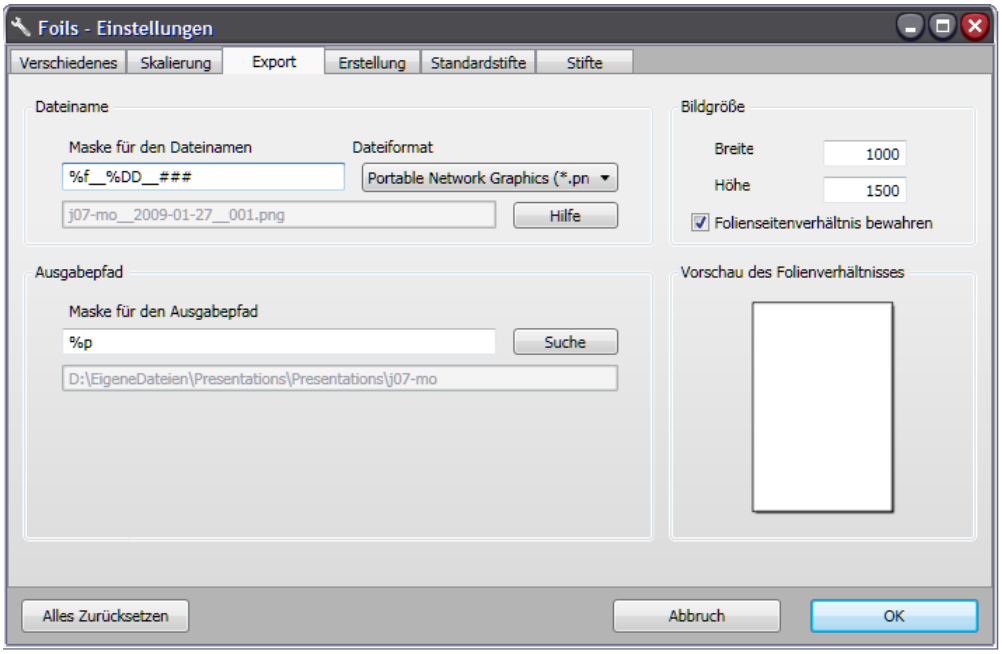

Abbildung 7.3.: Einstellungsdialog mit dem gewähltem Tab „Export".

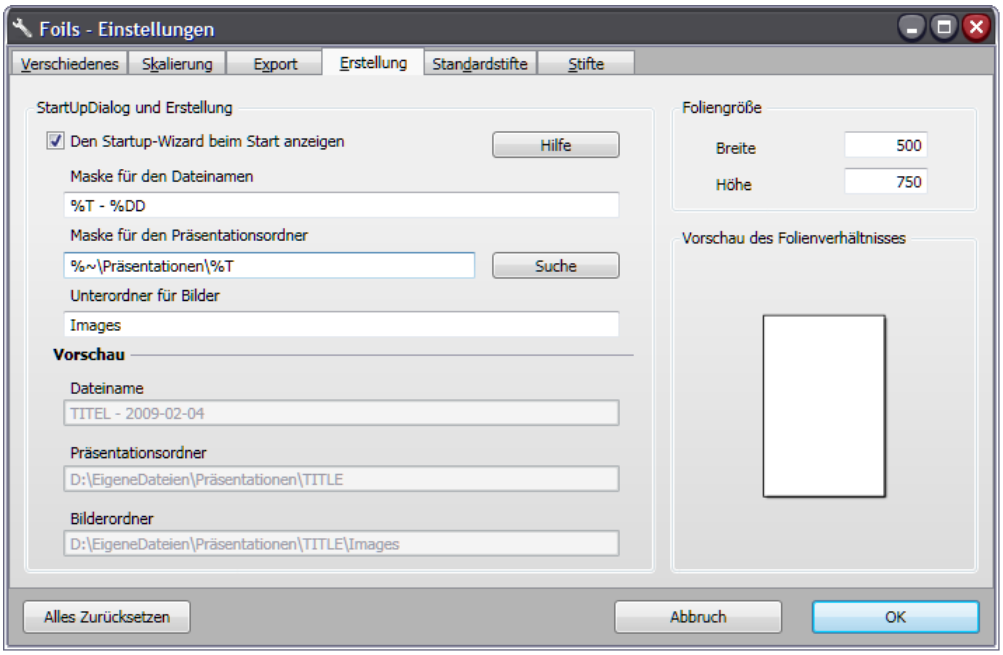

Abbildung 7.4.: Einstellungsdialog mit dem gewähltem Tab „Erstellung".

zurückgesetzt werden. Der Dialog ist mittels Tabs realisiert. Die einzelnen Tabs heißen Verschiedenes, Skalierung, Export, Erstellung, Standardstifte und Stifte.

Unter dem Tab „Verschiedenes", der in der Abb. 7.1 dargestellt wird, können alle Einstellungen, die aus der Tabelle 7.1 bekannt sind, erreicht und verändert werden.

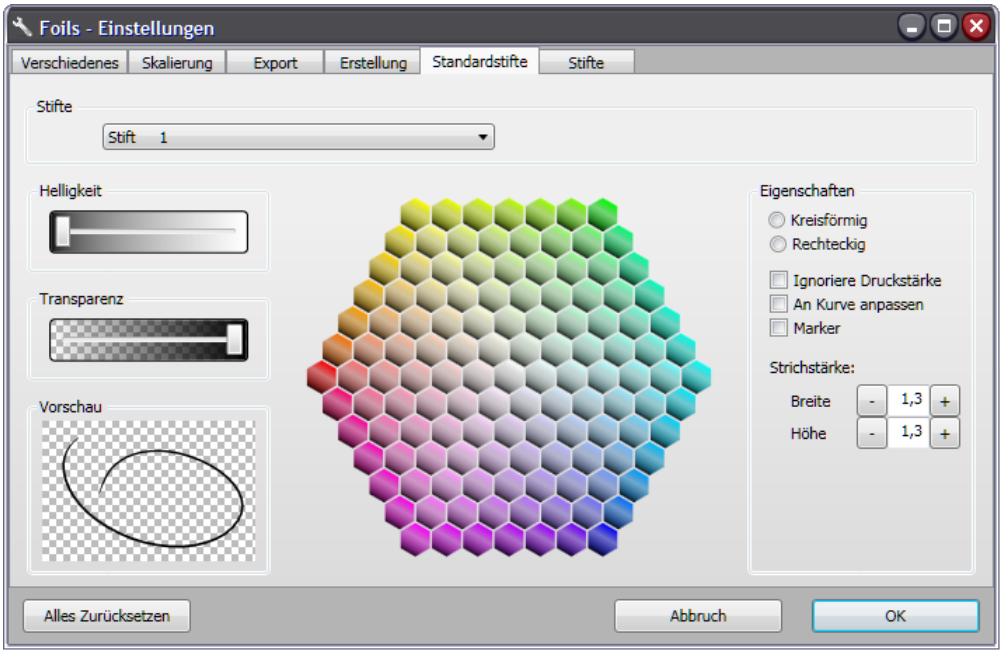

Abbildung 7.5.: Einstellungsdialog mit dem gewähltem Tab „Standardstifte”.

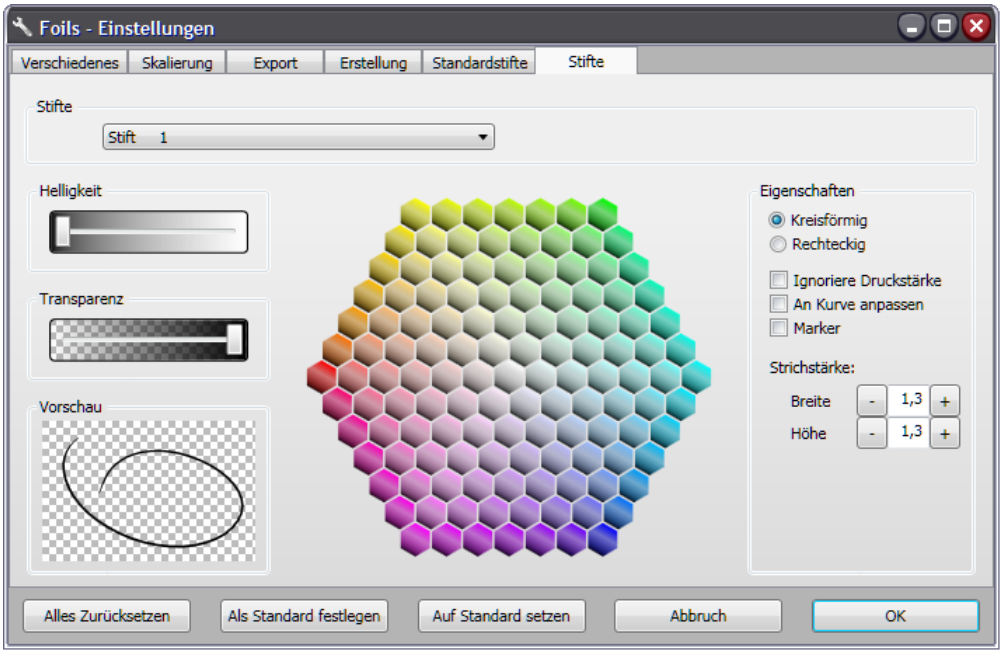

Abbildung 7.6.: Einstellungsdialog mit dem geöffnetem Tab „Stifte”.

Der Tab „Skalierung” bietet die Möglichkeit, das Skalierungverhalten zu beeinflussen. Die Einstellungen sind in der Tabelle 7.2 aufgeführt und ein Beispiel für den Dialog ist in der Abb. 7.2 abgebildet. Die Änderungen wirken sich direkt auf die Skalierung des Bearbeitungsfensters aus.

Unter dem Tab „Export", der in der Abb. 7.3 zu sehen ist, können die Einstellungen aus der Tabelle 7.3 verändert werden.

Der Tab „Erstellung" lässt es zu, die Einstellungen aus der Tabelle 7.4 zu bearbeiten. Der Dialog ist in der Abb. 7.4 dargestellt.

Die Abb. 7.5 zeigt den Tab „Standardstifte", der es ermöglicht jeden Stift zu ändern und die Änderungen zu speichern. Die Standardstifte werden erst dann verwendet, wenn eine neue Präsentation erstellt wird. Dieser und der nachfolgende Tab wurden mit Hilfe des Quellcodes von Shawn A. Van Ness [Nes09] erstellt.

Der letzte Tab „Stift" bietet die Möglichkeit die aktuellen Stifte zu bearbeiten und ist in der Abb. 7.6 zu sehen.

Unabhängig davon welcher Tab ausgewählt ist, werden drei Buttons am unteren Fensterrand dargeboten. Sie heißen „Alles Zurücksetzen", „Abbruch" und „OK". Durch Klicken auf „Alles Zurücksetzen" werden alle Einstellungen mit den Standardeinstellungen aus dem Schlüssel `HKEY_LOCAL_MACHINE` überschrieben. Der Button „Abbruch" bewirkt, dass die getätigten Einstellungen nicht übernommen werden und der Einstellungsdialog geschlossen wird. Hingegen übernimmt der Button „OK" alle Einstellungen und schließt den Einstellungsdialog.

Eine Besonderheit ist vorhanden, wenn der Tab „Stifte" ausgewählt ist. Es werden zwei weitere Buttons mit den Namen „Als Standard festlegen" und „Auf Standard setzen" sichtbar. „Als Standard festlegen" bewirkt, dass die Stifte als Standardstifte gesetzt werden. „Auf Standard setzen" bewirkt, dass die Stifte wieder auf ihre Standardeinstellungen gesetzt werden.

7.6. Gelöschte Folien wiederherstellen

A.3.12 ✓ (N) **U**: Falls eine Folie gelöscht wird, kann diese nicht mittels Undo wiederhergestellt werden. **L**: Das Löschen einer Folie soll mittels eines Undo-Schritts wieder rückgängig gemacht werden können.

Damit das Löschen einer Folie wieder rückgängig gemacht werden kann, wurde eine neue `DeleteSlideOperation`-Klasse mit der Basisklasse `UndoOperation` erstellt. Sie speichert die Slide, die gelöscht werden soll. Durch die `Do()`-Methode wird die zu löschende `Slide` aus der `SlideSequence` gelöscht. Beim Aufruf der `Undo()`-Methode wird die gespeicherte `Slide` wieder in die `SlideSequence` eingefügt.

7.7. Erweiterungen der Navigationsleiste

A.3.13 ✓ (N) Durch Doppelklick auf eine Folie F in der Navigationsleiste, soll zu F im Bearbeitungsfenster gewechselt werden. F ist dann im Präsentationsfenster auf der rechten Seite. Sofern F eine Vorgängerfolie V besitzt, wird V auf der linken Seite angezeigt. Ist dies nicht der Fall, befindet sich auf der linken Seite eine leere Folie.

A.3.14 ✓ (N) **U**: Derzeitig muss über das Drop-Down-Menü zu den gewünschten Folien navigiert werden. **L**: Zusätzlich zur Navigationsleiste soll es eine Blätterfunktion geben, mit der direkt zur vorherigen bzw. nachfolgenden Folie gesprungen werden kann.

Das Verhalten aus der Anforderung A.3.13 wurde umgesetzt. Das heißt, dass beim Wechseln der Folie im Bearbeitungsfenster, die Folien auf den Präsentationsansichten entsprechend ihre Positionen wechseln.

Die Blätterfunktion ist in der Navigationsleiste integriert, wie in Abb. 6.7 auf Seite 58 zu sehen ist. Falls die `NavigationBar` ausgeblendet wird, werden die Buttons, die für das Blättern zuständig sind, in der Menüleiste integriert.

7.8. Einbindung mehrerer Dateiformate

A.3.24 (✓)(-) Die Einbindung von PDFs, Source-Code, TIFF- und PNG-Dateien kann ermöglicht werden.

Die Einbindung von PDF oder Quellcode-Dateien wird in dieser Studienarbeit nicht erfolgen. Die Bilddateien TIFF und PNG sind hingegen zu den unterstützten Formaten hinzugefügt worden.

7.9. Zusammenfassung

Es wurde ein großer Teil der Anforderungen umgesetzt, aber ein kleiner Teil wurde nicht realisiert. Dieser Teil enthält noch einige Funktionalitäten die sich lohnen würden umgesetzt zu werden. Im nächsten Kapitel werden einige dieser Funktionalitäten genauer beschrieben und noch nicht genannte Möglichkeiten aufgeführt.

8. Ausblick

Es gibt einige Anforderungen, die im Rahmen dieser Studienarbeit nicht umgesetzt werden konnten, aber dem Benutzer fantastische Möglichkeiten bieten würden und zeigen, dass die Möglichkeiten des Programms noch nicht ausgeschöpft sind. Die noch nicht umgesetzten Anforderungen stellen damit einen Ansatz dar, in dessen Richtung das Programm noch weiter verbessert werden könnte. Einige interessante Möglichkeiten werden im Folgenden aufgeführt.

8.1. Vorschaufolien in der Navigationsleiste

Die Vorschaufolien in der Navigationsleiste werden momentan jeweils mit einem `SlideCanvas`-Objekt pro Folie dargestellt. Wenn mit der Scrollbar der Navigationsleiste versucht wird, die Folien durchzusehen, ist diese Aktion leider nicht sehr schnell. Dies wird wahrscheinlich durch das Neuzeichnen der Folien verursacht. Es sollte versucht werden, Vorschaubilder für die Folien zu generieren.

8.2. PDF in Foils

Das Laden von fertigen Präsentationen in Form von PDFs war ein gewünschtes Feature, das in der Anforderung A.3.25 spezifiziert wurde, aber nicht umgesetzt wurde. Es könnten Präsentation im PDF-Format gehalten werden und Vortragende wären sehr flexibel, falls ein Exkurs in ein Themengebiet stattfinden sollte, dass nicht von der vorbereiteten Präsentation abgedeckt wird. Auch wären Annotationen an den einzelnen Folien denkbar, um bestimmte Sachverhalte hervorzuheben oder schnell Korrekturen einzufügen. Vom Laden von fertigen PDF-Präsentationen in Foils könnten auch Benutzer profitieren, die Foils kaum verwenden würden.

8.3. Einfügen von Quellcode

In Foils ist es bis jetzt nur möglich, Bilder einzufügen oder direkt auf der Zeichenfläche zu schreiben oder zu malen. Besonders im Einsatz im Fachbereich der Informatik

als Lehrmittel, wäre es äußerst hilfreich dem Benutzer eine schnelle und einfache Methode an die Hand zu geben, den Quellcode in Präsentationen einzubinden. Denkbar wäre ein Importdialog, der die Auswahl der wichtigsten Codesegmente ermöglicht und nur diese einfügt.

8.4. Exportierung einer Präsentation

Momentan ist es möglich eine Präsentation als Bilder zu exportieren. Dabei liegt für jede Folie ein Bild im Ausgabeverzeichnis. Für Lehrzwecke wäre der Export in das PDF-Format sehr praktisch und wurde auch schon in der Anforderung A.3.23 gefordert. Dadurch läge ein PDF-Dokument vor, so dass alle Folien zu einem Dokument gebündelt wären und die Weitergabe einer Präsentation unkompliziert wird. Es wäre auch ein Deckblatt denkbar.

8.5. Seitenverhältnis von Bildern

Das Seitenverhältnis von Bilder wird in Foils standardäßig gewahrt und ist in der Anforderung A.2.4 festgehalten. Es kann natürlich vorkommen, dass ein Bild in einem Version vorliegt, in der es verzerrt ist und nur durch die Missachtung des Seitenverhältnisses richtig dargestellt werden könnte. In so einer Situation wäre es sehr hilfreich das Seitenverhältnis abzuschalten, was auch in der Anforderung A.3.8 vorgesehen war. So eine Funktion könnte evtl. durch einen Rechtsklick auf das Bild zur Verfügung gestellt werden oder mit einer Checkbox, die bei der Auswahl des Bildes am Bild selber sichtbar wird und somit anwählbar ist.

8.6. Bilder zuschneiden

In der Anforderung A.3.20 wurde schon angedeutet, dass das Zuschneiden von vorliegenden Bilder eine gewünschte Möglichkeit wäre. Der Vortragende kommt bestimmt irgendwann in die Verlegenheit ein Bild benutzen zu müssen, das zu groß ist. Eine Möglichkeiten wären das Bild zu verwenden und das Beste aus der momentanen Situation zu machen. Eine weitere Möglichkeit wäre Foils zu verlassen und mit einen Bildbearbeitungsprogramm das Bild zuzuschneiden. In so einer Lage, wäre das schnelle Zuschneiden von einem Bild in Foils sehr praktisch und würde den Vortragenden nicht in Bedrängnis bringen.

8.7. Fortsetzen und Zusammenführen mehrere Präsentationen

Bei einer Reihe von Präsentation ist es sehr nützlich nochmal mit ein paar Folien das Wissen der Zuhörer aufzufrischen, also eine Präsentation fortzusetzen. Dazu wäre eine Funktion beim Erstellung der Präsentation erforderlich, bei der die vorherige Präsentation ausgewählt werden kann und nach der Erstellung direkt verfügbar ist.

Es könnte also sein, dass Kopien der Folien der vorherigen Präsentation in die neue Präsentation eingefügt wurden. Eine weitere Möglichkeit wäre in einer Auswahl alle Folien der vorherigen Präsentation zur Verfügung zu stellen, so dass bei Bedarf auf die ein oder andere Folie zurückgegriffen werden kann.

Falls später mal die Präsentation als ganzes nochmal bearbeitet oder vorgetragen werden soll, wäre es sehr umständlich jede einzelne Präsentation einer Präsentationsreihe jeweils zu öffnen. Für so einen Fall könnte die Möglichkeit eingeräumt werden, mehrere Präsentation zu einem Präsentationsdokument zusammenzuführen. Diese Option würde sich auch wunderbar ergänzen lassen mit der Funktionalität aus Präsentationen PDFs zu erstellen.

Die Ideen für diese Funktionalitäten sind in den Anforderungen A.3.16 und A.3.17 festgehalten

8.8. Zusammenfassung

Alle diese Möglichkeiten, die hier aufgeführt wurden, wären eine wunderbare Ergänzung zum bestehenden Programm, sind aber nicht im momentanen Programm Foils enthalten. Das letzte Kapitel fasst nochmal zusammen was erreicht werden sollte und in dieser Arbeit erreicht wurde.

9. Zusammenfassung

Das vorrangige Ziel dieser Arbeit war es, den Speicherverbrauch und viele Mängel des verhergehenden Programms zu verbessern, sodass das Arbeiten ohne Abstürze möglich ist. Weitergehend sollten einige Teile des Programms, die Mängel aufwiesen, verbessert oder geändert werden. Die großen Themen waren die Menüleiste des Bearbeitungsfensters, die Navigationsleiste, die Bilderauswahlleiste und der Startup-Dialog. Zusätzlich waren es noch eine Menge an neuen Funktionalitäten gewünscht, die in das Programm eingebaut werden sollten. Ihr Zweck ist es, das Präsentieren von Lehrinhalten effektiver zu gestalten. Ein großer Bereich dieser neuen Funktionalitäten decke die Personalisierung des Programms ab.

Während dieser Arbeit wurden die Ursache des hohen Speicherverbrauchs und alle Beanstandungen, die im Bug-Report im Abschnitt A.1 auf Seite 81 beschrieben sind, behoben. Nahezu alle Änderungen am Programm, die im Change-Requests im Abschnitt A.2 auf Seite 83 aufgeführt sind, wurden umgesetzt. Die Menüleiste und der Startup-Dialog sind im Zuge dessen überarbeitet worden. Eine Navigationsleiste ist entsprechend den Anforderungen hinzugefügt worden. Dadurch ist die Handhabung des Programms verbessert worden. Zwei Drittel der Anforderungen des Extension-Requests im Abschnitt A.3 auf Seite 85 sind umgesetzt worden. Dies umfasst die Personalisierung des Programms, eine Autosave-Funktion, die Erstellung eines Backups beim Speichern, das Wiederherstellen von gelöschten Folien und eine Programmverknüpfung mit den Präsentationsdateien. Außerdem ist durch die Umstellung auf die Windows Presentation Foundation das Programm tiefgreifend umstrukturiert worden.

Aus dieser Studienarbeit ist das Programm Foils hervorgegangen. Die Mängel der seiner Vorgängerversion sind nicht mehr enthalten und Foils umfasst nun viele neue Funktionalitäten, die das Arbeiten und Präsentieren noch effektiver gestalten. Damit steht ein Werkzeug zur Verfügung, dass die Lehre unterstützen und bereichern kann.

A. Änderungsanforderungen

Die nachfolgenden Anforderungen wurden in Absprache mit Herrn Prof. Dr. Jürgen Ebert und Herrn Dr. Volker Riediger erstellt. Die aufgeführten Anforderungen beziehen sich auf die Studienarbeit Tim Steffens [Ste05]. Dabei sind die Anforderungen folgendermaßen zu lesen: Zuerst kommt eine Beschreibung des Bugs (**B**) oder einer störenden oder zu ändernden Eigenschaft (**C**), dann folgt die vermutete Ursache (**U**) und danach ein Lösungsvorschlag bzw. ein Ziel (**L**). Desweiteren sind alle Anforderungen mit Prioritäten versehen: (+) hoch, (N) normal und (-) niedrig.

Anforderungen die bis zum jetzigen Zeitpunkt (*5. Februar 2009*) erledigt wurden, sind mit einem Häkchen (✓), geänderte Anforderungen mit **Änderung**, nicht umgesetzte mit **Nicht Umgesetzt** und neue Anforderungen mit **Neu** markiert.

A.1. Bugreport

A.1.1 ✓ (+) **B**: Nach längerem Betrieb steigt der Speicherverbrauch massiv an, bis kein Speicher mehr zur Verfügung steht. **L**: Die Ursache hierfür muss analysiert und behoben werden.

A.1.2 ✓ (+) **B**: Die vorhandene Undo-Funktion ist wahrscheinlich die Ursache des zu hohen Speicherverbrauchs. **U**: Es werden evtl. zu viele Undoschritte gespeichert. **L**: Es muss untersucht werden, ob die Undo-Funktion für den hohen Speicherverbrauch verantwortlich ist. Ist dies der Fall, muss die Anzahl der gespeicherten Undoschritte reduziert oder die Undo-Funktion komplett aus dem Programm entfernt werden.

A.1.3 ✓ (+) **B**: Bei wachsender Anzahl an Folien, wird die Navigation über das Drop-Down-Menü sehr langsam. **U**: Die Vorschaubilder der Folien werden stets neu generiert. **L**: Die Vorschaubilder sollen nur einmalig generiert werden oder durch eine Navigationsleiste abgelöst werden.

A.1.4 ✓ (+) **B**: Wenn eine Folie F, wobei F nicht die letzte Folie sei, im Bearbeitungsfenster ausgewählt ist und eine neue Folie N hinzugefügt wird, wechseln alle Fenster auf N und N wird vor F eingefügt. **L**: N soll hinter F eingeordnet werden und die Vorgängerfolie F links und N rechts im Präsentationsfenster dargestellt werden.

A.1.5 ✓ (N) **B**: Die Lasso-Funktion wird bei längerer Benutzung langsam. **U**: Zu oft wird die aktuelle Position abgetastet und werden die dabei ermittelten Punkte neugezeichnet. **L**: Die Anzahl der Abtastungen muss reduziert und so selten wie möglich neugezeichnet werden.

A.1.6 ✓ (N) **B**: Wenn eine Präsentation geöffnet ist, existiert keine einfache Möglichkeit direkt an die Stelle der letzten Bearbeitung zurückzukehren. **L**: Dies soll durch die Speicherung des derzeitigen Bearbeitungspunkts und der Folienpositionen im Präsentationsfenster gelöst werden.

A.1.7 ✓ (N) **B**: Das Öffnen einer Präsentation aus einer bestehenden Präsentation heraus führt zu einem Fehler und ist nur auf Umwegen möglich. **L**: Dieser Fehler soll korrigiert werden.

A.1.8 ✓ (N) **B**: Wird die Undo-Funktion verwendet, springt der Fensterausschnitt des Bearbeitungsfensters ganz nach oben. **L**: Die Ausschnittsposition soll auch nach Anwendung der Undo-Funktion unverändert sein.

A.1.9 ✓ (N) **B**: Wenn das Programm minimiert ist und maximiert wird, wird das Präsentationsfenster nicht ebenfalls maximiert. **L**: Das Präsentationsfenster soll sich entsprechend dem Bearbeitungsfenster maximieren oder minimieren. Falls das Bearbeitungsfenster nur normal dargestellt wird und nicht maximiert ist, soll das Präsentationsfenster maximiert sein.

A.1.10 ✓ (N) **B**: In der Taskleiste befinden sich zwei Taskleisteneinträge des Programms. **L**: Es soll nur der Taskleisteneintrag des Bearbeitungsfensters in der Taskleiste vorhanden sein.

A.1.11 ✓ (N) **B**: Beim Maximieren des Bearbeitungsfensters und Wechsel in den Vollbildmodus entsteht eine ungenutzte Region im Bearbeitungsfenster. **L**: Diese freie Fläche soll beseitigt werden.

A.1.12 ✓ (N) **B**: Beim Exportieren der Folien als Bilder werden Überlagerungen von Markern als schwarze Balken dargestellt. **L**: Die Exportierfunktion soll verbessert werden, sodass diese Überlagerungen korrekt abgespeichert werden.

A.1.13 ✓ (N) **B**: Die Benennung der Folien beim Exportieren ist ungünstig gelöst. **L**: Beim Exportieren soll es möglich sein, den Namen des zu exportierenden Folienpakets und den Start der Nummerierung festzulegen.

A.1.14 **Neu**

✓ (+)**B**: Beim Schreiben im Bearbeitungsfenster ist nicht gewährleistet, dass die Größe der Schreibfläche bei jeder Haltungsweise des Tablet PCs gleich ist. **U**: Die Zeichenfläche skaliert sich entsprechend der Breite des Bearbeitungsfensters. **L**: Eine Folie soll immer auf das Minimum aus der Breite und Höhe der aktuellen Bilschimgröße skaliert werden.

A.2. Change-Requests

A.2.1 ✓ (+) **U**: Der alte Name “Studienarbeit Tim Steffens” beschreibt das Programm nicht passend. **L**: Der neue Name des Programms muss „Foils“ lauten.

A.2.2 ✓ (+) **C**: Die derzeitige Dateiendung lautet „xml” und kann dazu führen, dass xml-Dateien geöffnet werden, die nicht zu Foils gehören. **L**: Das Dateiformat muss in „foils” umbenannt werden.

A.2.3 ✓ (+) **U**: Beim Einfügen eines Bildes in eine Folie wird dieses meistens zu klein skaliert oder es erscheint eine Fehlermeldung, die besagt, dass das Bild auf dieser Bildschirmseite nicht geeinfügt werden kann. **L**: Das Bild muss direkt auf die bestmögliche Größe skaliert werden, so dass es die Präsentationsfläche so gut wie möglich ausfüllt und die Seitenverhältnisse gewahrt bleiben. Außerdem soll die Fehlermeldung verschwinden.

A.2.4 ✓ (+) **C**: Beim Ändern der Größe eines Bildes bleibt das Seitenverhältnis nicht gewahrt. **L**: Es muss eine Option geben, die den Verlust des Seitenverhältnisses verhindert.
Änderung: Das Seitenverhältnis wird standardmäßig gewahrt. Dies kann nicht abgeschaltet werden.

A.2.5 ✓ (+) **C**: Derzeitig wird der Knopf für die Löschung eines Bildes von mehreren Funktionen überlagert und verwirrt so den Benutzer. **L**: Die sich überlagernden Funktionen müssen auf mehrere Knöpfe aufgeteilt werden.

A.2.6 ✓ (+) **C**: Die Funktionen, die es ermöglichen, auf der derzeitigen Folien nach oben oder unten zu navigieren, werden nicht genutzt **L**: Diese Funktionen müssen entfernt werden.

A.2.7 ✓ (+) **C**: Wenn eine Folie gelöscht wird, wird nicht gefragt, ob diese wirklich gelöscht werden soll. **L**: Es muss vorher mittels einer Dialogbox nachgefragt werden, ob dieser Vorgang wirklich gewünscht ist.

A.2.8 ✓ (+) **C**: Die Demofunktion hat sich beim Arbeiten mit dem Programm nicht als nützlich erwiesen. **L**: Sie muss entfernt werden.

A.2.9 ✓ (N) **C**: Beim Exportieren der Folien als Bilder, werden alle Bilder in einem Verzeichnis mit den Präsentationsdateien abgespeichert. **L**: Alle Bilder sollen in einem eigenen Ordner abgespeichert werden.

A.2.10 ✓ (N) **C**: Verwendete Symbole in ”Foils” sind nicht eindeutig bzw. ändern ihre Bedeutung. **L**: Alle Symbole sollen sich an schon bekannten Symbolen orientieren oder die dahinter stehende Bedeutung schnell und einfach vermitteln. Die Symbole sollen außerdem klar voneinander unterscheidbar sein.

A.2.11 ✓ (N) **C**: Oft liegen Symbole beieinander, die thematisch nicht zusammengehören. **L**: Es sollen alle Symbole nach ihren Hauptgebieten aufgeteilt werden. Diese Hauptgebiete sollen sein: Dateioperationen, Stifte, Undo-Operationen, Bearbeitunsoperationen, Operationen auf Folien, Navigation, Menüs und weitere Modi.

A.2.12 ✓ (N) **C**: Das Drop-Down-Menü ist zu unhandlich und beansprucht zu viel Zeit. **L**: Das Drop-Down-Menü soll durch eine Navigationsleiste ersetzt werden.

A.2.13 ✓ (N) Die Folien-Navigationsleiste soll der Abb. A.1 auf Seite 87 nachempfunden werden und sich standardmäßig am unteren Rand befinden.

A.2.14 (N) Die Position der Navigationsleiste soll veränderbar und personalisierbar sein und folgende vier Möglichkeiten bieten: oben, unten, rechts, links.
Nicht Umgesetzt: Wurde aus Zeitmangel nicht umgesetzt.

A.2.15 ✓ (N) Die Navigationsleiste soll des Weiteren eine Scrollbar besitzen, sowie vier Buttons für das Wechseln zum Anfang bzw. Ende der Präsentation, zu der vorherigen und der nächsten Folie.

A.2.16 ✓ (N) **C**: Der derzeitige Wizard-Dialog ist umständlich zu bedienen und bietet nicht viele - eigentlich selbstverständliche - Möglichkeiten. **L**: Beim Start von Foils soll ein neuer Wizard-Dialog erscheinen, der den Benutzer beim Erstellen und Öffnen von Folien unterstützt und kurze Bedienschritte ermöglicht.

A.2.17 ✓ (N) Der Wizard-Dialog soll bediener- sowie einsteigerfreundlich sein. Einsteigerfreundlich bedeutet, dass jede Auswahlmöglichkeit einen treffenden Namen und eine verständliche nähere Erklärung besitzt.

A.2.18 ✓ (-) In der Navigationsleiste wird als letzte Folie immer eine leere Folie angezeigt: Durch Doppelklick auf sie wird eine neue Folie am Ende der Präsentation erstellt.

A.2.19 ✓ (-) Der Wizard-Dialog bietet folgende Wahlmöglichkeiten: leere Präsentation, Liste zuletzt bearbeiteter Präsentationen und direkt zum Arbeitsplatz (Bearbeitungsfenster) wechseln.

A.2.20 (-) **C**: Die Radierer-Funktion, mit der Striche wegradiert werden können, ist zu ungenau. **U**: Dies liegt an einer zu ungenauen Abtastung. **L**: Die Funktion kann durch Durchstreichen für den Benutzer vereinfacht werden.
Änderung: Da die Radierer-Funktion des neuen `InkCanvas`-Objekts sehr genau funktioniert, braucht diese Anforderungen nicht weiter bearbeitet werden.

A.3. Extension-Requests

A.3.1 ✓ (+) **U**: Es ist nicht möglich, eine Datei über den Datei-Explorer direkt zu öffnen. **L**: Die Dateiendung „foils“ muss mit dem Programm direkt assoziiert und beim Öffnen die Präsentationdatei geladen werden.

A.3.2 ✓ (+) **U**: Es kann kein Bild während einer laufenden Präsentation im System gesucht und zur Präsentation hinzugefügt werden. **L**: Das Suchen und Hinzufügen eines Bildes während der laufenden Präsentation muss ermöglicht werden. Das neue Bild wird in den Bilderordner kopiert.

A.3.3 ✓ (+) **U**: Es existiert keine Autosave-Funktion. **L**: Die Autosave-Funktion muss nachgerüstet werden, sodass in einem bestimmten Zeitintervall und evtl. mit jedem Hinzufügen einer Folie gespeichert wird.

A.3.4 ✓ (+) **U**: Es ist vorgekommen, dass das Programm während des Speicherns abgestürzt ist. Die zu überschreibende Präsentationsdatei war nicht mehr existent. **L**: Beim Speichern muss zuerst ein Backup der bisherigen Präsentation erstellt werden. Danach erst darf die aktuelle Präsentation gespeichert werden.

A.3.5 ✓ (N) Die Autosave-Funktion soll individuell einstellbar sein.

A.3.6 (N) **U**: Es gibt keine richtige Unterstützung für Rechts-/Linkshänder, weswegen wichtige Programmelemente verdeckt werden können. **L**: Dies soll durch einstellbare Anordnungsmöglichkeiten von allen verfügbaren Werkzeugen gelöst werden.
Nicht Umgesetzt: Wurde aus Zeitmangel nicht umgesetzt.

A.3.7 ✓ (N) Bilder sollen im Auswahlmodus eine Vorschau besitzen.

A.3.8 (N) Die Wahrung des Seitenverhältnisses soll durch einen Knopf, während ein Bild ausgewählt ist, zu de- bzw. aktivieren sein.
Nicht Umgesetzt: Wurde aus Zeitmangel nicht umgesetzt.

A.3.9 ✓ (N) Folgende Einstellungen sollen getroffen werden können, die beim Starten automatisch angewendet werden: Fullscreen & Textmodus.

A.3.10 ✓ (N) **U**: Die Farbe der einzelnen Stifte kann nur in der xml-Datei geändert werden. **L**: Bei den Stiften sollen die Farben personalisierbar sein und automatisch gespeichert werden, dabei soll es aber immer noch die Möglichkeit geben, auf Standardeinstellungen zurückzugreifen.

A.3.11 ✓ (N) Das Ein- und Ausblenden der Buttonbeschriftung, genannt Textmodus, soll personalisierbar sein.

A.3.12 ✓ (N) **U**: Falls eine Folie gelöscht wird, kann diese nicht mittels Undo wiederhergestellt werden. **L**: Das Löschen einer Folie soll mittels eines Undo-Schritts wieder rückgängig gemacht werden können.

A.3.13 ✓ (N) Durch Doppelklick auf eine Folie *F* in der Navigationsleiste, soll zu *F* im Bearbeitungsfenster gewechselt werden. *F* ist dann im Präsentationsfenster auf der rechten Seite. Sofern F eine Vorgängerfolie *V* besitzt, wird *V* auf der linken Seite angezeigt. Ist dies nicht der Fall, befindet sich auf der linken Seite eine leere Folie.

A.3.14 ✓ (N) **U**: Derzeitig muss über das Drop-Down-Menü zu den gewünschten Folien navigiert werden. **L**: Zusätzlich zur Navigationsleiste soll es eine Blätterfunktion geben, mit der direkt zur vorherigen bzw. nachfolgenden Folie gesprungen werden kann.

A.3.15 (N) **U**: Die Radier-Funktion arbeitet nicht erwartungsgemäß, d.h. das nur sehr unsauber radiert wird. **L**: Damit eine ähnliche Funktion existiert, soll es eine Art Tipex-Funktion geben, die alles mit Weiß bzw. der Hintergrundfarbe überstreicht.
Änderung: Da die Radier-Funktion des neuen `InkCanvas`-Objektes sehr genau funktioniert, braucht diese Anforderungen nicht bearbeitet zu werden.

A.3.16 (N) **U**: Wenn eine neue Präsentation begonnen wird, wäre es hilfreich auf die vorherigen Folien einfach zugreifen zu können. **L**: Es soll ermöglicht werden, die letzten *x* Folien aus einem bestehenden Foliensatz zu übernehmen.
Nicht Umgesetzt: Wurde aus Zeitmangel nicht umgesetzt.

A.3.17 (N) **U**: Momentan wird zur Bearbeitung der Folien viel Zeit verbaucht. Folien zusammenzuführen ist überhaupt nicht möglich. **L**: Das Zusammenführen verschiedener Präsentationen soll schnell und einfach möglich sein.
Nicht Umgesetzt: Wurde aus Zeitmangel nicht umgesetzt.

A.3.18 ✓ (-) Die im Wizard-Dialog getroffenen Entscheidungen können gespeichert und beim nächsten Programmstart automatisch angewendet werden.

A.3.19 (-) Bilder können über das Windows-Clipboard (die Kopierfunktion) eingefügt werden.
Nicht Umgesetzt: Wurde aus Zeitmangel nicht umgesetzt.

A.3.20 (-) Zuschneiden und Maskieren von eingefügten Bildern kann möglich sein und muss - falls möglich - pro Instanz erfolgen.
Nicht Umgesetzt

A.3.21 ✓ (-) Einstellungen, die personalisiert werden können, sollen über einen Einstellungsdialog erreichbar sein und verändert werden können.

A.3.22 ✓ (-) Gespeicherte Einstellungen können für jeden Benutzer einzeln abgespeichert werden.

A.3.23 (-) Eine verlustfreie PDF-Erstellung mit evtl. einem Titel kann ermöglicht werden.
Nicht Umgesetzt: Wurde aus Zeitmangel nicht umgesetzt.

A.3.24 (✓)(-) Die Einbindung von PDFs, Source-Code, TIFF- und PNG-Dateien kann ermöglicht werden.

A.3.25 (-) PDFs können wie Präsentationen geöffnet und wie normale Präsentationen behandelt werden, sodass in den Folien Inhalte markiert, Bemerkungen angebracht oder sogar neue Folien hinzugefügt werden können.
Nicht Umgesetzt: Wurde aus Zeitmangel nicht umgesetzt.

A.3.26 (N) Wenn eine Folie, die im Querformat vorliegt, dargestellt werden soll, soll nur diese im Präsentationfenster komplett angezeigt werden.
Nicht Umgesetzt

A.3.27 (-) Es kann auf Wunsch statt einer Folie im Hochformat auch eine Folie im Querformat eingefügt werden.
Nicht Umgesetzt: Wurde aus Zeitmangel nicht umgesetzt.

A.4. Abbildungen

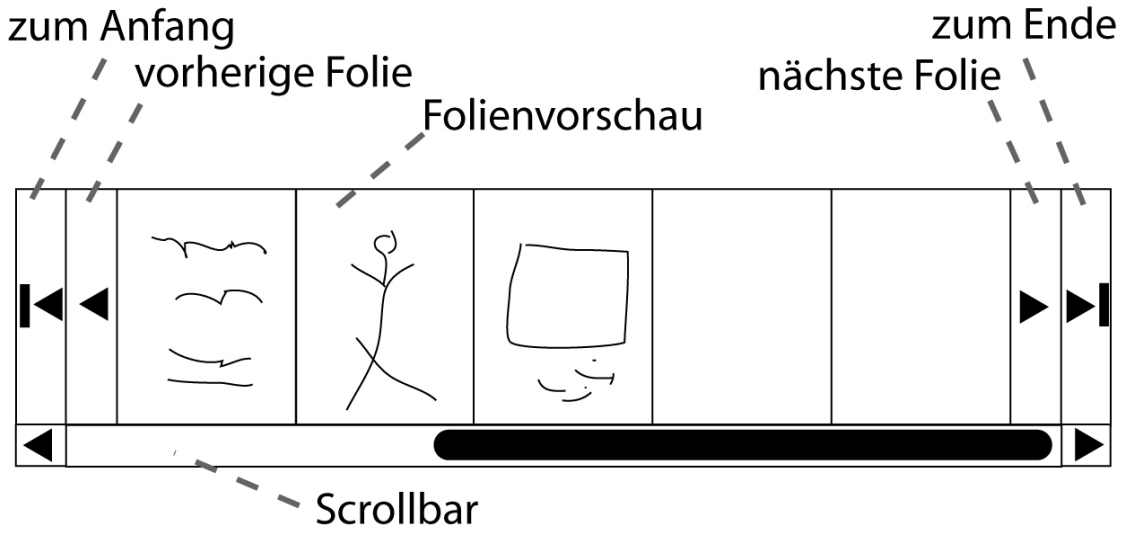

Abbildung A.1.: Die Navigationsleiste, die in Foils das Drop-Down-Menü ersetzen soll.

B. Mängelliste

Diese Mängelliste ist aus den Anforderungen der Studienarbeit von Tim Steffens [Ste05] und aus den Fehlerlisten und den Vorschlägen der Betreuer zusammengestellt worden.

B.1 System zu langsam bei längerem Betrieb

B.2 Fehlende Individualisierbarkeit und Speicherung von Einstellungen

B.3 Benutzerinterface überholungsbedürftig

B.4 Symbole nicht eindeutig

B.5 Einfache Navigation durch die Folien im Bearbeitungsfenster fehlt

B.6 Bei wachsender Anzahl an Folien Navigation über Drop-Down-Menü sehr langwierig

B.7 Keine Benutzerverwaltung / Profilverwaltung

B.8 Keine richtige Unterstützung für Rechts- / Linkshänder, daraus folgt Verdeckung von wichtigen Programmelemente

B.9 Keine direkte Speicherung in PDF-Format möglich

B.10 Keine Speicherung von Überlagerungen in den Bildern

B.11 Keine Textbausteine (handschriftlich oder maschinell)

B.12 Verbesserung der Lasso-Funktion notwendig

B.13 Navigationspfeile in der Symbolleiste überarbeiten oder entfernen

B.14 Undo-Funktion belastet den Speicher übermäßig

B.15 Einfügen von Bildern umständlich

B.16 Keine Autosave-Funktion vorhanden

B.17 Keine Möglichkeit zur letzten Folie zurückzukehren beim Öffnen einer alten Präsentation

B.18 Farben der Stifte nicht variierbar

B.19 Benennung der Folien beim Exportieren ungünstig

B.20 Öffnen aus einer bestehenden Präsentation heraus fehlerbehaftet, nur auf Umwegen möglich

C. Fragenkatalog

Hier stehen nach Kategorien geordnet mehrere Fragen, die im Rahmen der Anforderungserhebung und Mängelidentifizierung den Stakeholdern gestellt wurden.

C.1. Fragen zum Erscheinungsbild

C.1.1 Wie soll das Programm starten?

C.1.2 Hat es einen Startbildschirm, auf dem zunächst eine Auswahl von Aktionen getätigt werden kann?

C.1.3 Wie sollte so ein Startbildschirm aussehen?

C.1.4 Wenn es keinen Startbildschrim gibt, soll die zuletzt bearbeitete Folie geladen werden?

C.1.5 Wie soll die Aufteilung des Bildschirms für das Programm sein?

C.1.6 Ist die Aufteilung des bestehenden Programms bereits ausreichend ausgereift ?

C.1.7 Was stört Sie am derzeitigen Erscheinungsbild des Programms am meisten?

C.1.8 Soll das Erscheinungsbild anpassbar sein?

C.1.9 Sollen bestimmte, vorgenommene Veränderungen am Erscheinungsbild individuell abgespeichert werden?

C.2. Generelle Fragen

C.2.1 Wie würden Sie das Programm nennen, dass im Rahmen dieser Studienarbeit entwickelt werden soll?

C.2.2 Besitzt das Programm unnötige Eigenschaften?

C.2.3 Welche wichtigen Eigenschaften fehlen dem Programm?

C.2.4 Welche davon sind unabdingbar?

C.2.5 Welche sind optional?

C.2.6 Welche besonderen Eigenschaften fänden Sie wichtig, die für die Umsetzung im Rahmen dieser Arbeit allerdings ihrer Meinung nach zu aufwendig sind?

C.2.7 Welche wichtigen Eigenschaften hat das Programm, die unbedingt erhalten bleiben sollen?

C.2.8 Welche Eigenschaften sind bisher nicht zufriedenstellend umgesetzt, für das Arbeiten mit diesem Programm jedoch äußerst wichtig?

C.2.9 Welche sind gut umgesetzt?

C.2.10 Inwieweit sollte das Programm individualisierbar sein?

C.3. Fragen zu bestehenden Mängeln

C.3.1 Welche Mängel besitzt das Programm?

C.3.2 Welche dieser Mängel sind am Gravierendsten?

C.3.3 Welche dieser Mängel sind tolerierbar?

C.4. Fragen zur Softwareentwicklung

C.4.1 Welches Betriebssystem würden Sie für dieses Programm vorziehen?

C.4.2 Welche Programmiersprache würde für Sie in Frage kommen?

C.4.3 Was sind softwaretechnische Mängel an dem bestehenden Programm?

C.4.4 Was würden Sie am Softwaredesign ändern?

C.4.5 Was gefällt Ihnen am Softwaredesign?

Habe ich eine Frage vergessen? Wenn ja, wie lautet die Fragen und ihre Anwort?

D. Diagramme zum Speicher- und Zeitverbrauch

Alle Diagramme der Reihe “jo7-mo” sind hier nochmals aufgeführt. So ist es möglich sie mit den anderen Diagrammen der verwendeten Präsentationen zu vergleichen.

Die Messungen wurden mit insgesamt vier Präsentationen, mit jeweils unterschiedlicher Anzahl an Folien durchgeführt. Die verwendeten Präsentationen sind während Vorlesungen in den letzten Jahren entstanden. Die Ergebnisse der Messungen wurden aufgearbeitet, sodass sie als Diagramme zur Verfügung stehen. Die Abb. D.1 beinhaltet vier Messergebnisse von verschiedenen Präsentationen. Jede der erstellten Graphiken beinhaltet hierbei je zwei Diagramme. Das obere Diagramm zeigt den Speicherverbrauch, das untere Diagramm die Dauer einer ausgeführten Operation. Die Skalierungen können dabei variieren. Im oberen Diagramm sind zwei Datensätze untergebracht, die zueinander unterschiedlich skaliert sind. Die x-Achse entspricht getätigten `Do()`-Operationen. `Do()`-Operation ist in diesem Fall jede Operation, die beim Ausführen der `Do()`-Funktion des Undo-Stacks, anfällt.

Weitere Anmerkungen zu den Diagrammen finden sie im Abschnitt 4.2 auf Seite 25.

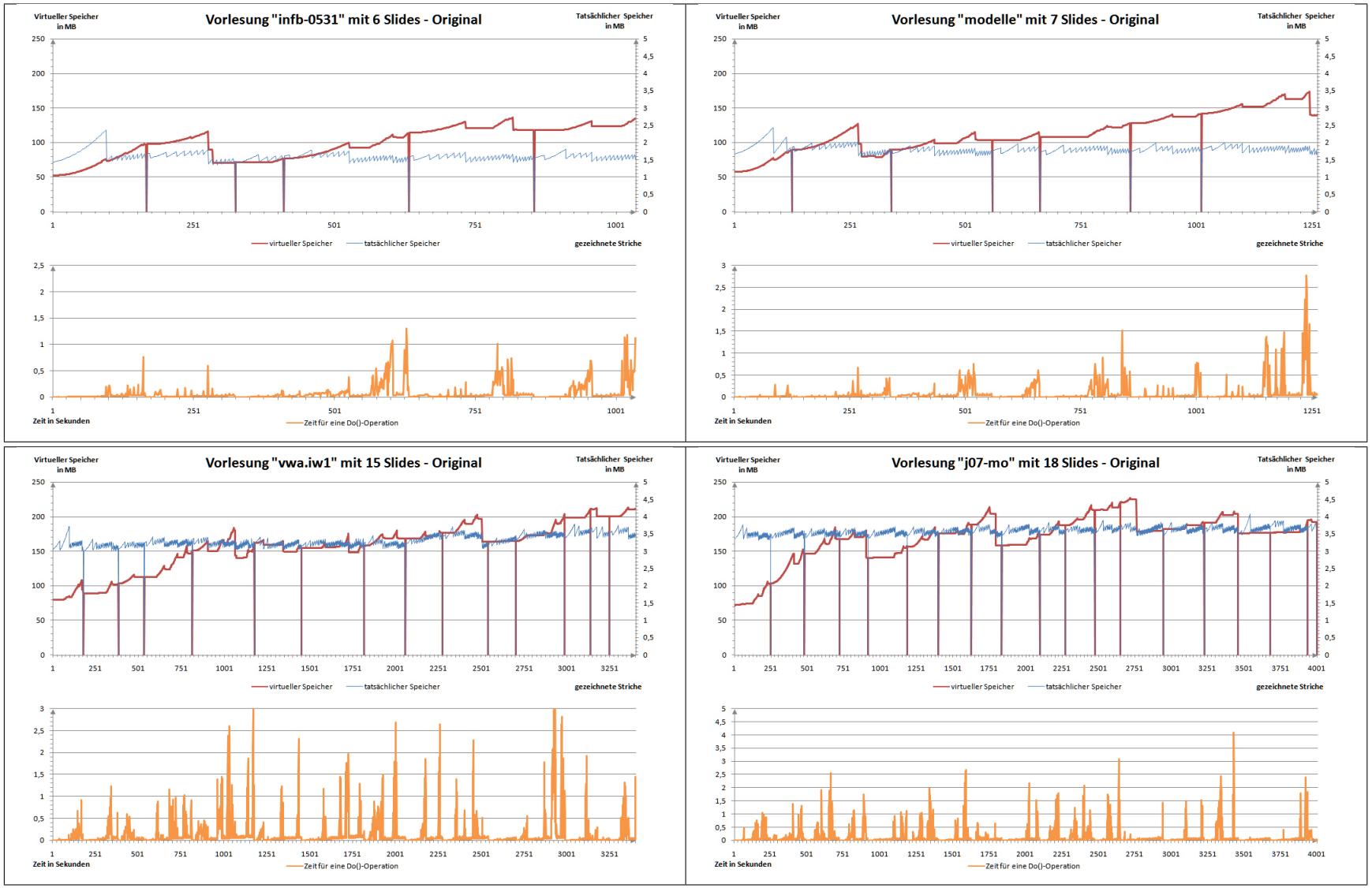

Abbildung D.1.: Messergebnisse der Originalversion von Foils.

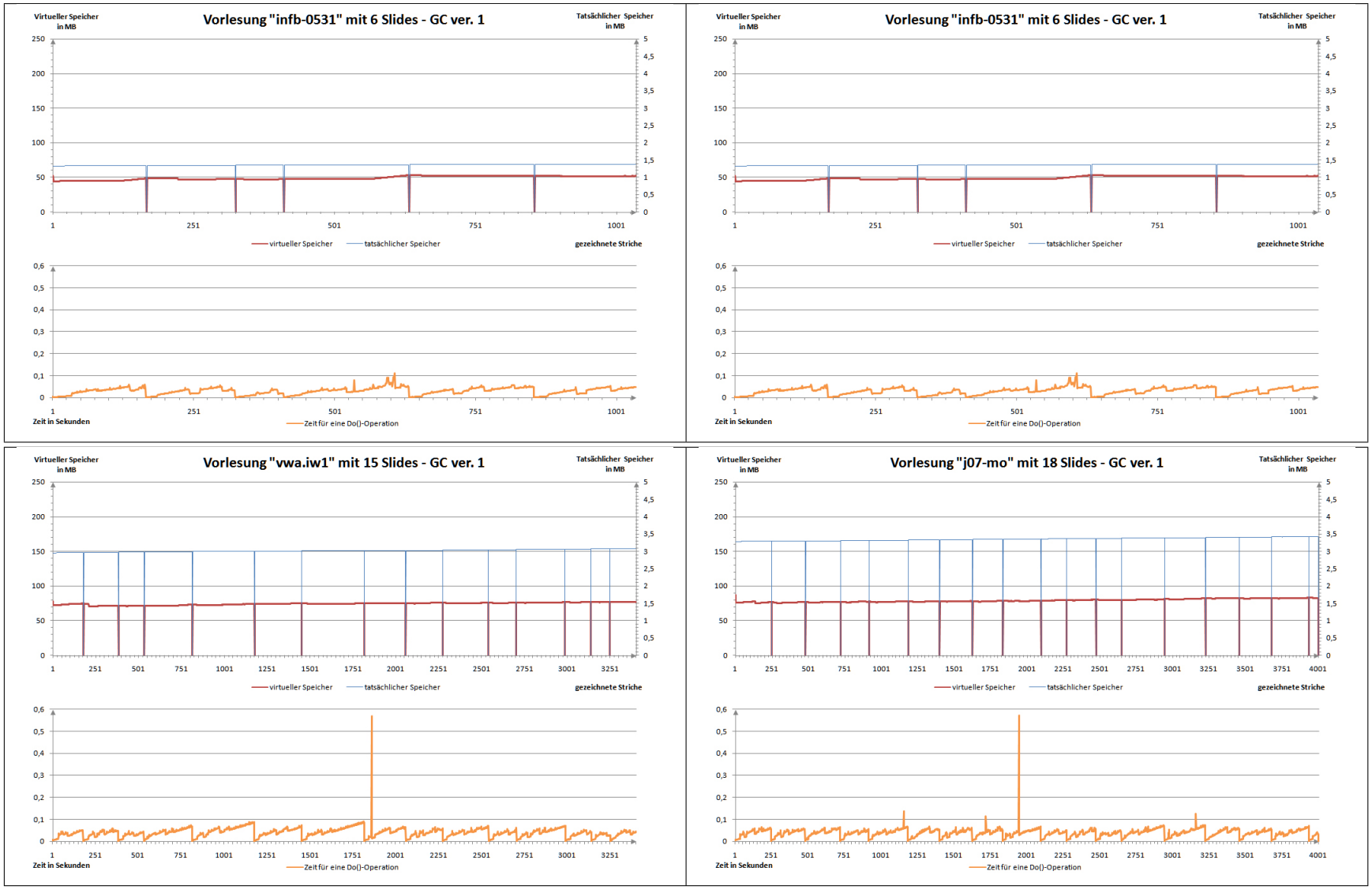

Abbildung D.2.: Auswirkungen nur mit dem GC-Aufruf `GC.Collect()`.

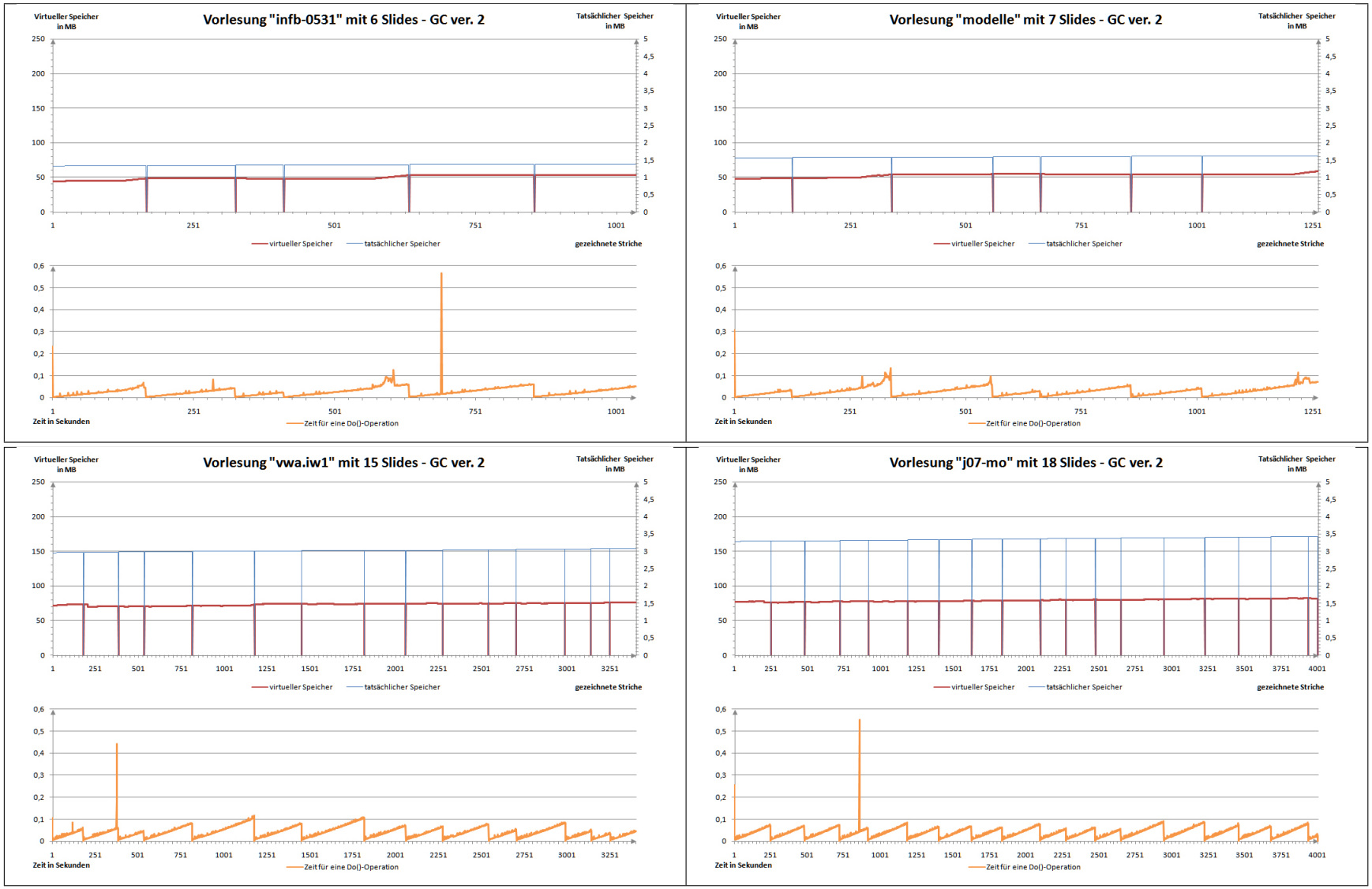

Abbildung D.3.: Auswirkung einer GC mit dem zusätzlichen Aufruf `GC.WaitForPendingFinalizers()`.

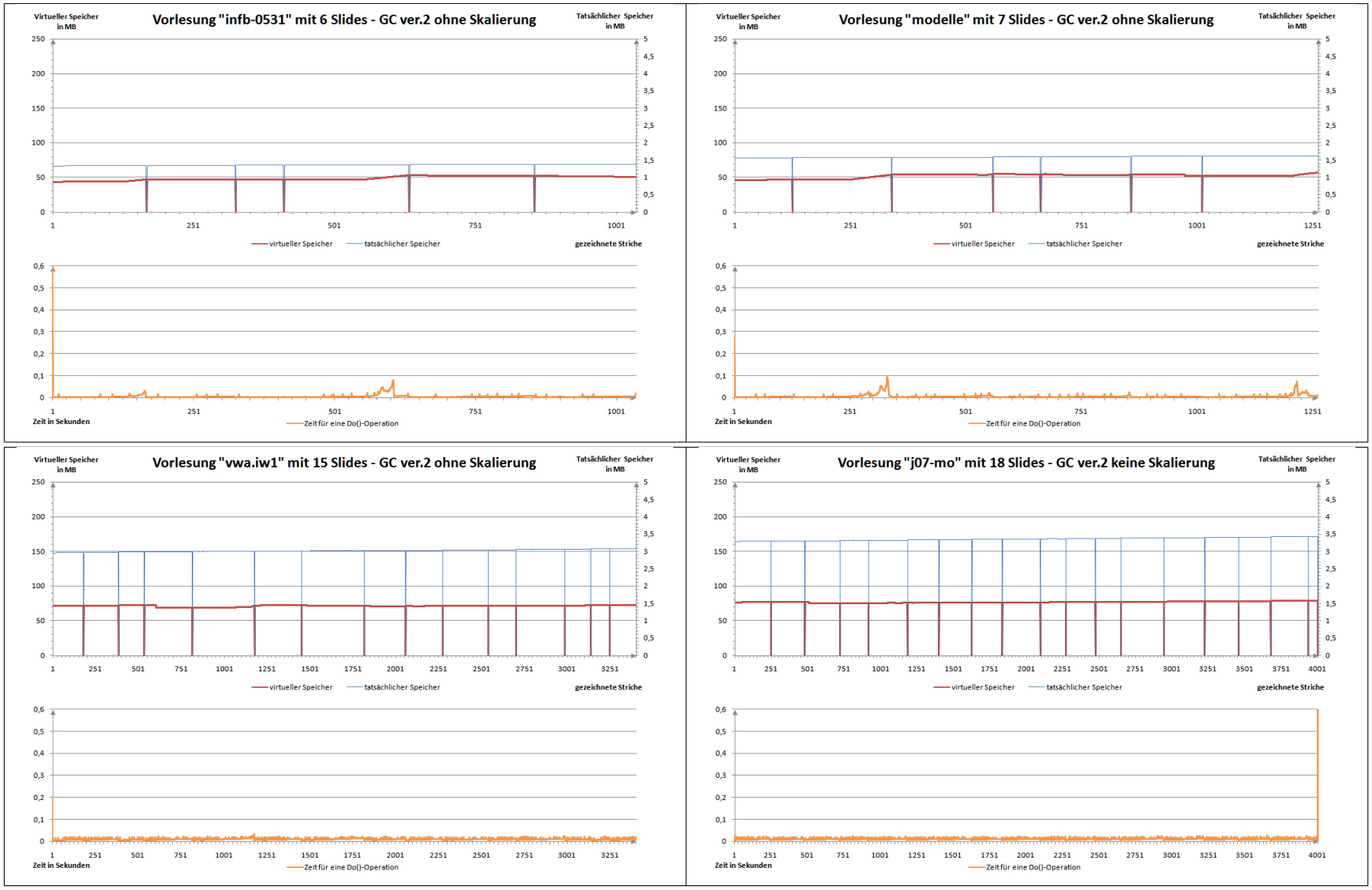

Abbildung D.4.: Richtige GC ohne Skalierung der `Ink`-Objekte.

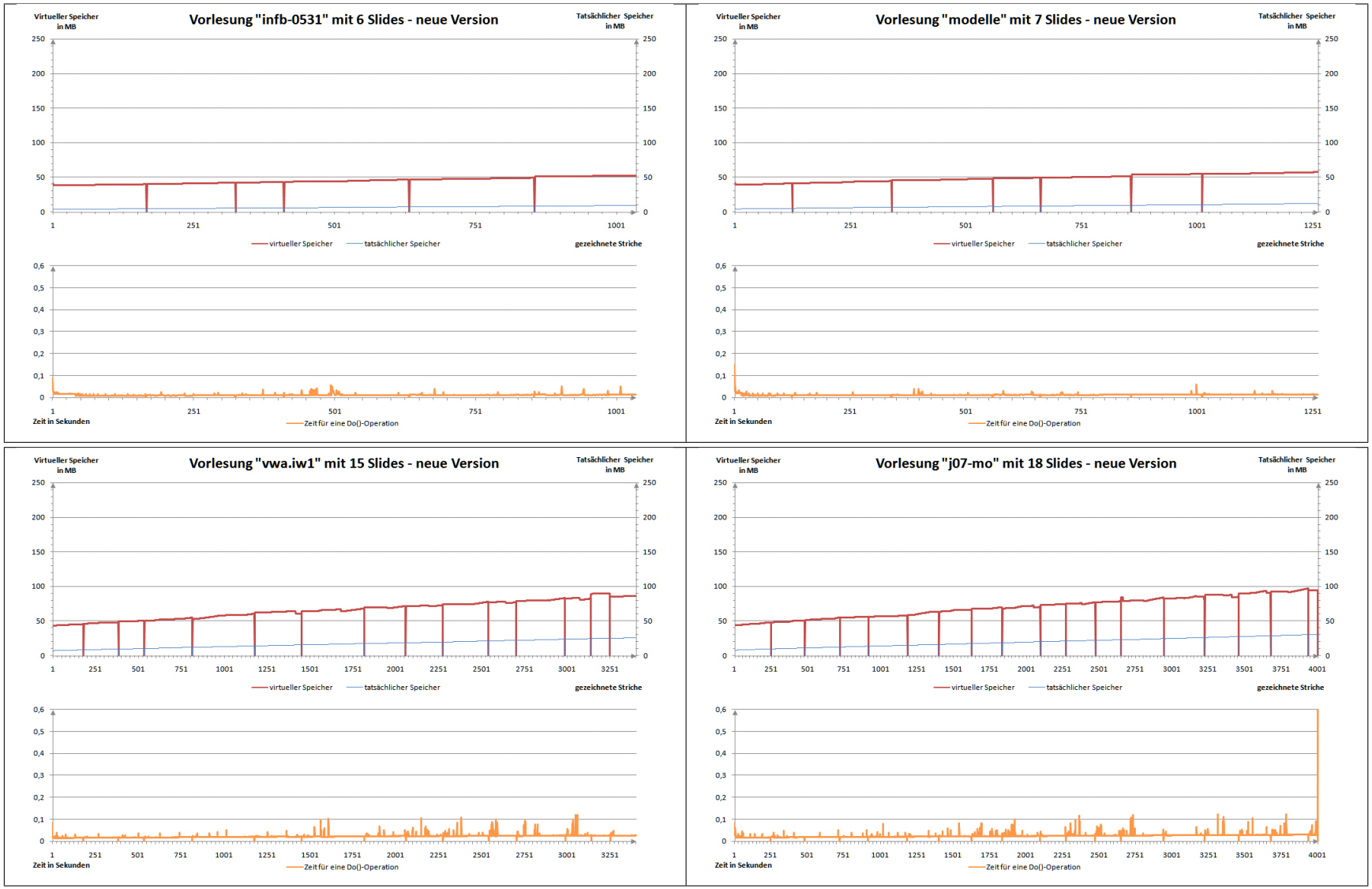

Abbildung D.5.: Messergebnisse für die neue Implementation.

E. Baustelle Foils

Bevor mit den Arbeiten an der Baustelle Foils begonnen werden kann, müssen zunächst ein paar Hürden genommen werden.

Zum Entwickeln kann ich die Programme

- Visual Studio 2008 mit Service Pack 1
- Expression Blend 2 mit Service Pack 1

empfehlen. Die Systemvoraussetzungen für das Entwickeln mit den Entwicklungsumgebungen sind

- .Net Framework 3.5 mit Service Pack 1 und
- 2 GB Arbeitsspeicher.

Das Service Pack 1 für .Net Framework 3.5 sollte unbedingt installiert werden, weil ein paar Unterschiede in den Basisbibliotheken der WPF existieren, die den Code sonst nicht auf Anhieb kompilieren lassen. 2 GB an Arbeitsspeicher sind notwendig, wenn gleichzeitig Visual Studio und Expression Blend betrieben werden.

Außerdem kann ich die Windows SDK Version 6.1 empfehlen, da sie viele Programmbeispiele enthält und so Denkanstöße bietet.

E.1. Wegweiser in der Projektstruktur

Im Hauptverzeichnis “Foils” des Quellcods gibt es vier wichtige Dateien und weitere vier wichtige Ordner. Die vier Dateien sind App.xaml, MainMethod.cs, Presentation.cs und ResourceDictionary.xaml. Hinter der Datei App.xaml verbirgt sich die Klasse `App`. Sie ist für das Starten der GUI-Oberfläche verantwortlich. In der Klasse `MainMethod` ist nur eine `main`-Methode, in der das Programm startet. Die Klasse `Presentation` bietet die Lade-, Speicher- und Exportierfunktionen für das Programm. In der Datei ResourceDictionary.xaml sind Styles definiert für die verschiedenen Elemente in Foils.

Die vier Verzeichnisse im Wurzelverzeichnis sind: BaseClasses, GUI, Operations und Resources. Im Verzeichnis BaseClasses sind die verschiedene Basisklassen zu finden, ohne die Foils nicht funktionieren würde. Das Verzeichnis GUI beherbergt alle

Elemente, die für die Benutzungsoberfläche verwendet werden. Im Verzeichnis „Operations" sind alle Kommando-Objekte einschließlich der `UndoStack`-Klasse enthalten. Das letzte Verzeichnis beinhaltet alle Bilder, die in Foils verwendet werden.

E.2. Die Replay-Funktion

```
1    public StartUpDialog()
2    {
3        ...
4        _ShouldBeReplayed.Visibility = Visibility.Visible;
5    }
```

Listing E.1: Ein C#-Destruktor

Die in Abschnitt 4.1 auf Seite 23 vorgestellte Replay-Funktion ist in der normalen Version von Foils nicht verfügbar. Damit sie genutzt werden kann muss in der Klasse `StartUpDialog` die Zeile aus Listing E.1 zum Konstruktor hinzugefügt werden.

Im StartUp-Dialog erscheint dann eine Checkbox mit dem Namen "Beim Öffnen einer Präsentation, diese abspielen.". Wird sie aktiviert und danach eine Präsentation geöffnet, wird diese Präsentation abgespielt. Da diese Methode zum Messen des Speicher- und Zeitverbrauchs gedacht war, wird im Ausführungsverzeichnis eine CSV-Datei mit dem Namen "results of <Dateiname der Präsentation> - SlideCanvas.csv" erstellt. Die eigentlichen Methoden, die für das Abspielen einer Präsentation wichtig sind, sind in der Klasse `Presentation` enthalten.

F. Inhalt der CD

Auf der CD sind die Verzeichnisse

- bin,
- sa-steffens,
- sample,
- src und
- thesis

enthalten.

Im Verzeichnis „bin" befindet sich das Programm in zwei Versionen. Die Version „Foils - Setup.exe" beinhaltet neben Foils noch .Net 3.5 SP 1. Es installiert automatisch die erforderlichen Bibliotheken und Foils auf dem Zielrechner. Die andere Version „Foils - Setup (minimal).exe" ist nur eine Minimalinstallation. Es wird nur Foils installiert.

Das Verzeichnis „sa-steffens" enthält den kompletten Quellcode der Studienarbeit von Tim Steffens. Eine Installationsdatei für die Tablet PC SDK 1.7 ist ebenfalls enthalten.

Das Verzeichnis „sample" beinhaltet eine Beispielpräsentation. Sie ist eine Präsentation, die im Sommersemester 2007 im Rahmen des Java Vorkurses gehalten wurde.

Das Verzeichnis „src" enthält den kompletten Quellcode dieser Studienarbeit. Das Unterverzeichnis „Foils" enthält die Projektdatei für Visual Studio 2008. Diese Projektdatei ist ebenfalls die Projektdatei für das Setup-Projekt im Verzeichnis „src\Setup". Die Quelldateien für die verwendeten Icons befinden sich im Unterverzeichnis „icons". Dort ist auch das Archiv aus der Quelle [Her09] enthalten. Im Unterverzeichnis „Test" befindet sich eine Vorversion zu Foils, die die beschriebenen Korrekturen aus dem Kapitel 4 auf Seite 23 enthält.

Das Verzeichnis „thesis" beinhaltet diese Studienarbeit als PDF-Datei und in ihrer Rohform als Lyx-Dokument. Lyx ist ein Programm mit dem sich Latex-Dokumente einfach verfassen und verwalten lassen. Das Programm ist auf der Webseite `http://www.lyx.org/` zu finden. Es enthält auch eine Funktion, mit der es möglich ist, die Dokumente als Latex-Dateien (*.tex) zu exportieren.

Abbildungsverzeichnis

1.1. Anwendungsszenario für Foils. 1
1.2. Erste Version von Foils. Kontrollfenster (*links*) und das Präsentationsfenster mit den Folien (*rechts*). 2

3.1. Ein Klassendiagramm des Undo-Stacks und den verwendeten Kommando-Objekten. 15
3.2. Klassendiagramm der aktuellen internen `Slide`-Zusammensetzung und deren Darstellung durch ein `SlideCanvas`-Objekt. 21

4.1. Diagramm zum Zeitbedarf und Speicherverbrauch der unveränderten Foils Version. 25
4.2. Gegenüberstellung zweier Messreihen der gleichen Präsentation. 26
4.3. Messergebnisse mit einem GC-Aufruf nach einer ausgeführten Operation. 27
4.4. Messergebnisse für den korrekten GC-Aufruf. 28
4.5. Diagramm über die Zeitaufteilung mehrerer `InkChangeOperation`-Objekte. 29
4.6. Messergebnisse ohne Skalierung und mit korrektem GC-Aufruf. 31
4.7. Messergebnisse der neuen Implementierung von Foils. 32

5.1. Eine exportierte Folie mit Transparenzen. *Links*: Variante aus der Original Fassung von Foils; *rechts*: eigentliches Aussehen der Folie. 43
5.2. Der Exportierdialog von Foils um aus Folien Bilder zu generieren. . . . 44
5.3. Skalierungsverhalten bei unterschiedlicher Benutzung des Tablet PC's. Oben ist jeweils ein Bearbeitungsfenster zu sehen, unten das dargestellte Resultat. 46

6.1. Gezogener Rahmen zum Vergrößern des Bildes (*links*). Skalierte Bilder mit gesetzter `Stretch`-Property `Uniform` (*Mitte*) und `UniformToFill` (*rechts*) 51
6.2. Das neue Design nach dem Starten des Programms. 52
6.3. Werkzeugleiste mit (*oben*) und ohne Beschriftungstext (*unten*). 53
6.4. Das Dateimenü von Foils. 54
6.5. Einstellungsdialog für einen Stift. 54

6.6. Aktiviertes Bilderauswahlmenü auf der rechten Seite, durch dessen Hilfe Bilder in eine Präsentation eingefügt werden können. 55
6.7. Erste Umsetzung der `NavigationBar`. 58
6.8. Das Kontextmenü mit weiteren Operationen, die durch die Navigationsleiste ausgeführt werden können. 59
6.9. Der neue Wizarddialog mit Beispieldaten. 60
6.10 Dialog zum Erstellen einer Präsentation. 60

7.1. Einstellungsdialog mit dem gewähltem Tab „Verschiedenes”. 70
7.2. Einstellungsdialog mit dem gewähltem Tab „Skalierung”. 70
7.3. Einstellungsdialog mit dem gewähltem Tab „Export”. 71
7.4. Einstellungsdialog mit dem gewähltem Tab „Erstellung”. 71
7.5. Einstellungsdialog mit dem gewähltem Tab „Standardstifte”. 72
7.6. Einstellungsdialog mit dem geöffnetem Tab „Stifte”. 72

A.1. Die Navigationsleiste, die in Foils das Drop-Down-Menü ersetzen soll. . 87

D.1. Messergebnisse der Originalversion von Foils. 94
D.2. Auswirkungen nur mit dem GC-Aufruf `GC.Collect()`. 95
D.3. Auswirkung einer GC mit dem zusätzlichen Aufruf `GC.WaitForPendingFinalizers()`. 96
D.4. Richtige GC ohne Skalierung der `Ink`-Objekte. 97
D.5. Messergebnisse für die neue Implementation. 98

Tabellenverzeichnis

4.1. Ausführungszeiten für verschiedene Versionen des Programms Foils und Präsentationen. 32

5.1. Generelle Maskenoptionen. 45
5.2. Spezielle Muster für Ordner. 45
5.3. Spezielle Muster für Bilddateinamen zu exportierender Folien. 45

7.1. Allgemeine Einstellungen für das Arbeiten mit Foils. 67
7.2. Skalierungseinstellungen. 68
7.3. Exporteinstellungen 68
7.4. Erstellungseinstellungen 69
7.5. Stifteinstellungen des ersten Stifts. Alle restlichen sechs Stifte sind analog definiert. 69

Listings

2.1. Beispiel für eine Eigenschaft in C#. 5
2.2. Ein C#-Destruktor 8
2.3. Die compilerseitige Übersetzung des Destruktors in die Methode `Finalize()`. 8
2.4. Programmbeispiel für einen Destruktor mit dem `IDisposable`-Interface 10
2.5. Schematisches Beispiel für die Verwendung der **`using`**-Anweisung . . . 12

5.1. XML-Gerüst einer alten Präsentationsdatei von Foils 39
5.2. XML-Gerüst einer Präsentationsdatei von Foils. 40
5.3. Zu importierende Funktion aus der DLL-Datei: gdi32.dll. 43

E.1. Ein C#-Destruktor 100

Literaturverzeichnis

[Bur09] BURBECK, Steve: *Applications Programming in Smalltalk-80(TM): How to use Model-View-Controller (MVC)*. http://st-www.cs.uiuc.edu/users/smarch/st-docs/mvc.html. Version: Januar 2009

[GHJV94] GAMMA, Erich ; HELM, Richard ; JOHNSON, Raplh ; VLISSIDES, John: *Design Patterns - Elements of Reusable Object-Oriented Software*. Addison Wesley, 1994 (2. Auflage). – 127–134, 163–174, 233–242, 293–304 S.

[Gno] *Gnome ART*. http://art.gnome.org/

[Her09] HERRERA, Silvestre: *Dropline Neu!* http://art.gnome.org/download/themes/icon/1100/ICON-DroplineNeu.tar.bz2. Version: Januar 2009

[Hof09] HOFSTETTER, Ueli: *Heapverwaltung und Garbage Collection*. http://seal.ifi.uzh.ch/fileadmin/User%5fFilemount/Vorlesungs%5fFolien/Seminar%5fSE/SS06/hofstetter%5fgarbage%5fcollection.pdf. Version: Januar 2009

[Hub08] HUBER, Thomas C.: *Windows Presentation Foundation - Das umfassende Handbuch*. Galileo Press, 2008 (1. Auflage)

[JS03] JARRETT, Rob ; SU, Philip: *Building Tablet PC Applications*. Microsoft Press, 2003 (1. Auflage)

[Mic09a] MICROSOFT: *InkCanvas Editing Modes Sample*. http://msdn.microsoft.com/en-us/library/aa972158.aspx. Version: Januar 2009

[Mic09b] MICROSOFT: *Object.Finalize-Methode*. http://msdn.microsoft.com/de-de/library/system.object.finalize.aspx. Version: Januar 2009

[Mic09c] MICROSOFT: *State information is lost when you draw Ink to an image by using the Renderer object*. http://support.microsoft.com/kb/829824/en-us. Version: Januar 2009

[Nes09] NESS, Shawn A. V.: *ColorComb: Yet Another Color-Picker Dialog for WPF*. http://weblogs.asp.net/savanness/archive/2006/12/05/colorcomb-yet-another-color-picker-dialog-for-wpf.aspx. Version: Januar 2009

[Sha08] SHARP, John: *Visual Studio C# 2008 - Schritt für Schritt*. Microsoft Press Deutschland, 2008 (1. Auflage). – 289–303 S.

[Ste05] STEFFENS, Tim: *Entwicklung eines computergestützten Systems zur Verbesserung der visuellen Darbietung von Lehrinhalten*, Universität Koblenz-Landau, Campus Koblenz, Studienarbeit, März 2005

[Str09] STRIGL, Daniel: *High-Performance Timer in C#*. `http://www.codeproject.com/KB/cs/highperformancetimercshar.aspx`. Version: Januar 2009

[Wac09] WACOM: *Wie der kabel- und batterielose Wacom-Stift funktioniert*. `http://www.wacom-europe.com/_bib_user/downloads/tech_bam_de.pdf`. Version: Januar 2009

[Wil09] WILKINSON, Andrew: *DropDownButtons in WPF*. `http://andyonwpf.blogspot.com/2006/10/dropdownbuttons-in-wpf.html`. Version: Januar 2009