

Fachpraktische Besondere Lernleistung

---

Entwickeln eines Computerprogramms zum Einscannen  
von ISBN-Barcodes und Speicherung der  
Literatureinträge im BibTeX-Format

---

Leipzig, 18. Januar 2021

vorgelegt von  
Mika Miosge

**Außenbetreuer:** Stephan Keller (Softwareentwickler bei der Informatik DV GmbH)

**Innenbetreuer:** Matthias Richter (Fachlehrer für Informatik und Mathematik)

# Einleitung

In unserer heutigen Bildungslaufbahn kommt in der Regel niemand um das Schreiben wissenschaftlicher Arbeiten herum. Ein beliebtes Programm zum Erstellen dieser Arbeiten ist  $\text{\LaTeX}$ . Dieses ist kein What-you-see-is-what-you-get-Programm<sup>1</sup>, sondern es gibt bestimmte Makros, welche die Formatierung des Textes vorgeben. Dies bringt einige Vorteile zum Beispiel ist es einfacher mathematische Formeln auszudrücken. Außerdem wird es durch die individuelle und genau einstellbare Formatierung des Textes einfacher, sich an strenge formale Richtlinien, zum Beispiel einer Besonderen Lernleistung zu halten, da die Formatierung vom Computer übernommen wird.

Auch Quellen lassen sich leicht in  $\text{\LaTeX}$  anlegen. Dazu gibt es ein eigenes Dateiformat, welches  $\text{BibTeX}$  genannt wird. In dieser  $\text{BibTeX}$ -Datei sind die Informationen über die Werke verzeichnet, wie zum Beispiel Titel, Autor oder Verlag. Um das verwendete Werk zu zitieren, benötigt man einen  $\text{BibTeX}$ -Key, welcher für die eindeutige Identifizierung des Werkes im Text zuständig ist. Wenn man diesen Key mit dem Zitatbefehl von  $\text{\LaTeX}$  benutzt, wird das Werk automatisch an der Textstelle aufgeführt und im Literaturverzeichnis angehängt. Dabei kann man auch die Zitierweise oder die Anordnung des Literaturverzeichnisses individuell gestalten.

Jedoch ist es aufwendig, die Daten der verwendeten Werke manuell einzugeben. Um dieses Problem zu beheben, entwickle ich ein Programm, in dem die ISBN des Buches eingescannt wird, die  $\text{BibTeX}$ -Datei und der  $\text{BibTeX}$ -Key ausgewählt werden und die wichtigsten Daten des Werks danach automatisch der  $\text{BibTeX}$ -Datei hinzugefügt werden.

---

<sup>1</sup>Ein What-you-see-is-what-you-get-Programm bezeichnet ein Programmtyp von Texteditoren, welche das Dokument genauso anzeigen, wie es auch ausgegeben (z. B. ausgedruckt) werden würde.

# Inhaltsverzeichnis

<b>1</b>	<b>Theorie der Softwareentwicklung</b>	<b>4</b>
1.1	Klassische Softwareentwicklung	4
1.1.1	Funktionsweise der klassischen Softwareentwicklung	4
1.1.2	Ziele der klassischen Softwareentwicklung	6
1.1.3	Das Wasserfallmodell	6
1.2	Agile Softwareentwicklung	7
1.2.1	Leitsätze der Agilen Softwareentwicklung	7
1.2.2	Pair Programming	8
1.2.3	Ziele der Agilen Softwareentwicklung	8
1.2.4	Extreme Programming	9
1.3	Vergleich klassischer und agiler Softwareentwicklung	10
<b>2</b>	<b>Dokumentation</b>	<b>12</b>
2.1	Idee	12
2.1.1	Python	12
2.1.2	L <sup>A</sup> T <sub>E</sub> X	12
2.1.3	Quellen in L <sup>A</sup> T <sub>E</sub> X	13
2.1.4	Problem	14
2.1.5	Konzept des Programms	14
2.2	Umsetzung	15
2.2.1	Benutztes Modell der Softwareentwicklung	15
2.2.2	Graphical User Interface (GUI)	16
2.2.3	Scannereinbindung	19
2.2.4	Verarbeitung der Daten	22
2.2.5	GitHub	24
2.2.6	Docker	24
<b>3</b>	<b>Abschlussbetrachtung</b>	<b>26</b>
3.1	Zusammenfassung	26
3.2	Ausblick	27
3.3	Diskussion	27
3.4	Fazit	28

Literaturverzeichnis

Abbildungsverzeichnis

Quellcodeverzeichnis

Selbstständigkeitserklärung

# 1 Theorie der Softwareentwicklung

## 1.1 Klassische Softwareentwicklung

Die klassische oder auch statische Softwareentwicklung beschreibt Softwareentwicklungsprozesse, in denen der Software Development Life Cycle im Idealfall einmal durchgeführt wird.

Diese Modelle der Softwareentwicklung wurden eher für große Projekte konzipiert, in welchem viele Entwickler mitwirken müssen und die Anforderungen klar definiert sind.

### 1.1.1 Funktionsweise der klassischen Softwareentwicklung

In der klassischen Softwareentwicklung gibt es einen festen Ablaufplan, den sogenannten Software Development Life Cycle (SDLC). Dieser wird im Laufe der Entwicklung der Software einmal durchlaufen und liefert an dessen Ende die entwickelte Software. (Vgl. [Leau u. a. 2012](#), S.162)

Der SDLC unterscheidet sich leicht von den verschiedenen klassischen Modellen, wie zum Beispiel dem V-Modell oder dem Wasserfallmodell. Jedoch lässt sich der SDLC in der Regel auf 6 Grundarbeitsschritte zusammenfassen. Diese lassen sich in [Abbildung 1](#) erkennen. (Vgl. [Stoica u. a. 2013](#), S.64 – 65)

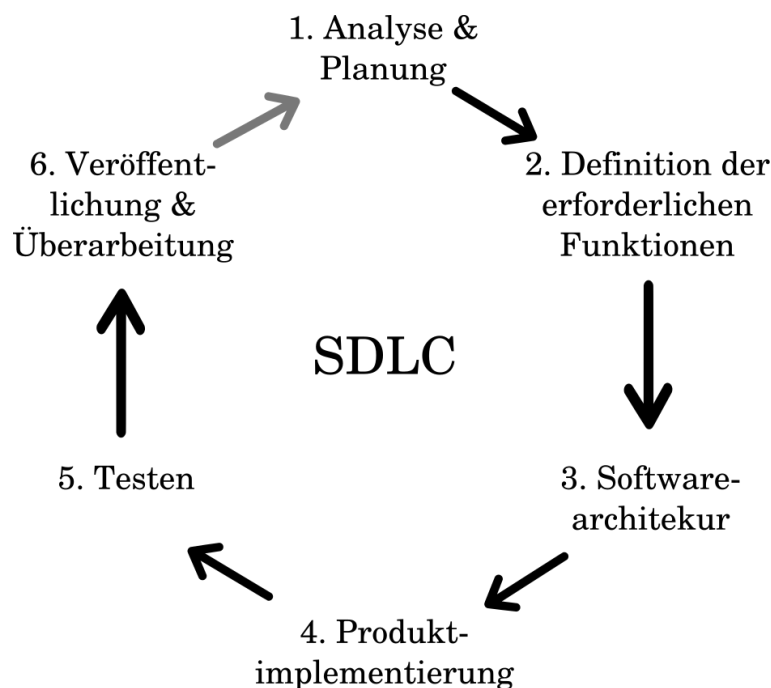


Abbildung 1: Der Software Development Life Cycle

## Analyse und Planung

Die erste Phase des SDLC besteht aus der Analyse und Planung des Programms.

In dieser Phase wird ein Projektplan aufgestellt, der festlegt, wie das Programm aussehen soll und welche Schritte durchlaufen werden.

Außerdem wird eine sogenannte Machbarkeitsstudie durchgeführt. Diese beschäftigt sich damit, ob das Projekt aus wirtschaftlicher, betrieblicher und technischer Sicht realisierbar ist. Das Ergebnis dieser Studie enthält verschiedene Softwareentwicklungsmethoden, welche dann nach geringstem Implementierungsrisiko ausgewählt werden können. Zudem wird die ungefähre Dauer der Entwicklung abgeschätzt. (Vgl. [Leau u. a. 2012](#), S.162 – 163)

Weiterhin wird in dieser Phase geplant, welche Funktionen die Software enthalten soll. Auch Projektrisiken, welche die Funktionen mitbringen, sollten identifiziert werden, um diese zu minimieren. (Vgl. [Stoica u. a. 2013](#), S.65 – 66)

Ist dies geschafft, wird zur zweiten Phase übergegangen.

## Definition der erforderlichen Funktionen

Im der zweiten Phase des SCLD wird eine sogenannte „Software Requirement Specification“ (SRS) aufgestellt. In jener werden die genauen Funktionen der Software sowie der Ansatz der Programmierung beschrieben.

Die in der zweiten Phase klar definierten und dokumentierten Funktionen werden in dieser Phase mit dem Kunden rückgesprochen und ggf. überarbeitet. (Vgl. [Stoica u. a. 2013](#), S.65)

Die darauffolgende dritte Phase behandelt die Softwarearchitektur.

## Softwarearchitektur

In der dritten Phase des SDLC wird das Design des Programms ausgearbeitet. Dabei werden Aufbaumöglichkeiten der Produkte ausgearbeitet, welche dann die Konzepte der Architektur des Programms darstellen. Diese Programminfrastruktur wird meist in Diagrammen oder Modellen dargestellt (vgl. [Leau u. a. 2012](#), S.162–163).

Das Entwicklerteam diskutiert dann diese Möglichkeiten; Vorteile und Nachteile der einzelnen Modelle werden schließlich anhand bestimmter Kriterien, wie zum Beispiel Risiko, Dauer oder Produktrobustheit, abgewogen und sich für das überzeugendste Konzept entschieden. (Vgl. [Stoica u. a. 2013](#), S.65)

Dieses Konzept wird dann in der vierten Phase implementiert.

## Produktimplementierung

In der vierten Phase des SDLC startet die Implementierung des Programms.

Es wird auf die, in den Phasen davor gelegten, Grundsteine aufgebaut und somit der Code geschrieben.

Dabei wird sich strikt an die Vorgaben gehalten und die einzelnen Aufgaben werden an verschiedene Teams verteilt, um effiziente Arbeit zu ermöglichen. (Vgl. [Stoica u. a. 2013](#), S.65)

Wenn die Implementierung abgeschlossen ist, wird das Programm in der fünften Phase getestet.

## Testen des Produkts

Beim Testen des Produkts achten die Testenden vor allem auf Softwarefehler, welche so schnell wie möglich von den Entwicklern entfernt werden. Da das Programm meist auch schon von den Entwicklern während der Implementierung getestet wird, kann diese Phase für das Einholen einer zweiten, unvoreingenommenen Meinung verwendet werden. (Vgl. [Stoica u. a. 2013](#), S.66)

Wenn alle Fehler behoben wurden, kann das Projekt in der letzten Phase veröffentlicht werden.

## Veröffentlichung und Überarbeitung

Wenn alle vorherigen Phasen des SDLC abgearbeitet wurden, erfolgt ein Teilrelease des Programms. Dabei wird die Software erst zuerst kleineren Teil des Marktes zur Verfügung gestellt und die Reaktion unter realen Marktbedingungen abgewartet.

Wenn keine Fehler gefunden wurden und das Feedback positiv ist, kann das Programm auf dem gesamten Markt veröffentlicht werden.

Finden sich hingegen Fehler, werden diese behoben und dann der globale Release vorgenommen.

Nach der Veröffentlichung auftretende Fehler werden mit Updates behoben. (Vgl. [Stoica u. a. 2013](#), S.66)

### 1.1.2 Ziele der klassischen Softwareentwicklung

Die klassische Softwareentwicklung gibt einen strikten Ablaufplan (SDLC) vor, welcher „Schritt für Schritt“ abgearbeitet wird.

Dadurch sollen Kommunikationsschwierigkeiten und Missverständnisse bei der Architektur sowie der Implementierung des Programms – vor allem durch große Teams bedingt – vermieden werden.

Außerdem ist es, durch die festgelegten Funktionen am Anfang, leichter die Kosten des Projekts abzuschätzen, einen Zeitplan aufzustellen und die Arbeit effizient zu verteilen (vgl. [Leau u. a. 2012](#), S.163).

Nun wird ein spezielles Modell der klassischen Softwareentwicklung – das Wasserfallmodell – vorgestellt.

### 1.1.3 Das Wasserfallmodell

Das Wasserfallmodell wurde 1970 von Winston Royce eingeführt. Damals wurden Computer fast ausschließlich von ihren Entwicklern benutzt und sie waren generell noch nicht weit verbreitet. (Vgl. [Laplante u. Neill 2004](#), S.10)

Jedoch kommt das Modell auch in der heutigen Zeit – obwohl sich vor allem die Computertechnik extrem weiterentwickelt hat – noch bei großen Projekten, zum Beispiel bei Regierungen oder anderen großen Unternehmen zum Einsatz (vgl. [Alshamrani u. Bahattab 2015](#), S.106).

#### Aufbau des Wasserfallmodells

Das Wasserfallmodell funktioniert nach einem einfachen Prinzip: Ist eine Phase abgeschlossen, wird die nächste begonnen.

Dabei gibt es 5 Phasen, welche in der [Abbildung 2](#) dargestellt sind.

Die Phasen ähneln in Struktur und Merkmalen denen des in [Abbildung 1](#) dargestellten SDLC.

Um ein Programm mit dem Wasserfallmodell entwickeln zu können, müssen die Anforderungen an die Software schon von Anfang an genau definiert sein, da aufgrund des festen Ablaufplans die Anforderungen später nur unter hohen Kosten verändert werden können. (Vgl. [Balaji u. Murugaiyan 2012](#), S.27)

#### Vorteile des Wasserfallmodells

Das Wasserfallmodell ist durch die Gliederung in Phasen, also einzelne Arbeitsschritte mit klaren Anforderungen, welche nacheinander ausgeführt werden, leicht zu verstehen und anzuwenden. Dies ist zum Beispiel von Vorteil, wenn es neues Teammitglied zum Team stößt, da es sich so schnell einarbeiten kann.

Außerdem wird durch das stufenweise Vorgehen eine einfache Koordination gewährleistet. In jeder Phase muss ein bestimmtes, festgelegtes Ziel erreicht werden. Das ist einfach zu verstehen und durchzuführen. (Vgl. [Stoica u. a. 2013](#), S.67)

Jedoch hat das Wasserfallmodell auch Nachteile.

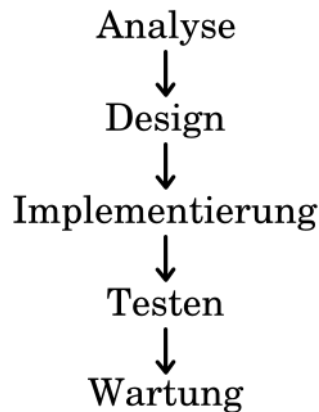


Abbildung 2: Wasserfallmodell

### Nachteile des Wasserfallmodells

Ein Nachteil des Wasserfallmodells ist, dass es keine Flexibilität bei Anforderungsänderungen gibt, denn wenn diese eintreten, muss das Projekt neu geplant werden, wodurch von vorn begonnen werden muss. Zudem gibt es keine Prototypen des Produktes, bis alle Phasen durchlaufen wurden. Dadurch, weiß man bis zuletzt nicht, ob das Produkt den Anforderungen entspricht.

Außerdem ist es schwierig, Probleme, welche beim Testen entstehen, zu beheben, da man dafür in die Systemdesignphase (zweite Phase) zurückkehren muss und somit damit alle nachfolgenden Phasen nochmals durchschreiten muss. (Vgl. [Stoica u. a. 2013](#), S.67)

## 1.2 Agile Softwareentwicklung

Als bekanntester Entwickler der agilen Softwareentwicklung gilt Kent Beck. Er veröffentlichte mit anderen Softwareentwicklern 2001 das „Manifest für Agile Softwareentwicklung“, welches diese Richtung der Softwareentwicklung maßgeblich förderte und zu ihrer Bekanntheit beitrug (vgl. [Beck u. a. 2013](#), S.1).

Die agile Softwareentwicklung ist durch eine hohe Anzahl der Durchläufe eines SDLC gekennzeichnet, wodurch schnell auf Veränderungen, zum Beispiel, in den Anforderungen, reagiert werden kann.

Durch die veränderbaren Anforderungen ist eine gute Kommunikation nötig, wodurch agile Softwareentwicklung meist nur in kleineren Gruppen erfolgen kann.

### 1.2.1 Leitsätze der Agilen Softwareentwicklung

Die Leitsätze des agilen Modells wurden von 11 Softwareentwicklern, unter der Führung von Kent Beck, im „Manifest für Agile Softwareentwicklung“ festgehalten. Es gibt vier wesentliche Leitsätze der agilen Softwareentwicklung, welche sie von anderen Softwareentwicklungsmodellen abgrenzt:

- **„Individuen und Interaktionen** mehr als Prozesse und Werkzeuge“ ([Beck u. a. 2013](#), S.1)  
Damit ist gemeint, dass vor allem die Individuen, also die Programmierer und Kunden, für die Entwicklung eines Programms wichtig sind und diese auch beachtet werden.

- **„Funktionierende Software** mehr als umfassende Dokumentation“ (Beck u. a. 2013, S.1)  
Hiermit ist gemeint, dass die Entwickler dem Kunden mit funktionierender Software mehr überzeugen können, als mit der Dokumentation, da so der Fortschritt „greifbar“ wird.
- **„Zusammenarbeit mit dem Kunden** mehr als umfassende Vertragsverhandlungen“ (Beck u. a. 2013, S.1)  
Dieses Prinzip erweitert die oberen insofern, dass die Zusammenarbeit mit dem Kunden der Software geschätzt werden sollte. Dazu zählt zum Beispiel auch das nachfolgende Prinzip.
- **„Reagieren auf Veränderung** mehr als das Verfolgen eines Plans“ (Beck u. a. 2013, S.1)  
Das Reagieren auf Veränderungen ist eines der Hauptmerkmale der agilen Softwareentwicklung, denn durch das schnelle Einstellen auf Veränderungen kann ein Wettbewerbsvorteil durch schnelle Anpassung geschaffen werden.

Außerdem gibt es verschiedene weitere Programmiertechniken, zum Beispiel das „Pair Programming“, die typisch für die agile Softwareentwicklung sind und die Entwickler damit beim Programmieren unterstützen. Dies soll der folgende Abschnitt erläutern.

## 1.2.2 Pair Programming

Agile Softwareentwicklung ist außerdem durch verschiedene Arbeitstechniken gekennzeichnet, die verstärkt das Gruppengefühl und damit die Motivation steigern sollen. Dazu zählt zum Beispiel das Pair Programming.

Dabei lösen zwei Entwickler eine Aufgabe an einem gemeinsamen Rechner.

Ein Programmierer ist dabei der „Driver“ und der andere „Observer“. Der Driver hat die Kontrolle über das bzw. die Eingabegeräte. Das bedeutet, dass er den Code schreibt oder designt.

Der Observer hingegen „überwacht“ den Driver. Seine Aufgabe sind es:

- auf Fehler aufmerksam zu machen, welche vom Driver begangen werden,
- sich Alternativen auszudenken,
- Ressourcen zu finden, welche zur Entwicklung gebraucht werden oder
- die Auswirkungen der aktuellen Arbeit einzuschätzen und im Gesamtkontext zu beachten. (Vgl. Williams u. a. 2000, S.3 – 4)

Im Entwicklungsprozess werden die Rollen immer in regelmäßigen Zeitabständen getauscht. Pair Programming ist sehr erfolgreich und wird daher oft in der Industrie eingesetzt.

Es verspricht eine höhere Qualität des Produktes mit geringerem Zeitaufwand. Dies wurde auch durch eine Studie der University of Utah belegt. Außerdem stellte sich heraus, dass sich die Programmierer beim Pair Programming sicherer in ihrer Arbeit fühlen und mehr Spaß an der Arbeit haben, wenn sie zu zweit arbeiten anstatt allein.

Die Studie kommt zu dem Ergebnis, dass Software, die mit Pair Programming programmiert wurde, schneller fertig und qualitativ hochwertiger, als die gleiche Software, die allein programmiert wurde, ist. (Vgl. Williams u. a. 2000, S.9)

## 1.2.3 Ziele der Agilen Softwareentwicklung

Aus den bereits aufgezeigten Leitsätzen der agilen Softwareentwicklung, können nun die Ziele und die damit verbunden Vorteile abgeleitet werden.

Schon die Prinzipien, die im „Agilen Manifest“ geschrieben wurden, geben Aufschluss darüber, was die Entwickler, welche dieses Modell der Softwareentwicklung nutzen, erreichen wollen.

Es wird Wert auf „Individuen und Interaktionen, funktionierende Software, Zusammenarbeit mit dem Kunden und das Reagieren auf Veränderung“ (Beck u. a. 2013, S.1) gelegt. Dies wird einerseits durch die Abarbeitung des SDLC bei jeder Funktionsentwicklung, jedoch auch durch neue Arbeitsmethoden, wie zum



Beispiel das Pair Programming, erreicht.

Somit liegt der Fokus der Softwareentwicklung eher auf den Menschen, welche am Prozess der Entwicklung beteiligt sind. Der Kunde mit seinen Wünschen im Vordergrund. Deshalb ist das Modell so ausgelegt, dass die Entwickler schnell auf Kundenwünsche bzw. -anregungen reagieren können.

Außerdem empfinden die Nutzer der agilen Softwareentwicklung die akribische Dokumentation des Schaffensprozesses als Behinderung der Arbeit und bauen lieber auf „funktionierende Software“ (Beck u. a. 2013, S.1).

## 1.2.4 Extreme Programming

Ein Beispiel eines agilen Modells ist Extreme Programming (auch „XP“ genannt). XP wurde von Kent Beck (Vorantreiber des „Agiles Manifests“) entwickelt. Ihm war beim Entwickeln mit dem Wasserfallmodell aufgefallen, dass die Kunden am Anfang des Projektes, die Anforderungen noch nicht genau wussten. Dadurch wurden die Entwickler vor massive Probleme gestellt, weil die Anpassung der Programme schwierig im langen SDLC umzusetzen war. Beck kam auf die Idee, die Laufzeit des SDLC zu verkürzen - also den SDLC auf jede Entwicklung einzeln anzuwenden, anstatt auf das ganze Programm - und statt wenigen bzw. einer Wiederholung mehrere Wiederholungen des SDLC ablaufen zu lassen (siehe [Abbildung 3](#)). (Vgl. Beck 1999, S.70)

### Aufbau von XP

Beim XP soll keine weitreichende Zukunftsplanung betrieben werden, da sich die Anforderungen jederzeit ändern können. Stattdessen soll alles Schritt für Schritt geplant und ausgeführt werden, d. h. erst einen Teil des Projektes abschließen und nachfolgend den nächsten planen und ausarbeiten.

Zudem hat Beck viele Arbeitstechniken benannt, die XP ausmachen.

Beck zählt hierbei zum Beispiel die Methode des „Planning Game“ (Beck 1999, S.71). Diese Praktik beinhaltet, dass der Kunde Umfang und Zeitpunkt, der Lieferung von Prototypen bzw. Ergebnissen, auf der Grundlage der Schätzung der Programmierer bestimmt. Außerdem bedeutet dies, dass die Programmierer nur jene Inhalte implementieren, welche vom Kunde explizit gewünscht werden.

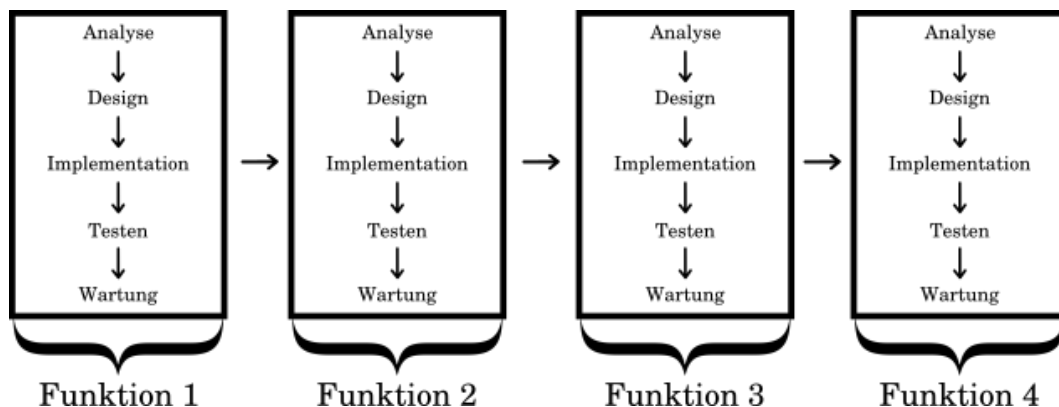


Abbildung 3: XP Ablaufschema

Zu den vielfältigen Praktiken des XP gehört auch „Simple Design“ (Beck 1999, S.71). Dies bedeutet, dass der Code so einfach wie möglich gehalten werden sollte. Das heißt:

- kein doppelter Code,
- so wenige Klassen wie möglich und
- so wenige Methoden wie möglich.

Kent Beck nennt hier den Leitspruch „Say everything once and only once.“ (Beck 1999, S.71). Außerdem zählt Kent Beck zu den Techniken des XP „Tests“ (Beck 1999, S.71). In XP sollen alle Programme ausgiebig getestet werden, um Fehler immer schnell zu finden. Ein weiteres Prinzip ist „Continuous Integration“ (Beck 1999, S.71). Hierbei wird neuer Code schon beim Schreiben in den schon vorhandenen Code eingefügt und auch im gesamten Programm getestet. Wenn dieser Code nicht im gesamten Umfeld funktioniert, wird er verworfen. Zudem wird in XP eine „40-hour week“ (Beck 1999, S.71) empfohlen, weil sich zu viel Arbeit negativ auf die Arbeitsmoral auswirkt. Beck meint dabei, dass niemand eine zweite Überstundenwoche in Folge leisten sollte. Dadurch soll die Arbeitsmoral gesteigert werden. Die Entwicklungsarbeit findet in einem „Open workspace“ (Beck 1999, S.71) statt. Das Team arbeitet in einem großen Raum mit kleinen Abgrenzungen an den Seiten mit einzelnen Programmierern. Zusammenfassend lässt sich sagen, dass XP viele Merkmale des Agilen Manifests, wie zum Beispiel die 40-Stunden-Woche (als Individuumsbezug), das Programmieren der Programmteile einzeln (um schneller auf Veränderungen reagieren zu können), aufgreift. (Vgl. Jeffries u. a. 2000, S.8 – 11) Somit ergeben sich Vorteile und Nachteile von XP.

### **Vorteile von XP**

Der größte Vorteil des XP ist, dass schnell auf Änderungen in den Anforderungen reagiert werden kann und dadurch möglicherweise hohe Kosten verhindert werden. Außerdem treten durch die klare Kommunikation zwischen Entwickler und Kunde weniger Missverständnisse über die Software auf. (Vgl. Balaji u. Murugaiyan 2012, S. 29)

Ein weiterer Vorteil ist die humane Seite von XP. Es gibt in diesem Modell viel Bezug zum Individuum. Jeder Entwickler wird einzeln betrachtet und wertgeschätzt, wodurch die Arbeitsmoral gestärkt wird.

### **Nachteile von XP**

Ein Nachteil von XP wird durch die benötigte hohe Kommunikation zwischen den Entwicklern hervorgerufen. Dadurch fällt es, vor allem bei großen Teams, schwer, „alles im Blick“ zu behalten. Zudem ist XP einsteigerunfreundlich, weil nur die erfahrenen Programmierer die Entscheidungen über die Vorgehensweise treffen. (Vgl. Balaji u. Murugaiyan 2012, S. 29)

## **1.3 Vergleich klassischer und agiler Softwareentwicklung**

Nun wurde sowohl die klassische, als auch die agile Softwareentwicklung, genau untersucht. Im Folgenden werden die beiden Modelle verglichen und geschildert, für welche Projekte und Gegebenheiten sich die jeweiligen Softwareentwicklungsmodelle eignen.

Zuerst wird die Kommunikation in den einzelnen Modellen analysiert.

Die klassische Softwareentwicklung sieht eine formale Sprache zwischen den Projektbeteiligten vor, sodass eine Kommunikation auf formaler Ebene entsteht.

Beim agilen Modell hingegen, empfiehlt sich eine ungezwungene und offene Kommunikation, um die Zwischenmenschlichkeit zu steigern und ein besseres Arbeitsverhältnis zu erzeugen.

Die Zielgruppe der agilen Softwareentwicklung sind Programmierer, welche schon viele Erfahrungen, auch mit agiler Softwareentwicklung, gesammelt haben. Diese können das Projekt gut leiten und alles „im Auge“ behalten, um auch, bei kurzfristigen Änderungen der Anforderungen, flexibel sein zu können.

Die klassische Softwareentwicklung ist hingegen auch für Anfänger geeignet, da alle Abläufe nach dem SDLC klar geregelt sind, sodass jeder sie einfach verstehen und ausführen kann.

Wie gerade angedeutet, gibt es in der klassischen Softwareentwicklung nach dem SDLC eine klar festgelegte Hierarchie, das bedeutet, dass die Rollen klar verteilt sind und der Vorgesetzte das „Sagen“ hat. Zudem ist das Verhältnis zum Kunden bei der klassischen Softwareentwicklung sehr eingeschränkt. Der Kunde benötigt klare Vorstellungen von dem Produkt, was er bekommen möchte. Er sollte also schon im Voraus sehr genau wissen, was das Produkt für Anforderungen erfüllen soll. Auf Veränderungen im SDLC kann nicht einfach und schnell eingegangen werden, weil die Phasen nacheinander ablaufen und mit wechselnden Anforderungen neu begonnen werden müssen.

Hingegen ist die Beziehung zum Kunden bei der agilen Softwareentwicklung flexibel. Da das Produkt immer „Schritt für Schritt“ weiterentwickelt wird, können veränderte Anforderungen schnell eingebaut werden. Das bedeutet, dass der Kunde seine Anforderungen ändern und das Entwicklerteam agil darauf reagieren kann.

Das agile Modell beruht auf gleichgestellter Zusammenarbeit der Projektbeteiligten. Es soll gewährleistet sein, dass alle Teammitglieder auf der gleichen Ebene stehen und gleich wichtig für das Gelingen des Projektes sind.

Es wurden die zwischenmenschlichen Beziehungen in den jeweiligen Modellen betrachtet, jetzt folgen die informatischen Unterschiede der Modelle.

Durch die bereits erwähnte Agilität in der agilen Softwareentwicklung halten sich auch die Kosten für einen „Neustart“ gering, da die einzelnen Programmteile immer noch verwendet werden können und nicht erneut „von Null“ begonnen werden muss.

Dagegen geht dies bei der statischen Softwareentwicklung nicht, da der SDLC neugestartet werden und alle Phasen erneut durchlaufen werden müssen. Dadurch steigen auch die Kosten stark an.

Aus den nun verglichenen Attributen der verschiedenen Softwareentwicklungsmodelle geht hervor, dass sich die agile und statische Softwareentwicklung maßgeblich unterscheiden, da die Zielsetzung der Modelle für unterschiedliche Zwecke gedacht ist.

In der statischen Softwareentwicklung hat die Stabilität des Entwicklungsprozesses und des Produktes die höchste Priorität, dafür wird aber die Flexibilität vernachlässigt. Daraus ergibt sich, dass dieses Modell gut für große Projekte, von großen Firmen mit vielen Mitarbeitern geeignet ist, da es wichtig ist, eine geordnete Struktur bei vielen Aufgaben und Mitarbeitern zu besitzen.

Bei der agilen Softwareentwicklung hat die Agilität bzw. Flexibilität den größten Stellenwert, dafür ist die Entwicklung nicht so stabil wie beim klassischen Ansatz, da mit vielen Teammitgliedern die Koordination schwierig ausfallen kann. Daraus lässt sich ableiten, dass ein agiler Ansatz eher für kleine Projekte mit einem kleinen Team geeignet ist, da so Agilität, als auch Stabilität, hergestellt werden kann. Dabei steht der Kunde im Vordergrund und es kann von den Entwicklern aktiv auch auf größere Kundenwünsche eingegangen werden.

Jedoch müssen auch die Umstände der Entstehungszeit der Modelle beachtet werden.

Während die klassische Softwareentwicklung, wie der Name verrät, schon seit circa 50 Jahren existiert und angewendet wird, ist die agile Softwareentwicklung mit circa 20 Jahren relativ jung.

Die klassische Softwareentwicklung schuf damals die Ursprünge. Die Computer waren teils noch riesige Maschinen und für einen Großteil noch nicht zugänglich. Die Programme, die programmiert wurden, hatten von Anfang an feste Anforderungen, welche sich nicht bzw. nur geringfügig veränderten.

Heutzutage ist das allerdings gar nicht mehr denkbar. Viele Dinge, vor allem in der digitalen Welt, können sich rasant ändern, wodurch Änderungen der Anforderungen, zum Beispiel durch Updates von relevanten Programmen, kaum noch auszuschließen sind.

Daher lässt sich sagen, dass heute oftmals Mischformen aus beiden Modellen angewendet werden.

Zum Beispiel gibt es zwar mehrere Iterationen des SDLC, aber nicht alle Leitsätze der agilen Softwareentwicklung werden beachtet.

## 2 Dokumentation

In diesem Kapitel soll ein Einblick in den Schaffensprozess gegeben werden. Dabei wird sowohl die Idee des Programms erklärt als auch die konkrete Ausführung.

### 2.1 Idee

Damit die Idee verstanden werden kann, müssen erst einmal die Grundlagen für das Programm erklärt werden. Dazu zählt die Programmiersprache, in welcher das Programm geschrieben ist, Python, als auch die Software, auf die das Programm aufbaut,  $\text{\LaTeX}$ .

Nach der Erklärung dieser Grundlagen, wird das Problem, welches das Programm lösen soll, dargelegt und anschließend das Konzept des Programms in den einzelnen Schritten erläutert.

#### 2.1.1 Python

Für das Programm wird die Programmiersprache Python benutzt.

Python wurde in den frühen 90er Jahren vom niederländischen Professor Guido von Rossum am Stichting Mathematisch Centrum in den Niederlanden entwickelt.

Python ist ein OpenSource-Programm, das bedeutet, dass jeder den Quellcode einsehen und ohne Einschränkungen benutzen und verändern kann.<sup>1</sup>

Aus folgenden Gründen wird Python benutzt:

1. Da Python schon seit mehreren Jahren im Unterricht benutzt wird, sind viele Befehle und die Syntax bekannt.
2. Python zeichnet sich außerdem durch seine einfach verständliche Sprache aus. Die Syntax ist leicht verständlich. Außerdem ist es einfach, Module hinzuzufügen, welche zahlreiche weitere Funktionen importieren, welche man nutzen kann, um die Arbeit erleichtern.
3. Zudem ist der Code sehr übersichtlich.
4. Außerdem ist die Sprache plattformunabhängig und für alle gängigen Betriebssysteme (MacOS, Windows und Linux) verwendbar.

#### 2.1.2 $\text{\LaTeX}$

$\text{\LaTeX}$  ist ein Programm zum Erstellen von Dokumenten. Es basiert auf dem von Donald E. Knuth entwickelten Textsatzsystem  $\text{\TeX}$  und vereinfacht dieses durch Makros.

Das Programm wurde Anfang der 1980er Jahre von Leslie Lamport entwickelt.

Da er ein Buch mithilfe von  $\text{\TeX}$  schreiben wollte und ihm die Befehle zu kompliziert waren, entwickelte er vereinfachte Befehle (Makros) und fügte diese in seinem Programm  $\text{\LaTeX}$  zusammen.

Er schrieb für  $\text{\LaTeX}$  auch ein Benutzerhandbuch, von dem er dachte, dass es sich nicht verkauft. Die Realität zeigte aber, dass das Programm viele Vorteile hatte und seinem Zweck - dem Schreiben wissenschaftlicher

---

<sup>1</sup>aus <https://docs.python.org/3/license.html>

Texte, vor allem für Wissenschaftler und Ingenieure - erfüllte. Auch deshalb verkaufte sich das Handbuch sehr gut. (Vgl. [Lamport 2020](#), S.48-49)

L<sup>A</sup>T<sub>E</sub>X hat einige Merkmale, die es besonders machen.

Es ist kein klassisches „Schreibprogramm“, da es nicht nach dem „What-you-see-is-what-you-get-Prinzip“ funktioniert. Das bedeutet, dass das, was geschrieben wird, nicht automatisch das ist, was am Ende im Dokument steht. Es lassen sich viele Formatierungen mit Befehlen ausführen, zum Beispiel wird mit dem Befehl `\LaTeX{}` das L<sup>A</sup>T<sub>E</sub>X-Logo dargestellt.

Dies bringt dem Programm einige Vorteile. Durch diese Befehle, lassen sich besonders komplexe mathematische Formeln darstellen. Außerdem ermöglichen die Befehle, dass eine einheitliche Formatierung eingehalten werden kann, ohne dass der Verfasser selbst formatieren muss. Es stellt also eine enorme Arbeitserleichterung dar, vor allem bei wissenschaftlichen Arbeiten mit hohen formalen Ansprüchen.

Außerdem erleichtert L<sup>A</sup>T<sub>E</sub>X auch die Quellenangaben und das Erstellen eines Quellenverzeichnis. Dafür wird das dazu entwickelte Programm „BibT<sub>E</sub>X“ verwendet.

### 2.1.3 Quellen in L<sup>A</sup>T<sub>E</sub>X

Um Quellen in L<sup>A</sup>T<sub>E</sub>X anzugeben, wird eine Bibliotheksdatei des Typs „.bib“ benötigt.

Diese Datei ist die Datenbank der Quellen des zugehörigen Dokuments. In einer bestimmten Syntax werden die Informationen über das Buch abgespeichert (siehe [Abbildung 4](#)).

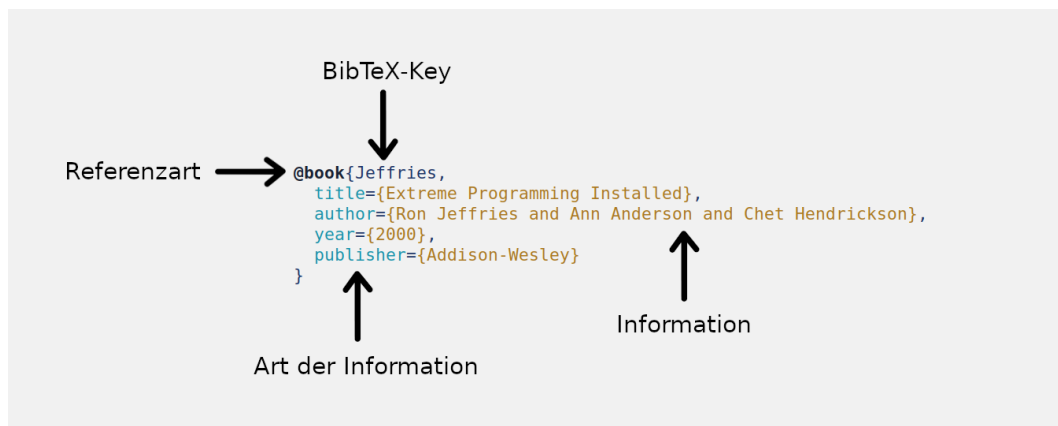


Abbildung 4: Aufbau eines Eintrags

In der [Abbildung 4](#) lässt sich unter anderem die Referenzart feststellen. Diese bestimmt, wie der Eintrag im Literaturverzeichnis formatiert ist. Es lässt sich zwischen verschiedenen Referenzarten, wie zum Beispiel Buch, Artikel oder auch Abschlussarbeit unterscheiden.

Außerdem ist für jeden Eintrag der BibT<sub>E</sub>X-Key essenziell. Dieser wird dazu verwendet, um dann im Text auf den Eintrag zu verweisen, den Eintrag eindeutig zu identifizieren und Verwechslungen zu vermeiden.

Schließlich ist die Information in der Datei eingebettet.

Die Syntax für die Einträge ist in [Abbildung 4](#) zu erkennen. Zuerst ist die Art der Information (zum Beispiel Titel, Autor oder Erscheinungsjahr) gespeichert. Danach wird dieser die Information, durch ein =, zugewiesen.

Wenn dieser Eintrag fertig ist, kann er sofort in der Arbeit verwendet werden. Dafür müssen allerdings bestimmte Voraussetzungen im Dokument geschaffen werden.

Bevor das Dokument mit dem Befehl `\begin{document}` beginnt, wird der Stil des Quellenverzeichnisses mit dem Befehl `\bibliographystyle{•}` festgelegt. Je nachdem, welcher Stil festgelegt wird, wird das Quellenverzeichnis und die -verweise anders gestaltet. Somit existiert eine große Vielfalt an Formatierungen und es ist außerdem möglich eigene zu erstellen.

Da nun die Grundlagen erklärt wurden, folgt nun die Erläuterung des Problems, welches das Programm lösen soll.

#### 2.1.4 Problem

Ein Problem, welches sich aus den vielen Möglichkeiten von  $\text{\LaTeX}$  ergibt, tritt bei der Angabe der Quellen auf. Es ist sehr aufwendig, die Daten des verwendeten Werks manuell in die BibTeX-Datei einzutragen. Dazu benötigt man viele Angaben und muss viel Code in die Bib-Datei schreiben. Beispielsweise müsste man bei jedem Buch die ISBN einzeln abtippen. Das ist eine aufwendige und fehleranfällige Arbeit.

Um dieses Problem zu beheben, wird ein Programm entwickelt, in dem die ISBN des Buches eingescannt wird, die BibTeX-Datei und der BibTeX-Key ausgewählt werden und die wichtigsten Daten des Werks danach automatisch der BibTeX-Datei hinzugefügt werden.

#### 2.1.5 Konzept des Programms

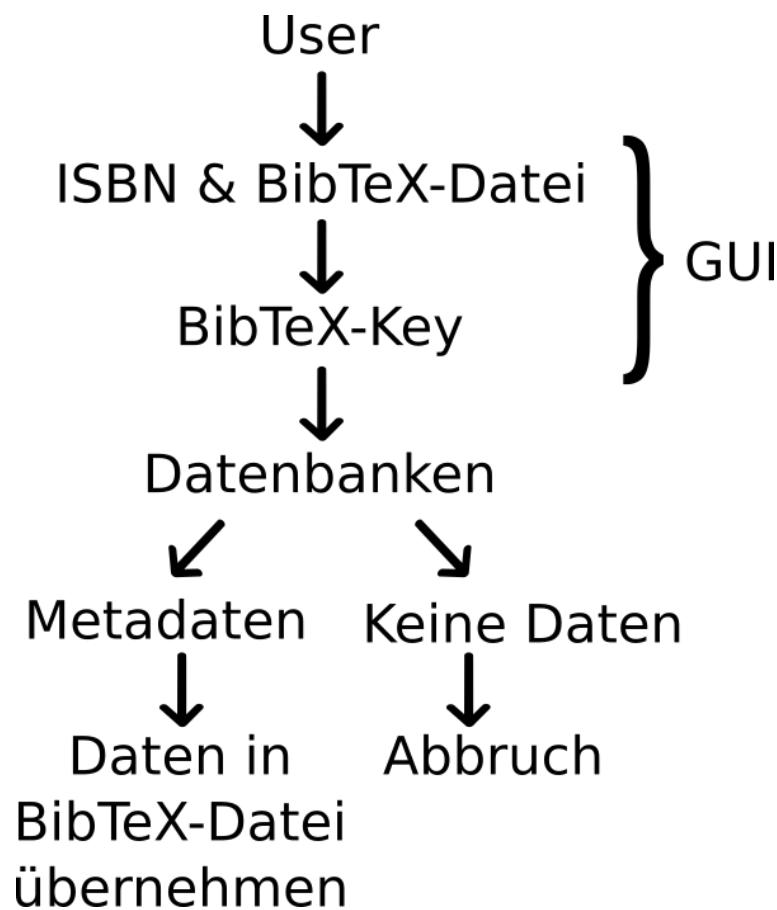


Abbildung 5: Ablaufplan des Programms

Abbildung 5 stellt den Ablaufplan des Programms dar.

Ganz am Anfang steht, wie immer bei einem Programm, der User, welcher das Programm verwenden will. Dafür öffnet er das Programm.

Dann kommt ihm das Graphical User Interface (GUI) entgegen. Dies ist der Teil des Programms, in welchem der User mit dem Programm direkt interagiert. Hier erfolgt die Dateneingabe.

Zum einen gibt der User die ISBN des Buches ein, welches er in seiner Arbeit zitieren will. Außerdem wird er aufgefordert, die BibTeX-Datei auszuwählen, in welcher der neue Eintrag gespeichert werden soll. Nach Angabe aller benötigten Daten gibt der Benutzer als letzte Interaktion mit dem GUI den BibTeX-Key ein.

Anschließend sucht das Programm mit der ISBN in verschiedenen Datenbanken nach Metadaten über das Buch mit der angegebenen ISBN.

Wenn keine Daten vorhanden sind, dann wird folglich nichts zurückgegeben und es kommt zum Abbruch des Programms, da kein Eintrag erzeugt werden kann.

Wenn Daten zum Buch vorhanden sind, werden diese an das Programm zurückgegeben und diese Daten in der BibTeX-Datei gespeichert, die der Benutzer im GUI angegeben hat.

Damit ist dann das Ziel des Programmes erfüllt und der Vorgang abgeschlossen.

## 2.2 Umsetzung

Da das Programm auf einem Computer mit dem Betriebssystem „Xubuntu“ entwickelt wurde, welches auf Linux basiert, ist es möglicherweise nur auf Linux-Betriebssystemen voll funktionsfähig.

### 2.2.1 Benutztes Modell der Softwareentwicklung

Für die Entwicklung des Programms wurde eine Mischform beider Modelle gewählt. Jede Phase der Entwicklung baute auf dem SDLC auf, jedoch natürlich in einer abgewandelten Form ohne Veröffentlichung.

Gut zu erkennen ist dies am Beispiel des Graphical User Interface (siehe [Unterabschnitt 2.2.2](#)).

Zuerst wurde das GUI mit dem Programm „Pencil“ modelliert, welches der Planung, der Definition der erforderlichen Funktionen und der Architektur des Programms dient.

Danach folgt die Implementierung mithilfe des Moduls „AppJar“ mit der Programmiersprache Python.

Darauf folgt die Phase des Testens, welche sich durch Python und die Entwicklungsumgebung als einfacher, als in anderen Sprachen zeigt, da, wie in [Abbildung 6](#) zu erkennen, der Fehler und die -art einfach angegeben wird.

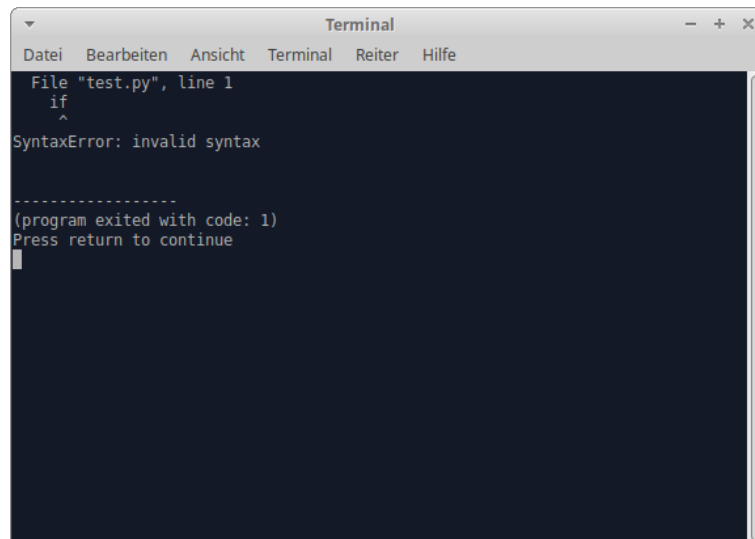


Abbildung 6: Fehlermeldung

Im Beispiel handelt es sich um einen **SyntaxError**, also einen Fehler der Syntax.

Es wird auch angezeigt, dass sich dieser in Zeile 1 befindet. Hinter einer `if`-Anweisung wird als Syntax ein Doppelpunkt benötigt.

## 2.2.2 Graphical User Interface (GUI)

Das GUI ist einer der wichtigsten Teile eines Programms. Es ist der Programmteil, mit welchem der Nutzer aktiv interagieren kann. Das GUI legt außerdem die Architektur der Anwendung fest. Damit dabei keine Fehler auftreten, wird das GUI in einem Programm, hier Pencil, modelliert.

### Entwurf des GUI in Pencil

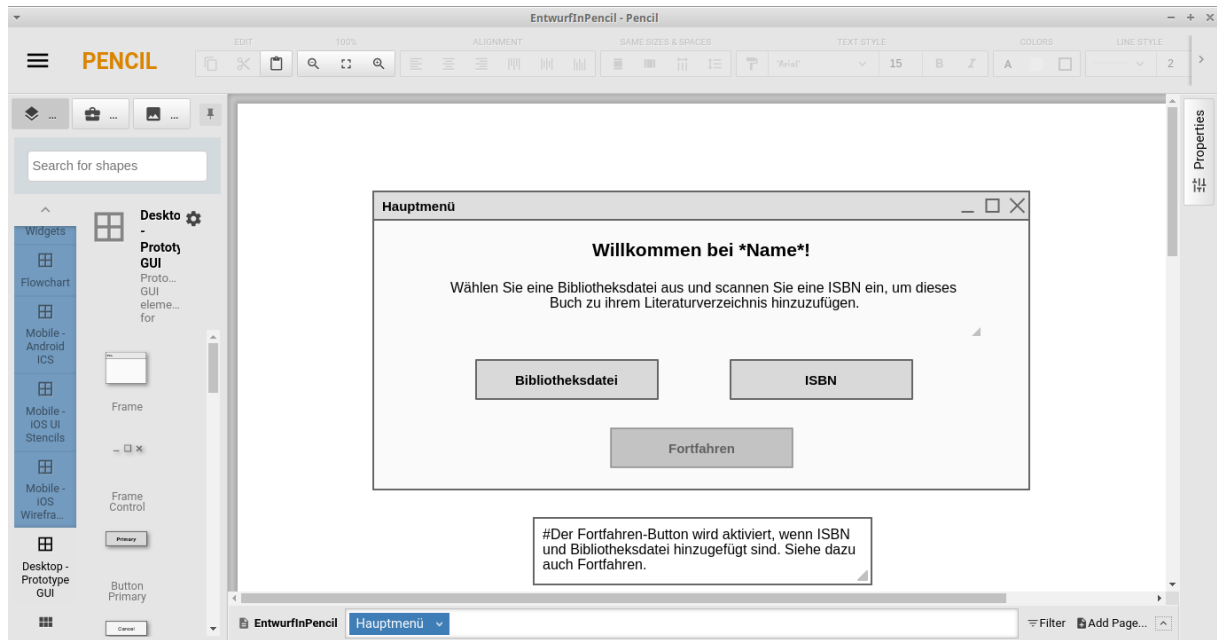


Abbildung 7: Entwurf des Hauptmenüs des GUI in Pencil

Wie in [Abbildung 7](#) erkenntlich ist, ist Pencil ein Programm zum Entwerfen der Graphical User Interfaces. Dafür lassen sich auf der linken Seite die einzelnen Elemente eines GUI auswählen. Dabei besteht die Auswahl zwischen unterschiedlichen Designs, wie zum Beispiel Android-, iOS- oder Desktop-Designelemente. Wenn die Kategorie ausgewählt wurde, lassen sich die einzelnen Elemente einfach per Drag-and-Drop auf die Fläche ziehen. So ein Element ist zum Beispiel der Rahmen des Fensters oder die Buttons (wie in [Abbildung 7](#)). Die einzelnen Elemente lassen sich editieren, zum Beispiel kann dem Fenster einen Titel, hier Hauptmenü, gegeben werden.

Außerdem können Anmerkungen hinzugefügt werden. Diese erscheinen dann wie in [Abbildung 7](#) als grau umrandete Box. Dort werden Bemerkungen hineingeschrieben, die so nicht aus dem Entwurf abzulesen sind. Im Beispiel wird gesagt, wann der Fortfahren-Button aktiviert wird, da dies nicht direkt aus der Abbildung hervorgeht.

Ein weiteres Feature des Programms ist das Verlinken von Unterfenstern. Dies ist in [Abbildung 8](#) dargestellt. Da die Buttons nicht wirklich funktionsfähig sind, werden die einzelnen Unterfenster im Menü als sogenannte „Child Pages“ erstellt, um die einzelnen Verknüpfungen zu sehen und einen schnellen Wechsel zwischen den Fenstern gewährleisten zu können.

Nach dem Entwerfen des GUI zeigt der nächste Arbeitsschritt dessen Implementierung.



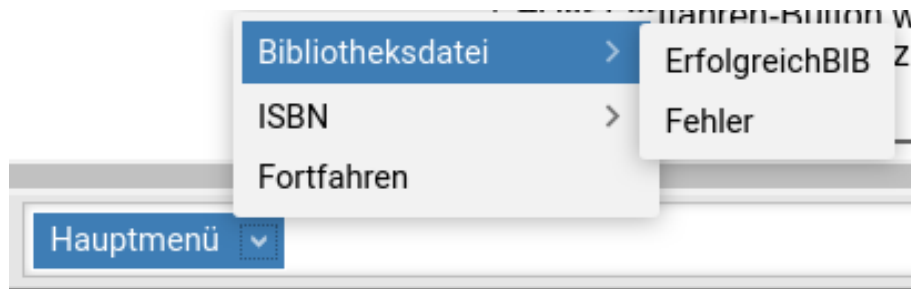


Abbildung 8: Verlinken von Unterfenstern in Pencil

## Programmieren des GUI

Für die Programmierung des GUI wird das Python-Modul „AppJar“<sup>2</sup> benutzt.

„AppJar“ ist ein, von Richard Jarvis entwickeltes, Modul, welches es erlaubt, einfach GUIs mit Python zu erstellen.

Im Folgenden werden die verschiedenen Bestandteile des GUI erklärt und auch am Code des Programms erläutert.

Im Folgenden wird der Aufbau der Menüs erklärt.

Es gibt zuerst das Hauptmenü, welches der User immer als Erstes sieht.

### Python-Code 1: Hauptmenü

```

1 #Hauptmenü
2 bell=gui("Hauptmenu", "800X200")
3 bell.setResizable("False")
4 bell.setStretch("Coulumn")
5 bell.setSticky("ew")
6 bell.setFont(size=12, family="Sans_Serif")
7 bell.addLabel("titel", "Willkommen bei ISBN2BibTeX!")
8 bell.getLabelWidget("titel").config(font=("Sans_Serif", "20", "bold"))
9 bell.setStretch("None")
10 bell.addLabel("untertitel1", "Wählen Sie eine Bibliotheksdatei aus und")
11 bell.addLabel("untertitel2", "scannen Sie eine ISBN ein, um dieses Buch zu ihrem Literaturverzeichnis hinzuzufügen.")
12 bell.setStretch("both")
13 bell.addButtons(["Bibliotheksdatei", "ISBN", "Einstellungen", "Abbrechen"], pressHauptmenu)
14 bell.addButton("Fortfahren", pressHauptmenu)
15 bell.disableButton("Fortfahren")

```

Wie in Zeile 2 vom [Python-Code 1](#) zu erkennen ist, wird dort das GUI festgelegt. Mit dem Befehl `gui("Hauptmenu", "800X200")` wird erst der Name des Hauptfensters (Hauptmenu) und die Größe des Fensters, in diesem Fall mit 800 mal 200 Pixeln, festgelegt.

Danach folgt die Konfiguration des Hauptmenüs, welche für alle anderen Fenster auch übernommen wird, zumindest solange bei diesen nichts geändert wird. In Zeile 6 des [Python-Code 1](#) wird zum Beispiel die Schriftgröße und Schriftart festgelegt.

Mit dem Befehl `addLabel()`, wie beispielsweise in Zeile 7, wird ein sogenanntes Label hinzugefügt. Dies ist eine Zeile von Schrift. In Zeile 8 wird das Label aus Zeile 7 bearbeitet und in die Schriftgröße 20 versetzt. Zudem erscheint die Schrift fett ("bold").

Mit dem Befehl `addButtons(["Bibliotheksdatei", "ISBN", "Einstellungen", "Abbrechen"], pressHauptmenu)` werden die Buttons des Menüs hinzugefügt. In den eckigen Klammern steht dabei, wie die Buttons heißen. `pressHauptmenu` ist der Befehl, welcher ausgeführt wird, wenn ein Button geklickt wird.

<sup>2</sup>siehe <http://appjar.info/>

Der Code des Befehls `pressHauptmenu` ist in [Python-Code 2](#) zu sehen.

#### Python-Code 2: Buttonanweisungen des Hauptmenüs

```
1 #Hauptmenübutton
2 def pressHauptmenu(btn):
3     if btn=="Abbrechen":
4         bell.stop()
5     if btn=="Bibliotheksdatei":
6         bell.hide("Hauptmenu")
7         bell.showSubWindow("Bib")
8     if btn=="ISBN":
9         #schauen, ob Scanner verbunden ist
10        if scanner=="---":
11            bell.hide("Hauptmenu")
12            bell.showSubWindow("Isbn")
13        else:
14            bell.hide("Hauptmenu")
15            bell.showSubWindow("IsbnScan")
16    if btn=="Fortfahren":
17        if bell.yesNoBox("Fortfahren", "Wollen_Sie_diese_Daten_übermitteln?",
18            parent=None)==True:
19            l.append(bell.stringBox("Bibtexkey_eingeben", "Bitte_geben_Sie_den
20                Bibtexkey_ein._Dieser_wird_in_LaTeX_verwendet_um_das_Werk_zu_
21                zitieren."))
22            bell.infoBox("Erfolgreich", "Sie_haben_Ihre_Daten_erfolgreich_ü
23                bermittelt._Ihr_Buch_wurde_im_Literaturverzeichnis_hinzugefügt",
24                parent=None)
25            bell.stop()
26    if btn=="Einstellungen":
27        bell.hide("Hauptmenu")
28        bell.showSubWindow("Einstellungen")
```

In [Python-Code 2](#) ist zu erkennen, dass `pressHauptmenu(btn)` als Funktion mit dem Befehl `def` definiert wird. `btn` steht hierbei als Variable, für den Button, der gedrückt wird.

Die `if`-Anweisungen in den Zeilen 4, 6, 9, 17 und 22 geben an, was passiert, wenn der jeweilige Button gedrückt wird.

Zum Beispiel wird der Button "Bibliotheksdatei" geklickt. Dann ist die `if`-Anweisung aus Zeile 6 erfüllt. Damit wird zuerst die Anweisung in Zeile 7 ausgeführt und das "Hauptmenu"-Fenster mit dem Befehl `hide()` versteckt.

Danach wird die Anweisung in Zeile 8 ausgeführt und mit dem Befehl `showSubWindow("Bib")` wird das Unterfenster "Bib" geöffnet. Unterfenster (englisch „subwindow“) sind in „AppJar“ alle Fenster, welche nicht das Hauptfenster (hier „Hauptmenü“) sind.

#### Python-Code 3: Code der Bibliotheksdateiauswahl

```
1 #Bibliotheksdatei
2 bell.startSubWindow("Bib", "Bibliotheksdatei", transient=True, modal=True)
3 bell.setStretch("None")
4 bell.addLabel("l1Bib", "Bibliotheksverzeichnis_auswählen")
5 bell.getLabelWidget("l1Bib").config(font=("Sans_Serif", "14", "bold"))
6 bell.addLabel("l2Bib", "Bitte_wählen_sie_ein")
7 bell.addLabel("l3Bib", "Literaturverzeichnis_des_Typs_bib_aus")
8 bell.setStretch("both")
9 bell.addButtons(["Datei_hinzufügen", "Zurück"], pressBib)
10 bell.setSize(400, 150)
11 bell.stopSubWindow()
```

Der [Python-Code 3](#) stellt den Code eines solchen Unterfensters dar. Wie sich erkennen lässt, unterscheidet sich der Aufbau des Codes eines Unterfensters nicht wesentlich von dem des Hauptfensters ([Python-Code 1](#)). Ein Unterschied ist, dass das Unterfenster mit dem Befehl `startSubWindow` in Zeile 2 gestartet wird und in Zeile 11 mit dem Befehl `stopSubWindow` gestoppt wird.

Allerdings ist es auch bei einem Subwindow so, dass zu diesem mit dem Befehl `addLabel()` ein Text hinzugefügt und dieser auch bearbeitet werden kann. Zudem besteht die Möglichkeit Buttons mit dem Befehl `addButtons` hinzuzufügen.

So wird das Programm nach dem in [Abschnitt 2.2.2](#) vorgestellten Konzept programmiert. Hierbei gibt es noch einige Besonderheiten, welche im Folgenden näher erläutert werden.

#### Python-Code 4: Boxen

```
1  if bell.yesNoBox("Fortfahren", "Wollen_Sie_diese_Daten_übermitteln?",  
2      parent=None)==True:  
3      l.append(bell.stringBox("Bibtexkey_eingeben", "Bitte_geben_Sie_den  
    _Bibtexkey_ein._Dieser_wird_in_LaTeX_verwendet_um_das_Werk_zu_  
    ziteren."))  
    bell.infoBox("Erfolgreich", "Sie_haben_Ihre_Daten_erfolgreich_ü  
    bermittelt._Ihr_Buch_wurde_im_Literaturverzeichnis_hinzugefügt",  
    parent=None)
```

Im [Python-Code 4](#) ist der Code für sogenannte Boxen oder auch Pop-Ups zu erkennen.

Es gibt, wie in [Python-Code 4](#) zu erkennen ist, verschiedene Arten solcher Boxen.

In der `yesNoBox` aus Zeile 1 kann man, wie der Name sagt, „Ja“ oder „Nein“ auswählen. Je nachdem wird für „Ja“ `True` und für „Nein“ `False`, also ein Wert des Datentyps `bool` zurückgegeben. Im Programm wird diese Box benutzt, um sicherzugehen, dass der User fortfahren möchte.

In Zeile 2 wird eine `stringBox` erzeugt. Diese hat ein Eingabefeld mit Text, welcher den Datentyp `string` hat. Es wird der eingegebene Text ausgegeben. Hier wird diese Box dazu genutzt, den BibTeX-Key einzugeben. Eine weitere Box ist die `infoBox`, welche eine Information anzeigt und einen „OK“-Button enthält. Durch Drücken dieses Buttons, wird die nächste Anweisung ausgeführt. Hier wird diese `infoBox` benutzt, um dem User mitzuteilen, dass die Daten erfolgreich übermittelt wurden.

Zudem gibt es noch weitere Boxen, zum Beispiel die `errorBox`, welche einen Fehler anzeigt oder die `openBox`, mit welcher sich Dateien auswählen lassen.

#### Python-Code 5: Links

```
1  bell.addLink("Was_macht_Isbn2BibTeX?", HilfeLink)  
2  bell.addWebLink("Was_ist_BibTeX?", "https://de.wikipedia.org/wiki/BibTeX")  
3  bell.addWebLink("Was_ist_eine_ISBN?", "https://de.wikipedia.org/wiki/  
    Internationale_Standardbuchnummer")
```

In [Python-Code 5](#) lassen sich verschiedene Links erkennen. Mit dem Befehl `addLink` wird ein Link auf dem ersten eingegebenen Argument erzeugt. Wenn der Link angeklickt wird, wird die Funktion, in diesem Fall `HilfeLink`, ausgeführt. Hier wird diese Funktion genutzt, um zu erklären, was die Aufgabe des Programms ist.

Zudem gibt es mit der Funktion `addWebLink` in Zeile 2 und 3 die Möglichkeit, auf eine Website zu verweisen. Diese wird dann geöffnet, wenn der Text, welcher den Link enthält (erstes Argument der Funktion), angeklickt wird. In diesem Fall wird auf Websites verwiesen, welche erklären, was BibTeX (Zeile 2) und eine ISBN (Zeile 3) ist.

Ein weiterer Teil des Programmes ist die Integration des Scanners, welche ebenfalls in der GUI stattfindet.

### 2.2.3 Scannereinbindung

Die Scannereinbindung ist ein weiteres Feature des Programms, welche die Benutzung erleichtern soll.

Bei der Einbindung des Scanners sind die Möglichkeiten des Entwickelns allerdings an die Umgebung eines bestimmten Computers gebunden, da die Mittel zum Testen mit einem anderen Scanner nicht verfügbar waren.

Die Scannereinbindung erfolgt mit dem Modul `PySerial`.

### Python-Code 6: Erkennen der angeschlossenen Scanner

```

1 #Scannerliste
2 try:
3     scannerliste=([comport.device for comport in serial.tools.list_ports.comports()])
4 except:
5     scannerliste=[]

```

Dazu wird zuerst versucht die Scannerliste anzulegen. Dies geschieht in Zeile 3. Mit dem Befehl werden alle Ports des Computers durchsucht und der Liste hinzugefügt. Dabei können allerdings Fehler auftreten. Wenn ein Fehler auftritt, tritt die `except`-Anweisung in Kraft, wodurch die Scannerliste leer gelassen wird, wodurch die weitere Funktionstüchtigkeit des Programms gewährleistet werden kann.

Um den Scanner benutzen zu können, muss dieser im Menü ausgewählt werden.

### Python-Code 7: Scannermenü

```

1 #Scanner
2 bell.startSubWindow("Scanner", "Scanner_□auswählen", modal=True)
3 bell.setStretch("column")
4 bell.setSize(600, 250)
5 bell.addLabel("l1Scanner", "Scanner_□auswählen.")
6 bell.addListBox("Scanner", ["---"])
7 scannercounter=0
8 if scannerliste != []:
9     for device in scannerliste:
10         try:
11             scannerOut=""
12             scannerOutListe=([comport.device for comport in serial.tools.
13                             list_ports.comports()])
14             scannerOutListe.append([comport.product for comport in serial.
15                                 tools.list_ports.comports()])
16             scannerOutListe.append([comport.manufacturer for comport in serial.
17                                 tools.list_ports.comports()])
18             scannerOut+=scannerOutListe[scannercounter]
19             scannerOut+=":□" + scannerOutListe[-2][scannercounter] + "□(" +
20                 scannerOutListe[-1][scannercounter] + ")"
21             bell.addListItem("Scanner", scannerOut)
22             scannercounter+=1
23         except:
24             None
25 bell.setListBoxWidth("Scanner", 600)
26 bell.addNamedButton("OK", "OKScanner", pressScanner)
27 bell.stopSubWindow()

```

Dafür gibt es ein Fenster in den Einstellungen, welches im [Python-Code 7](#) und [Abbildung 9](#) dargestellt ist. In Zeile 2 des [Python-Code 7](#) startet das Unterfenster. In den Zeilen 3 bis 5 erfolgt die Einstellung und Anpassung des Fensters. In der sechsten Zeile wird eine `ListBox` hinzugefügt. Eine `ListBox` ist eine Box, in welcher die Einträge untereinander aufgelistet werden und man einen Eintrag auswählen kann (siehe [Abbildung 9](#)). In Zeile 8 bis Zeile 20 werden der `ListBox` weitere Einträge hinzugefügt. Dafür wird mit der `try`-Anweisung gearbeitet, um Fehlermeldungen zu vermeiden. In den Zeilen 11 bis 16 wird der Eintrag mithilfe des `PySerial`-Moduls in eine verständliche Form umgewandelt. So werden Port, Produkt und Hersteller angegeben. Wenn der Scanner ausgewählt wurde, kann er dann beim Scannen eingesetzt werden. Dafür muss der ISBN-Button im Hauptmenü gedrückt werden. Der Code für diesen ist im [Python-Code 8](#) zu erkennen.

### Python-Code 8: ISBN-Button im Hauptmenü

```

1 if btn=="ISBN":
2     #schauen, ob Scanner verbunden ist
3     if scanner=="---":
4         bell.hide("Hauptmenu")
5         bell.showSubWindow("Isbn")
6     else:
7         bell.hide("Hauptmenu")
8         bell.showSubWindow("IsbnScan")

```

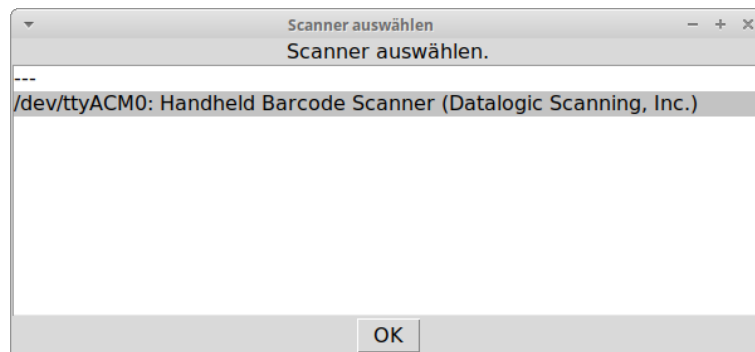


Abbildung 9: Scannermenü

Es ist zu erkennen, dass in Zeile 3 des [Python-Code 8](#) mit einer `if`-Anweisung gearbeitet wird. Diese ist erfüllt, wenn kein Scanner verbunden ist. Dann wird das „normale“ ISBN-Fenster gezeigt, in welcher der ISBN manuell eingegeben werden kann.

Sonst wird die `else`-Anweisung in Zeile 6 ausgeführt. Diese öffnet das ISBN-Scanner-Unterfenster, welches den Button aus [Python-Code 9](#) enthält.

Python-Code 9: ISBN-Button mit angeschlossnem Scanner

```

1  #ISBN-Scan-Button:
2  def pressIsbnScan(btn):
3      if btn=="Scannen":
4          #print scanner
5          ser=serial.Serial(scanner)
6          isbn=ser.read(13)
7          #print isbn
8          #Checken, ob Eingabe ISBN ist.
9          #isbn=canonical(ser.read(13))
10         if is_isbn13(isbn)==True:
11             #nichts
12             if l==[]:
13                 l.append(isbn)
14                 bell.hideSubWindow("IsbnScan")
15                 bell.infoBox("Erfolgreich", "Sie haben Ihre ISBN
16                             erfolgreich hinzugefügt.", parent="IsbnScan")
17                 bell.show("Hauptmenu")
18             #nur Bib
19             elif l[0][-1]=="b" and l[0][-2]=="i" and l[0][-3]=="b" and l
20                 [0][-4]=="." and len(l)==1:
21                 l.append(isbn)
22                 bell.enableButton("Fortfahren")
23                 bell.hideSubWindow("IsbnScan")
24                 bell.infoBox("Erfolgreich", "Sie haben Ihre ISBN
25                             erfolgreich hinzugefügt.", parent="IsbnScan")
26                 bell.show("Hauptmenu")
27             #nur ISBN
28             elif len(l)==1 and l[0][-1]!="b" and l[0][-2]!="i" and l[0][-3]!="
29                 b" and l[0][-4]!=".":
30                 if bell.yesNoBox("ISBN überschreiben", "Wollen Sie die
31                             ISBN überschreiben?", parent="IsbnScan")==True:
32                     del l[0]
33                     l.append(isbn)
34                     #print l
35                     bell.hideSubWindow("IsbnScan")
36                     bell.infoBox("Erfolgreich", "Sie haben Ihre ISBN
37                             erfolgreich hinzugefügt.", parent="IsbnScan")
38                     bell.show("Hauptmenu")
39             #beides
40             elif len(l)==2:
41                 if bell.yesNoBox("ISBN überschreiben", "Wollen Sie die

```

```

36         ISBN_überschreiben?", parent="IsbnScan")==True:
37             del l[1]
38             l.append(isbn)
39             bell.enableButton("Fortfahren")
40             bell.hideSubWindow("IsbnScan")
41             bell.infoBox("Erfolgreich", "Sie haben Ihre ISBN erfolgreich hinzugefügt.", parent="IsbnScan")
42             bell.show("Hauptmenu")
43
44     else:
45         bell.errorBox("Fehler", "Es ist ein Fehler aufgetreten!" + "\n" +
46             "Bitte überprüfen Sie die ISBN und versuchen Sie es erneut.",
47             parent="Isbn")

```

Dieser Button ermöglicht es dem Nutzer, den Scanner zu benutzen.

Zuerst wird in Zeile 5 der Scanner mithilfe des PySerial-Moduls ausgewählt.

Mit dem `read`-Befehl in Zeile 6 wird der Scanner aufgefordert, 13 Zeichen – die Länge einer ISBN – zu erfassen und in der `isbn`-Variable zu speichern. Wenn dies erfolgt ist, wird in Zeile 10 untersucht, ob die erfassten Zeichen eine valide ISBN sind.

Ist dies erfüllt, wird die ISBN der Liste angehängt (je nachdem, wie viele Objekte schon in der Liste vorhanden sind).

Alle relevanten Daten wurden somit in der GUI entweder per Hand eingegeben oder mit dem Scanner erfasst. Danach werden diese nun im Hintergrund vom Programm weiter verarbeitet.

## 2.2.4 Verarbeitung der Daten

Die Verarbeitung der eingegebenen Daten ist der Teil des Programmablaufs, welcher dem Nutzer den Mehrwert bringt.

Im GUI werden die relevanten Daten in einer Liste der folgenden Syntax gespeichert:

Liste=[Bibliotheksdatei, ISBN, BibTeX-Key].

Nun wird mit der ISBN des Werks in verschiedenen Datenbanken nach Informationen über das Buch, sogenannte Metainformationen bzw. Metatext, gesucht.

### Python-Code 10: Finden der Metadaten

```

1  #Metadaten finden und in Bibtex bei Variable metatext speichern
2  if bibtex(meta(isbn, service="goob"))==None:
3      try:
4          metatext = bibtex(meta(isbn, service="openl"))
5          #Metatext in Utf-8 codieren, da sonst Fehler auftreten könnten.
6          metatext = metatext.encode("utf-8")
7      except:
8          try:
9              metatext = bibtex(meta(isbn, service="wiki"))
10             metatext = metatext.encode("utf-8")
11         except:
12             metatext="Fehler!"
13 else:
14     metatext=bibtex(meta(isbn, service="goob"))
15     metatext = metatext.encode("utf-8")

```

Diese Suche in den Datenbanken ist in [Python-Code 10](#) zu erkennen.

Für die Suche wird das Modul `isbnlib`<sup>3</sup> verwendet. In diesem Modul gibt es den Befehl `meta` (Zeile 2, 4, 9 und 14), welcher die wichtigsten Metadaten aus der jeweiligen Datenbank sucht.

Wie aus Zeile 2, 4, 9 und 14 zu erkennen ist, benötigt der Befehl 2 Argumente. Das erste Argument

<sup>3</sup>siehe: <https://pypi.org/project/isbnlib/>

ist die ISBN und das zweite Argument ist die Datenbank, welche durchsucht werden soll. Es können die OpenLibrary-API<sup>4</sup> (`service="openl"`), die Google-Books-API (`service="goob"`) und die Wikipedia-API (`service="wiki"`) durchsucht werden.

Mit dem `bibtex`-Befehl, in welchem der `meta`-Befehl ausgeführt wird, werden die übermittelten Daten in die BibTeX-Form formatiert.

Zudem wird der Metatext mit dem Befehl `encode("utf-8")` encodiert, damit keine Fehler mit Umlauten entstehen.

Wie sich im [Python-Code 10](#) erkennen lässt, werden hier alle relevanten Datenbanken, der oben genannten, durchsucht. Dafür wird in Zeile 2 mit einer `if`-Anweisung begonnen, welche erfüllt ist, wenn keine Daten (`None`) mit der Google-Books-API gefunden werden können.

Bei Erfüllung der Anforderung der `if`-Anweisung wird die `try`-Anweisung in Zeile 3 ausgeführt, welche versucht Daten mit der OpenLibrary-API abzurufen. Wenn hier ein Fehler ausgegeben wird, kommt es `except`-Anweisung ausgeführt, welche wiederum in eine `try`-Anweisung mündet.

In dieser `try`-Anweisung (Zeile 8) wird mithilfe der Wikipedia-API versucht, Metadaten zur ISBN zu finden. Wenn auch dies fehlschlägt, wird der Metatext **Fehler!** (Zeile 12) gespeichert.

Wenn schon mithilfe der Google-Books-API Metadaten gefunden werden, dann wird die `else`-Anweisung in Zeile 13 ausgeführt und der gefundene Metatext gespeichert und codiert.

Python-Code 11: Einsetzung des BibTeX-Key

```
1 #Metatext modifizieren, so dass bibtexkey, der von Nutzer angegeben wurde verwendet werden kann.
2 if len(isbn)==10:
3     isbn=to_isbn13(isbn)
4
5 metatext=metatext.replace(canonical(isbn), bibtexkey, 1)
```

Nachfolgend wird der BibTeX-Key im Eintrag - wie im [Python-Code 11](#) zu sehen - ersetzt, denn der `meta`-Befehl fügt die ISBN-13 als BibTeX-Key automatisch hinzu.

Aufgrund der Diskrepanz zwischen den beiden BibTeX-Keys ist es notwendig, den vom User eingegebenen BibTeX-Key mit dem vom `meta`-Befehl zu ersetzen. Um die ISBN-13 zu ersetzen, muss die ISBN, falls sie als ISBN-10 eingegeben wurde, in eine ISBN-13 umgewandelt werden. Dies geschieht in Zeile 2 und 3.

Daraufhin wird in Zeile 5 mit dem Befehl `replace` der BibTeX-Key ersetzt.

Damit ist der Metatext im richtigen Format und kann in die Bibliotheksdatei eingefügt werden kann.

Python-Code 12: Anzeigen der Erfolgreich- bzw. Fehlermeldung

```
1 #wenn Infos empfangen wurden, werden sie der vom Nutzer angegebenen Bibdatei angehängt.
2 if metatext != "Fehler!":
3     bibdatei=open(directory, mode = "a+")
4     bibdatei.write(metatext + "\n")
5     app1=gui("erfolg")
6     app1.hide("erfolg")
7     app1.infoBox("Erfolgreich", "Ihr_Buch_wurde_dem_Literaturverzeichnis_hinzugefügt!")
8     exit()
9     #app1.go()
10 else:
11     app1=gui("fehler")
12     app1.hide("fehler")
13     app1.errorBox("Fehler", "Fehler!_Keine_Daten_gefunden!")
14     exit()
15     #app1.go()
```

Der Code für das Einfügen in die Bibliotheksdatei ist im [Python-Code 12](#) dargestellt.

Die `if`-Anweisung in Zeile 2 wird erfüllt, wenn Metadaten vorhanden sind. Dann wird die Bibliotheksdatei mithilfe des Befehls `open(directory, mode="a+")` geöffnet. `directory` steht dabei als Variable für die Bibliotheksdatei und der Modus `a+` steht für „append“, was „anhängen“ bedeutet, denn der Eintrag soll ans Ende der Bibliotheksdatei angehängt werden.

<sup>4</sup>API (application programming interface) ist die Programmierschnittstelle, mit welcher man im Quellcode, auf zum Beispiel eine Datenbank, eines anderen Programms zugreifen kann.

Anschließend wird der Metatext mithilfe des `write` Befehls (siehe Zeile 4) in die zuvor geöffnete Datei eingefügt. Wenn dies ausgeführt wurde, wird in Zeile 7 eine `infoBox` mithilfe von „AppJar“ erzeugt, welche die „Erfolreich“-Meldung anzeigt. Danach wird das Programm mit dem Befehl `exit()` in Zeile 8 geschlossen. Falls keine Metadaten gefunden wurden, wird die `else`-Anweisung in Zeile 10 ausgeführt und eine `errorBox`, also eine Fehlermeldung, mithilfe von „AppJar“ erzeugt, bevor das Programm mithilfe des `exit()`-Befehls beendet wird.

## 2.2.5 GitHub

GitHub ist eine Website, welche auf dem `git`-Versionskontrollsystem aufbaut. Auf der Website können Entwickler Projekte (auch Repositorys genannt) anlegen und ihren Fortschritt speichern. Im sogenannten „fork & pull“-Modell können verschiedene Zweige des Projektes erstellt werden, in denen dann verschiedene Ansätze ausprobiert werden können.

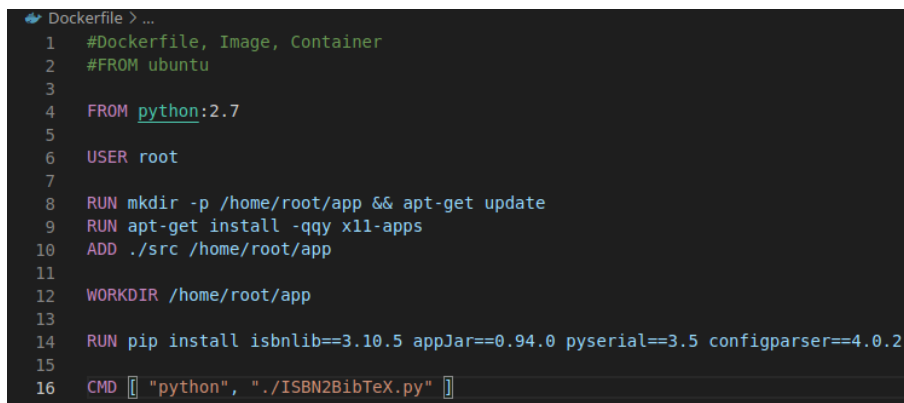
Dabei wird der Code online hochgeladen und kann auch von jedem Nutzer der Plattform angesehen und heruntergeladen werden. Dabei zählt GitHub mit über 10 Millionen angelegten Repositorys zu den größten und beliebtesten Plattformen. (Vgl. [Kalliamvakou u. a. 2014](#), S. 92)

Auch für das vorliegende Programm wurde mit GitHub gearbeitet, damit der Fortschritt gespeichert wird und auch online einsehbar ist<sup>5</sup>.

## 2.2.6 Docker

Docker ist ein Containerprogramm. Container sind Einheiten zur Speicherung von Software und all ihren Abhängigkeiten, wie zum Beispiel Modulen. Dadurch wird das Ausführen und der Transport von Software auf verschiedene Geräte erleichtert. Es kann als Image geteilt werden. Images sind die Abbilder von Containern, welche dann auf anderen Computern zum Erzeugen von Containern genutzt werden können.

Zum Erzeugen eines solchen Containers gibt es ein „Dockerfile“, welches Anweisungen gibt, welche Befehle im Container ausgeführt werden sollen. Ein solches „Dockerfile“ lässt sich in [Abbildung 10](#) erkennen.



```
Dockerfile > ...
1  #Dockerfile, Image, Container
2  #FROM ubuntu
3
4  FROM python:2.7
5
6  USER root
7
8  RUN mkdir -p /home/root/app && apt-get update
9  RUN apt-get install -qqy x11-apps
10 ADD ./src /home/root/app
11
12 WORKDIR /home/root/app
13
14 RUN pip install isbnlib==3.10.5 appJar==0.94.0 pyserial==3.5 configparser==4.0.2
15
16 CMD ["python", "./ISBN2BibTeX.py"]
```

Abbildung 10: Dockerfile

Im Dockerfile wird als Erstes in Zeile 4 das Image Python 2.7 eingefügt. Dies wird im Container benötigt, um „ISBN2BibTeX“ auszuführen. Als Nächstes wird der User `root` als aktiv ausgewählt (siehe Zeile 6) und in Zeile 8 und 9 werden Kommandozeilenbefehle ausgeführt, welche das Arbeitsverzeichnis des Containers erzeugen, updaten und „X11“<sup>6</sup> für die Benutzung des GUI installieren.

In Zeile 12 wird das Arbeitsverzeichnis als das vorher erzeugte festgelegt und in Zeile 14 werden die Module,

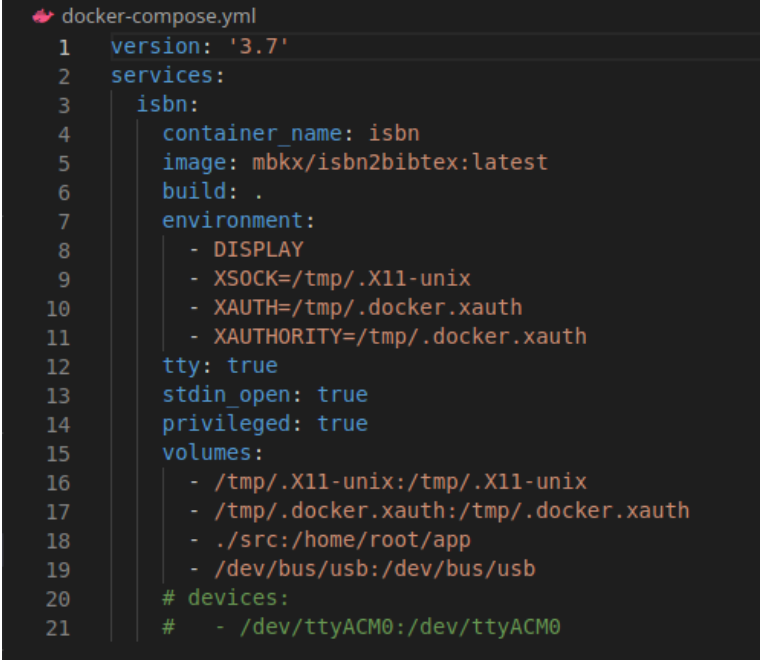
<sup>5</sup>siehe: <https://github.com/mbkx/bell>

<sup>6</sup>X11 ist eine Software, welche den grundlegenden „Werkzeugkasten“ für ein GUI bereitstellt.



welche für „ISBN2BibT<sub>E</sub>X“ benötigt werden installiert.  
Schlussendlich wird das Programm mit dem CMD-Befehl in Zeile 16 ausgeführt.

Zudem wird das Tool „Docker Compose“ verwendet. Dieses konfiguriert die Anwendung und ist eigentlich für Anwendungen, welche aus mehreren Containern bestehen, ausgelegt. Allerdings kann „Docker Compose“ auch mit nur einem Container verwendet werden. Die Docker-Compose-Datei – wie in [Abbildung 11](#) zu erkennen – ist sozusagen ein erweitertes Dockerfile. Dies ermöglicht es mehrere Container gleichzeitig zu starten, aber auch die Syntax erleichtert. In dieser Docker-Compose-Datei wird in Zeile 3 der „Service“ `isbn`



```
1 version: '3.7'
2 services:
3   isbn:
4     container_name: isbn
5     image: mbkx/isbn2bibtex:latest
6     build: .
7     environment:
8       - DISPLAY
9       - XSOCK=/tmp/.X11-unix
10      - XAUTH=/tmp/.docker.xauth
11      - XAUTHORITY=/tmp/.docker.xauth
12     tty: true
13     stdin_open: true
14     privileged: true
15     volumes:
16       - /tmp/.X11-unix:/tmp/.X11-unix
17       - /tmp/.docker.xauth:/tmp/.docker.xauth
18       - ./src:/home/root/app
19       - /dev/bus/usb:/dev/bus/usb
20     # devices:
21     #   - /dev/ttyACM0:/dev/ttyACM0
```

Abbildung 11: Docker-Compose-Datei

definiert, welcher „ISBN2BibT<sub>E</sub>X“ ist.

Dabei wird als erstes der Containername als `isbn` festgelegt und das Image von „ISBN2BibT<sub>E</sub>X“ geladen (siehe [Abbildung 10](#)).

Von Zeile 7 bis Zeile 12 werden `environment`-Variablen festgelegt, welche für den Zugriff des Containers auf den Bildschirm des Hosts benötigt werden, um das GUI anzuzeigen.

Außerdem werden `tty`, `stdin_open` und `privileged` auf `true` gesetzt, um die Scannereinbindung in den Container zu ermöglichen.

Dazu dient auch das `volume dev/bus/usb`, welches die USB-Ports übergibt.

Die `volumes` in Zeile 15 und 16 werden für das GUI benötigt und das `volume` in Zeile 18 übergibt den Ordner mit den benötigten Dateien.

Mit dem Kommandozeilenbefehl `docker-compose up`, wird der Container gestartet.

Außerdem bietet es sich an, den Container mit dem Befehl `docker-compose run --rm isbn bash` zu starten, um die Kommandozeile des Containers zu starten. Somit kann nachvollzogen werden, ob das Programm auch seinen Zweck erfüllt.

## 3 Abschlussbetrachtung

Am Ende werden die Ergebnisse zusammengefasst. Zudem wird ein Ausblick gegeben und anschließend ausgewertet, sowie kritisch betrachtet.

### 3.1 Zusammenfassung

Es folgt die Zusammenfassung der Arbeit, welche – wie auch die Arbeit – in Theorie- und Praxisteil gegliedert ist.

#### Theorie

Die Theorie der Softwareentwicklung lässt sich in zwei große Teile, die klassische und die agile Softwareentwicklung, unterteilen.

Zum einen ist das die klassische Softwareentwicklung. Bei dieser wird der SDLC im Idealfall einmal ausgeführt. Das bedeutet, dass alle großen Phasen der Entwicklung – also Analyse, Definition der erforderlichen Inhalte, Architektur, Implementierung, Testen und Veröffentlichung – einmal durchlaufen werden.

Dadurch entstehen einige Vor-, aber auch Nachteile.

Vorteile sind zum Beispiel die leichte Anwendung und Verständnis des Modells oder die einfache Koordination der einzelnen Entwicklungsabschnitte.

Allerdings kann durch die stufenweise Ausführung schlecht auf Veränderungen in den Anforderungen der Software eingegangen werden, was einen Nachteil darstellt.

Ein Beispielmmodell für die klassische Softwareentwicklung ist das Wasserfallmodell. Diese Art der Softwareentwicklung eignet sich vor allem für große Firmen und Projekte.

Einen anderen Ansatz der Softwareentwicklung verfolgt die agile Softwareentwicklung.

Bei diesem Ansatz wird versucht, eine hohe Anpassungsfähigkeit an Veränderungen (Agilität) zu erzeugen. Das kann durch verschiedene Methoden geschehen.

Eine davon ist das Wiederholen des SDLC – der schon aus der klassischen Softwareentwicklung bekannt ist – für nicht mehr nur das ganze Programm, sondern für jeden Programmteil. Dadurch ist jeder Programmteil einzeln funktionsfähig und bei Veränderungen müssen nicht alle, sondern nur die betroffenen Programmteile, verändert werden.

Ein Mitbegründer der agilen Softwareentwicklung ist Kent Beck, welcher mit anderen Softwareentwicklern 2001 das „Agile Manifest“ veröffentlichte. In diesem sind verschiedene Leitsätze und Prinzipien festgehalten, welche die „Essenz“ der agilen Softwareentwicklung enthalten.

Beck ist u. a. auch Begründer von Extreme Programming, welches ein praktisches Modell für die agile Softwareentwicklung darstellt.

Jedoch hat auch das agile Modell Schwächen. Durch die hohe Agilität ist eine hohe Kommunikation zwischen den einzelnen Entwicklern erforderlich, was bei großen Projekten mit vielen Developern schwierig gewährleistet werden kann.

Deshalb ist der agile Ansatz eher für kleinere Projekte beziehungsweise Entwicklergruppen geeignet.

Es muss aber auch erwähnt werden, dass dies nur Modellvorstellungen sind. Sie werden selten in Reinform, sondern eher als Mischung verwendet. So finden sich zum Beispiel viele Methoden der klassischen, als auch der agilen Softwareentwicklung, bei dieser Programmentwicklung wieder (siehe [Unterabschnitt 2.2.1](#)).

## Programm

Es wurde ein Programm entwickelt, welche die Arbeit mit Quellenangaben in L<sup>A</sup>T<sub>E</sub>X erleichtern soll. Das Programm trägt den Namen „ISBN2BibT<sub>E</sub>X“, welches auch die Funktionen des Programms widerspiegelt. Das Programm wurde mithilfe der Programmiersprache Python entwickelt. Es kann die ISBN eines Buchs aufnehmen (manuell oder mithilfe eines Scanners) und diese in einen Eintrag des Literaturverzeichnis umwandeln. Dabei werden die relevanten Daten, wie Titel, Autor und Erscheinungsjahr, automatisch eingesetzt. Der erste Programmteil, der entwickelt wurde, war das GUI. Dieses Graphical User Interface wurde zuerst mit dem Programm „Pencil“ modelliert und danach mithilfe des „AppJar“-Moduls in Python programmiert. Dabei wurde auch der Scanner durch das Modul „pySerial“ eingebunden. Danach folgt die Verarbeitung der Daten und das Suchen der Metadaten mithilfe des Moduls „isbnlib“. Die Dokumentation des Entwicklungsprozesses erfolgt mit „GitHub“, welches auf „Git“ aufbaut. Um das Programm auch anderen User zur Verfügung zustellen, wird „Docker“ bzw. „Docker Compose“ benutzt, mit welchen das Programm für die meisten Linux-Systeme funktioniert.

## 3.2 Ausblick

Das Programm erfüllt viele Funktionen, welche in der ursprünglichen Planung vorgesehen waren, wie beispielsweise die Scannereinbindung. Allerdings gibt es einige Probleme mit den anderen Betriebssystemen (MacOS, Windows), da sich manche Befehle und Einstellungen von den Linux-Einstellungen unterscheiden. Deshalb könnte es ein zukünftiges Ziel sein, dass das Programm für alle Betriebssysteme zugänglich wird.

Außerdem gibt es noch weitere Funktionen, die ergänzt werden sollen, um das Programm zu verbessern. Dazu zählt zum Beispiel die wiederholte Eingabe von Einträgen, die Auswahl der Datenbank, in welcher die ISBN gesucht werden soll oder auch das Hinzufügen von anderen Datenbanken zur Suche.

Zudem wäre ein Erfolg, wenn das Programm, zum Beispiel bei Schülern, welche eine Besondere Lernleistung erbringen, getestet und später eingesetzt werden kann. So kann die Arbeit mit Quellen und dem Quellenverzeichnis erleichtert werden. Dazu soll ein Tutorial entwickelt werden, welches eine Anleitung gibt, wie das Programm installiert und funktionsfähig gemacht werden kann.

## 3.3 Diskussion

In diesem Abschnitt wird eine kritische Auseinandersetzung mit allen Teilen des Programms vorgenommen. Die meisten Probleme traten bei der Gestaltung des GUI und den damit verbundenen Funktionen auf. Hierbei gestaltet es sich schwierig, die Kompatibilität mit anderen Betriebssystemen, wie zum Beispiel Windows und MacOS zu gewährleisten, ohne Abstriche, zum Beispiel beim Design, zu verzeichnen. Das Modul „AppJar“ erwies sich dabei durchaus als gute Option. Jedoch ist das Modul eigentlich nur für kleinere Projekte gedacht und baut auf dem „Tkinter“-Modul auf. Somit hätte auch gleich dieses genutzt werden können, um umfassendere Funktionen und Designoptionen zu ermöglichen – auch wenn dies keinen großen Einfluss auf die Funktionalität des Programms hätte.

Ein weiteres Problem tritt bei der Scannereinbindung auf. Es ist dabei schwierig – als einzelner Entwickler ohne großes Budget – die Funktionsfähigkeit mehrerer Scanner zu gewährleisten, da es nicht einfach ist, diese Scanner zu erhalten, da diese meist preisintensiv sind. Dadurch ist die Scannereinbindung nur mit einem Scanner (Datalogic GFS4170) und nur für Linux verfügbar. Die Verfügbarkeit für andere Scanner und Betriebssysteme soll ergänzt werden.

Ein weiteres Problem tritt beim Containerprogramm „Docker“ auf, welches eigentlich die Nutzung auf den meisten Betriebssystemen ermöglichen sollte. Jedoch ist „Docker“ nicht für GUIs ausgelegt, weil bei diesen der Bildschirm des benutzten Geräts benutzt wird. Jedoch hat der Container keinen Bildschirm, deshalb muss der Container Zugriff auf den Bildschirm des Computers erhalten. Jedoch ist dieser Zugriff mit verschiedenen Prozessen und Variablen auf den unterschiedlichen Betriebssystemen verbunden. Dadurch ist das Programm mit „Docker“ nur für Linux-Systeme mit „X11“ verfügbar.

## 3.4 Fazit

Zusammenfassend lässt sich sagen, dass das Programm funktioniert und alle wichtigen Funktionen enthalten sind, sodass das Programm auf jedem Betriebssystem arbeitet. Auf Windows und MacOS müssen die entsprechenden Module noch installiert werden (`pip install [Modul]`) und das Programm im Terminal ausgeführt werden muss (`python ISBN2BibTeX.py` im Verzeichnis, in dem das Programm gespeichert ist). Für Linux ist das Programm am weitesten entwickelt. Dort ist es auch mit Docker verfügbar. Dadurch wird ermöglicht, dass jede Person mit einem Computer mit Linux-Betriebssystem und „X11“ das Programm benutzen kann.

Das Programm erleichtert die Arbeit mit dem Quellenverzeichnis in L<sup>A</sup>T<sub>E</sub>X. Dafür bedarf es der Eingabe der ISBN (scannen oder manuell), sowie der Angabe der Bibliotheksdatei, um daraus einen fertigen Eintrag im Literaturverzeichnis zu erzeugen.

Zwischen der Idee für das Programm und dem schlussendlichen Produkt lagen allerdings viele Arbeitsschritte.

Zuerst wurde das gesamte Programm geplant und in einzelne Teile „zerlegt“, welche dann nacheinander genau geplant und implementiert wurden, sodass letztendlich das Programm entstand. Auch mit allen fertigen Programmteilen wurde das Programm noch weiterentwickelt und es wurde versucht, das Programm für alle Betriebssysteme zur Verfügung zu stellen.

Allerdings ist es dabei schwierig, die Funktionsfähigkeit für alle Betriebssysteme und Scanner zu testen, da es keinen einfachen Zugang zu allen Betriebssystemen und Scannerarten gibt.

Schlussendlich lässt sich erhoffen, dass das Programm die meist schwierigen und komplexen Quellenangaben bei fachwissenschaftlichen Arbeiten – zum Beispiel der BeLL – erleichtert und Zeit spart, welche dadurch anderweitig verwendet werden kann.

# Literaturverzeichnis

- [Alshamrani u. Bahattab 2015] ALSHAMRANI, Adel ; BAHATTAB, Abdullah: A comparison between three SDLC models waterfall model, spiral model, and Incremental/Iterative model. In: *International Journal of Computer Science Issues (IJCSI)* 12 (2015), Nr. 1, S. 106
- [Balaji u. Murugaiyan 2012] BALAJI, S ; MURUGAIYAN, M S.: Waterfall vs. V-Model vs. Agile: A comparative study on SDLC. In: *International Journal of Information Technology and Business Management* 2 (2012), Nr. 1, S. 26–30
- [Beck 1999] BECK, K.: Embracing change with extreme programming. In: *Computer* 32 (1999), Oktober, Nr. 10, 70–77. <http://dx.doi.org/10.1109/2.796139>. – DOI 10.1109/2.796139. – ISSN 00189162
- [Beck u. a. 2013] BECK, Kent M. ; BEEDLE, Mike ; BENNEKUM, Arie van ; COCKBURN, Alistair ; CUNNINGHAM, Ward ; FOWLER, Martin ; GRENING, James ; HIGHSMITH, Jim ; HUNT, Andy ; JEFFRIES, Ron ; KERN, Jon ; MARICK, Brian ; MARTIN, R. C. ; MELLOR, Steve J. ; SCHWABER, Ken ; SUTHERLAND, Jeff ; THOMAS, Dave: Manifesto for Agile Software Development, 2013
- [Fritzsche u. Keil 2007] FRITZSCHE, Martin ; KEIL, Patrick: Kategorisierung etablierter Vorgehensmodelle und ihre Verbreitung in der deutschen Software-Industrie. (2007)
- [Jeffries u. a. 2000] JEFFRIES, Ron ; ANDERSON, Ann ; HENDRICKSON, Chet: *Extreme Programming Installed*. Addison-Wesley, 2000
- [Kalliamvakou u. a. 2014] KALLIAMVAKOU, Eirini ; GOUSIOS, Georgios ; BLINCOE, Kelly ; SINGER, Leif ; GERMAN, Daniel M. ; DAMIAN, Daniela: The promises and perils of mining GitHub. In: *Proceedings of the 11th working conference on mining software repositories*, 2014, S. 92–101
- [Lamport 2020] LAMPORT, Leslie: My Writings. (2020). <http://lamport.azurewebsites.net/pubs/pubs.pdf>
- [Laplante u. Neill 2004] LAPLANTE, Phillip A. ; NEILL, Colin J.: The demise of the waterfall model is imminent. In: *Queue* 1 (2004), Nr. 10, S. 10–15
- [Leau u. a. 2012] LEAU, Yu B. ; LOO, Wooi K. ; THAM, Wai Y. ; TAN, Soo F.: Software development life cycle AGILE vs traditional approaches. In: *International Conference on Information and Network Technology* Bd. 37, 2012, S. 162–167
- [Stoica u. a. 2013] STOICA, Marian ; MIRCEA, Marinela ; GHILIC-MICU, Bogdan: Software Development: Agile vs. Traditional. 17 (2013), Dezember, Nr. 4/2013, 64–76. <http://dx.doi.org/10.12948/issn14531305/17.4.2013.06>. – DOI 10.12948/issn14531305/17.4.2013.06. – ISSN 14531305, 18428088
- [Williams u. a. 2000] WILLIAMS, L. ; KESSLER, R.R. ; CUNNINGHAM, W. ; JEFFRIES, R.: Strengthening the case for pair programming. In: *IEEE Software* 17 (2000), August, Nr. 4, 19–25. <http://dx.doi.org/10.1109/52.854064>. – DOI 10.1109/52.854064. – ISSN 07407459

# Abbildungsverzeichnis

1	Der Software Development Life Cycle. Selbständig erstellte Grafik . . . . .	4
2	Wasserfallmodell. Selbständig erstellte Grafik nach: Fritzsche 2007, S.9 . . . . .	7
3	XP Ablaufschema. Selbständig erstellte Grafik . . . . .	9
4	Aufbau eines Eintrags. Selbständig erstellte Grafik . . . . .	13
5	Ablaufplan des Programms. Selbständig erstellte Grafik . . . . .	14
6	Fehlermeldung. Selbständig erstellte Grafik . . . . .	15
7	Entwurf des Hauptmenüs des GUI in Pencil. Selbständig erstelltes Foto, aufgenommen am: 28.12.2020 . . . . .	16
8	Verlinken von Unterfenstern in Pencil. Selbständig erstelltes Foto, aufgenommen am: 28.12.2020	17
9	Scannermenü. Selbständig erstelltes Foto, aufgenommen am: 28.12.2020 . . . . .	21
10	Dockerfile. Selbständig erstelltes Foto, aufgenommen am: 28.12.2020 . . . . .	24
11	Docker-Compose-Datei. Selbständig erstelltes Foto, aufgenommen am: 28.12.2020 . . . . .	25

# Quellcodeverzeichnis

1	Hauptmenü . . . . .	17
2	Buttonanweisungen des Hauptmenüs . . . . .	18
3	Code der Bibliotheksdateiauswahl . . . . .	18
4	Boxen . . . . .	19
5	Links . . . . .	19
6	Erkennen der angeschlossenen Scanner . . . . .	20
7	Scannermenü . . . . .	20
8	ISBN-Button im Hauptmenü . . . . .	20
9	ISBN-Button mit angeschlossnem Scanner . . . . .	21
10	Finden der Metadaten . . . . .	22
11	Einsetzung des BibTeX-Key . . . . .	23
12	Anzeigen der Erfolgreich- bzw. Fehlermeldung . . . . .	23

## Anmerkung

Alle Quellcodes sind aus dem Programm entnommen und damit selbständig erstellt.

# Danksagung

Schlussendlich möchte ich mich bei allen Menschen bedanken, welche mich bei dem Erstellen dieser Arbeit und beim Erstellen des Programms unterstützt haben.

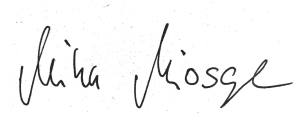
In erster Linie sind dies natürlich meine Betreuer (Matthias Richter und Stephan Keller), an welche ich mich bei Fragen immer wenden konnte und die mir auch bei der Themenfindung und den anderen Teilen der Entwicklung der Arbeit weitergeholfen haben.

Außerdem möchte ich meiner Familie danken, die mich immer tatkräftig unterstützte und an die ich mich immer wenden konnte, wenn ich Probleme hatte.



# Selbstständigkeitserklärung

Hiermit erkläre ich, dass ich die vorliegende Besondere Lernleistung selbstständig und ohne unerlaubte fremde Hilfe angefertigt, keine anderen als die angegebenen Quellen und Hilfsmittel verwendet und die den verwendeten Quellen und Hilfsmitteln wörtlich oder inhaltlich entnommenen Stellen als solche kenntlich gemacht habe.

A handwritten signature in black ink, reading "Mika Miosge". The signature is written in a cursive style with a large, stylized 'M' and 'M'.

Leipzig, 15. Januar 2021

Mika Miosge