Phasen, Techniken und Strategien der Softwareentwicklung anhand eines Kursprojekts im Modul SWE-II

Miklas Boskamp, Martin Böttcher, David Golla, Matthias Klassen, Fabian Klose
Hochschule für Wirtschaft und Recht Berlin

***Zusammenfassung*—In den Modulen Software-Engineering I und II des Informatikstudiums werden aktuelle Technologien und Vorgehensmodelle der Softwareentwicklung gelehrt. Um den Studierenden die Möglichkeit zu geben, das in den Lehrveranstaltungen vermittelte Wissen in der Praxis anzuwenden, wurden Gruppen gebildet, die im Laufe des Moduls eine frei wählbare Software entwickeln sollten.**

In der Textverarbeitungssoftware Latex kann der Nutzer ein Literaturverzeichnis aus einer Bib-Datei generieren lassen. Um das Erstellen der Bib-Datei nach dem BibTeX-Standard zu erleichtern, wurde als Kursprojekt die Erstellung einer Software mit einer grafischen Oberfläche namens BibFileGenerator gewählt. Mit ihr ist der Nutzer in der Lage ein Literaturverzeichnis anzulegen und als Bib-Datei zu speichern, ohne dass Kenntnisse über den BibTeX-Standard notwendig sind.

Die Entwicklung des BibFileGenerators fand unter strenger Beachtung der verschiedenen Phasen, Techniken und Strategien beim Software-Engineering statt. So wurden zum Anfang die Anforderungen der Software ermittelt und dazugehörige Diagramme zur Visualisierung und Diskussion mit den Stakeholdern erstellt. Bevor die Codierung dann startete, wurde über die Bedienoberfläche, das Programmierparadigma und die dazugehörigen Probleme nachgedacht. Dadurch entstand ein genauer Plan und die Codierung konnte zwischen den Entwicklern aufgeteilt werden. Parallel zur Entwicklung wurde die Software durch das Team getestet.

I. Einleitung

Für das Erstellen von wissenschaftlichen Arbeiten und anderen Dokumenten wird oft Latex verwendet. Dabei handelt es sich um eine freie Software zur Textverarbeitung [1], die meist an Hochschulen und Universitäten Anwendung findet. Der Autor erstellt Textdateien, in denen hervorzuhebende Abschnitte mit entsprechenden Befehlen gekennzeichnet werden [2]. Der in dieser Textdatei entstehende Quellcode kann mit Hilfe eines Übersetzers (Compilers) z.B. zu einer PDF-Datei umgewandelt werden. In der Regel werden Latex-Dokumente jedoch in einer Entwicklungsumgebung angefertigt, da diese Vorteile wie die Hervorhebung von Befehlen im Text oder die Ausgabe der compilierten Datei mit wenigen Knopfdrücken bietet. Im Vergleich zu anderen Programmen zur Textverarbeitung erfordert Latex eine längere Einarbeitungszeit, ermöglicht dafür aber eine präzise Gestaltung des Dokuments [2].

In einem Latex-Dokument besteht die Möglichkeit ein Literaturverzeichnis zu generieren. Dafür muss zusätzlich eine Bib-Datei mit Informationen zur verwendeten Literatur angelegt und in das Dokument eingebunden werden. In dieser Bib-Datei ist allerdings eine strenge Syntax vorgeschrieben, wodurch erneut Recherchen über die korrekte Nutzung und eine Einarbeitung erfolgen müssten. Um das Generieren eines Literaturverzeichnisses zu erleichtern, wird als Kursprojekt eine Software mit einer grafischen Oberfläche entwickelt, die es ermöglicht eine Bib-Datei zeitnah und intuitiv anzulegen. Der Nutzer soll mithilfe der Software ein Verzeichnis mit den nötigen Informationen anlegen können, welches daraufhin als Bib-Datei gespeichert wird. Dieses kann daraufhin über wenige Befehle im Latex-Dokument eingebunden werden.

Zum Lernen der korrekten und effizienten Arbeitsweise bei der Entwicklung von Software, wird im Laufe des Projekts besonders auf die Einhaltung und Nutzung von Phasen, Techniken und Strategien der Softwareentwicklung geachtet.

II. Analyse

Bevor das Programmieren der Software beginnen kann, muss die erste Phase der Softwareentwicklung, die Analyse, durchlaufen werden. In ihr werden die Anforderungen für die Software identifiziert, analysiert und dokumentiert. Um die nötigen Anforderungen zu bestimmen, sollten Informationen von Stakeholdern beschaffen werden. Stakeholder sind Personen mit besonderen Interessen an der Software. Zu ihnen können sowohl die Auftraggeber und Entwickler, als auch die zukünftigen Nutzer der Software zählen. Sie haben nicht nur Wünsche und Vorstellungen für die zu entwickelnde Software, sondern können auch die Kunden für das Endprodukt sein. Damit die Stakeholder zufrieden mit der Software sind, ist es wichtig während der Entwicklung mit ihnen in Kontakt zu bleiben. Um Informationen von ihnen zu erhalten, können verschiedene Verfahren angewandt werden. Es können beispielsweise Interviews von Einzelpersonen, Workshops mit mehreren Personen oder Umfragen, die große Gruppen von Stakeholdern erreichen und dabei kostengünstig sind, durchgeführt werden. Wichtig ist, dass die Informationen Klarheit über die Ziele der zu entwickelnden Software verschaffen. Die Entwickler müssen verstehen, welche Ziele mit der Software erreicht werden sollen, welche Abhängigkeiten zwischen den Zielen herrschen und welche Randbedingungen beachtet werden müssen. (Vgl. zu diesem Abschnitt [3])

Die ermittelten Ziele werden daraufhin granuliert, um Anforderungen zu entwickeln, die darstellen, wie die Ziele erreicht werden können. Um die Anforderungen strukturiert zu dokumentieren, werden sie zunächst klassifiziert. Dabei können sie z.B. in Oberkategorien wie „grafische Oberfläche“

und „Programmlogik“ und daraufhin in weitere Unterkategorien geteilt werden. Natürlich können auch Prioritäten festgelegt werden, die z.B. Auskunft darüber geben, in welcher Version der Software eine Anforderung umgesetzt werden soll. Generell unterscheidet man zwei Arten von Anforderungen: Die funktionalen Anforderungen sind die Ziele für die Funktionalitäten des Systems. Dazu zählt z.B. das Sortieren von Namen nach dem Alphabet. Nutzeranforderungen hingegen beschreiben die Ziele und Ansprüche die Interaktionen durch den Nutzer erfordern, wie z.B. die Eingabe des Namens. (Vgl. zu diesem Abschnitt [3])

Stehen die Anforderungen fest, sollte eine Rücksprache mit den Stakeholdern geführt werden. Dies ist wichtig, um einerseits Unklarheiten auf der Entwicklerseite abzuschaffen und andererseits Fehler aufzudecken oder Ergänzungen durch die Stakeholder vorzunehmen. Dafür ist es notwendig den Stakeholdern die Anforderungen möglichst verständlich zu präsentieren. Um das zu erreichen, können z.B. UML-Diagramme wie Klassen-, Use-Case- oder Sequenzdiagramme erstellt werden, die die Software modellieren. (Vgl. zu diesem Abschnitt [3])

In diesem Kursprojekt legte das Entwicklerteam die Anforderungen der Software in Meetings fest, ohne dass weitere Personen hinzugezogen wurden. Dies war möglich, da jedes Teammitglied zu den Stakeholdern gehörte und eine Vorstellung hatte welche Funktionen die Software beinhalten sollte. Nach dem ersten Meeting wurden alle in Betracht kommenden Ideen durch Protokollanten notiert und eine erste Vorstellung der Software entstand. Die gesammelten Anforderungen wurden daraufhin analysiert. Dafür mussten sie granuliert werden, sodass jede Anforderung genau einer Funktionalität entsprach. Nun wurden die atomaren Anforderungen in die Oberkategorien Frontend, für die grafische Oberfläche, und Backend, für die Logik des Programms, unterteilt. Weiterhin fand eine Klassifizierung in Programmabläufe (Use-Cases) und die Prioritätensetzung statt. Das ist hilfreich für die Entwickler, um die Strukturierung der Software schnell zu verstehen. Das Resultat wurde in einer Excel-Tabelle in schriftlicher Form festgehalten und mit zusätzlichen Informationen, wie Randbedingungen und Abhängigkeiten, versehen. Nach weiteren Meetings entstanden außerdem ein Use-Case- und ein Aktivitätsdiagramm eines Programmablaufs (siehe Abb. 1). Von Zeit zu Zeit auftretende Mängel und Ergänzungen wurden in den darauffolgenden Meetings schnellstmöglich behoben und getätigt.

III. Entwurf

Nach der Analysephase folgt die Entwurfsphase. In dieser werden die Anforderungen, welche während der Analyse definiert wurden, in eine Systemarchitektur übersetzt. Schließlich entsteht ein objektorientiertes Design, welches die technische Lösung darstellt. Dieses Modell kann auch als objektorientiertes Programm auf höherer Abstraktionsebene gesehen werden. Durch den Entwurf wird das System auf eine einfache Art und Weise beschrieben, wodurch die Kommunikation zwischen

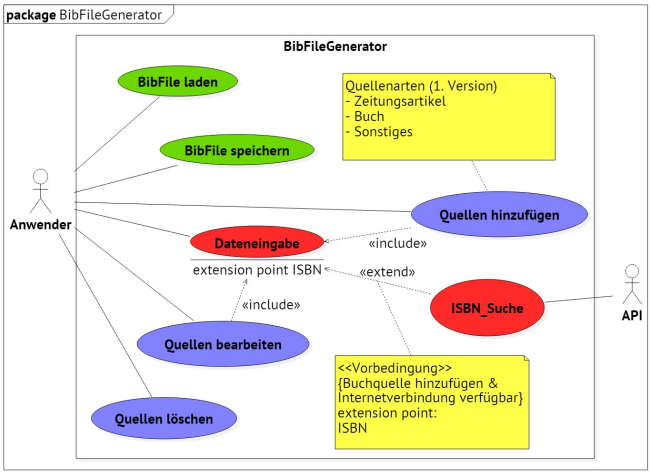

Abbildung 1. Use-Case-Diagramm des BibFileGenerators

den Stakeholdern erleichtert wird. Zudem ist die Architektur der Ausgangspunkt zur Implementierung. Die Entwickler codieren das Programm auf Grundlage dieses Entwurfs [4].

Während der Entwurfsphase wird auch erkennbar, wie viele Schichten das Programm schließlich haben wird. So gibt es eine 2-Schichten-Architektur, in der die Applikation nur in Logik und Datenbank getrennt wird. Weiterhin werden oft 3-Schichten-Architekturen genutzt, in der zwischen dem Client und der Datenbank noch ein Webserver steht. Die dritte Schichten-Architektur ist die Service-orientierte Architektur. Dort sind alle Services eingenständige Komponenten, das heißt jeweils ein eigenes kleines Programm [5].

Ein weiterer Teil der Entwurfsphase, ist der Entwurf der Bedienoberfläche. Er ist ein entscheidender Bestandteil der Softwareentwicklung, da dies die Schnittstelle mit dem Nutzer ist. Dabei müssen verschiedene Faktoren berücksichtigt werden [6]:

- begrenztes Kurzzeitgedächnis
- alle machen Fehler
- unterschiedliche körperliche Fähigkeiten
- unterschiedliche Vorlieben für Interaktionen

Zudem wurden von Sommerville [7] Grundsätze für den Entwurf von Bedienoberflächen definiert:

- Benutzervertrautheit (Oberfläche sollte Bezeichnungen und Begriffe aus der Erfahrungswelt der Nutzer verwenden)
- Konsistent (vergleichbare Operationen werden auf dieselbe Weise veranlasst)
- Minimale Überraschung (Nutzer sollten nicht vom Systemverhalten überrascht werden)
- Wiederherstellbarkeit (Wiederherstellungsmechanismen für Nutzer)
- Benutzerführung (beim Auftreten von Fehler aussagekräftige Rückmeldungen, inklusive kontextsensitive Hilfsmittel für den Nutzer)
- Benutzervielfalt (für verschiedene Arten von Nutzern geeignete Interaktionsmöglichkeiten)

Neben den Grundsätzen für den Entwurf von Bedienoberflächen nach Sommerville sind 2006 vom Europäischen Komitee für Normung in der „DIN EN ISO 9241 Teil 110: Grundsätze der Dialoggestaltung“ Dialoggrundsätze für interaktive Systeme definiert worden:

- Aufgabenangemessenheit (Funktionalität und Dialog basieren auf den charakteristischen Eigenschaften der Arbeitsaufgabe und nicht auf der eingesetzen Technologie)
- Selbstbeschreibungsfähigkeit (für Nutzer ist es offensichtlich, in welchem Dialog, an welcher Stelle im Dialog er sich befindet, welche Handlungen unternommen werden können und wie diese ausgeführt werden können)
- Steuerbarkeit (Nutzer kann Dialogablauf starten, Richtung und Geschwindigkeit beeinflussen)
- Erwartungkonformität (Dialog entspricht allgemein anerkannten Koventionen)
- Fehlertoleranz (beabsichtigtes Arbeitsergebnis kann trotz erkennbar fehlerhaften Eingaben mit keinem oder minimalen Korrekturaufwand erreicht werden)
- Individualisierbarkeit (Nutzer kann Darstellung von Informationen ändern)
- Lernförderlichkeit (Dialog unterstützt Nutzer beim Erlernen der Nutzung des interaktiven Systems)

Neben den oben genannten Grundsätzen, stellt sich die Frage, wie ein Nutzer mit dem System interagieren kann [6]:

- Direkte Manipulation
- Menüauswahl
- Ausfüllen einer Eingabemaske
- Befehlssprache
- Natürliche Sprache

Ein weiterer wichtiger Aspekt bei der Gestaltung der Oberfläche ist der Einsatz von Farbe. Dabei ist dieser Teil des Entwurfs von Bedienoberflächen einer der schwierigsten, da die menschliche Farbempfindung unterschiedlich ist. In verschiedenen Berufen und Kulturen gibt es unterschiedliche Vereinbarung über die Bedeutung von Farben.

Der Einsatz von Farben sollte zurückhaltend geschehen. Mittels Farbänderungen können Änderungen des Systemzustandes angezeigt werden, wodurch auch unterschiedliche Farbcodes den Nutzer bei seiner Arbeit unterstützen können. Es muss dafür gesorgt werden, dass Farbcodes konsistent genutzt werden [6].

Sollte während der Laufzeit ein Fehler auftreten oder sollten Informationen über den Systemstatus für den Nutzer relevant sein, werden Meldungen genutzt. Bei dem Entwurf dieser sind folgende Grundsätze zu beachten [6]:

- Kontext (Meldungen sollten den Nutzer bei seiner derzeitigen Tätigkeit unterstützen)
- Erfahrung (ein System sollte dem Nutzer die Möglichkeit geben, die Genauigkeit der Meldungen zu steuern)
- Fähigkeiten (Meldungen sollten auf die Fähigkeiten und Erfahrungen der Nutzer zugeschnitten sein)
- Stil (eine Meldung sollte positiv formuliert sein, eine aktive Anrede beinhalten und nicht verletzend oder komisch sein)
- Kultur (Gestaltung von Meldungen sollten auf die Kultur des Landes, in der das System genutzt wird, angepasst werden)

Nach der Gestaltung einer Oberfläche sollte diese zur ihrer Benutzerfreundlichkeit bewertet werden. Dazu wurden verschiedene Attribute definiert [8]:

- Erlernbarkeit („Wie lange dauert es, bis ein neuer Benutzer produktiv mit dem System arbeiten kann?“)
- Antwortgeschwindigkeit („Wie gut passt die Antwortzeit des System zur Arbeitsweise des Benutzers?“)
- Stabilität („Wie tolerant ist das System gegenüber fehlerhaften Eingaben?“)
- Wiederherstellbarkeit („Wie gut kann sich das System von Eingabefehlern erholen?„)
- Anpassungsfähigkeit („Wie eng ist das System auf ein einzelnes Arbeitsmodell zugeschnitten?“)

Schließlich ist die Frage zu klären, wie eine Oberfläche entwickelt werden kann. Normalerweise werden dazu Klassenbibliotheken genutzt. Außerdem können mittels GUI-Builder Oberflächen einfach zusammengestellt werden (beispielsweise per Drag-and-Drop) [8].

Für die Steuerung von Benutzerinteraktionen wird häufig das Model-View-Controller(MVC)-Modell genutzt. Dies löst das Problem, dass die GUI oft geändert wird, indem das System in Verabeitung, Output und Input aufgeteilt wird. Das Model kapselt die Daten, die View stellt die Informationen dar und der Controller empfängt Events [6].

Da das Projekt bereits im dritten Semester inklusive ersten Implementierungen begann, wurde diese Phase zum Teil übersprungen. Von einem Mitglied des Teams wurde dennoch bereits eine Klassenstruktur programmiert. Zudem wurde das MVC-Modell genutzt. Da diese bereits implementierte Klassenstrucktur inklusive des MVC-Modells sinnvoll ist, wurde im Laufe des vierten Semester daran nichts geändert. Das Projektteam hat ebenso erkannt, dass der BibFileGenerator eine Service-orientierte Architektur hat. Durch die Quellensuche mit Hilfe der Internationale Standartbuchnummer (ISBN) oder der Digitaler Objektbezeichner (engl.: Digital Object Identifier, DOI) wird beispielsweise ein Service von Google genutzt, welcher unter Umständen ebenfalls weitere Services nutzt.

Bei dem Oberflächendesign wurde sich an der Norm DIN EN ISO 9241 Teil 110 orientiert. So wurde sehr auf die Selbstbeschreibungsfähigkeit geachtet. In jedem Fenster und jeder View des BibFileGenerators ist ersichtlich was zu tun ist, dass ein Nutzer zu seinem Ziel gelangt. Außerdem war die Fehlertoleranz ein wichtiges Thema, indem z.B. statt eines Fehlers eine Meldung gezeigt wird, wenn man einen neuen Eintrag hinzufügen möchte und dabei nicht alle Pflichtfelder dieses Quellentyps ausgefüllt wurden. Bei dem Entwurf dieser Meldungen wurden die Fähigkeit der Nutzer beachtet. So wird nur eine kurze Meldung gezeigt, die dennoch genug Information enthält, damit ein noch nicht so erfahrener Nutzer weiß, was er tun kann. Schließlich wurde auch auf die Attribute einer benutzerfreundlichen Software geachtet. So

stehen die Erlernbarkeit und die Stabilität des Produktes im Vordergrund. Der Umgang mit dem BibFileGenerator ist leicht zu erlernen, da er sich an ähnlichen Programmen orientiert und nur wichtige Features bietet. Zudem wird bei einer falschen Eingabe eine Meldung gezeigt, wodurch die Stabilität des Programmes gesichtert ist.

IV. Codierung

Die Codierungsphase begann mit der Auswahl einer Programmiersprache und der passenden IDE[1]. Die Wahl fiel auf Java SE als Programmiersprache und Eclipse als Entwicklungsumgebung, da das Team dort die größte gemeinsame Schnittstelle besaß und alle Teammitglieder bereits mindestens einen Einstieg in Java gemacht hatten. Zusätztlich bietet Java den Vorteil, dass standardmäßig plattformunabhängig entwickelt wird. So konnte parallel unter Windows und MacOS gearbeitet werden. Ein weiterer Vorteil von Java als Entwicklungssprache war, dass wegen der weit verbreiteten Verwendung von Java zahlreiche Tools und Bibliotheken zur freien Nutzung vorhanden sind.

Zur Versionierung hat sich das Team entschlossen, das Projekt auf github.com und mit dem Eclipse Plugin EGit[9] zu verwalten. Weitere Plugins für Eclipse, die zur Entwicklung verwendet wurden sind: Project Lombok[10] (automatische Generierung von Getter/Setter-Methoden und Konstruktoren), JSON-simple[11] (Parser für JSON Dokumente in Java) und ObjectAid[12] (automatische Generierung von Klassendiagrammen in Eclipse). Verwaltet wurden die Tools über den Eclipse Marketplace und mittels Maven[13], einem Projektmanagement-Tool, mit dem Abhängigkeiten von Projekten verwaltet werden können.

Die Plattform github.com hatte im Entwicklungsprozess mehrere zentrale Funktionen. Neben der bereits erwähnten Funktion als Versionskontrolle, hat das Team das Ticket System und Projekt-Board von GitHub verwendet. Angelehnt an das Scrum Projektmanagement Framework, wurde das Board in “Backlog“, “In Progress“, “Deferred“ und “Done“ unterteilt. Nach dem Kanban Verfahren wurden Tickets an die Entwickler vergeben. Dabei wurden die einzelnen Stärken und Interessen der einzelnen Teammitglieder nach Möglichkeit berücksichtigt. So haben sich manche fast ausschließlich mit der Programmierung des Backends beschäftigt, während andere die View erstellt haben. Dabei konnten beide ”Fraktionen”, dank git branching, parallel und ungestört arbeiten. Es wurden jeweils Git-Branches für Frontend, Backend und gegebenenfalls für größere Aufgaben erstellt, damit die Entwicklung verschiedener Funktionen sich nicht gegenseitig behindert. Im master-Branch wurden alle Teilentwicklungen zusammengefasst.

Zum Designen der Oberfläche wurde sich im Team für die Klassenbibliothek JavaFX[14] entschieden. Zusammen mit dem SceneBuilder[15] lassen sich Oberflächen per Drag-and-Drop erstellen. Zudem lässt sich zu jeder View ein Controller definieren, wodurch das Model-View-Control(MVC)-Modell unterstützt wird. Das MVC Entwurfsmuster wurde über den gesamten Entwicklungsprozess eingehalten. Jeder View, die mit dem SceneBuilder erstellt wurde, wurde eine Controller-Klasse zugeordnet. Die Controller reagieren auf Eingaben des Benutzers auf der Oberfläche und verändern dann Daten im Model und aktualisieren die Anzeige der Views. Die verschiedenen Controllerklassen leiten alle von einem Basiscontroller ab. Codewiederholungen, durch identische Attribute oder ähnliche Methoden, wurden durch konsequente Vererbung vermieden.

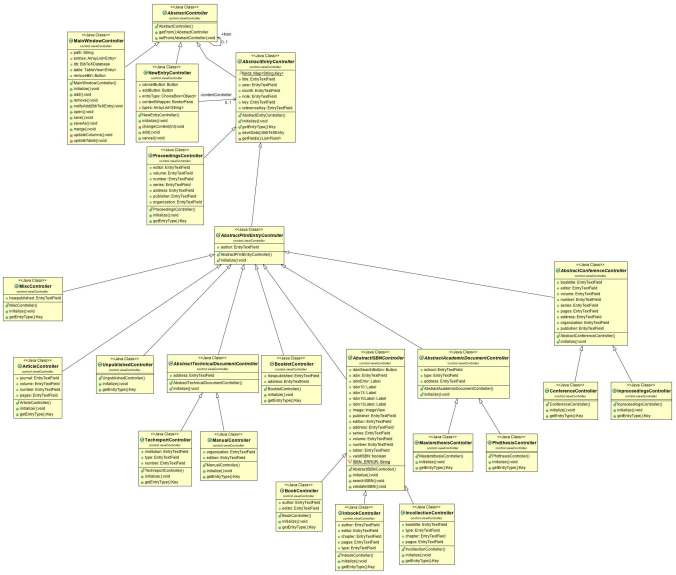

Abbildung 2. Klassendiagramm der ViewController-Klassen

Die Suche nach ISBNs und das automatische Finden der bibliographischen Informationen über das zugehörige Buch, wurde im Vorfeld von den Stakeholdern als Hauptaufgabe der BibFileGenerator Software definiert. Der Umsetzung dieser Funktion wurde dementsprechend viel Zeit gewidmet. Zunächst wurden verschiedene Quellen für Informationen zu Büchern und anderen gedruckten Werken verglichen. In Frage kamen unter anderem die ISBN Datenbank ISBNdb[16] und die Amazon Web Services (AWS) Product Advertising API[17]. Beide APIs mussten verworfen werden, da erstgenannte inkonsistente Ergebnisse lieferte und die AWS API zu komplex und nur mit Verschlüsselter Übertragung arbeitet. Im Endeffekt hat sich das Team für die Nutzung der Google Books API v1[18] entschieden. Die Nutzung ist im kleinen Rahmen kostenlos und Anfragen liefern zufriedenstellende, konsistente Ergebnisse. Da der Dienst von Google betrieben wird, kann davon ausgegangen werden, dass die Datenbanken gut gepflegt sind und der Dienst dauerhaft verfügbar ist. Zudem können Anfragen an die Datenbank mittels HTTP GET Befehl formuliert werden. Die API liefert eine Antwort im JSON Format zurück.

Für die Entwicklung des ISBN Services wurden verschiedene Eingabevalidatoren entwickelt, die nur bei korrekt einge-

[1] integrated development environment - (engl.) Integrierte Entwicklungsumgebung

gebener ISBN eine Anfrage an die API auslösen. Die Antwort der Google Datenbank (im JSON Format) musste möglichst schnell und performant ausgewertet werden. Um Zeit zu sparen wurde hierfür die Bibliothek json-simple[11] verwendet, welche via Maven eingebunden wurde. Json-simple ging in Tests verschiedener JSON Parser als Sieger in puncto Schnelligkeit bei kleinen und großen JSON-Dateien hervor[19]. Später wurde nach dem Schema der ISBN Suche eine Suche für DOI[2] Nummern entwickelt. Die Eingaben werden hier ebenfalls vor dem Absenden einer Anfrage an den Daten-Provider validiert. Als Datenquelle dient die CrossRef REST API[20]. Diese ist ebenfalls mittels HTTP GET Befehlen zu erreichen und liefert, genau wie die Google Books API, ihre Ergebnisse in JSON. Die Implementierung konnte stark von der bereits bestehenden ISBN Suche abgeleitet werden.

Der Export der eingegebenen und aus dem Internet gesammelten Daten und der Import von vorhandenen .bib Dateien sollte ebenfalls durch eine Bibliothek unterstützt werden. Die Wahl fiel auf die in Java programmierte Bibliothek jBibTeX[21], welche auf GitHub frei verfügbar ist. Genutzt wurde jedoch eine leicht modifiziert Kopie des Original-Repositories[22]. Die Bibliothek übernimmt das lesen (parsen) von .bib Dateien, das Verwalten mehrerer Einträge in einer .bib Datei sowie die Ausgabe aller Einträge in einer neuen Datei.

Das Thema Dokumentation und Codekommentierung, war dem Entwicklungsteam von Beginn an sehr wichtig. Jede Klasse und jede Methode hat einen JavaDoc Kommentar in englischer Sprache erhalten. Bei Bedarf kann so eine komplette Dokumentation des Codes mit Eclipse generiert werden. Bei der Erstellung von Variablen, Methoden, Klassen und Paketen wurde darauf geachtet sprechende Namen zu vergeben und einer logische Kapselung der Softwarefunktionen in Pakete zu folgen. Dank der automatischen Code Formatierung von Eclipse sind alle Dateien gut lesbar und auch für Menschen außerhalb des Entwicklungsteams schnell nachzuvollziehen.

Zu Beginn der Entwicklung begann das Projekt als privates und damit nicht öffentlich sichtbares GitHub Repository. Bereits während der Entwicklungsphase entschied sich das Team jedoch dazu die Software frei, das heißt als Open Source Anwendung, zur Verfügung zu stellen.[3]

V. Test

Ein Großteil der Tests lief parallel zur Entwicklung. Hier wurden einzelne Funktionen und Teilprogramme einzeln getestet und ihre Integration ins Gesamtsystem erneut auf Funktionalität überprüft. Durch die Arbeitsverteilung in Front- und Backend fand im weitesten Sinne eine Top-Down-Integration statt. Allerdings wurde auf unnötige Simulationen verzichtet. Für viele spätere Funktionen wurden bereits Buttons, Schnittstellen oder auch nur Platzhalter bereitgestellt, aber noch nicht mit Funktionen oder Simulationen versehen. Auf die Weise konnten neue, nachträgliche Funktionen des Backend einfach ins Frontend integriert und getestet werden, ohne dort jedes Mal Anpassungen vornehmen zu müssen.

Insgesamt war die Aufgabe des Testens natürlich durch den Hauptentwickler abgedeckt. Dadurch, dass die anderen Teammitglieder aber parallel kleine Veränderungen und Anpassungen vorgenommen haben und diese auch selbst auf Funktionalität überprüften, konnten deutlich mehr kleinere Fehler noch vor der Integration ermittelt werden. Um nur ein einfaches Beispiel zu erwähnen, wurden teilweise bei Eingabefeldern Label oder ID vertauscht, so dass hier die eingegebenen Daten an falsche Stellen im Bib-Datei hinterlegt worden wären, wenn dies nicht direkt beim überarbeiten des Quellcodes aufgefallen wäre.

Speziell für die Validierung von ISBNs und zum Testen des Verhaltens bei verschiedenen oder falschen Eingaben solcher, wurde eine extra JUnit Test entwickelt, die eine entsprechende Auswahl an Fällen und Varianten der Falscheingaben, sowie auch unterschiedlicher Notationen aufgreift. Hierzu wurden entsprechend Äquivalenzklassen gebildet, um möglichst alle Unterschiede abzuarbeiten und gleichzeitig Wiederholungen zu vermeiden. Entscheidend bei diesem Test ist das Verhalten und entsprechende Rückgaben des Programmes.

Tests in Bezug auf die nichtfunktionalen Eigenschaften beschränkten sich auf ein Minimum. So ist der Aspekt der Sicherheit bei diesem Programm nicht äußerst relevant. Da beim Aufruf externer Daten diese nicht ausgeführt, sondern lediglich deren Inhalte gelesen und interpretiert werden, können von außen keine Eingriffe in das eigentliche Programm vorgenommen werden. Dadurch, dass nicht interpretierbare Daten verworfen werden und auch bei korrekten Daten lediglich deren Inhalte kopiert werden, ist ausreichend für Sicherheit gesorgt.

Auch im Bereich der Benutzbarkeit waren nur bedingt Tests notwendig. Da wie oben beschrieben das Frontend weitestgehend unabhängig vom Backend entwickelt wurde, sind hier nachträglich keine Veränderungen vorgenommen worden, die an der ursprünglichen Benutzbarkeit etwas verändert hätten und somit Wiederholungen der ersten Tests erfordert hätten. Überhaupt beliefen sich die Tests hier auf die intuitive Nutzbarkeit der Oberfläche durch den Nutzer. Hauptsächlich die Kollegen im Team und im kleinen Rahmen auch durch vereinzelte Tests zufällig ausgewählter Personen, die mit der Entwicklung nichts zu tun hatten und somit auch mit der Funktionsweise nicht vertraut waren.

Da die Leistungsfähigkeit größtenteils von der Internetanbindung des ausführenden Betriebssystems abhängig ist, waren auch hier die Tests auf ein Minimum beschränkt. Hier wurden lediglich kleine Beobachtungen bei der Integration der API's gemacht, welche genutzt wurden, um Online-Datenbanken mit ISBNs abzufragen.

Zusätzlich wird mit Beendigung der Entwicklungsphase ein ausgiebiger Test aller Funktionalitäten stattfinden, um letzte Fehler zu vermeiden und die Funktionsfähigkeit zu gewährleisten.

[2]Digital Object Identifier - Identifizierungsnummer für digitale Objekte, beispielsweise Artikel aus einer Fachzeitschrift

[3]Das Repository ist hier zu finden: https://github.com/mboskamp/BibFileGenerator.git

VI. Wartung

Während der Entwicklung wurde entschieden die BibFile-Generator Software als ein Open Source Projekt bereitzustellen. Dementsprechend ist das Programm kostenfrei erhältlich. Außerdem ist der Quellcode für jeden frei zugänglich. Werden in der offiziellen Version Fehler gefunden, können diese gemeldet werden. Auftretende Fehler können in zukünftigen Versionen entfernt werden, jedoch ist der Nutzer für die Nutzung zukünftiger Updates selber verantwortlich. Er muss selbst entscheiden ob er diese nutzen und installieren will. Durch den frei zugänglichen und dementsprechend frei zu bearbeitenden Code, ist jeder Nutzer selbst für die Wartung seiner Version zuständig. Das Projekt befindet sich in GitHub und kann somit durch andere bearbeitet werden. So können alle gefundenen Fehler auch direkt vom Nutzer behoben werden, falls das nötige Wissen vorhanden ist. Die Beurteilung der Commits wird regelmäßig durchgeführt. Dementsprechend helfen alle Nutzer bei der Wartung mit.

VII. Ergebnisdiskussion

Zum aktuellen Zeitpunkt wurden die gesteckten Ziele weitestgehend erreicht. Dabei sind sowohl die Ziele zur Benutzbarkeit, als auch die zum Funktionsumfang gemeint. Bei der Benutzeroberfläche wurden hierbei Oberfläche und Bedienelemente so ausgelegt, dass die Grundsätze nach Sommerville [7] realisiert wurden. In geringem Umfang konnten dabei auch verschiedene Möglichkeiten eingebunden werden, welche die gleichen Optionen ansteuern. So wurde erreicht, dass ein größerer Umfang verschiedener Nutzer, nach eigenen Gewohnheiten und Erfahrungen, die Optionen finden und nutzen können. Gleichwohl konnten unerwartete Systemverhalten und dadurch auftretende Überraschungen vermieden und der Umfang an möglichen Fehlermeldungen reduziert werden. Jegliche Meldungen (Fehler, Warnungen und Hinweise) sind in einfachem Deutsch gehalten, dadurch für den Nutzer gut verständlich und entsprechend mit der aktuellen Situation verknüpfbar. Durch Nutzung der im Betriebssystem eingestellten Farbschemen, ist die Oberfläche hier auch individuell für jeden Nutzer in einem für ihn angenehmen Erscheinen dargestellt.

Das Hauptziel, Literaturverzeichnisse zu erstellen und als Bib-Datei zu speichern, sowie viele für das Projekt ebenso wichtige Nebenziele, konnten realisiert werden. Die Google Books API war hier ein wichtiger Bestandteil, um mit Hilfe von einer ISBN-Suche Daten wie Titel und Autor automatisiert nutzen zu können. Später wurde sogar eine Möglichkeit implementiert, mit Hilfe eines weiteren Services von Google, nach DOI zu suchen und die Daten genauso übernehmen zu können. Dadurch wurde erreicht, dass auch für wissenschaftliche Artikel Daten automatisiert zurechtgelegt werden und so neben den Büchern beide am häufigsten genutzten Quellen auf einfachem und schnellem Weg in die Bib-Datei aufgenommen werden können. Durch die Funktion die eigenen Dateien später erneut laden und verändern zu können, wurde die Nutzbarkeit für den Anwender deutlich erhöht.

VIII. Zusammenfassung und Ausblick

Die eigens festgelegten Ziele konnten also sowohl im Frontend, als auch im Backend weitestgehend erreicht werden. Letztendlich führten kleinere Probleme dazu, dass der Zeitrahmen für alle über die eigentlichen Ziele hinausgehenden Wünsche nicht ausreichte. So gab es leider noch keinen Erfolg bei der Umsetzung des Zusammenfügens verschiedener Bib-Dateien. Weiterhin sind Fehler beim Anzeigen der Quelleninformationen aufgetreten, weshalb nachträglich keine Änderungen an ihnen möglich sind. Dadurch entstand allerdings eine Idee, diese auch an anderer Stelle verändern zu können. Möglichkeiten dies zu realisieren wurden bereits gefunden, sodass dies in naher Zukunft noch umgesetzt werden könnte. Darüber hinaus steht noch aus, ein Feature hinzuzufügen, die dem Nutzer erlaubt, die angezeigten Spalten auszuwählen und darüber hinaus deren Reihenfolge zu verändern. Abschließend lässt sich feststellen, dass sowohl die Zusammenarbeit gut funktioniert hat als auch, dass die wichtigsten Ziele erfolgreich umgesetzt werden konnten. Der geringe zur Verfügung stehende Zeitrahmen konnte für mehr als das Nötigste verwendet werden und wurde entsprechend gut ausgenutzt. Dennoch sind Ideen und Wünsche im Voranschreiten der Umsetzung entstanden, die bei leicht größerem Zeitrahmen noch umgesetzt werden könnten und so sowohl den Funktionsumfang des Programms, als auch die Zufriedenheit der Entwickler erhöhen könnten.

Literatur

[1] "Latex." [Online]. Available: http://www.latex-project.org/contact/

[2] "Latex." [Online]. Available: https://de.wikipedia.org/wiki/LaTeX

[3] P. D. D. M. Díaz, "Software engineering i - techniques eliciting requirements," Präsentation, September 2016.

[4] ——, "Software engineering ii - phase entwurf (ii)," Präsentation, Januar 2017.

[5] ——, "Software engineering ii - phase entwurf (i)," Präsentation, Januar 2017.

[6] ——, "Software engineering ii - entwurf von bedienoberflächen (i)," Präsentation, Januar 2017.

[7] I. Sommerville, *Software Engineering*. Pearson Studium, 2007, vol. 8.

[8] P. D. D. M. Díaz, "Software engineering ii - entwurf von bedienoberflächen (ii)," Präsentation, Januar 2017.

[9] "Egit," Webseite, März 2017, http://www.eclipse.org/egit/.

[10] "Project lombok," Webseite, März 2017, https://projectlombok.org/.

[11] Apache, "json-simple at githib.com," Webseite, März 2017, https://github.com/fangyidong/json-simple.

[12] "Objectaid uml explorer," Webseite, März 2017, http://www.objectaid.com/.

[13] Apache, "Apache maven," Webseite, März 2017, https://maven.apache.org/.

[14] Oracle, "Java platform, standard edition (java se) 8 - client technologies," Webseite, März

2017, http://docs.oracle.com/javase/8/javase-clienttechnologies.htm.
[15] ——, “Javafx scene builder - a visual layout tool for javafx applications,” Webseite, März 2017, http://www.oracle.com/technetwork/java/javase/downloads/javafxscenebuilder-info-2157684.html.
[16] ISBNdb, “Isbndb,” Webseite, März 2017, http://isbndb.com/.
[17] A. W. Services, “Amazon advertising api,” Webseite, März 2017, https://partnernet.amazon.de/gp/advertising/api/detail/main.html.
[18] Google, “Google books api v1,” Webseite, März 2017, https://developers.google.com/books/docs/overview.
[19] J. Dreyfuss, “The ultimate json library: Json.simple vs gson vs jackson vs jsonp,” Webseite, März 2017, http://blog.takipi.com/the-ultimate-json-library-json-simple-vs-gson-vs-jackson-vs-json/.
[20] CrossRef, “Crossref rest api,” Webseite, März 2017, https://github.com/CrossRef/rest-api-doc.
[21] jbibtex, “Java bibtex api,” Webseite, März 2017, https://github.com/jbibtex/jbibtex.
[22] jbibtex and mboskamp, “Fork der java bibtex api (leicht verändert),” Webseite, März 2017, https://github.com/mboskamp/jbibtex.