hochschule mannheim

Fakultät Informationstechnik
Studiengang Technische Informatik

Bachelorarbeit

Arbeitstitel: Anforderungen und Testen

Florian Brunotte

Abschlussarbeit

zur Erlangung des akademischen Grades

Bachelor of Science

Vorgelegt von	Florian Brunotte
am	30.01.2021
Referent	Prof. Dr. Ing. Mark Hastenteufel
Korreferent	Prof. Dr. Martin Damm

Schriftliche Versicherung laut Studien- und Prüfungsordnung

Hiermit erkläre ich, dass ich die vorliegende Arbeit selbstständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel benutzt habe.

Mannheim, 30.01.2021 _____

Florian Brunotte

Zusammenfassung

Ziel dieser Bachelorarbeit war die Erstellung einer leichtgewichtigen Software mit der Requirements und Testcases aufgenommen werden können und durch Testruns validiert werden. (Unterschied Validierung und Verifikation?) Eingesetzt wird diese Software im Rahmen von kleinen Semesterprojekten an der Hochschule Mannheim. Geschrieben wurde sie in als Webapplikation in Python und Django während die Visualiisierung und das User Interface durch HTML und CSS erstellt wurde.

Inhaltsverzeichnis

Kapitel 1

Motivation - Warum wurde diese Software geschrieben?

Es ist vorzuziehen, dass...

Motivation: Die Zielgruppe für diese Arbeit und für diese Software sind die Hochschulangehörigen, die innerhalb des Studiums Anforderungen und Tests verwalten müssen.

Es soll eine leichtgewichtiges Softwaretool entwickelt werden zum Anforderungs- und Testmanagement. Dabei sollte man sich auf die Kernfunktionalitäten beschränken, damit man sich auf das Wesentliche konzentrieren kann, dem Schreiben von Anforderungen und dem Ausführen von Tests. Andere Softwaretools, die am Markt verfügbar sind, sind für den Einsatz in der Lehre nicht geeignet, da sie zu komplex sind und auch Geld kosten können. Sollte man nicht bezahlen, werden bestimmte Funktionen gesperrt. Insgesamt sind sie zu schwergewichtig und man verliert das eben beschriebene wesentliche aus dem Blick. Eine intuitive und simple Software, wie „Anforderungen und Testen (WIP)" soll hierbei Abhilfe schaffen.

Hier werden dann noch manche Tools aufgezählt. Vielleicht mit Bilder die Limitierung und die Überfülle zeigen.

Kapitel 2

Anforderungsanalyse - Welche Anforderungen hat das Projekt?

Zuerst hat Herr Hastenteufel ein Dokument mit Anforderungen gesendet. Nach Absprache per Mail wurden noch folgende Sachen geklärt:

- Die Kategorie für die Requirements werden erstmal als einfache Kategorien wie 1, 2, 3 realisiert. Später vielleicht Auswahl aus der Art der Requirement: Stakeholderanforderung, aber dazu nochmal in SOE reingucken
- Die Namen Requirement, Testcase und Testrun werden benutzt, um alles einheitlich zu lassen
- Da der Admin und der Professor als Nutzer sehr ähnlich sind, wurden die beiden Rollen mit ihren Anforderungen zusammengelegt als Professor

Ein Klassendiagramm und ein Zustandsdiagramm wurden erstmal nicht gemacht, da sie nicht für sinnvoll gehalten wurden. Aber das wurde nochmal nachgefragt.

Es wurden ein Use Case Diagramm skizziert, mehrere Activity Diagramme, die die Use Cases verfeinern skizziert und die UIs wurden auch skizziert. Daneben wurde auch das ER-Modell schon in Draw.io gezeichnet und die Tabellen daraus wurden transformiert. Somit hat man für den Prototyp bereits ein Datenmodell mit dem man testen kann. Die Diagramme werden sich wahrscheinlich noch ändern im Verlauf des Projekts, aber für den Anfang sind sie in den Abbildungen 1, 2, 3... zu sehen. Das war der Start des Projekts.

Es wurden ebenfalls erste Skizzen zu den UIs vollzogen und abgesprochen mit dem Kunden, also Herrn Hastenteufel. Die Rückmeldungen waren:

> Documentation is like sex: when it is good, it is very, very good; and when it is bad, it is better than nothing.
>
> *Dick Brandon*

Kapitel 3

Implementierung - Wie erstellt man eine Django-Applikation?

mo: Warum sollte man Django verwenden ist in der Tabelle tab dargestellt: mo: Geschichte von Django und die Popularität mo: den generellen Aufbau und Ablauf eines HTTP Request kann man in der Abbildung abb sehen. In Django Applikationen werden die verschiedenen Aufgaben von verschiedenen Dateien erledigt. In URLS.py werden die Views zu den HTML Seiten gelinkt. Sie verknüpfen also die spezifischen URL Patterns mit den gewünschten View Funktionen. In den Views werden noch die HTML Seiten mit Daten versehen aus den Modellen, die man zuvor definiert hat. Eine View verarbeitet die HTTP requests und sendet basierend auf diesen die HTTP responses. Hierbei ist die render Funktion entscheidet. Models sind Python objects und stellen so Funktionen zur Verfügung zum erstellen, ändern und löschen der Daten, aber auch um die Daten in die Datenbank zu speichern mit Querys. Die Templates können dynamisch HTML Seiten erzeugen. Sie haben noch Platzhalter für die richtigen Inhalte. Es müssen aber keine HTML Dateien sein. Django nennt diese Methode Model View Template und das kann man vergleichen mit dem Model View Controller Architektur.

mo: Das Model View Controller Architektur kann auf folgenden Weg beschrieben werden... und mit der Abbildung abb auch erklärt werden.

mo: Die Kommunikation mit der Datenbank wird von Django erledigt, das heißt die beiden Sachen sind entkoppelt voneinander. Man kann so die Models einfach definieren und dann die Datenbank auswählen.

mo: Python unterstützt die Objektorientierte Programmierung bzw. ist eine objektorienritere Programmiersprache.

mo: Django hat von sich aus einen Development Webserver dabei, den man benutzen

kann um seine Anwendung zu testen.

mo: zur Instalation wurde in PyCharm ein neues Projekt angelegt und dann in dem virtual environment kann mit pip die verschiedenen Python Erweiterungen runtergeladen werden. Man muss sich hierbei entscheiden ob man es global installiert, dann kann man aber nur von jeder Sache nur eine Version gleichzeitig da sein. Bei einem virtual Environmnet kann man dann in einer eigenen Version verschiedene Versionen unterbringen. Die Sachen gelten dann auch nur in dem Environment. Die hier verwndeten Versionen von Python ist 3.7.2, von Django ist 3.1.2 und von PIP ist 1. Die verwendete Datenbank ist Postgres und die Version davon ist 3.7. In der Tabelle sind die verwendeten Versionen und Software mit Download Link angegeben. Virtualwrapper mit virtualvenv ist auch verwendet worden. In Pycharm wird bei einem neuen Projekt auch ein neues venv gemacht. Zum venv gibt es die verschidenen Befehle workon, deactivate, remove.

Mit PyCharm ein neues Projekt starten. Dabei wird ein neues virtual Enviroment gebildet in dem man alles installieren kann, das hat aber keine Auswirkungen auf andere Installationen, wie die Hauptinstallation von Python. Neuer Ordner wurde gebildet in dem Bachelorarbeit Ordner für den Prototyp. Als nächstes sollte man Django installieren. Im Terminal von PyCharm kann alles gemacht werden, wie Django installieren. Mit Pip install django kann man dann django installieren. Ebenfalls sollte hier bereits die Datenbank wie Postgres installiert werden. Nach anlegen des Projekts kann die Datenbank auch schon registriert werden. Wichtig hierbei sind das Passwort für den Benutzer und der Benutzername neben dem Namen der Datenbank.

Dann einfach das Tutorial machen von Django Documentation.

Django entstand in einem Nachrichtenumfeld als Einführung in Django

Man startet ein neues Django Project mit `django-admin startproject anforderungenundteste`, dabei sollte der Name nicht wie django oder test heißen, das könnte zu Problemen führen `cd ./anforderungenundtesten` um in den richtigen Ordner zu kommen, der die `manage.py` Datei enthält. Durch diesen Schritt wurde in dem ausgewählten Ordner ein Django Projekt erstellt und dazu notwendiger Code autogeneriert. In der Tabelle 3.1 können die bis jetzt generierten Dateien und Ordner gesehen werden.

über python manage.py runserver kann der Developerserver aufgerufen werden und man sieht, dass die Installation geklappt hat. Man kann hierbei auch die IP Adresse so ändern, dass alle Geräte im gleichen Netzwerk Zugriff haben. Weitergehend kann unter PyCharm die Run Configuration so angepasst werden, dass bei einem Run der runserver Befehl aufgerufen wird. Über STR C kann der Server wieder beendet werden.

Es gibt dabei einen Unterschied zwischen einem Projekt und einer App in Django,

Datei	Inhalt
/anforderungenundtesten	Der äußere Ordner ist ein Container für das Projekt, der Name ist nicht wichtig und kann beliebig unbenannt werden
manage.py	In dieser Datei sind die Command Line Befehle enthalten, wie zum Beispiel runserver
anforderungenundtesten/	der innere Ordner ist der eigentliche Python Package mit diesem Namen werden alle Dateien importiert, wie zum Beispiel anforderungenundtesten.urls

Tabelle 3.1: Die erstellten Dateien und Ordner nach dem django-admin startproject anforderungenundtesten Befehl

Datei	Inhalt
/aut	Der Ordner beinhaltet die App und die nachfolgenden Dateien
manage.py	In dieser Datei sind die Command Line Befehle enthalten, wie zum Beispiel runserver
anforderungenundtesten/	der innere Ordner ist der eigentliche Python Package mit diesem Namen werden alle Dateien importiert, wie zum Beispiel anforderungenundtesten.urls

Tabelle 3.2: Die erstellten Dateien und Ordner nach dem django-admin startproject anforderungenundtesten Befehl

so kann ein Projekt mehrere Apps enthalten und eine App kann in verschiedenen Projekten enthalten sein. Man sollte auf die Wiederverwendbarkeit achten. Eine App ist auch etwas, dass etwas macht, wie ein Blog oder eine Datenbank auslesen und Daten darstellen. Ein Projekt ist dann eine Sammlung von Apps und Einstellungen für eine explizite Webseite.

Über den Befehl `python manage.py startapp aut` wird eine neue App angelegt und auch hier wird zuerst wieder Code autogeneriert. Dieser ist in der Tabelle 3.2 dargestellt mit seinen Komponenten und den verschiedenen Funktionen.

Hierbei kam der Begriff Boiler Code auf, der Codeabschnitte beschreibt, die wenige Änderungen haben und an vielen Stellen auftauchen können. Beispiele sind in den Listings 1 und 2 dargestellt. Man sieht in dem 1. Lisitng ein Shebang für die Programmiersprache Python, damit wird dem Compiler klar das es sich hier um Python Code handelt. Im 2. Lisiting sieht man die Basis von einem HTML Dokument. Mit diesem Template können viele verschiedene Websites generiert werden, aber diese Tags sind notwendig.

Die App muss unter den installierten Apps angemeldet werden

Danach sollten die URLs definiert werden und zusammengeführt werden. Es gibt

eine url Datei für das Projekt und eine für die App. Von dem Projekt kann man auch die Grundseite umleiten lassen auf die Seite der App. Die verschiedenen Routen der Seiten sind in der Abbildung dargestellt. In dieser Datei kann man über den Path angeben welche View für welchen HTTP Request ausgeführt werden soll. Das soll also eine Verlinkung sein.

Die andere url Datei vom Hauptprogramm, also anforderungenundtesten, ist dafür zuständig den Hauptpfad zu deklarieren. In dieser Arbeit ist der Hauptpfad „aut“. Danach wird das url File von der App inkludiert mit `include()` Daneben wird auch die admin Seite angegeben.

In den Settings kann auch noch die TIMEZONE angepasst werden.

mo: die 2 anderen Einstellungen, die man beachten sollte sind der Secret Key und DEBUG. Mit dem Secret Key kann Django die Security Sachen machen, darum sollte man ihn geheim halten und schützen während der Produktionsphase. Mit dem DEBUG Keyowrd kann man mehr Informationen bei einem Fehler bekommen, wenn die App dann fertig ist, sollte er False sein.

Es müssen auch die Befehle `makemigrations` und`migrate` angewandt werden, da sie sämtliche Änderungen an den Datenbanken und den Modellen ausführen. Mit `migrate` werden alle notwendigen Datenbank Tabellen erstellt für die unter InstalledApps angegebenen Apps. In der Postgres Umgebung können diese Tabellen auch gesehen werden.

Zuletzt wird mit runserver der Development Server erstellt und zum Laufen gebracht unter der Adresse kann man dann lokal die App testen oder man kann auch für alle Geräte im gleichen Netz die App verfügbar machen. Das ist nützlich, um die Webseiten auf mobilen Geräten oder generell Geräten mit verschiedenen Displaygrößen zu testen.

Die Datenbank, in diesem Fall Postgres muss auch eingerichtet werden, dazu gibt es in settings.py die Einstellungen zu Databases. Diese müssen verändert werden, damit das Django-Projekt mit einer Postgres Datenbank funktioniert. Es müssen der Name und das Passwort des Benutzers zum Anmelden benutzt werden, nicht das Masterpassword.

Danach können die Modelle definiert werden. ORM von Python ist hierbei wichtig. Die Modelle werden aus den Tabellen, die wiederrum aus den ER-Modellen hergeleitet werden, erstellt. Bei Modellen hat Django das Dont Repeat Yourself Prinzip, welches besagt dass...

Die ER-Modelle sind in der Abbildung 1 dargestellt....weiter erklären. Die Modelle sind in Abbildung 1 dargestellt... auch weiter erklären. Zuerst werden einfache Modelle angelegt in Django mit Foreign Keys und Char Feldern. Der Primary Key ist

definiert so wie er als Standard definiert wäre, aber das wurde überschrieben, um den einzelnen Schlüsseln die richtigen Namen zu geben. Ansonsten ist der Primary Key einfach ein AutoField und hat Integer Werte die bei 1 anfangen. Die Foreign KEys sind auch damit OK, dass sie keinen Wert am Anfang haben.

Durch diese Beschreibung der Modelle kann Django die entsprechenden Datenbanken erstellen und für den Entwickler die API bereitstellen.

Nachdem man die Modelle geschrieben hat muss man das Modul noch für den Admin zur Bearbeitung freischalten bzw. registrieren. Der Admin oder Superuser muss auch erstellt werden. Unter installed Apps muss man dann aut noch eintragen.

Nachdem die Modelle erstellt/geschrieben wurden, kann man mit makemigrations und migrate die Tabellen/Relationen in die Datenbank einspielen. Mit makemigrations werden die migrations erstellt als Datei um noch mal drüber zu gucken mit den SQL Befehlen. Dabei konnte ich auch den Fehler mit dem Default DateTimeField ausbessern. Da die Migrationen nacheinander gemacht werden. Jetzt kann man bereits über den Admin die Daten einpflegen. Dazu ist zu sagen, dass das bis jetzt nur der Admin kann und die Admin Seite zwar Funktional ist, aber im Sinne von UI/Aussehen noch verbessert werden kann. Mit dem Befehl `python manage.py sqlmigrate 0001` kann man die SQL Anweisungen zu dem entsprechenden Migrate sehen in der Konsole, z.b. zu 0001 initial. Der Befehl zeigt nur die Befehle an, die gemacht werden ohne diese tatsächlich zu machen. Mit dem Befehl `check` kann auch vorher gecheckt werden, ob es Probleme gibt mit den Migrationen ohne das diese schon gemacht werden.

Die Schritte beim Arbeiten mit der Datenbank und den Modellen ist wie folgt:

- Die Modelle anpassen in models.py
- mit makemigrations die migrations für die Änderungen erstellen
- mit migrate die migrations anwenden
- optional: check um Fehler zu finden
- optional: sqlmigrations um die SQL Befehle zu sehen

Der Admin muss zuerst angelegt werden über create superuser. Und die Modelle müssen noch angemeldet werden, damit der Admin diese bearbeiten kann. Dann kann man über die Admin Seite bereits Daten einfügen, ändern und löschen. Im nächsten Schritt wird die Admin Seite aber angepasst, damit sie besser aussieht und auch neue Funktionalitäten hat.

Mit dem runserver Befehl kann dann der Development Server aufgerufen werden, der die Seiten, die unter den Urls und den Views definiert sind anzeigt. Dazu kommt

dann auch noch die admin Seite unter der die Modelle mit ersten Daten versorgt werden können und auch angezeigt werden können

mo: Mit der RedirectView kann man die GrundURL auf eine App lenken. Das macht hier Sinn, da es nur eine App geben wird aut. mo: Damit die Seite auch noch static Files unterstützt, wie JS und CSS kann man auch noch eine Zeile hinzufügen. mo: makemigration und migrate sind die wichtigen Befehle für die Datenbank

mo: Django übernimmt die Kommunikation mit der Datenbank, darum muss man bei der Definition der Models nicht darauf achten.

Darunter fällt auch das JOIN von verschiedenen Tabellen, wie die einzelnen Tabellen miteinander verbunden wurde, kann man in den Abbildungen 1 sehen. Mit so Pfeilen zwischen den Schlüsseln und so

Das UI wird aber später kommen, da man erstmal einen Prototyp haben möchte der die Grundlegenden Funktionen einer Datenbank hat: Daten eintragen, löschen, ändern und anzeigen.

Das erste UI was man auch klicken kann wurde in Axure erstellt. Diese Dateien konnte man als HTML exportieren und auch CSS und Javascript war dabei. Der nächste große Schritt ist jetzt diese Dateien in das Django Projekt einzubinden.

Bei der Admin Seite werden nur die Modelle angezeigt, die unter admin.py registriert sind. Darum müssen alle Modelle zuerst registriert werden

Im nächsten Schritt werden die Views kreiert, die man braucht. Django vergleicht welche View angezeigt werden soll durch die URL, darum mussten die Views auch erst verknüpft werden um jetzt benutzt werden zu können. Bei diesen Views kann man auch aus der Anfrage Variablen abfangen. Das heißt man kann zusätzliche Variablen angeben oder zum Beispiel ein Requirement im Detail angucken, wenn die entsprechende ID angegeben wurde in der URL. So kann man mit `<int:>` etwas abfangen Nachdem auf diesem Weg erste einfache Views geschrieben wurden, kann man jetzt das auf dem Weg machen, wie es auch in Django vorgesehen ist. Man erstellt templates, also html Dateien und tut sie in den entsprechenden Ordner. Der Pfad ist so gewählt, dass die templates wieder den Apps zugeordnet werden können. In den Templates wird dann auch Django Code verwendet mit den Klammern und den Befehlen wie if und so.

Zum einen muss man die richtigen Befehle benutzen um in den Views die richtigen Werte der Models zu bekommen. Das ist eine Qual. Danach sollte man die Views Sachen übergeben an das HTML und in die entsprechenden Felder die Sachen ausgeben. Dabei wird bei mir bis jetzt eine Liste mit Dictionarys übergeben. Das muss dann entsprechend mit 2 For Schleifen ausgegeben werden. Dann kann man so die Daten die bis jetzt in der Datenbank sind ausgeben in der HTML Datei, das war echt

ein langer Weg.

Um die richtigen Werte zu bekommen hat man zuerst die Elemente gefiltert nach dem richtigen Projekt. Das Projekt sollte später von irgendwo herkommen, wie zum Beispiel automatisch und unsichtbar für den Nutzer. Die Nutzer sind ja sowieso eingeteilt in die Gruppen und gehören nur zu einem Projekt. Danach werden alle Elemente dazu geholt, damit man auch auf die Attribute zugreifen kann wie Name oder so. Hier im Beispiel wird einfach alles ausgegeben. Mit den Forms die später kommen sollen kann man so vielleicht dann noch Sachen Filtern oder so. Damit sollte man dann auch endlich den anderen Weg beschreiten könne, dass man die Daten in die Datenbank schreibt ohne die Admin Seite benutzen zu müssen. Es ist bei alldem sehr wichtig auch die richtige view in den URLS verknüpft zu haben.

Mit der render Funktion kann man eine Abkürzung machen. Man muss dann nicht mehr zuerst das Template laden, sondern kann es sofort in der render Funktion angeben.

Man kann den Apps in den url EInstellungen auch einen Namen geben, das ist vor allem nützlich, wenn man mehrere Apps hat und man aber in den Templates auf eine bestimmte zugreifen möchte. In diesem Beispiel ist der appname dann aut und man kann so auf die views zugreifen mit aut:requirement. Auf diese Weise können jetzt alle Links zu den verschiedenen Django Templates, also HTML Dateien geschreiben werden und man kann die Seite bereits navigieren.

Das Dashboard und die entsprechenden Seiten sind in der Abbildung abb angezeigt.

Die HTML Siten wurden mit einer Testversion von Axure erstellt. Mit der Seite kann man HTML Seiten erstellen und bereits ausprobieren. Bei der Integration muss daraug geachtet werden, dass die ganzen Dateien, also auch die CSS Dateien in den richtigen Ordnern sind. Die Links in den Dateien müssen noch angepasst werden an die static Variable. Die ganzen ressourcen, die mit src anfangen und die href müssen jetzt noch /static/ davor stehen haben, da die Sachen in diesem Ordner zu finden sind.

Kapitel 4

LaTeX-Sachen

4.1 Latex Sachen als Section

Die Ziele des Templates sind wie folgt:

- Beispiele der typischen Verwendung von LaTeX und dessen Erweiterungen geben, die viele im Rahmen von Abschlussarbeiten üblichen Anforderungen abdeckt.
- Nahe am LaTeX-Standard halten mit wenigen weit verbreiteten Erweiterungen, um problemlosen Einsatz und Erweiterbarkeit sicher zu stellen.
- Die Einhaltung der Formalien an der Hochschule Mannheim in der Fakultät Informationstechnik vereinfachen.

~~durchgestrichen~~ angezeigt.

Nehmen Sie Kapitel 1 nicht mit dazu. Schreiben Sie Inhalte und keine Leerphrasen. Verwenden Sie nicht das „nächste“ oder „folgende“ Kapitel sondern immer als Zahl das wievielte Kapitel. Verlassen Sie sich auf LaTeX und nummerieren Sie nie selbst sondern referenzieren Sie. Jedes Kapitel außer das Erste muss vorkommen. In Kapitel 4 führen wir in die LaTeX-Umgebung kurz ein und geben eine Übersicht über die Tools, die notwendig sind ein Dokument zu erstellen. In Kapitel **??** stellen wir das Layout sowie einige Idiome zum Textsatz mit LaTeX vor. In Kapitel **??** besprechen wir das Einbinden und Erstellen von Fließobjekten wie Bilder, Tabellen und Listings. Hinweise zum Schreibstil, mathematischem Formelsatz und zur Literatur sind in Kapitel **??** gesammelt. Abschließend fassen wir in Kapitel 5 die Vorteile und Features, hervorzuheben sind die gute Qualität und Satz, von LaTeX für Ihre Abschlussarbeit noch einmal zusammen.

Plattform	LaTeX-Distribution	Editor
Linux/Unix	TeX Live	Texmaker, Emacs
MacOSX	TeX Live, MacTex	Texmaker, TeXShop
Windows	TeX Live, MiKTeX	Texmaker, TeXstudio

Tabelle 4.1: LaTeX-Distributionen und Editor je Plattform

in Tabelle 4.1.

(zum Beispiel `thesis.tex`)

(Times New Roman) When Apollo Mission Astronaut Neil Armstrong first walked on the moon, he not only gave his famous "one small step for man, one giant leap for mankind" statement but followed it by several remarks, usual communication traffic between him, the other astronauts and Mission Control. Just before he re-entered the lander, however, he made this remark *Good luck Mr. Gorsky*.

(Helvetica) Many people at NASA thought it was a casual remark concerning some rival Soviet Cosmonaut. However, upon checking, there was no Gorsky in either the Russian or American space programs. Over the years many people questioned Armstrong as to what the *Good luck Mr. Gorsky* statement meant, but Armstrong always just smiled. On July 5, 1995 in Tampa Bay FL, while answering questions following a speech, a reporter brought up the 26 year old question to Armstrong. This time he finally responded. Mr. Gorsky had finally died and so Neil Armstrong felt he could answer the question.

(Palatino) When he was a kid, he was playing baseball with a friend in the backyard. His friend hit a fly ball, which landed in the front of his neighbor's bedroom windows. His neighbors were Mr. & Mrs. Gorsky. As he leaned down to pick up the ball, young Armstrong heard Mrs. Gorsky shouting at Mr. Gorsky. *Oral sex! You want oral sex?! You'll get oral sex when the kid next door walks on the moon!*

Machen Sie Fußnoten[1] immer ohne einleitendes Leerzeichen und innerhalb des Satzes, also nie nach einem Punkt. Fußnoten sind ganze Sätze mit Satzzeichen. Fußnoten sind Inhalte, die nicht für das Verständnis notwendig sind[2]. Juristen verwenden Fußnoten zur Quellenangabe. Wir sind keine Juristen und distanzieren uns (nicht nur) von dieser Praxis deutlich. Setzen Sie Fußnoten sehr sehr sparsam ein.

Referenzieren Sie innerhalb des Dokuments, zum Beispiel auf das Kapitel **??** in dem es unter anderem um Bilder geht und das auf Seite **??** los geht, mit `\ref` (meistens) oder `pageref` (sehr selten). Verwenden Sie vor dem Befehl zum Referenzieren immer ein ~. Das ist ein nicht umbrechbares Leerzeichen und ~~Kapitel~~
~~1~~, also der Zeilenumbruch vor der Nummerierung, wird vermieden.

~~Es macht keinen Sinn aus irgendwelchen Gründen~~

[1] Das ist eine Fußnote.

[2] Fußnoten haben übrigens nichts mit Noten oder Musik zu tun.

~~erscheinen sie noch so sinnvoll~~
~~Zeilenbrüche im Fließtext einzuführen.~~
~~Sie~~
~~wollen eigentlich etwas anderes.~~

Sie sollten Abkürzungen (AKÜ) bei ersten Vorkommen definieren. Schreiben Sie das Wort zuerst aus und dann die Abkürzung in Klammern. Danach können Sie die AKÜ verwenden. Meistens sollten Sie jedoch auf Abkürzungen verzichten. Schreiben Sie lieber *beispielsweise, zum Beispiel, und so weiter, beziehungsweise* statt *bspw., z.B., usw., bzw.*.

Setzen Sie die drei verschiedenen Bindestriche -, – und — richtig ein. Der einfache Bindestrich - wird bei Worttrennungen, wie AKÜ-Fimmel, eingesetzt (im Quelltext mit `-`). Der etwas längere Streckenstrich – wird bei Streckenangaben, wie die Strecke Mannheim–Karlsruhe oder von 10:00–11:45 eingesetzt (im Quelltext mit `--`). Der Gedankenstrich — ist bei Einschüben — wie zum Beispiel hier — einzusetzen (im Quelltext mit `---`). ~~Es ist auf keinen Fall ein Leerzeichen um einen Binde - strich oder einen Strecken -- strich und immer ein Leerzeichen um einen Gedanken --- strich~~.

Name	Adresse	Wohnort	Telefon
Susi Sinnlos	Eichenstrasse 5	12345 Unterstadt	24927749242
Horst Kurz	Schnellweg 17	42420 Rapid	999
Jochanaan Leuchtentrager	Hochstraße zu	666 Hell	1-800-33845

Tabelle 4.2: Adressliste

Die entsprechenden vordefinierten Umgebungen heißen `table` für Tabellen und `figure` für Abbildungen. Mit den optionalen Argumenten `htbp`, das steht für *here, top, bottom, page*, geben Sie LaTeX den Tipp, dass Sie am liebsten das Fließobjekt *hier* an dieser Stelle haben möchten. Wenn das nicht geht, dann eben am *Anfang* der Seite, und wenn das nicht geht (weil es zum Beispiel ein Kapitelanfang ist) ans *Ende* der Seite. Wenn das alles nicht klappt, dann halt auf eine Extra-*Seite*. Beherzigen Sie folgende Tipps zu Fließobjekten:

- Jede Tabelle, jedes Bild und jedes Listing ist ein Fließobjekt.
- Zentrieren Sie Bilder und Tabellen.
- Jedes Fließobjekt hat eine Bildunterschrift (Caption) mit einem Label und wird im Text passend referenziert.

Schreiben Sie nie, ~~wie man unten in der Tabelle sehen kann~~, da Sie nie wissen und auch nicht wissen sollen, ob die Tabelle wirklich „weiter unten“ ist. Verwenden Sie statt relativer Positionsangaben Referenzen mit Zahlen, die Sie durch das Label

erhalten, wie zum Beispiel „wie sie in Tabelle 4.2 sehen können“. Verwenden Sie kurze und prägnante Bildunterschriften, die nicht länger als eine Zeile lang sind. Alles was mehr als eine Zeile hat gehört in den Fließtext. Sie sollten für die Fließobjekt Caption keinen Satz bilden und daher auch keinen Punkt am Ende haben. Die Caption ist eine Unterschrift und gehört unter das Fließobjekt.

Es ist möglich, wenn auch nicht empfohlen, Bilder an den Rand einer Seite zu klatschen, wie wir das mit dem GNU-Logo in Abbildung 4.1 gemacht haben. Das ganze ist ein netter Effekt für Graphiken, wie zum Beispiel ein Logo, die nicht zum Verständnis des Texts gebraucht werden und wenig Details aufweisen. Der Effekt sollte aber nicht überstrapaziert werden, 1–2 Mal je Abschlussarbeit sollte, wenn überhaupt, reichen. Außerdem funktioniert `wrapfigure` nicht immer sehr stabil.

Abbildung 4.1: GNU-Logo [3, 4]

Bitte nehmen Sie **nie** JPG oder PNG für Vektorgrafiken, also Zeichnungen mit Linien oder anderen geometrischen Objekten, sondern ausschließlich PDF. Binden Sie also **nie** Vektorgraphiken verpixelt ein.

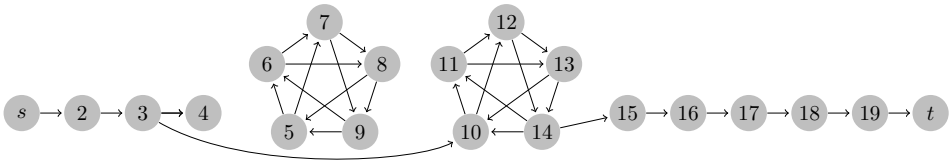

Abbildung 4.2: Automaten mit tikz [2]

Neben langen Listings sind natürlich kurze prägnante Listings in Pseudocode (oder Python ;-) viel angenehmer, wie in Listing 4.1 der effiziente GGT.

```
1 def ggt(x, y):
2     while x != 0:
3         x,y = y%x, x
4     return y
```

Listing 4.1: ggT — kurz und gut

Die Parameter für Listings sollte man für das ganze Dokument gleich lassen. Wenn man mal unbedingt wechseln will, dann ist das auch möglich, wie zum Beispiel bei Listing 4.2, das den ggT in Java mit einem anderen Zeichensatz zeigt.

```
public static int ggt(int x, int y) {
    while (x != 0) {
        int h = x;
        x = y%x;
        y = h;
    }
    return y;
}
```

Listing 4.2: ggT — Java

Der verwendete serifenlose Zeichensatz sieht vielleicht schöner aus, aber der variable Zeichenabstand kann bei Listing störend sein. Der in den Beispielen Listing **??** und Listing 4.1 verwendete Zeichensatz mit festem Zeichenabstand ist für Quellcode meist zu bevorzugen. Wir können natürlich auch C++-Quellcode setzen, bei Listings LaTeX in den Kommentaren verwenden und Listings aus Dateien einlesen wie in 4.3.

```
1 int gcd(int x, int y) { // greatest common divisor
2    while (x != 0) { // x ≠ 0
3       int h = x; // prepare swap
4       x = y%x;
5       y = h;
6    }
7    return y;
8 }
```

Listing 4.3: gcd — C/C++

Idealerweise verwenden Sie spätestens jede zweite Seite ein Bild. Ein Bild lockert auf und „sagt mehr als tausend Worte“. Vermeiden Sie Aufzählungen.

Eine Formel kann im Fließtext integriert sein, wie zum Beispiel $\sum_{i=1}^{n} i = \frac{n\cdot(n+1)}{2}$ oder separat und referenzierbar gesetzt werden wie die Folgende:

$$\sum_{i=1}^{n} i = \frac{n \cdot (n+1)}{2} \tag{4.1}$$

Im LaTeX-Quelltext werden beide Arten von Formeln gleich geschrieben aber anders gesetzt. Achten Sie zum Beispiel auf das Summenzeichen und die Positionen des Index. Die Gleichung 4.1 ist natürlich vom Fließtext aus referenzierbar. Man kann auch schreiben: (4.1) ist natürlich vom Fließtext aus referenzierbar. Im Fließtext kann man auch gerne auf den Bruch verzichten und $\sum_{i=1}^{n} i = (n \cdot (n+1))/2$ schreiben, was meist etwas lesbarer ist. Alternativ geht auch $\sum_{i=1}^{n} i = ½ \cdot n \cdot (n+1)$. Achten Sie

bei Formeln darauf als Multiplikationszeichen $\cdot$ zu verwenden und nicht $*$. Ich kenne einen Kollegen, der ansonsten dadurch sehr erregt wird.

Sie können viele Symbole, wie die griechischen Buchstaben $\alpha, \beta, \gamma, \ldots$; logische Symbole wie $\forall, \exists, \nexists, \wedge, \vee, \neg, \Rightarrow, \Leftrightarrow$; Mengensymbole wie $\in, \cup, \cap, \subseteq, \not\supset, \uplus, \ldots$; andere Symbole $\to, \sqsubseteq, \sim, \models, \vdash, \infty, \emptyset, \mathbb{N}, \mathbb{R}, \ldots$; oder zusammengesetzte Gleichungen wie die Definition der 91er-Funktion [1] verwenden.

$$f(x) = \begin{cases} x - 10 & \text{gdw} \quad x > 100 \\ f(f(x+11)) & \text{sonst} \end{cases}$$

Definition 4.1 *Sei $\varepsilon = 0$.*

Satz 4.1 *Für alle positiven ganzen Zahlen n gilt $\sum_{i=1}^{n} i = \frac{n \cdot (n+1)}{2}$.*

Beweis Vollständige Induktion:

- *Induktionsanfang ($n = 1$):* Es gilt

$$\sum_{i=1}^{1} i \;=\; 1 \;=\; \frac{1 \cdot (1+1)}{2}.$$

- *Induktionsschritt ($n \to n+1$):* Es gelte die Induktionsvoraussetzung (IV):

$$\sum_{i=1}^{n} i = \frac{n \cdot (n+1)}{2}$$

  Wir zeigen, dass dann auch

$$\sum_{i=1}^{n+1} i = \frac{(n+1) \cdot (n+2)}{2}$$

  gilt wie folgt:

$$\begin{aligned} \sum_{i=1}^{n+1} i &= (\sum_{i=1}^{n} i) + (n+1) \\ &=_{\text{IV}} \frac{n \cdot (n+1)}{2} + (n+1) \\ &= \frac{n \cdot (n+1)}{2} + \frac{2 \cdot (n+1)}{2} \\ &= \frac{n \cdot (n+1) + 2 \cdot (n+1)}{2} \\ &= \frac{(n+2) \cdot (n+1)}{2} \end{aligned}$$

□

Sie müssen den Dreisatz *Definition*, *Satz* und *Beweis* nicht verwenden, wenn Sie kein sehr formales Thema haben. Eine sehr formale Aufarbeitung von bekanntem Inhalt gefolgt von einem nicht so formalen eigenen Anteil sollte man meist vermeiden.

`http://www.csie.ntu.edu.tw/~cjlin/libsvm/`, sondern einen passenden publizierten Artikel zitieren.

```
$ bibtex thesis1
$ bibtex thesis2
```

`bibtex`

Kapitel 5

Zusammenfassung und Ausblick

Abschließend kann man sagen, dass...

Literaturverzeichnis

[1] Zohar Manna and Amir Pnueli. Formalization of properties of functional programs. *J. ACM*, 17:555–569, July 1970.

Online-Quellen

[2] TeXample.net, Automata and Petri nets examples. `http://www.texample.net/tikz/examples/automata-and-petri-nets/`. [letzter Zugriff: 9. Jan. 2018].

[3] The GNU Logo.png. `http://en.wikipedia.org/wiki/File:The_GNU_logo.png`. [letzter Zugriff: 9. Jan. 2018], Optimager commonswiki, 21:48, 20 October 2005, unter Free Art Licence.

[4] Free Art Licence. `http://artlibre.org/licence/lal/en`. [letzter Zugriff: 9. Jan. 2018].