Karlsruher Institut für Technologie

FAST

Title of the Thesis
Second Title Line

Master's Thesis/Bachelor's Thesis von

My Name

an der Fakultät für Maschinenbau
Institut für Fahrzeugsystemtechnik (FAST)

Erstgutachter:	Prof. A
Zweitgutachter:	Prof. B
Betreuender Mitarbeiter:	M.Sc. C
Zweiter betreuender Mitarbeiter:	M.Sc. D

xx. Month 20XX – xx. Month 20XX

Ich versichere wahrheitsgemäß, die Arbeit selbstständig angefertigt, alle benutzten Hilfsmittel vollständig und genau angegeben und alles kenntlich gemacht zu haben, was aus Arbeiten anderer unverändert oder mit Änderungen entnommen wurde.

PLACE, DATE

Please replace with actual values

. . . . . .

(My Name)

Zusammenfassung

Deutsche Zusammenfassung

Inhaltsverzeichnis

Abbildungsverzeichnis

Tabellenverzeichnis

1. Introduction

This is the SDQ thesis template. For more information on the formatting of theses at SDQ, please refer to `https://sdqweb.ipd.kit.edu/wiki/Ausarbeitungshinweise` or to your advisor.

1.1. Example: Citation

A citation: [2] For referencing, see Abschnitt 1.2

1.2. Example: Figures

Abbildung 1.1.: SDQ logo

A reference: The SDQ logo is displayed in Abbildung 1.1. (Use `\autoref{}` for easy referencing.)

1.3. Example: Tables

abc	def
ghi	jkl
123	456
789	0AB

Tabelle 1.1.: A table

1.4. Example: Todo-Note

Meaningless text.

Replace with meaningful text. (This note is only shown in draft mode.)

1.5. Example: Formula

One of the nice things about the Linux Libertine font is that it comes with a math mode package.

$$f(x) = \Omega(g(x)) \, (x \to \infty) \;\Leftrightarrow\; \limsup_{x \to \infty} \left| \frac{f(x)}{g(x)} \right| > 0$$

2. First Content Chapter

The content chapters of your thesis should of course be renamed. How many chapters you need to write depends on your thesis and cannot be said in general.

Check out the examples theses in the SDQWiki:
`https://sdqweb.ipd.kit.edu/wiki/Abschlussarbeit/Studienarbeit`

Of course, you can split this .tex file into several files if you prefer.

2.1. First Section

...

2.1.1. A Subsection

...

2.1.1.1. A Subsubsection

2.2. Second Section

...

3. Second Content Chapter

…

3.1. First Section

…

3.2. Second Section

…

Add additional content chapters if required by adding new .tex files in the `sections/` directory and adding an appropriate `\input` statement in `thesis.tex`.

4. Erstellung studentischer wissenschaftlicher Arbeiten mit der *KITreprt*-Klasse

Die *KITreprt*-Klasse dient vorrangig der Erstellung studentischer wissenschaftlicher Texte wie zum Beispiel Bachelor- oder Masterarbeiten. Aufbauend auf dem *Koma-Script*[1] bietet diese Klasse eine KIT-konforme Titelseite, bindet die empfohlenen Pakete standardmäßig ein und versucht ein möglichst ansprechendes Seitenlayout zu erzeugen. In diesem Kapitel werden die Aspekte dieser Klasse vorgestellt und ein paar generelle Hinweise zur Verwendung von LaTeX aufgezeigt. Dieses Dokument wurde mit dieser Klasse erstellt und der Quelltext soll als weiterführendes Beispiel dienen.

4.1. Die *KITreprt*-Klasse

4.1.1. Verwendung der Klasse

Die Klasse wird mit dem Befehl `\documentclass[english,ngerman]{KITreprt}` eingebunden. Die Sprache kann mit den Befehlen `\selectlanguage{ngerman}` und `\selectlanguage{english}` zwischen dem Deutschen und dem Englischen umgeschalten werden. Dokumente mit der *KITreprt*-Klasse sollten direkt mit *Pdflatex* in Pdf-Dokumente übersetzt werden und unbedingt *UTF-8* als Zeichenkodierung verwenden.

4.1.2. Die Einrichtung der Titelseite

Das Layout der Titelseite ist komplett in der Klassendefinition enthalten und es werden keine weiteren Dateien, zum Beispiel für das Logo, benötigt. Für die Daten auf der Titelseite müssen TeX-Makros definiert (bzw. neu definiert werden). Sind diese nicht gesetzt erscheinen auf der Titelseite in roter Schrift Anweisungen, wie diese gesetzt werden müssen. Um ein Makro neu zu definieren reicht zum Beispiel die Anweisung `\renewcommand{\myshorttitle}{Die offizielle HIS-\LaTeX-Vorlage}`. Die möglichen Makros sind in Tabelle 4.1 aufgelistet. Eine Ausnahme bilden die Makros `\advisor`, `\advisortwo`, `\reviewer` und `\reviewertwo`. Dies müssen mit `\newcommand` komplett neu definiert werden, um auf der Titelseite zu erscheinen. Ist keines dieser vier Makros definiert, muss ein spezielles Makro definiert werden, damit kein roter Text zu sehen ist: `\renewcommand{\noadvisors}{}`.

4.1.3. Vordefinierte Pakete

Die folgenden Pakete werden in der *KITreprt*-Klasse definiert und müssen daher nicht von Hand eingebunden werden.

babel Lokalisierte Trennung und Schriftsatz.

fontenc Umlaute und Sonderzeichen.

[1] `http://www.komascript.de/`

Makro	Inhalt
`\myname`	*Name des Autors*
`\mythesis`	*Vordefiniert (zweisprachig):* `\termpaper` *(Seminararbeit)*, `\mastersthesis`, `\bachelorsthesis`, `\protocol`, `\studienarbeit`, `\diplomarbeit`, *oder eigene Bezeichnung*
`\myshorttitle`	*Name der Arbeit*
`\mytitle`	*Optionaler Untertitel oder leer*
`\timestart`	*Anfangsdatum*
`\timestart`	*Datum der Abgabe*
`\advisor`	*Name des betreuenden Mitarbeiteiters (Seminararbeiten)*
`\reviewer`	*Name des Referenten*

Tabelle 4.1.: Die Makros die für die Titelseite definiert werden müssen. Tabellen in wissenschaftlichen Texten sollten nach Möglichkeit nie vertikale Linien verwenden.

inputenc *Utf-8* Zeichenkodierung. Dokumente müssen diese Kodierung verwenden, um mit der Klasse verwendet werden zu können.

graphicx Für die Einbindung von Graphiken in moderenen Formaten.

amsmath,amssymb,amsthm Mathematische Symbole und Extrapakete.

subfigure Mehrere Einzelbilder in der selben Abbildung.

booktabs Ansprechende Tabellen (siehe Tab. 4.1).

listings Darstellung von Quelltexten (siehe Abschnitt 4.4).

helvet,courier,mathptmx Alternative Schriften (z.B. Serifenlose Schrift für Überschriften.

lastpage Erlaubt die letzte Seite zu referenzieren

natbib Paket zum Zitieren für die Naturwissenschaften. Der Stil *abbrvnat* ist voreingestellt.

tikz,eso-pic,setspace,automark Pakete, die intern zum Beispiel für die Titelseite verwendet werden.

4.1.4. LaTeX-Befehle

Dieses Dokument soll nicht als ausführliche Einführung in LaTeXverstanden werden. Für eine Übersicht über alle Befehle, wie zum Beispiel`\emph` für hervorgehobenen Text, wird daher auf folgende Quellen verwiesen:

- Die deutsche Wikipedia-Seite zu LaTeX[2] verweist auf einige gute Einführungen.
- Der LaTeXCompanion von Mittelbach, Goossens, Braams, Carlisle und Rowley [3].

[2] http://de.wikipedia.org/wiki/LaTeX

4.2. Verweise und Zitate

4.2.1. Verweise

Um auf Stellen im selben Dokument zu verweisen wird der `\ref`-Befehl verwendet. Dazu müssen an den Betreffenden Stellen Markierungen (sogenannte Labels) mit dem`\label`-Befehl gesetzt werden. Üblicherweise werden so Textstellen wie Kapitel und Abschnitte oder Abbildungen und Tabellen markiert. Bei Textstellen können direkt nach dem entsprechenden Befehl die Markierungen gesetzt werden, bei Abbildungen sowie Tabellen muss dies nach dem Festlegen der Überschrift passieren. Die Verweise können von LaTeXerst nach wiederholtem Übersetzen korrekt aufgelöst werden. Es empfiehlt sich die Markierungsbezeichnungen zu strukturieren, zum Beispiel alle Abbildungen mit einem vorausgehenden *fig:* zu kennzeichnen.

4.2.2. Zitate

Für das Zitieren soll unbedingt *Bibtex*[3] verwendet werden. Das *natbib*-Paket[4] stellt zusätzliche Befehle zum Zitieren zur verfügung:

`citet` Dieser Befehl erzeugt Zitate, die direkt im Text erscheinen: Asfour u. a. [1],

`citet*` Das Asterisk lässt alle Koautoren erscheinen: Asfour, Regenstein, Azad, Schröder, Vahrenkamp und Dillmann [1]

`citep` Mit diesem Befehl werden Zitate in Klammern angezeigt: [1]

Wichtig: Nicht nur direkte Zitate, die in Anführungszeichen (im Deutschen der Befehl `\gqq`) gekennzeichnet werden müssen, sondern auch alle Gedanken, Ideen oder Ansätze, die nicht vom Autor stammen müssen korrekt und vollständig zitiert werden.

4.2.3. Zeilenumbrüche

Die *cite-* und *ref*-Befehle sollten immer durch eine vorangestellte Tilde (~) – das steht in LaTeX ein geschütztes Leerzeichen – direkt mit vorhergehenden Wort verbunden werden, damit kein unerwünschter Zeilen- oder Seitenumbruch dazwischen erfolgen kann.
Zum Beispiel: `Der Roboter Armar-III~\citep{Asfour2006} hat sieben Freiheitsgrade pro Arm.`

4.3. Bilder und Tabellen

In LaTeXsollten Bilder und Tabellen prinzipiell nicht an einer festen Stelle an den Text gekoppelt werden. Vielmehr gibt es dafür Umgebungen (sogenannte *floating*-Umgebungen), die die Bilder automatisch in der Nähe und an typographisch sinnvoller Stelle platzieren. Für Abbildungen steht dafür die *figure-* und für Tabellen die *tabular*-Umgebung zur Verfügung.

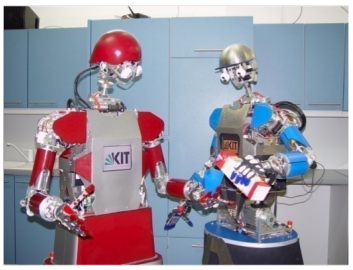
4.3.1. Bilder

Die Verwendung der *figure*-Umgebung soll hier anhand eines kurzen Beispiels in Listing 4.1 erläutert werden. Die resultierende Abbildung ist Abb. 4.1.

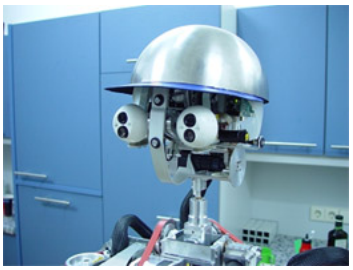
[3] Weitere Informationen unter `http://www.bibtex.org/de/`
[4] `http://www.ctan.org/pkg/natbib`

```
\begin{figure}[htbp]
\begin{center}
% Die beiden Teilbilder
\subfigure[Die humanoiden Roboter ARMAR-IIIa und
    ARMAR-IIIb am His.]{
\label{fig:example2a}
\includegraphics[width=0.3\textwidth]{Example1}}
% horizontaler Abstand  von 1cm.
\hspace{1cm}
\subfigure[Der Kopf des Roboters ARMAR-IIIa]{
\label{fig:example2b}
\includegraphics[width=0.3\textwidth]{Example2}}
% Ueberschrift und Verweismarke fuer die gesamte
    Abbildung.
\caption{Die Roboter des HIS.}
\label{fig:example}
\end{center}
\end{figure}
```

Listing 4.1: Beispiel für eine Floatumgebung.



(a) Die humanoiden Roboter ARMAR-IIIa und ARMAR-IIIb am HIS.



(b) Der Kopf des Roboters ARMAR-IIIa

Abbildung 4.1.: Die Roboter des HIS.

Die eckigen Klammern umfassen die optionalen Parameter, die die Platzierung der Abbildung (nicht zwingend) beeinflussen. Die Parameter sind Präferenzen und LaTeX versucht sie der Reihe nach zu erfüllen:

h Direkt an der Stelle, wo es definiert wurde.

t Oben, am Anfang der Seite.

b Unten, am Ende der Seite.

p Gesondert, auf einer Extraseite für Abbildungen.

In der Umgebung ist eine *center*-Umgebung eingeschlossen, die das Bild zentriert erscheinen lässt. Die beiden *subfigure*-Befehle erzeugen die Einzelbilder der Abbildung. Die optionalen Parameter

in eckigen Klammern erzeugen die Bildunterschriften der Einzelbilder. Gefolgt in den geschweiften Klammern können Markierungen für Verweise auf die Einzelbilder angelegt werden (z.B. für Abb. 4.1(a)) und mittels `\includegraphics` wirklich die Graphikdatei in das Dokument eingebunden werden. Wenn das Bild im Suchpfad liegt (siehe unten) ist keine vorangehende Pfadangabe und Dateierweiterung nötig. Der *caption*-Befehl weiter unten erlaubt eine Überschrift für die gesamte Abbildung anzugeben. Die Markierung mittels *label* wird in Abschnitt 4.2.1 behandelt. Die geöffneten Umgebungen müssen mit einem *end* wieder geschlossen werden. Anstelle der *subfigure*-Befehle kann auch ein einzelnes *includegraphics* stehen. Der Graphikpfad kann wie folgt gesetzt werden (man beachte die doppelten geschweiften Klammern und dass die Anweisung vor dem Dokumentanfang stehen muss):

```
\graphicspath{{./images/}}
```

4.3.2. Tabellen

Die Umgebung für Tabellen ähnelt stark derer für Bilder – inklusive Überschriften und Textmarken. Leider sind die Tabellen selbst relativ kompliziert in LaTeX. Daher wird an dieser Stelle auf den Quelltext dieses Dokuments und andere Quellen verwiesen[5]. In Tabellen sollten generell vertikale Linien vermieden werden, um ein modernes Erscheinungsbild zu garantieren. Das *booktabs*-Paket[6] ermöglicht hier die Verwendung unterschiedlich dicker Linien mit `\toprule`, `\midrule` und `\bottomrule` (siehe Tab. 4.1).

4.4. Quelltexte

Um Quelltexte (engl. Listings) wie in Listing 4.1 zu setzen sollte das *listings*-paket[7] verwendet werden. Listings in *float*-Umgebungen werden mit abgerundeten Rahmen gemäss der Titelseite versehen und unterstützen Syntaxhighliting (farbliche Kennzeichnung von Spracheigenheiten). Als Standard-Sprache ist nach einbinden der Klasse *C++* voreingestellt.

So kann man zum Beispiel mit `\lstinline[language={[LaTeX]TeX}]!\emph!` LaTeX-Befehle im Text eingebettet anzeigen. Für weitere Beispiele sollte der Quelltext dieses Dokuments untersucht werden.

$$e^x = \frac{e^{2\dot{x}}}{||x||} \tag{4.1}$$

[5] `http://en.wikibooks.org/wiki/LaTeX/Tables`

[6] `http://ctan.org/tex-archive/macros/latex/contrib/booktabs/`

[7] `http://www.ctan.org/tex-archive/macros/latex/contrib/listings/`

5. Hinweise für studentische Hilfswissenschaftle

,

In diesem Kapitel werden einige Themen, die für die Arbeit an den *Humanoids and Intelligence Systems Laboratories* wichtig sind, vorgestellt. Die Themen umfassen Bereiche wie Programmiersprachen, Versionsverwaltung und den Webservices, die für die Arbeit am Institut nützlich sind. Diese kurze Übersicht soll nur den Einstieg in die Materie vereinfachen verweist auf weiterführende Literatur.

5.1. C++ Templates

5.1.1. Generische Programmierung – Quelltexterzeugung durch Schablonen

```
1    class DoubleTupel {
2    private:
3    double a[2];
4    public:
5    DoubleTupel(double[])
6    {...};
7    double add(){
8    return a[0]+a[1];
9    };
10   };
```

Listing 5.1: Eine einfache *Tupel*-Klasse für den Datentyp *double*.

```
1    class IntTupel {
2    private:
3    int a[2];
4    public:
5    DoubleTupel(int[])
6    {...};
7    int add(){
8    return a[0]+a[1];
9    };
10   };
```

Listing 5.2: Eine einfache *Tupel*-Klasse für den Datentyp *int*.

Wenn noch verlangt wird, dass es Tupel mit unterschiedlicher Anzahl Elemente für jeden Datentyp existieren sollen, so explodiert die Anzahl der möglichen Kombinationen. Es wäre deswegen wünschenswert, dass ein Mechanismus existierte, der genau diese Kombinationen anhand einer Schablone (engl. template) generiert. Genau diese Funktion gibt bereits in C++[1]. Mit Hilfe des `templates` Schlüsselwortes lassen sich genau solche Schablonen erzeugen. Ein Tupel, das solch einer Schablone (siehe Listing 5.3) entspricht kann nun einfach durch die Angabe des Datentyps und der Anzahl der Elemente erzeugt werden (Listing 5.4). Man beachte, dass den Parametern der Schablone *(engl. template parameters)* Standardwerte zugewiesen werden können (im Beispiel die Anzahl der Elemente).

[1] Diese Mechanismen existieren unter dem Begriff *generics* ebenfalls in anderen Sprachen wie C# und Java (ab Version 5)

```
1    template<typename T,
2    int size=2>
3    class Tupel {
4    private:
5    T a[size];
6    public:
7    T Tupel(T[])
8    {...};
9    T add(){
10   T sum = T[0];
11   for (int i=1;i<size;i++)
12   sum=sum+a[i];
13   };
14   };
```

Listing 5.3: Eine Schablone für die *Tupel*-Klassen.

```
1    int main() {
2    Tupel<double,4> vierer;
3    Tupel<int> zweier;
4    };
```

Listing 5.4: Verwendung der Schablone.

Achtung! Wird die *Tupel*-Klasse für einen Datentyp erzeugt, für den der Operator „+“ nicht definiert ist (zum Beispiel für Zeichenfolgen), führt dies zu einer schwer zu verstehenden Fehlermeldung des Compilers. Als Ursprung des Fehlers wird insbesondere nicht die Erzeugung der Klasse mit einem ungeeigneten Parameter sein, sondern die Zeile an der die Addition stattfindet. Gerade wenn die Schablonen von jemand Anderem verfasst wurden, ist dann der Fehler nur sehr schwer zu finden.

Eine weitere Einschränkung stellt die Tatsache dar, dass Schablonen (entgegen der Spezifikation) bei allen üblichen Compilern *komplett* in einer Header-datei definiert werden müssen.

5.1.2. Wichtige Templates des C++-Standards

Im C++-Standard (bzw. in den Standardbibliotheken) finden sich eine Reihe nützlicher Klassen, die sich die Möglichkeiten des generischen Programmierens zunutze machen. In diesem Abschnitt werden zwei der gebräuchlichsten vorgestellt: *vector* und *map*. Bei beiden handelt es sich um so genannte Containerklassen, d.h. sie können Objekte anderer Datentypen aufnehmen – ganz wie die *Tupel*-Klasse im letzten Abschnitt. Beide Klassen sind in dem Kontext *(engl. namespace)* „std::“ definiert, das bedeutet damit der Compiler die Namen richtig auflösen kann, muss entweder `std::` vor den Bezeichner gesetzt werden oder der aktuelle Kontext zum Beispiel mit `using namespace std;` erweitert werden.

Die *vector*-Klasse kann wie ein dynamisch wachsendes Datenfeld *(engl. array)* oder ein Stapel *(engl. Stack)* verwendet werden. Um die Klasse zu verwenden muss die Klassendeklaration mittels `#include <vector>` eingebunden werden.

Die *map*-Klasse stellt ein assoziatives Array zur Verfügung. Damit kann einem Elements oder „Schlüssel “eines beliebigen Datentyps (wobei alle Schlüssel des selben Typs sein müssen) ein Element eines anderen Datentyps zugeordnet werden. Zum Beispiel lassen sich auf diese Weise dynamische Felder erzeugen deren Elemente mit Namen (*string*-Objekten) adressiert werden können.

Des Weiteren gibt es in den Standardbibliotheken nützliche Klassen, wie zum Beispiel Sortieralgorithmen für die Containerklassen und die Stringklasse *std::string*, die die veralteten C-Strings ersetzen soll. Die in C übliche *printf()* Funktion sollte durch das Objekt *std::cout* ersetzt werden.

```
1   #include <vector>
2   #include <iostream>
3   int main() {
4   using namespace std;
5   vector<int> v;
6   v.push_back(1);
7   v.push_back(12);
8   cout << "1: " << v[0]
9   << "2:" << v[1] << endl;
10  }
```

Listing 5.5: Die *vector*-Klasse.

```
1   #include <map>
2   #include <string>
3   #include <iostream>
4   int main() {
5   std::map<char,int> m;
6   // F\"uge der map Elemente
7   // hinzu:
8   m['a']=1;
9   m['b']=5;
10  cout << "a: " << m['a']
11  << "b: " << m['b']
12  << endl;
13  }
```

Listing 5.6: Die *map*-Klasse.

5.1.3. Zukünftige Erweiterungen und die Boost-Bibliotheken

Die *Boost*-Bibliotheken[2] haben das Ziel den Funktionsumfang der Standardbibliotheken dem anderer moderner Programmiersprachen wie Java, C# oder Python anzugleichen. Ein Komitee entscheidet in einem strengen Prozess zur Qualitätssicherung darüber, welche Bibliotheken zu Boost hinzugefügt werden. Ein Großteil dieses Komitees ist darüber hinaus an der Verabschiedung des neuen C++-Standards C++-11 beteiligt gewesen, so dass viele der Boost-Bibliotheken in den neuen Standard übernommen wurden. Von den Bibliotheken werden zur Zeit unter Anderem folgende Bereiche abgedeckt: die Verarbeitung von Zeichenketten, weitere Container-Klassen, Algorithmen, funktionale Programmierung (λ-Kalkül), Nebenläufigkeit, Datenstrukturen, Ein- und Ausgabe, Bildverarbeitung und mehr. Wie bei den Standardbibliotheken sind die Funktionen in einem gesonderten Kontext `boost::` gesammelt. Eine sehr gute deutschsprachige Einführung kann man im Internet[3] finden.

Die *boost::shared_ptr*-Klasse Im Gegensatz zu Java oder C# weisst C++ keine automatische Speicherverwaltung auf. Die Lebensdauer eines Objektes wird nur dann automatisch festgelegt, wenn es auf dem Stack angelegt wird (d.h. ohne die Verwendung von *new*) und zwar genau dann wenn sein Gültigkeitsbereich verlassen wird.
Objekte, die mit *new* erzeugt wurden, existieren bis der *delete*-Operator auf sie angewendet wird. Dies hat kann zwei Probleme zur Folge haben. Zum einen müssen alle Objekte die mittels *new* angelegt werden auch wirklich von Hand mit *delete* gelöscht werden um Speicherlecks zu verhindern. Zum anderen sind die Folgen wenn man umgekehrt versucht auf ein Objekt zu zugreifen, dass bereits nicht mehr existiert, gravierend. Es wird in diesem Fall keine Ausnahme *(engl. exception)* ausgelöst wie bei den meisten modernen Sprachen. Anstelle dessen erfolgt meist eine unerlaubte Speicherzugriffsverletzung, die das sofortige Beenden des Programms ohne jeglichen Hinweis

[2] `http://www.boost.org/`
[3] unter `http://www.highscore.de/cpp/boost/titelseite.html`

```
1#include <boost/shared_ptr.hpp>
2
3class A {
4void method(){};
5};
6
7void main() {
8
9// \"ubernehme nur die shared_ptr in den aktuellen Kontext
10using boost:shared_ptr;
11
12shared_ptr<A> a(new A);
13
14// Erh\"ohe Referenzenz\"ahler
15shared_ptr<A> b = a;
16
17// Erniedrige Referenzenz\"ahler
18a.reset();
19
20b->method();
21
22A* c = b.get() // Gef\"ahrlich
23
24// b verliert bei Programmende die  G\"ultigkeit
25// und das Objekt vom Typ A wird zerst\"ort.
26} ;
```

Listing 5.7: Die Verwendung der *shared_ptr*-Klasse.

auf die Stelle des Fehlers zur Folge haben, oder es gar können Daten ausgelesen werden, die zu einem völlig anderen Datenblock gehören. Die Klasse *boost::shared_ptr*, die mit `#include <boost/shared_ptr.hpp>` verwendet werden kann, ergänzt C++ um einen Mechanismus, der genau solche Fehler verhindern soll. Dazu wird anstelle normaler Zeiger die Klasse *boost::shared_ptr* verwendet, die einen Zeiger auf das Zielobjekt beinhaltet. Die Klasse zählt mit, wie viele solcher Referenzen auf das selbe Objekt verweisen und gibt erst dann den Speicher des Objektes frei, wenn alle ihren Gültigkeitsbereich verlassen haben oder dereferenziert wurden (sogenanntes *reference counting*). Wenn konventionelle Zeiger nicht mit diesen Referenzen vermischt werden, können unerlaubte Zugriffe nicht mehr passieren – bei dem Versuch wird von dem *shared_ptr* eine Exception ausgelöst die sich zurückverfolgen lässt.

Die *shared_ptr* sind als Schablone realisiert und können so Zeiger auf jeden beliebigen Datentyp aufnehmen, zum Beispiel: `boost::shared_ptr<A> a(new A)` legt eine neue Referenz auf ein Objekt vom Typ *A* an. Auf die Klassenelemente kann dann einfach zugegriffen werden: `a->method()` (siehe Listing 5.7).

5.1.4. Metaprogrammierung

...

5.2. Versionsverwaltung mit Mercurial

Die Verwendung einer Versionsverwaltung hat in vielerlei Hinsicht Vorteile. Zu den wichtigsten gehört, dass man zwischen den unterschiedlichen Stadien des Projektes hin- und herspringen kann. Man kann genau verfolgen, wer zu welchem Zeitpunkt Änderungen vorgenommen hat, und diese auch Rückgängig machen (wenn man zum Beispiel einen Wichtigen Bestandteil gelöscht hat). Des Weiteren können mehrere Personen an dem selben Projekt arbeiten; die Versionsverwaltung kümmert sich um die Zusammenführung der jeweiligen Arbeiten. Dies ist auch von Vorteil, wenn man selbst mit mehreren Rechnern arbeitet. Nicht zu vernachlässigen ist auch die Möglichkeit, die Software für Backups zu benutzen. *Mercurial* (dt. Quecksilber) ist ein Vertreter dieser Programme. Es zeichnet sich dadurch aus, dass sich die Bedienung stark am Urvater SVN anlehnt, es frei erhältlich ist (OpenSource) und auf allen wichtigen Plattformen verfügbar ist. Befehle beginnen stets mit `hg`, dem chemischen Zeigen für Quecksilber im Periodensystem. Die wichtigsten (meistens ausreichend) Befehle sind in Listing 5.8 zu sehen. *Git*, das vom Linux-Erfinder Linus Torvalds entwickelt wurde ähnelt Mercurial in sehr vielen Aspekten und lässt sich analog bedienen. Beide unterscheiden sich von SVN dadurch, dass sie verteilte Versionsverwaltungen sind. Das bedeutet, dass nicht zwingend ein einzelner Zentraler Server existieren muss, sondern jede lokale Kopie für sich ein selbständiges Repository darstellt. Daraus resultieren unter Anderem, die Vorteile, dass kein ständiger Kontakt zum Server gehalten werden muss, und dass Abspaltungen vom Hauptentwicklungszweig sehr einfach erstellt und später wieder rückgeführt werden können.

```
1
2$> hg initialize <Projektname>
3# Erzeugt neues (leeres) Projekt.
4# Das Verzeichnis <Projektname> wird neu angelegt.
5
6$> hg clone <Quelle> [<Ziel>]
7# Erstellt eine Kopie eines Projektes.
8# Quelle und Ziel k\"onnen auf unterschiedlichen Rechnern liegen.
9
10$> hg push [<Ziel>]
11# \"ubertr\"agt lokale \"Anderungen zu dem angegebenem Ziel
12
13$> hg pull [<Quelle>]
14# \"ubertr\"agt lokale \"Anderungen zu dem angegebenem Ziel
15
16$> hg add <File1> [<File2> ...]
17# F\"ugt Dateien und Ordner dem Archiv hinzu
18
19$> hg ci -m"<Text>"
20# Speichert die aktuelle Version.
```

Listing 5.8: Die wichtigsten Mercurialbefehle.

Sollen Quelle und Ziel permanent gespeichert werden (um zukünftig nicht mehr angegeben zu werden, so können diese in einer Einstellungsdatei `.hg/hgrc` abgelegt werden (siehe Listing 5.9).

```
1  [paths]
2  default = ssh://i61p100/Path/To/Project
3  default-push = ssh://i61p100/Path/To/Project
```

Listing 5.9: Konfiguration von Mercurial

Literatur

[1] T. Asfour u. a. “ARMAR-III: An Integrated Humanoid Platform for Sensory-Motor Control”. In: *IEEE/RAS International Conference on Humanoid Robots (Humanoids)*. Genova, Italy, Dez. 2006, S. 169–175.

[2] Steffen Becker, Heiko Koziolek und Ralf Reussner. “The Palladio Component Model for Model-driven Performance Prediction”. In: *Journal of Systems and Software* 82 (2009), S. 3–22. DOI: 10.1016/j.jss.2008.03.066. URL: http://dx.doi.org/10.1016/j.jss.2008.03.066.

[3] Frank Mittelbach u. a. *The LaTeX Companion*. 2. Aufl. Addison-Wesley, Apr. 2004.

A. Evaluation

…

A.1. First Section

…

A.2. Second Section

…

A.3. Third Section

…

B. Conclusion

…

C. Anhang

C.1. First Appendix Section

Abbildung C.1.: A figure

…