ingenieur
wissenschaften
htw saar

Hochschule für
Technik und Wirtschaft
des Saarlandes
University of
Applied Sciences

**Bachelor-Thesis**

zur Erlangung des akademischen Grades
Bachelor of Science (B. Sc.)
an der Hochschule für Technik und Wirtschaft des Saarlandes
im Studiengang Praktische Informatik
der Fakultät für Ingenieurwissenschaften

# Eine LaTeX-Vorlage für Abschlussarbeiten im Bereich Informatik/ Mechatronik-Sensortechnik an der htw saar

vorgelegt von
Max Muster

betreut und begutachtet von
Prof. Dr.-Ing. André Miede
Prof. Dr. Thomas Kretschmer

Saarbrücken, Tag. Monat Jahr

# Selbständigkeitserklärung

Ich versichere, dass ich die vorliegende Arbeit (bei einer Gruppenarbeit: den entsprechend gekennzeichneten Anteil der Arbeit) selbständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel benutzt habe.

Ich erkläre hiermit weiterhin, dass die vorgelegte Arbeit zuvor weder von mir noch von einer anderen Person an dieser oder einer anderen Hochschule eingereicht wurde.

Darüber hinaus ist mir bekannt, dass die Unrichtigkeit dieser Erklärung eine Benotung der Arbeit mit der Note „nicht ausreichend“zur Folge hat und einen Ausschluss von der Erbringung weiterer Prüfungsleistungen zur Folge haben kann.

*Saarbrücken, Tag. Monat Jahr*

Max Muster

# Zusammenfassung

Kurze Zusammenfassung des Inhaltes in deutscher Sprache, der Umfang beträgt zwischen einer halben und einer ganzen DIN A4-Seite.

Orientieren Sie sich bei der Aufteilung bzw. dem Inhalt Ihrer Zusammenfassung an Kent Becks Artikel: `http://plg.uwaterloo.ca/~migod/research/beckOOPSLA.html`.

*We have seen that computer programming is an art,*
*because it applies accumulated knowledge to the world,*
*because it requires skill and ingenuity, and especially*
*because it produces objects of beauty.*

— Donald E. Knuth [5]

# Danksagung

Hier können Sie Personen danken, die zum Erfolg der Arbeit beigetragen haben, beispielsweise Ihren Betreuern in der Firma, Ihren Professoren/Dozenten an der htw saar, Freunden, Familie usw.

# Inhaltsverzeichnis

# 1 Einleitung

## 1.1 LaTeX installieren und einrichten

### 1.1.1 Unter Windows

Als LaTeX-Distribution unter Windows steht *MikTeX* zu Verfügung, die als freie Software im Internet erhältlich ist. *MikTeX* unterstützt Windows XP, Vista und Windows 7. Neben *MikTeX* wird noch ein PostScript-Interpreter benötigt, z.B. GhostScript, zu finden auf Chip.de.

*Wichtig:* Bei *MikTeX* unbedingt Vollinstallation auswählen, sonst sind eventuell benötigte Packages nicht vorhanden.

### 1.1.2 Unter Linux

Unter Linux existiert die LaTeX-Distribution *texlive*, die als aktuelle Version aus den Paketquellen geladen werden kann (unter Ubuntu mit `apt-get install texlive-full`). Auch hier ist ganz wichtig, die volle Distribution zu laden, damit alle Packages zur Verfügung stehen.

## 1.2 Entwicklungsumgebungen

Hat man die passende Distribution installiert, bieten sich vielerlei Möglichkeiten an ein LaTeX-Projekt anzugehen oder einzelne Dokumente zu editieren. Unter Windows könnten dies folgende sein:

**TeXnicCenter** Umfangreiche Entwicklungsumgebung mit Projektorganisation und Autovervollständigung

**TeXLipse** Eclipse-Plugin, das alle Vorteile der Eclipseumgebung mit LaTeX verbindet

**TeXmaker** Einfacher LaTeX-Editor mit Pdf-Direktvorschau

Unter Linux stehen bereit:

**Gummi** Ebenfalls einfacher LaTeX-Editor mit Direktvorschau

**TeXLipse** Auch für Linux erhältlich

**Kile** Umfangreiche Entwicklungsumgebung, ähnlich wie TeXnicCenter

Nach der Installation muss die Entwicklungsumgebung eingerichtet werden; dazu finden sich viele Anleitungen im Internet, die genau erklären, welche Distribution auf welche Weise eingerichtet wird. Insbesondere sollte der PDF-Viewer festgelegt werden, damit bei Gummi und TeXmaker die Direktvorschau funktioniert. Manchmal kommt es vor, dass die Ausgabe nach dem Kompilieren Umlaute und Sonderzeichen nicht richtig darstellt. Unter Linux hängt dies mit den unterschiedlichen Zeichensätzen zusammen, die unterstützt werden. Um diese Vorlage zu verwenden ist es notwendig, den verwendeten Zeichensatz des Editors bzw. der Entwicklungsumgebung auf den in diesem Dokument verwendeten Zeichensatz umzustellen: UTF-8 ohne BOM (Byte Order Mark).

## 1.3 Werkzeuge

**JabRef** Ein Literaturverwaltungsprogramm, welches das *BibTeX*-Format einsetzt und mithilfe einer graphischen Oberfläche das Anlegen von Literaturverzeichnissen vereinfacht.

## 1.4 Struktur und Gebrauch der Vorlage

Die vorliegende Vorlage für Abschlussarbeiten besteht aus einer internen Struktur, die grundsätzlich nicht verändert werden sollte.

### 1.4.1 Struktur der Vorlage

**htw-i-mst-config.tex** Enthält alle zu ladenden Packages, Styleparameter für Hyperlinks, Codelistings und Literaturverzeichnis sowie globale Parameter für Tabellen und Beschriftungen. Im Besonderen befinden sich hier die Variablen für den eigenen Namen, Titel, Datum der Arbeit, den betreuenden Professor etc.

**htw-i-mst-vorlage.tex** Dies ist die Hauptdatei, in der alle notwendigen **.tex*-Dateien eingebunden werden, die zu dem Dokument gehören. Es empfiehlt sich die interne Struktur *nicht* zu verändern. Eigene Kapitel werden an der dafür markierten Stelle eingebunden.

**Bibliography.bib** Zentrale Datei für die Literaturangaben, welche man z.B. mit JabRef verwalten kann.

**Chapters/** Ablageort für alle selbst angelegten Kapitel der Arbeit. Die Aufteilung in eigene Dateien erleichtert die Übersicht über den Quellcode.

**Graphics/** Ablageort für alle im Dokument benötigten Grafikdateien. Gerne darf man hier Unterverzeichnisse zur besseren Strukturierung anlegen.

**Examples/** Dieser Ordner enthält die in dieser Vorlage beigefügten LaTeX-Beispiele, welche vor der Abgabe der Arbeit selbstverständlich gelöscht werden sollten.

**Frontbackmatter/** In diesem Ordner sind all jene Dateien abgelegt, die – außer dem Kerntext in *Chapters/* – die Gesamtheit der Abschlussarbeit ausmachen.

- **Titlepage.tex** Definiert die Titelseite der Abschlussarbeit. Diese Datei muss normalerweise nicht verändert werden.
- **Abbreviations.tex** Hier werden alle Abkürzungen hinterlegt, die im Dokument verwendet werden.
- **Abstract.tex** Eine kurze Zusammenfassung der Abschlussarbeit wird in diese Datei eingefügt.
- **Acknowledgements.tex** Dort finden Danksagungen ihren Platz.
- **ConfidentialityClause.tex** Beinhaltet den Sperrvermerk und ist nur zu verwenden, falls dies beispielsweise vom beteiligten Unternehmen gefordert wird.
- **Contents.tex** Enthält wichtige Eintragungen in die *Table-of-Contents*. Diese Datei muss normalerweise nicht geändert werden.
- **Declaration.tex** Enthält die Selbständigkeitserklärung. Diese Datei darf nicht geändert werden.

**Colophon.tex** Enthält einen Hinweis auf die Urheber dieser Vorlage. Diese Datei darf nicht geändert werden.

**ListOfs.tex** Enthält die Einträge für die Tabellen- und Abbildungsverzeichnisse etc. und muss gewöhnlich nicht verändert werden.

### 1.4.2 Gebrauch der Vorlage

Grundsätzlich ist nicht viel zu tun, um die Vorlage für Abschlussarbeiten zu verwenden. Man entpackt den Hauptordner in das gewünschte Verzeichnis und nutzt die Dateien so, wie in Abschnitt 1.4.1 beschrieben. Danach werden *alle* Dateien gespeichert und die Hauptdatei, *htwsaar-i-mst-vorlage.tex*, mehrfach kompiliert (LaTeX benötigt mehrere Durchgänge um z.B. Referenzen richtig zuzuordnen). Hat man Änderungen in *Bibliography.bib* bzw. *Bibliography.tex* vorgenommen oder neue Zitate z.B. mittels `\cite` eingefügt, muss erst mit *BibLaTeX* und anschließend mitdem entsprechenden LaTeX-nach-PDF-Compiler übersetzt werden.

# 2 Herausforderungen

In dem folgendem Kapitel werden die Herausforderungen und Schwierigkeiten von Miroservices beschrieben. Dabei werden die Anforderungen an das Netzwerk beschrieben, welches mit den Lasten von Verteilten Systemen zurecht kommen muss (2.1). Darauf folgend wird auf die vermutlich größte Herausforderung eingegangen, der Architektur. Eine schlechte Architektur kann den Vorteil von Microservices zunichte machen und muss daher fortlaufend angepasst werden (2.2). Danach wird die benötigt Infrastruktur beschrieben, die bei Microservices um ein vielfaches größer ist und mehr Aufmerksamkeit bedarf als bei einer mololithischen Entwicklung (2.3).

## 2.1 Netzwerk

Da Microservices verteilte Systeme sind, werden Aufrufe untereinander über das Netzwerk verschickt. Damit verbunden sind Latenz und Antwortzeiten die mit steigender Netzwerkauslastung exponentiell steigen.

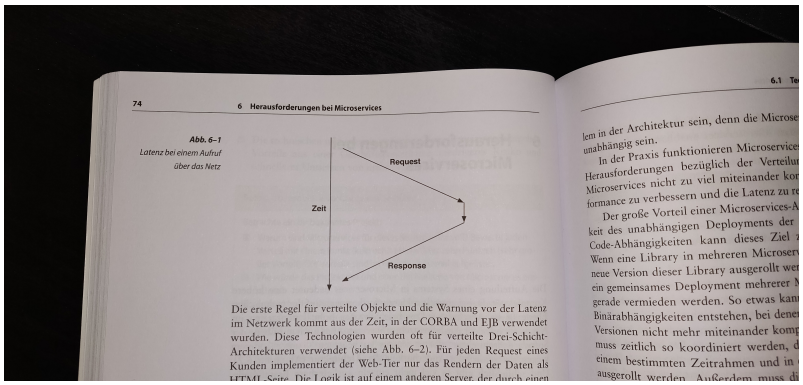


Abbildung 2.1: Latenz bei einem Netzwerkaufruf

In Fig. 2.1 ist ein typischer Aufruf über ein Natzwerk zu sehen. Während die Anfrage über das Netzwerk verschickt, von einem anderen Service bearbeitet und die Antwort wieder zurück geschickt wird muss der ursprüngliche Service warten. Selbst wenn die Latenz einer Nachricht nur 0,5 ms beträgt, könnte ein Prozessor mit 3 GHz in dieser Zeit 1,5 Millionen Operationen durchführen. Daher sollte auf die Entfernung der einzelnen Services geachtet werden und diese so nah beieinander zu halten wie möglich. Auch sollte darauf geachtet werden, dass das Netzwerk nicht überlastet wird. Subnetze bieten sich bei dieser Problemstellung an.

Eine hohe Kommunikation zwischen Microservices spricht jedoch auch für eine schlechte Skalierung der Services und für zu vielen Abhängigkeiten. Solche Abhängigkeiten sollen jedoch vermeiden werden, da dadurch der Vorteil von Microservices verloren geht - das unabhängige Deployment. Mehr zum Thema Architektur im folgendem Unterkapitel (2.2).

Ein weiteres Problem welches bei verteilten Systemen auftreten kann, ist der Ausfall eines Teilsystems. Wenn dies geschieht, müssen unterschiedliche Maßnahmen getroffen werden, damit nicht das gesamte System zum Erliegen kommt. So muss es Pläne geben, was passiert, wenn eine Request keine Antwort bekommt. Dafür kann es verschiedene Gründe geben. Es kann beispielsweise daran liegen, dass die Nachricht im Netzwerk nicht richtig weitergeleitet wurde. Eine weitere wahrscheinliche Möglichkeit ist ein abgestürzter Gegenspieler. Im ersten Fall kann durch eine erneut gesendete Anfrage das Problem gelöst werden, im zweiten Fall würde dies jedoch ohne weiteren Maßnahmen zu einer Endlosschleife führen. Dies gilt es zu erkennen und geeignete Fehlerfälle mitzubeachten. Währenddessen muss der abgestürzte Microservice neu gestartet werden, am Besten wäre es, wenn mehrere Instanzen des Services aktiv wären und im Falle eines Absturzes diese kurzzeitig die zusätzlichen Anfragen bearbeiten könnten. Gleichzeitig wird im Hintergrund der Service automatisch neu gestartet und kann danach weiterarbeiten.
Im Falle von großen Lastwechseln kann dieser Mechanismus auch dazu genutzt werden bei Bedarf neue Instanzen eines Services zu starten, damit keine zu großen Warteschlangen entstehen und die Antwortzeiten gering bleiben. Ein Aufteilen der Requests auf die einzelnen Instanzen kann dabei dann mittels Service Discovery und Load Balancing geschehen.

## 2.2 Architektur

Bei Microservices ist es wichtig, dass die fachliche Struktur mit der Organisationsstruktur übereinstimmt (Gesetz von Conway). Die unterschiedlichen fachlichen Funktionen werden dabei in Microservices aufgeteilt. Ein Team der Organisation ist dabei für einen oder mehrere Microservices zuständig. Die Microservices anderer Teams werden dagegen als Black-Box gesehen, von denen nur die Schnittstelle bekannt ist. Daher können die Microservices auch in unterschiedlichen Technologien implementiert sein. Die einzige Gemeinsamkeit, die alle Services haben müssen, ist eine einheitliche Schnittstelle. Eine valide Option hierfür wäre eine REST Schnittstelle.
Durch die verschiedenen Technologien ist es schwer einen Gesamtüberblick über das System zu behalten. Falls dies jedoch benötigt wird, ist es ratsam einen Technologiestack vorzuschreiben. Dies kann auch partiell durchgeführt werden, falls beispielsweise einheitlich geloggt werden soll.

**Änderungen** Durch die Aufteilung in kleine Microservices sind Änderungen und Refactorings innerhalb eines Microservices sehr leicht umzusetzen. Bei größeren Änderungen oder Fehlern ist es auch leicht den gesamten Microservice zu ersetzen und neu zu implementieren.
Falls neue Anforderungen Änderungen an mehreren Microservices erfordern, ist dies sehr viel mehr Arbeit. Kommt dies öfters vor, kann dies auch für eine schlechte Architektur sprechen, die überarbeitet werden sollte. Bei Microservice-übergreifenden Änderungen bei denen mehrere Teams beteiligt sind, müssen diese koordiniert werden und sich miteinander abstimmen. Auch können die Services in unterschiedlichen Technologien entwickelt worden sein oder Bibliotheken in unterschiedlichen Versionen nutzen. Dies alles macht den Vorteil von Microservices zunichte und ist sehr Aufwändig. Daher ist das initiale Erstellen der Architektur ein essentieller Punkt, der über den Erfolg des Projektes maßgeblich mitentscheidet.

**Anpassen der Architektur** Wie bereits gezeigt, ist die Architektur ein essentieller Part eines Microservice Systems. Durch eine gute Architektur können Anforderungen nach-

haltig schnell umgesetzt werden (Micorservice-intern) und es können die Technologien genutzt werden, die für den Anwendungsfall optimal sind.
Jedoch ändert sich die optimale Architektur mit neuen Anforderungen. Wenn die Architektur daraufhin (agil) angepasst wird, ergibt sich daraus auch kein Problem, jedoch wenn man an der alten Architektur festhält.
Gründe für eine Anpassung der Architektur können folgende sein:

- Ein Microservice ist zu groß und muss aufgeteilt werden. Zeichen hierfür ist beispielsweise eine schlechte Verständlichkeit oder auch eine Größe, mit der nicht einmal ein ganzes Team zurecht kommt. Ebenfalls kann es vorkommen dass ein Micorservice mehrere Themengebiete umfasst.

- Ein Microservice bearbeitet Funktionen, die in einem anderen Microservice besser aufgehoben wären. Erkennbar sind solche Funktionen durch viel Kommunikation zwischen den beiden Services. Ebenfalls können Bereiche in einem Microservice sehr wenig miteinander zu tun haben. Diese können dann aus Gründen der Übersichtlichkeit in neue Microservices getrennt werden.

- Wenn eine Funktion von mehreren Microservices genutzt werden soll, sollte eine Architekturänderung ebenfalls in Betracht gezogen werden.

Um diese Probleme zu lösen gibt es verschiedene Lösungsansätze:

**Gemeinsame Bibliotheken** Wenn Code gemeinsam genutzt werden soll, so kann dieser in Bibliotheken ausgelagert werden. Dies setzt voraus, dass die darauf zugreifenden Microservices in der selben Technologie entwickelt werden.
Dies bedeutet jedoch auch, dass die Microservices von einander abhängig werden und die Arbeit an der Bibliothek koordiniert werden muss.
Durch 3rd. Party Bibliotheken können so ebenfalls Probleme entstehen. So kann in manchen Laufzeitumgebungen jeweils nur eine Version einer Bibliothek genutzt werden. Das heißt, dass wenn in der neuen Bibliothek die externe Bibliothek XY v2.0 benötigt wird, muss der Code des Microservices, wenn er die Bibliothek XY benötigt, diese ebenfalls in der Version 2.0 nutzen.
Wegen diesen Problemen wird eine Wiederverwendung von Code in Microservices nicht forciert.

**Code übertragen** Ein anderer Weg eine Funktion in einem Microservice verfügbar zu machen, ist die Codeübertragung. Der Ansatz ist wie bei einer gemeinsamen Bibliothek, nur dass keine Abhängigkeiten entstehen. Dadurch kann eine lose Kopplung zwischen zwei Abhängigen und stark kommunizierenden Microservices wiederhergestellt werden. Dabei spielt es keine Rolle ob der Code im ursprünglichem Microservice bestehen bleibt oder nicht, wodurch jedoch Redundanzen entstehen. Dies bedeutet dass Bug Fixes an mehreren Stellen vorgenommen werden müssen, dagegen können sich die Microservices/Funktionen unabhängig voneinander in unterschiedliche Richtungen weiterentwickeln.
Es muss darauf geachtet werden, dass durch das Hinzufügen von Funktionen der Microservice nicht zu groß wird und zu einem Monolithen mutiert.

**Gemeinsamer Service** Anstatt den Code in eine Bibliothek auszulagern, kann dieser auch in einen neuen Microservice überführt werden. Dadurch werden auch verschiedenen Technologien unterstützt. Mit dieser Methode werden die Micorservices auch klein gehalten, was vorteilhaft für die Übersichtlichkeit und Verständlichkeit des Systems ist. Diese neuen Microservices sind dann reine Backend-Services, da

diese keine Benutzeroberfläche besitzen.
Dieses Verfahren kann auch gut dazu genutzt werden wenn ein Microservice zu groß wird. Durch Auslagern von Funktionen kann die Wartbarkeit wieder erhöht werden.

**Neu schreiben** Mit der Zeit kann es vorkommen dass ein Microservice schwer wartbar wird. Dies kann beispielsweise durch einen "historisch gewachsenen" Code geschehen, oder auch durch die Auswahl einer nicht praktikablen Technologie. Gerade bei solchen Problemen haben Microservices ihren Vorteil.
Durch ihre kleine Größe kann der gesamte Service ohne größeren Aufwand neu entwickelt werden. Dabei können neue Erkenntnisse über die Domäne direkt in die Neuimplementierung einfließen. Ein Wechseln der Technologie ist ebenfalls kein Problem, solang die Schnittstellen gleich bleiben.

## 2.3 Infrastruktur

Auch bei der Infrastruktur gibt es bei Microservices Hürden und Herausforderungen. So werden viel mehr Server oder Virtuelle Maschinen benötigt als bei einem Monolithen. Die verschiedenen Services müssen aufgrund ihrer Unabhängigkeit auf einzelnen Servern/VM's liegen. Dies kann bspw. an verschiedenen Versionen von Bibliotheken liegen oder auch an dem unabhängigen Deployment der Services.
Einer der großen Vorteile von Micorservices ist das lastabhängige Starten zusätzlicher Instanzen. Dies muss automatisch und auf einer neuen VM geschehen. Um jedoch überhaupt erstmal festzustellen, wie groß die Last ist, muss ein geeignetes Monitoring auf allen Services aktiv sein. Aufgrund der verschiedenen Technologien kann dies zu einem großen Problem werden.
Ebenfalls sollte ein Monitoring für die Problemfindung vorhanden sein. Dies muss auf die einzelnen Services angepasst sein, damit Schwachstellen erkannt und optimiert werden können.
Auch bei der Entwicklung gibt es im Vergleich zu einem Monolithen Nachteile. So muss für jeden einzelnen Microservice eine Versionskontrolle eingerichtet werden. Eine Build Pipeline pro Micorservice ist ebenfalls empfehlenswert. Diese Entwicklungsanforderungen bedeuten für das Unternehmen zusätzliche Infrastruktur wie auch erhöhte Administration.

# 3 Fallstudie

**Szenario** Für unseren Online-Shop soll eine neue Benutzerverwaltung entwickelt werden, nachdem die alte nicht mehr Wartbar ist. Dazu zählt das Anmelden und neu registrieren von Nutzern wie auch das ausgeben der Benutzerdaten.

**Gründe für einen Microservice** Da man aus den Problemen der Vorherigen Benutzerverwaltung lernt, soll die neue Verwaltung auch langfristig Wartbar bleiben.
Auch soll jedes Teilgebiet der Anwendung von einem einzelnen Team umgesetzt werden.

# Literatur


[1] Jon Bentley. *Programming Pearls*. Addison–Wesley, 1999.

[2] Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest und Clifford Stein. *Introduction to Algorithms*. The MIT Press, 2001.

[3] Gunter Dueck. *Duecks Trilogie: Omnisophie – Supramanie – Topothesie*. `http://www.omnisophie.com`. Springer, 2005.

[4] Donald E. Knuth. „Big Omicron and Big Omega and Big Theta“. In: *SIGACT News* 8.2 (1976), S. 18–24.

[5] Donald E. Knuth. „Computer Programming as an Art“. In: *Communications of the ACM* 17.12 (1974), S. 667–673.

[6] Ian Sommerville. *Software Engineering*. Boston, MA, USA: Addison-Wesley, 1992.

# Abbildungsverzeichnis



# Tabellenverzeichnis

# Listings

# Abkürzungsverzeichnis

# Anhang

# A Erster Abschnitt des Anhangs

In den Anhang gehören „Hintergrundinformationen“, also weiterführende Information, ausführliche Listings, Graphen, Diagramme oder Tabellen, die den Haupttext mit detaillierten Informationen ergänzen.

Dies hier ist ein Blindtext zum Testen von Textausgaben. Wer diesen Text liest, ist selbst schuld. Der Text gibt lediglich den Grauwert der Schrift an. Ist das wirklich so? Ist es gleichgültig, ob ich schreibe: „Dies ist ein Blindtext“ oder „Huardest gefburn“? Kjift – mitnichten! Ein Blindtext bietet mir wichtige Informationen. An ihm messe ich die Lesbarkeit einer Schrift, ihre Anmutung, wie harmonisch die Figuren zueinander stehen und prüfe, wie breit oder schmal sie läuft. Ein Blindtext sollte möglichst viele verschiedene Buchstaben enthalten und in der Originalsprache gesetzt sein. Er muss keinen Sinn ergeben, sollte aber lesbar sein. Fremdsprachige Texte wie „Lorem ipsum“ dienen nicht dem eigentlichen Zweck, da sie eine falsche Anmutung vermitteln. Dies hier ist ein Blindtext zum Testen von Textausgaben. Wer diesen Text liest, ist selbst schuld. Der Text gibt lediglich den Grauwert der Schrift an. Ist das wirklich so? Ist es gleichgültig, ob ich schreibe: „Dies ist ein Blindtext“ oder „Huardest gefburn“? Kjift – mitnichten! Ein Blindtext bietet mir wichtige Informationen. An ihm messe ich die Lesbarkeit einer Schrift, ihre Anmutung, wie harmonisch die Figuren zueinander stehen und prüfe, wie breit oder schmal sie läuft. Ein Blindtext sollte möglichst viele verschiedene Buchstaben enthalten und in der Originalsprache gesetzt sein. Er muss keinen Sinn ergeben, sollte aber lesbar sein. Fremdsprachige Texte wie „Lorem ipsum“ dienen nicht dem eigentlichen Zweck, da sie eine falsche Anmutung vermitteln. Dies hier ist ein Blindtext zum Testen von Textausgaben. Wer diesen Text liest, ist selbst schuld. Der Text gibt lediglich den Grauwert der Schrift an. Ist das wirklich so? Ist es gleichgültig, ob ich schreibe: „Dies ist ein Blindtext“ oder „Huardest gefburn“? Kjift – mitnichten! Ein Blindtext bietet mir wichtige Informationen. An ihm messe ich die Lesbarkeit einer Schrift, ihre Anmutung, wie harmonisch die Figuren zueinander stehen und prüfe, wie breit oder schmal sie läuft. Ein Blindtext sollte möglichst viele verschiedene Buchstaben enthalten und in der Originalsprache gesetzt sein. Er muss keinen Sinn ergeben, sollte aber lesbar sein. Fremdsprachige Texte wie „Lorem ipsum“ dienen nicht dem eigentlichen Zweck, da sie eine falsche Anmutung vermitteln.

## Kolophon

Dieses Dokument wurde mit der LaTeX-Vorlage für Abschlussarbeiten an der htw saar im Bereich Informatik/Mechatronik-Sensortechnik erstellt (Version 2.1). Die Vorlage wurde von Yves Hary und André Miede entwickelt (mit freundlicher Unterstützung von Thomas Kretschmer, Helmut G. Folz und Martina Lehser). Daten: (F)10.95 – (B)426.79135pt – (H)688.5567pt