Informationen für Bachelor-/Masterarbeiten

Fachgebiet Industrielle Automatisierungstechnik

Dieses Dokument dient als Information für Studierende, welche am Fachgebiet Industrielle Automatisierungstechnik eine Bachelor- oder Masterarbeit verfassen.

Inhalt:

- Bewertungsgrundlage
- Notenschlüssel
- Code-Convention des FG Industrielle Automatisierungstechnik

---

Stand: 02/2021

Institut für Werkzeugmaschinen und Fabrikbetrieb

Fachgebiet Industrielle Automatisierungstechnik
Prof. Dr.-Ing. Jörg Krüger

Technische Universität Berlin

Bewertungsgrundlage

Ordnung zur Regelung des allgemeinen Studien- und Prüfungsverfahrens, (AllgStuPo), 2013

§ 46 Abschlussarbeiten
(1) Die Abschlussarbeit ist eine Prüfungsarbeit und zugleich Teil der wissenschaftlichen Ausbildung. Mit ihr soll die Kandidatin oder der Kandidat zeigen, dass sie oder er in der Lage ist, innerhalb einer vorgegebenen Frist ein Problem aus ihrem oder seinem Studiengang selbstständig nach wissenschaftlichen Methoden zu bearbeiten. […]

In der folgenden Tabelle sind aus der allgemeinen Formulierung zu Abschlussarbeiten abgeleitete Bewertungskriterien aufgeführt, welche der Beurteilung von am FG Industrielle Automatisierungstechnik verfassten Arbeiten dienen. Hierbei ist zu beachten, dass die genannten Teilaspekte lediglich für eine gemeingültige Orientierung dienen und keinen Anspruch auf Vollständigkeit für eine einzelne, individuelle Abschlussarbeit erheben.

	Gewicht
Wissenschaftliche Arbeitsweise	33%
Die zentralen Herausforderungen und Ziele (Forschungsfragen) sind explizit und exakt formuliert	
Sämtliche Teile der Aufgabenstellung werden adressiert	
Erreichte Ergebnisse werden entsprechend der Zielsetzungen methodisch bewertet	
Eigene Ergebnisse werden klar von bestehenden Erkenntnissen abgegrenzt	
Unerwartete Ergebnisse werden kritisch hinterfragt	
Kritische Aussagen bzw. Behauptungen werden durchweg mit Quellen belegt	
Sämtliche wichtigen bzw. bekannten Ansätze wurden im Stand der Technik berücksichtigt	
Es werden nur zitierfähige Quellen genutzt (kein Wikipedia o.ä.), außerdem werden überwiegend Originalquellen statt Sekundärliteratur referenziert.	
Die Wahl der genutzten Lösungsmethode (falls nicht vorgegeben) wird hinreichend begründet.	
Eine etwaige Implementierung ist gut strukturiert und besteht aus gut lesbarem Code.	
…	
Eigenständigkeit der Arbeit	33%
Der Kandidat entwickelt eigenständig neue Ideen und Lösungsansätze	
Bestehende/vorgeschlagene Ansätze werden weiterentwickelt und angepasst	
Der Kandidat war engagiert und hat selbstständig gearbeitet ohne ständig detaillierte Instruktionen vom Betreuer zu benötigen	
Neue Literatur/Quellen wurden selbstständig gesucht, anstatt lediglich vorgeschlagene Literatur zu verwenden	
…	
Darstellung im Schrifttum	33%
Die Gliederungsstruktur verfolgt einen klaren roten Faden	
Der rote Faden der Arbeit wird durch Kurzeinführungen und/oder Zusammenfassungen an wichtigen Stellen unterstützt	
Alle Abschnitte der Arbeit sind für die Ziele relevant und tragen zum Thema bei	

Die Ausdrucksweise ist prägnant und präzise, ohne umgangssprachliche Formulierungen oder lange, verschachtelte Satzkonstruktionen zu verwenden	
Auch schwierige Sachverhalte werden in einer leicht verständlichen Form beschrieben	
Der Text wird an geeigneten Stellen durch Übersichtstabellen und Abbildungen veranschaulicht	
Im Text wird auf die Abbildungen eingegangen und verwiesen	
Die Abbildungen und Tabellen sind in einem einheitlichen Design gestaltet	
Die Qualität der Abbildungen ist gut, d.h. gute Auswahl der Darstellungsform, gut lesbare Schrift innerhalb der Abb., sinnvolle Beschriftungen die zum Verständnis der Abb. ausreichen	
Es gibt ein übersichtliches Abbildungs- und Tabellenverzeichnis	
Begriffe, Bezeichnungen und Formelzeichen werden einheitlich und konsistent benutzt	
Abkürzungen/Formelzeichen sind in entsprechenden Verzeichnissen zusammengefasst	
Das Dokument ist frei von Rechtschreib- und Tippfehlern	
Die Formatierung des Dokuments ist ansprechend und ermöglicht eine gute Übersicht und Lesbarkeit	
…	
Sonstiges / Besonderheiten Besonderheiten können die Leistung auf- oder abwerten. Abwertende Besonderheiten sind mit negativen Prozentangaben zu vermerken. Insgesamt haben die Besonderheiten einen Einfluss von maximal ±5 Prozentpunkten, d.h. einem Notenpunkt.	**Bonus/ Malus**
Keine Besonderheiten = 0%	
Die Arbeit wurde in der Verteidigung sehr gut präsentiert und die Fragen wurden kompetent beantwortet	
Die Situation bei der externen Firma erschwerte dem Kandidaten die Arbeit	
Das Thema erforderte ein hohes Maß an theoretischen (Vor-)Betrachtungen	
Das Thema erforderte ein hohes Maß an Literaturrecherche oder es stand nur sehr wenig passende Literatur zur Verfügung	
Der vereinbarte Zeitplan wurde ohne besonderen Grund deutlich überschritten	
Es sind besondere, nicht abzusehende Schwierigkeiten während der Bearbeitungszeit aufgetreten	
…	
Gesamtbewertung	**Prozent:** **Note:**

Notenschlüssel

Aus der oben stehenden Tabelle ergibt sich eine Gesamtbewertung der Abschlussarbeit in Prozent. Daraus wird die Note gemäß des von der Ausbildungskommission der Fakultät V empfohlenen Schlüssels ermittelt:

Prozent	Note
mehr oder gleich 95,0 %	1,0
mehr oder gleich 90,0 %	1,3
mehr oder gleich 85,0 %	1,7
mehr oder gleich 80,0 %	2,0
mehr oder gleich 75,0 %	2,3
mehr oder gleich 70,0 %	2,7
mehr oder gleich 65,0 %	3,0
mehr oder gleich 60,0 %	3,3
mehr oder gleich 55,0 %	3,7
mehr oder gleich 50,0 %	4,0
weniger als 50,0 %	5,0

Das Urteil und dessen Definition ergeben sich auf Basis der Note nach §47, AllgStuPo 2013:

Note	Urteil	Definition
1,0 1,3	sehr gut	eine hervorragende Leistung
1,7 2,0 2,3	gut	eine über den durchschnittlichen Anforderungen liegende Leistung
2,7 3,0 3,3	befriedigend	eine Leistung, die insgesamt durchschnittlichen Anforderungen entspricht
3,7 4,0	ausreichend	eine Leistung, die trotz Mängeln den Anforderungen noch entspricht
5,0	nicht ausreichend	eine Leistung mit erheblichen Mängeln, die den Anforderungen nicht entspricht

Code-Convention des FG Industrielle Automatisierungstechnik

Guter Quelltext „hält länger“ und ist leichter zu warten. Diese Konvention beschäftigt sich mit dem Aussehen und der Struktur von Quelltext. Diese haben großen Einfluss auf die Lesbarkeit und Verständnis und damit auf die Wartung und Lebensdauer von früher investierter Arbeitszeit. Grob gesagt ist Ziel jeder einzelnen Regel, „schlechtes Verhalten“ zu erschweren und „gutes Verhalten“ zu fördern.

Grundlegende Regeln zum Programmieren

1. Optimiere den Code für den Leser, nicht für den Schreiber.
2. Jede Regel soll in der Bilanz einen positiven Effekt haben.
3. Wertschätze den Standard, aber vergöttere ihn nicht.
4. Sei konstant und konsequent im Umgang mit Regeln.
5. Wenn etwas „Ungewöhnliches“ passiert, hinterlasse Hinweise für den Leser.
6. Verhindere Konstrukte die überraschend oder gefährlich sind (z.B. side effects).
7. Halte den globalen Namensraum schmal.
8. Optimiere deinen Quelltext auf Kosten der Lesbarkeit erst dann, wenn es Anlass dazu gibt. (Dazu zählen beispielsweise Laufzeit, Speicherbedarf und Flexibilität.)

Datei-Grundlagen

- Alle Quell-Dateien werden in UTF-8 Codiert.
- Whitespaces:
  - Nur Zeilenende („Enter“) und Leerzeichen ( ) sind im Quelltext erlaubt.
  - Alle anderen „unsichtbaren“ Zeichen sind im Quelltext nur durch ihre Ersetzung mit Fluchtzeichen (Tab = \t) erlaubt.
  - Tabulatoren („Tab(s)“) sind als Zeichen explizit nicht erlaubt, da es keinen einheitlichen Standard bei Editoren für die Tab-Breite gibt.
  - Viele Editoren können allerdings Tabs automatisch in Leerzeichen übersetzen, sodass man beim Editieren zwar wie gewohnt Tabs verwenden kann, aber die Datei nur die Leerzeichen-Ersetzung enthält.
  - Editoreinstellung: Tabs werden in 4 Leerzeichen übersetzt
- Nicht-ASCII-Zeichen dürfen ausdrücklich im Quellcode erscheinen. Eine Ersetzung ist aufgrund der schlechteren Lesbarkeit zu vermeiden (z. B. "µs" statt "\u03bcs").
- Aber: Bezeichner bitte nur in ASCII

FALSCH	RICHTIG
`public class Tür {`	`public class Tuer {`

Formatierung von Klassen (objektorientierte Sprachen)

- Die Member einer Klasse sollten *einer* Ordnung folgen. Diese kann selbst gewählt werden, sollte sich aber nach inhaltlichen und nicht syntaktischen Kriterien richten, d. h. keine Sortierung nach static-Membern o. ä.

FALSCH

```
public static final int STANDARD_TUER = 0;
private int typ;
public static final int SCHIEBE_TUER = 1;
private static int instanceCounter;
private boolean klinke_vorhanden;
public static final int DREH_TUER = 2;
private int oeffnung;
protected static final float VERSION=1.0f;
```

RICHTIG

```
protected static final float VERSION=1.0f;

public static final int STANDARD_TUER = 0;
public static final int SCHIEBE_TUER = 1;
public static final int DREH_TUER = 2;

private static int instanceCounter;

private int typ;
private int oeffnung;
private boolean klinke_vorhanden;
```

- Überladene Methoden/Konstruktoren dürfen nicht getrennt werden.

FALSCH

```
public void oeffne() { … }
public void schliesse() { … }
public void oeffne(double winkel) { … }
public void schliesse(double winkel) { … }
```

RICHTIG

```
public void oeffne() { … }
public void oeffne(double winkel) { … }
public void schliesse() { … }
public void schliesse(double winkel) { … }
```

Klammersetzung

- Es wird dringend empfohlen die geschweiften Klammern immer zu setzen!
- **Warum?** - Oftmals ist es möglich eine einzelne Anweisung statt eines Blocks direkt hinter eine Bedingung zu schreiben. Das ist sehr kompakt, kann aber dadurch auch zu schwer erkennbaren Fehlern führen. Insbesondere bei nachträglichen Bearbeitungen und fehlerhaften Formatierungen ist dann nicht klar, was innerhalb und was außerhalb des Blocks ist.

One Statement per Line

- Eine Zeile enthält nur einen Befehl bzw. Bedingung.
- Mathematische Ausdrücke bilden eine Ausnahme, sofern diese erkennbar oder dokumentiert sind. Die Dokumentation kann mit Verweis auch extern existieren.

Zeilenende

- Aufgrund der Beschränkung der Zeilenbreite einiger Editoren, bzw. aufgrund Unterschiedlicher Arbeitsbedingungen (Monitor- /Auflösungsbreite) sollen Zeilen nicht mehr als 100 Zeichen enthalten (Projektindividuell auch 80).

Bezeichner

- Variablen-, Klassen- und Funktionsnamen und andere Bezeichner haben stets ihre Bedeutung als Bezeichnung. Ausnahmen sind:
  - Laufvariablen, sofern sie nicht in Begleitung anderer anonymer Variablen sind oder
  - technische oder mathematische Bezeichner, deren Bedeutung sich z. B. aus Formeln und damit anderer Dokumentation ergibt.

Exceptions

- Ausnahmen dürfen auch nur als solche auftreten und dürfen nicht Teil der Programmlogik sein.
- Wenn eine Methode Exceptions werfen kann, dann muss auch der Header darauf hinweisen (Indikator *throws*).
- Java: Wirft ein Statement wahrscheinliche eine RuntimeException, dann muss diese vor Ort auch behandelt werden, oder als Nicht-RuntimeException weitergeleitet werden.
- C/C++: Exceptions sind im Allgemeinen zu vermeiden und wahrscheinlich auftretende Exceptions direkt zu behandeln, da der Compiler nicht ihr potentielles Auftreten prüft.
- Wahrscheinliche Exception (Java-Beispiel):
  - `Double.parseDouble(String s) throws NumberFormatException`

Acknowledgements

- Datei „Acknowledgements.txt“ im Softwarepaket hinzufügen
- Template: „Dieser Quellcode entstand im Rahmen eines von [Geldgeber/Auftraggeber] geförderten Projektes [Projektname/Akronym, Projektnummer/Förderkennzeichen].“
- Autor(en) nennen (inklusive Konzeptgeber, etc.)
- Außerdem Englische Übersetzung

Lizenz

- WICHTIG: Eine Softwarelizenz ist immer anzugeben, da Dritte sonst weder die kompilierte Software, noch den Quelltext verwenden dürfen! Es ist immer (am besten automatisiert) die Lizenz anzugeben, unter der dieser Quelltext (und später auch das Kompilat) steht. Fehlt eine Lizenz und das führt bei späteren Kunden zu Schadensersatzforderungen, so kann man durch Rechtemängelhaftung auch finanziell zur Verantwortung gezogen werden.

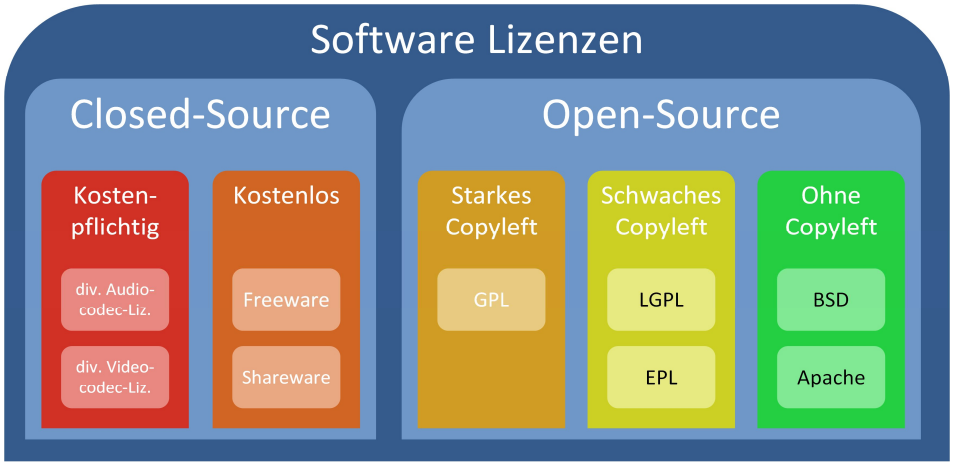

Abbildung 1: Übersicht Softwarelizenzen

- Sofern keine expliziten Gründe dagegen sprechen, ist die Fachgebiets-Lizenz zu verwenden (BSD 3-clause license – siehe Anhang). Diese ist auch kompatibel zur GPL-Lizenz, man kann die Software also gleichzeitig unter beiden Lizenzen veröffentlichen.

- Umgang mit Fremdlizenzen (siehe Abb. 1):
  - **Copyleft**-Lizenzen (z.B. GPL, LGPL, Mozilla Public License), vor allem solche mit starkem Copyleft, sind im Allgemeinen kritisch wenn es um die Weitergabe der Software sowie des (erweiterten) Quelltextes geht. Die Lizenz wird hier u. U. **vererbt**, d. h. man muss bei deren Weitergabe oft die eigene Software unter die gleiche Lizenz stellen sowie den Quelltext veröffentlichen. Die Verwendung sollte **immer mit dem jeweiligen Betreuer/Projektpartner absprechen.**

  - **Proprietäre Software(-Pakete)** verbieten in der Regel die kostenfreie Nutzung oder stellen sie unter kritische Auflagen und sind damit noch schwieriger zu verwenden als GPL-lizenzierte Software. Im Allgemeinen sind sie zu vermeiden, bzw. auf Ersatz zu überprüfen. Jede Verwendung muss mit dem Betreuer/Projektpartner geklärt werden.

- Häufige Pflichten bei Verwendung von Freien- und Open-Source-Softwarelizenzen (FOSS-Lizenzen):
  - Bei Verbreitung sind **ALLE beteiligten Urheber** zu nennen, also auch die Urheber aller verwendeten Bibliotheken. (Mitliefern des Quelltexts umgeht das Problem.)
  - Bei Verbreitung ist der **Lizenztext** einer Bibliothek oder eines Programms mitzuliefern.
  - Bei Verbreitung ist der **Haftungsausschluss** (engl. Disclaimer, meist in Großbuchstaben und Teil des Lizenztexts) mitzuliefern.
  - LGPL - obwohl nur mit „schwachem“ Copyleft versehen, ist hier dennoch Vorsicht geboten:
    - Der LGPL-Teil muss austauschbar bleiben.
    - Das Debuggen und Reverse Engineering vom LGPL-Teil darf nicht untersagt werden.
    - Durch dynamisches Linken kann so eine Trennung erfolgen, aber: auch dynamisch verlinkte Bibliotheken können den Status „abgeleitet“ erzeugen und ggf. das Copyleft auslösen.

Fachgebiets-Lizenz (BSD)