

# Terminal zur Bearbeitung von ARX-Daten mit R

ALEXANDER BEISCHL UND THUY TRAN  
*Technische Universität München*  
27. Januar 2017

## **Zusammenfassung**

Im Rahmen des Klinisches Anwendungsprojektes wurde die *Java*-Software *R-Terminal* implementiert. Dieses Programm integriert die Programmiersprache und Entwicklungsumgebung *R* in *Java*, sodass in der graphischen Benutzeroberfläche des *R-Terminals* Befehle und Code eingegeben werden können, welche von *R* ausgeführt werden.

# Inhaltsverzeichnis

# Kapitel 1

## Einführung

Für die Arbeit von Tabellen aus ARX, welche sich mit medizinischen Daten befassen, bietet sich R aufgrund der Unterstützung statistischer Funktionen an. Allerdings können ARX Tabellen nicht direkt in der R GUI eingelesen werden, was nicht technische, sondern rechtliche Gründe hat. ARX ist unter der Apache License lizenziert, während R als GNU Projekt unter GPL (GNU General Public License) lizenziert ist.

Obwohl beide Lizenzen für freie Software genutzt werden, welche den Nutzern kaum Restriktionen im Umgang mit der Software auferlegen, sind sie nicht zueinander kompatibel. Während Software unter Apache License in GPL Projekten integriert werden dürfen, umgekehrt GPL lässt dies nicht zu.

Diese Inkompatibilität kommt zustande, wenn Software unter Apache mit von GPL Software abgeleiteten Projekten geschrieben wird. Dadurch muss es unter GPL gestellt werden. In diesem Falle müsste ARX als Apache License Projekt unter GPL gestellt werden, um es mit R zusammen zu verwenden. Dies wiederum lässt die Apache License nicht zu, da jede Apache Software unter Apache lizenziert werden muss. Es ist weitgehend von GPL-Autoren interpretationsbedingt, was ein von GPL Software abgeleitetes Projekt ist. Um hierbei Komplikationen zu vermeiden, sollte nicht mal von Apache Software zu GPL Software verlinkt werden [?].

Das Ziel des RTerminals ist es, eine lizenzfreie Plattform zu bieten. Dadurch kann R aufgerufen werden und innerhalb des RTerminals ARX Daten eingelesen und bearbeitet werden. TODO Fabian bzgl. Lizenzen fragen + Einführung schreiben

## R

R ist eine Programmiersprache und Entwicklungsumgebung, die für statistische Berechnungen und Graphen unter John Chambers von Bell Laboratories entwickelt wurde. Sie ist ein *GNU*-Projekt, bei dem die Entwicklung von freier Software im Mittelpunkt steht. *R* weist große Ähnlichkeiten zu der Programmiersprache *S* auf, einem weiteren *GNU*-Projekt, welches weitgehend auch unter *R* läuft. [3]

*R* bietet standardmäßig alle Hauptfunktionen für die statistische Analyse von Datensätzen an und ist einfach zu erweitern, weswegen es vor allem für wissenschaftliche Arbeiten verwendet wird. Es ist mit *R* einfach, statistische Funktionen auf große Datenmengen anzuwenden. [3]

## ARX

*ARX* ist eine freie Software zur Anonymisierung von medizinischen Datensätzen, die von Fabian Prasser und Florian Kohlmayer vom Institut für medizinische Statistik und Epidemiologie an der Technischen Universität München entwickelt wurde. [?]

## Vorgaben

Für die Implementierung der Software wurde *Java* als Programmiersprache vorgegeben, außerdem sollte die graphische Benutzeroberfläche mit *SWT* realisiert werden.

Die graphische Benutzeroberfläche sollte aus unterschiedlichen Tabs aufgebaut sein. Ein Tab beinhaltet folgende Informationen zum Setup: verwendetes Betriebssystem, aktuell integrierte Version von *R* und Installationsort der *R*-Version. In einem weiteren Tab wird der Std-Output von *R* ausgegeben und *R*-Befehle können in diesem Tab eingegeben werden.

Beim Programmstart soll automatisch in den Standard-Installationsorten nach einer ausführbaren *R*-Executive gesucht werden. Außerdem soll die Möglichkeit bestehen, manuell eine *R*-Installation auszuwählen und diese mit der Software in *Java* zu integrieren.

# Kapitel 2

## Anwenderdokumentation

### 2.1 Übersicht

Das *R-Terminal* dient dazu, über eine externe Schnittstelle *R* aufzurufen und zu bedienen. Dies soll vor allem dazu genutzt werden, um Tabellen aus *ARX* einzuladen, auf diesen Skripte auszuführen, und gleichzeitig die in der Einführung beschriebenen Probleme zu umgehen. Das *R-Terminal* ist kompatibel mit Windows, der Linux Distribution Ubuntu und OS X, von denen jeweils die folgenden Versionen im Rahmen der Entwicklung getestet wurden:

- Windows 10 Education (Version 1511)
- OS X El Capitan (Version 10.11.1)
- macOS Sierra (Version 10.12.2)
- Ubuntu

In den folgenden Abschnitten werden die Grundfunktionen und zusätzliche Features des *R-Terminals* beschrieben.

### 2.2 Voraussetzungen

Um die Software *R-Terminal* auszuführen, muss *Java* sowie eine Standalone-Installation von *R* installiert sein. Es wird empfohlen das *R-Terminal* mit Eclipse auszuführen. Die Versionen, mit welchen die Software getestet wurde, sind unter ?? aufgeführt.

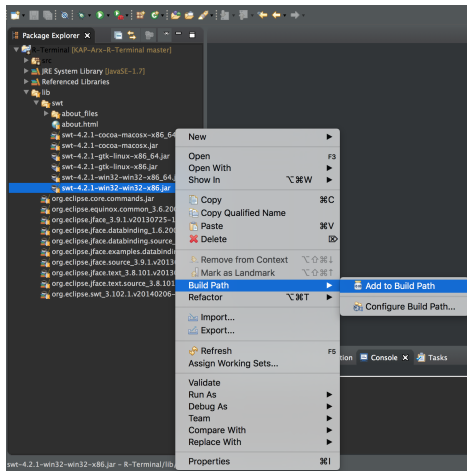


Abbildung 2.1: Eclipse: Hinzufügen der SWT-Library zum Build Path

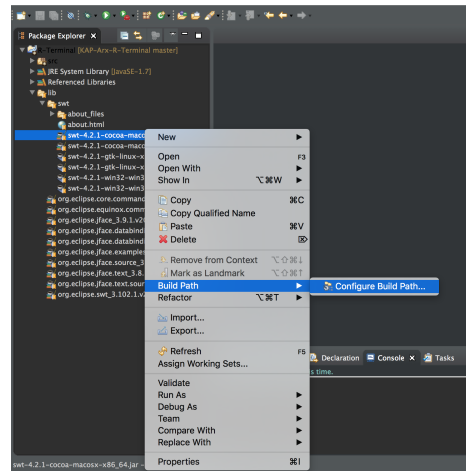


Abbildung 2.2: Eclipse: Diese SWT-Library ist bereits auf dem Build Path

## 2.3 Vorbereitung

Zunächst muss das Projekt *R-Terminal* in Eclipse importiert werden.

Um das *R-Terminal* ausführen zu können, müssen die betriebssystemspezifischen SWT-Libraries auf den Bild-Path gesetzt und die SWT-Libraries anderer Betriebssysteme entfernt werden. Im Folgenden wird genau beschrieben, wie dies unter *Eclipse* umgesetzt wird:

1. Navigieren Sie im *Eclipse* Package Explorer zu:

*R-Terminal* -> *lib* -> *swt*

2. Wählen Sie die betriebssystemspezifischen SWT-Libraries aus und öffnen Sie mit der rechten Maustaste, bzw. dem OS-spezifischen Äquivalent, für jede dieser Libraries das Auswahlfenster. Klicken Sie auf „Build-Path“. Sollte die Library bereits hinzugefügt sein, erscheint „Configure Build Path...“. Ist dies nicht der Fall, erscheint „Add to Build Path“, klicken Sie dieses an. Die beiden Fälle werden in Abbildung ?? und ?? dargestellt.

Betriebssystemspezifische SWT-Libraries:

- Windows:

- swt-4.2.1-win32-win32-x86\_64.jar
  - swt-4.2.1-win32-win32-x86.jar
  - Mac:
    - swt-4.2.1-cocoa-macosx-x86\_64.jar
    - swt-4.2.1-cocoa-macosx.jar
  - Linux:
    - swt-4.2.1-gtk-linux-x86\_64.jar
    - swt-4.2.1-gtk-linux-x86.jar
3. Überprüfen Sie nun, ob alle anderen Libraries des Ordners *swt* nicht auf dem „Build Path“ liegen. Gehen Sie hierfür zu:

*R-Terminal -> Referenced Libraries*

Überprüfen Sie, dass keine der genannten Libraries der oberen Aufzählung, die nicht Ihrem Betriebssystem zugehört, in diesem Ordner enthalten ist. Sollte sich eine SWT-Library für ein anderes Betriebssystem in diesem Ordner befinden, öffnen Sie mit einem Rechtsklick das Menü und wählen Sie aus:

*Build Path -> Remove from Build Path*

Dies wird auch in Abbildung ?? dargestellt.

Nun können Sie das *Java*-Programm Sie die „main“-Methode in *RMain* ausführen.

## 2.4 Benutzung des R-Terminals

In diesem Kapitel wird die Benutzung des *R-Terminals* beschrieben. Hierfür wird zunächst ein Überblick über die graphische Benutzeroberfläche in ?? und ?? gegeben. Außerdem wird die Nutzung gemeinsam mit dem Programm *ARX* in ?? erläutert.

Diese wird für das Betriebssystem OS X in Abbildung



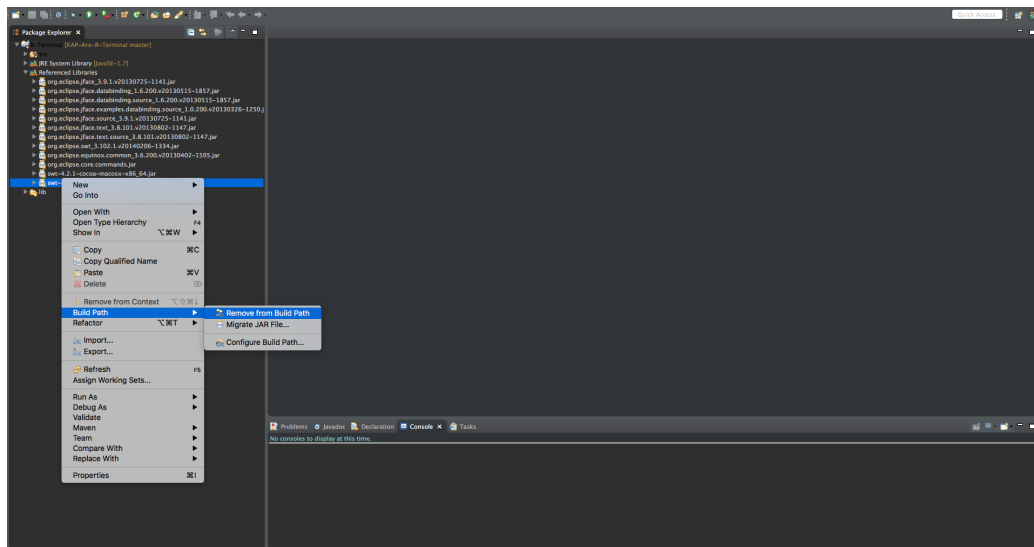


Abbildung 2.3: Eclipse: Entfernen der SWT-Library vom Build Path

## 2.4.1 Setup

Unter *Setup* wird entweder eine *R*-Version auf dem Rechner gesucht und automatisch ausgeführt oder es wird vom Benutzer selber der Pfad zu der gewünschten *R*-Version angegeben.

Hierbei gibt das Feld unter *Operating System* das Betriebssystem der benutzten Plattform an. Das Feld *Location* gibt den Pfad an, unter dem die automatische Suche die ausführbare *R*-Version gefunden hat. Konnte keine ausführbare *R*-Version gefunden werden, wird in dem Feld die Nachricht *No valid R-exec found!* angezeigt.

Das mit *Version* betitelte Feld gibt die Version an, die die gefundene *R*-Version hat. Wurde keine ausführbare *R*-Version gefunden oder eine ungültige *R*-Version angegeben, zeigt das Feld die Ausgabe *No valid R-Version selected*.

Sind auf dem Rechner mehrere *R*-Versionen vorhanden, von denen der Benutzer eine andere Version bevorzugt, als die durch die automatische Suche gefundene, gibt es die beiden unteren, im folgenden erläuterten Felder als Optionen.

*Select the R-Exec manually* öffnet eine Ordnerübersicht, durch die dann die zu der gewünschten *R*-Version navigiert werden kann. Hierbei sollte beachtet werden, dass nicht der Pfad zu der *R-GUI* angegeben werden sollte, welche oft zusammen mit dem *R-Executable* installiert wird, sondern zu der ausführbaren *R*-Datei.

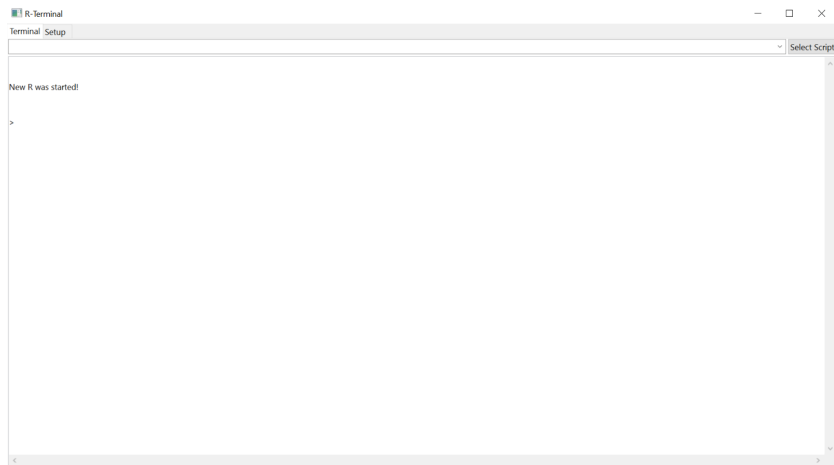


Abbildung 2.4: R-Terminal: *Terminal* unter Windows 10 Education (Version 1511)



Abbildung 2.5: R-Terminal: *Setup* unter Windows 10 Education (Version 1511)

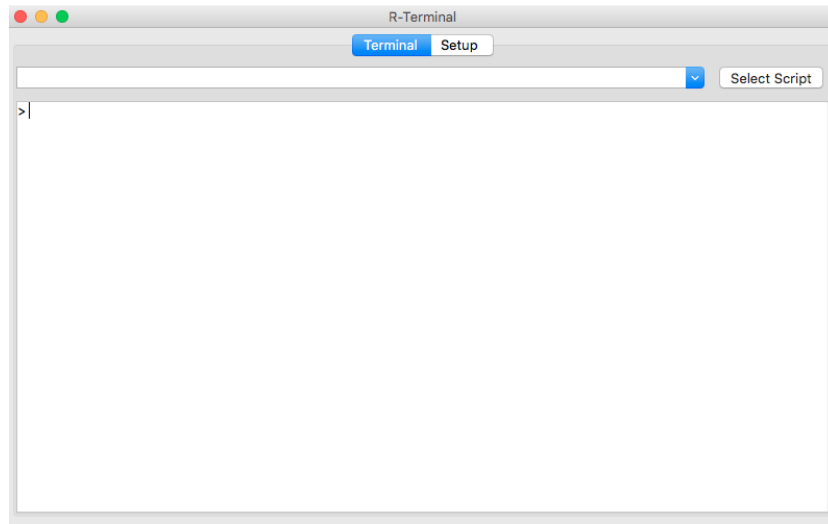


Abbildung 2.6: *R-Terminal: Terminal* unter OS X (Version 10.11.1)

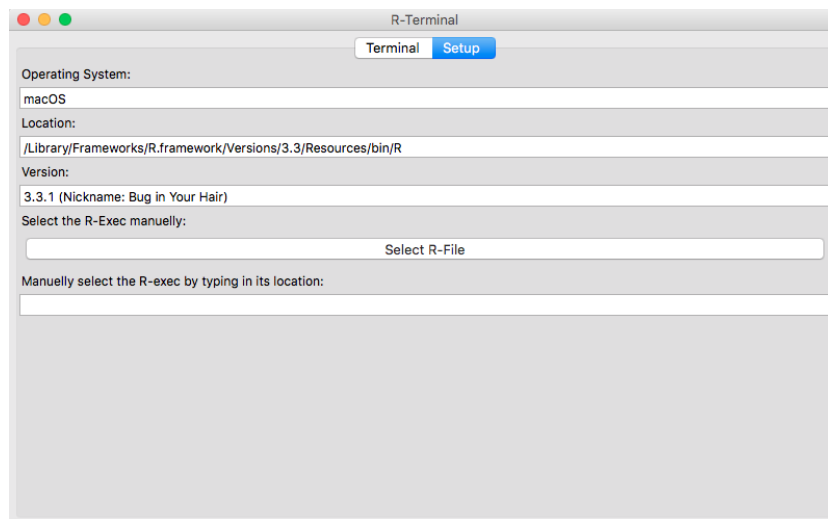


Abbildung 2.7: *R-Terminal: Setup* unter OS X (Version 10.11.1)

Das letzte Feld *Manually select the R-exec by typing in its location* gibt dem Benutzer die Möglichkeit, den bereits bekannten Dateipfad zur ausführbaren R-Datei anzugeben. Hierbei sollte ein absoluter Dateipfad angegeben werden, der die ausführbare R-Datei enthält (im Gegensatz dazu, nur den Ordner anzugeben, in dem die Datei liegt).

Wird in *Select the R-Exec manually* oder in *Manually select the R-exec by typing in its location* eine ungültige R Version oder ein ungültiger Dateipfad angegeben, wird unter *Operating System* und *Version* dieselbe Fehlermeldung angegeben, wie oben bei dem Fehlschlagen der automatischen Suche beschrieben.

## 2.4.2 Terminal

Nachdem die gewünschte R-Version gefunden wurde, kann diese nun unter dem Tab *Terminal* bedient werden. Das Eingabefeld kann benutzt werden, um R-kompatible Befehle auszuführen. Es werden bis zu 10 Befehle gespeichert. Diese können ausgewählt werden, indem auf den Pfeil auf der rechten Seite des Eingabefeldes geklickt wird. Dadurch öffnet sich ein Dropdown-Menü, aus dem der gewünschte Befehl ausgewählt werden kann. Dieser erscheint im Eingabefeld.

## 2.5 Benutzung von R für ARX

Dieser Abschnitt wird sich mit der Benutzung von R im Zusammenhang mit großen Datentabellen befassen, wie sie in ARX üblich sind. Dies soll die Benutzung von dem R-Terminal zur Verarbeitung von ARX-Daten ermöglichen.

### 2.5.1 Datentypen und Skalenniveau

In der medizinischen Statistik ist die Unterscheidung von den verschiedenen Stufen der Skalierbarkeit von großer Bedeutung. Für die Daten aus ARX sind vor allem die Eigenschaften wichtig, die Patientenmerkmale besitzen. So können Merkmale in quantitative und qualitative Merkmale unterteilt werden [?]. Es ist kein Problem, quantitative Merkmale darzustellen, da diese einfach durch numerische Werte ausgedrückt werden können. Die Grunddatentypen sind wie folgt:

- Numeric

- Integer
- Complex
- Logic

Wenn also eine Variable als Dezimalzahl ohne zusätzliche Bedingung deklariert wird, ist der Datentyp standardmäßig `numeric` (vgl. ??).

R-Befehl 2.1: Deklaration einer numerischen Variable `x`

```
> x = 13.4
```

Will man explizit einen Integer als Variable initialisieren, sieht der Befehl aus wie in ??. Es sollte zusätzlich beachtet werden, dass `TRUE` und `FALSE` in R jeweils die Ganzzahlwerte 1 und 0 besitzen.

R-Befehl 2.2: Deklaration einer ganzzahligen Variable `y`

```
> y = as.integer(4.5)
> y      # gibt y als Wert aus
[1] 4
```

Um eine Variable mit dem Typ `complex` zu initialisieren, wird der imaginäre Teil der komplexen Zahl bei der Initialisierung angegeben. Ein Beispiel dazu ist in ?? zu sehen.

R-Befehl 2.3: Deklaration einer komplexen Variable `z`

```
> z = 2 + 5i
```

Die Datentypen können jeweils durch den Befehl `class()` abgefragt werden. Die Ausgaben sind in ?? zu sehen.

R-Befehl 2.4: Abfrage des Datentypen

```
> class(x)
"numeric"
> class(y)
"integer"
> class(z)
"complex"
```

Nachdem nun die quantitativen Merkmale dargestellt werden können, stellt sich die Frage nach der Darstellung der qualitativen Merkmale.

Die qualitativen Merkmale werden in nominalskalierte und ordinalskalierte Merkmale unterteilt. Ordinalskalierte Merkmale haben eine vorbestimmte Ordnung untereinander, die nicht zwangsläufig eine lineare

Beziehung zueinander haben müssen. Nominalskalierte Merkmale sind hingegen eindeutige Kategorien, die untereinander keine Ordnung haben.

Die einfachste Möglichkeit, einer Variable ein qualitatives Merkmal zuzuordnen, ist sie als `character` zu definieren. Dies geschieht durch eine einfache Zuweisung wie in ??.

#### R-Befehl 2.5: Deklaration einer qualitativen Variable a

```
> a = "verheiratet"      #nominalskaliertes Merkmal
> class(a)
"character"
> b = "befriedigend"     #ordinalskaliertes Merkmal
> class(b)
"character"
```

Dies ist allerdings keine optimale Lösung, da hierdurch keine eindeutige, abgegrenzte Kategorisierung der Variablen stattfindet. Im Falle der ordinalskalierten Variablen kann auch keine Ordnung unter den Variablen festgelegt werden. Möchte man ein Histogramm erstellen, was bei qualitativen Merkmalen sehr wahrscheinlich ist, werden die Strings nicht als Instanz eines Attributs gezählt, sondern einfach nur als Strings. Um also eine bessere Darstellung für qualitative Merkmale zu erreichen, gibt es sogenannte `factor variables`. Hierbei kann ein Vektor aus Integers erstellt werden, wo jedem Integer ein `label` zugewiesen wird. Ein Beispiel dazu ist in ?? zu sehen.

#### R-Befehl 2.6: Erstellung einer factor variable

```
> v=c(0, 1, 1, 0, 1) #Vektorinitialisierung
> v
[1] 0 1 1 0 1          #Ausgabe von v
> einheiten=factor(v, labels=c("metrisch", "imperial"))
> einheiten
[1] metrisch imperial imperial metrisch imperial
Levels: metrisch imperial
```

Nun konnte den verschiedenen Werten im Vektor `v` ein `label` zugeordnet werden. Zu beachten ist hierbei, dass der `labels`-Vektor entweder 1 oder genauso viele Elemente beinhaltet, wie verschiedene Werte im Basisvektor `v` enthalten sind. Die Zeile `Levels` gibt die Ordnung der Merkmale an. Hierbei wurde *metrisch* auf den ersten Wert gesetzt, der im Vektor `v` vorkommt, *imperial* auf den ersten Wert, der von *metrisch* verschieden ist.

Dadurch konnte eine eindeutige Ordnung unter den Merkmalen festgelegt werden, nämlich *imperial* > *metrisch*. Dies ist für dieses einfache Beispiel allerdings sinnlos, da keine Ordnung unter dem metrischen und imperialen System besteht. Natürlich ist es ebenfalls möglich, eine *factor variable* direkt anhand von Strings zu erstellen, s. ??.

#### R-Befehl 2.7: Erstellung einer *factor Variable* mit Strings

```
> w=c("sehr_gut", "gut", "gut", "ausreichend",
      "mangelhaft", "sehr_gut", "befriedigend",
      "gut", "ausreichend", "gut", "befriedigend")
> factor(w)
 [1] sehr gut gut          gut ausreichend mangelhaft
 [6] sehr gut befriedigend gut ausreichend gut
[11] befriedigend
Levels:ausreichend befriedigend gut mangelhaft sehr gut
```

Es wurden mit der *factor*-Funktion direkt *Levels* erstellt, die allerdings nicht die gewünschte Ordnung haben, die wir von den Schulnoten im Beispiel erwartet hätten. Dies ist aufgrund der automatischen alphabetischen Sortierung der *Levels*. Für eine selbst erstellte Ordnung, siehe R-Befehl ??.

#### R-Befehl 2.8: Ordnen einer selbsterstellten *factor Variable*

```
> noten = ordered(w, levels=("sehr_gut", "gut",
                             "befriedigend", "ausreichend", "mangelhaft"))
> noten
 [1] sehr gut gut          gut ausreichend mangelhaft
 [6] sehr gut befriedigend gut ausreichend gut
[11] befriedigend
Levels: sehr gut < gut < befriedigend
< ausreichend < mangelhaft
```

Hier sieht man nun auch eine eindeutige Ordnung unter den Notenstufen [?]. Die Bedeutung der *factor*-Funktion in Anwendung auf Vektoren wird im folgenden Abschnitt erklärt.

Zusätzlich gibt es das *R*-Package *memisc*. Diese Erweiterung weist Objekten, die ein *item* sind, ein *level of measurement* (Skalenniveau) zu, das entweder ein *ordinal*, *nominal*, *interval* oder *ratio* ist. Je nach dem Skalenniveau wird das *item* in einen *factor* (*nominal*), einen geordneten Vector (*ordinal*) oder einen numerischen Vektor (*interval und ratio*) konvertiert.

## 2.5.2 Tabellen

In R wird üblicherweise zur Darstellung von Tabellen ein `data.frame` verwendet.

Ein `data.frame` ist eine Liste von Vektoren, welches mit dem Befehl `data.frame()` erstellt werden kann (s. ??) [?].

R-Befehl 2.9: `data.frame` aus 3 Vektoren

```
> v1 = c(1,2,3)
> v2 = c("a","b","c")
> v3 = c(FALSE, FALSE, TRUE)
> frame = data.frame(v1, v2, v3)
> frame
  v1 v2   v3
1  1  a FALSE
2  2  b FALSE
3  3  c  TRUE
> names(frame) = c("Zahl", "Buchstabe", "Bool")
> frame
  Zahl Buchstabe Bool
1    1         a FALSE
2    2         b FALSE
3    3         c  TRUE
```

Mithilfe der in Abschnitt ?? beschriebenen Funktionen können die Spaltenvektoren in das gewünschte Skalenniveau konvertiert werden.

Für die Nutzung von ARX mit dem *R-Terminal* können die bereits vorhandenen Daten auch aus ARX als CSVs eingeladen werden. Diese werden in ein `data.frame` gelesen, wie in Beispiel ?? gezeigt.

R-Befehl 2.10: Einlesen einer Tabelle

```
frame=read.csv(file="c:/test.csv",header=TRUE,sep=" ",")
```

Die hier verwendeten Argumente sind einmal das `file`, der `header` und `sep`. `file` gibt den Dateipfad zu der CSV-Datei an. Hierbei kann auch ein Pfad angegeben werden, der relativ zum Arbeitsordner ist. `header` gibt an, ob in der Datei die erste Zeile ein Header ist, in dem die Namen der Spaltenattribute stehen, oder ob keine Namen für die Spalten vorhanden sind und in der ersten Zeile direkt der Inhalt der Tabelle beginnt. `sep` gibt den Character an, durch den die Spalten getrennt sind.



Möchte man ein `data.frame` erweitern, z.B. durch eine neue Zeile oder eine neue Spalte, gibt es dafür die Funktionen `rbind` und `cbind`. Die Funktionen sind anzuwenden wie in ?? vorgestellt.

#### R-Befehl 2.11: `rbind` und `cbind`

```
#Hierbei auf die gleichen Spaltennamen achten
> frame2 = data.frame(Zahl=4,Buchstabe="d",Bool=TRUE)
#frame aus data.frame
> newframe=rbind(frame, frame2)
> newframe
  Zahl Buchstabe  Bool
1    1          a FALSE
2    2          b FALSE
3    3          c  TRUE
4    4          d  TRUE
#####
#Hier geht auch ein Vektor
> Nummer = c(4, 3, 2, 1)
> cbind(newframe, Nummer)
  Zahl Buchstabe  Bool Nummer
1    1          a FALSE    4
2    2          b FALSE    3
3    3          c  TRUE    2
4    4          d  TRUE    1
```

# Kapitel 3

## Entwicklerdokumentation

### 3.1 Verwendete Technologien

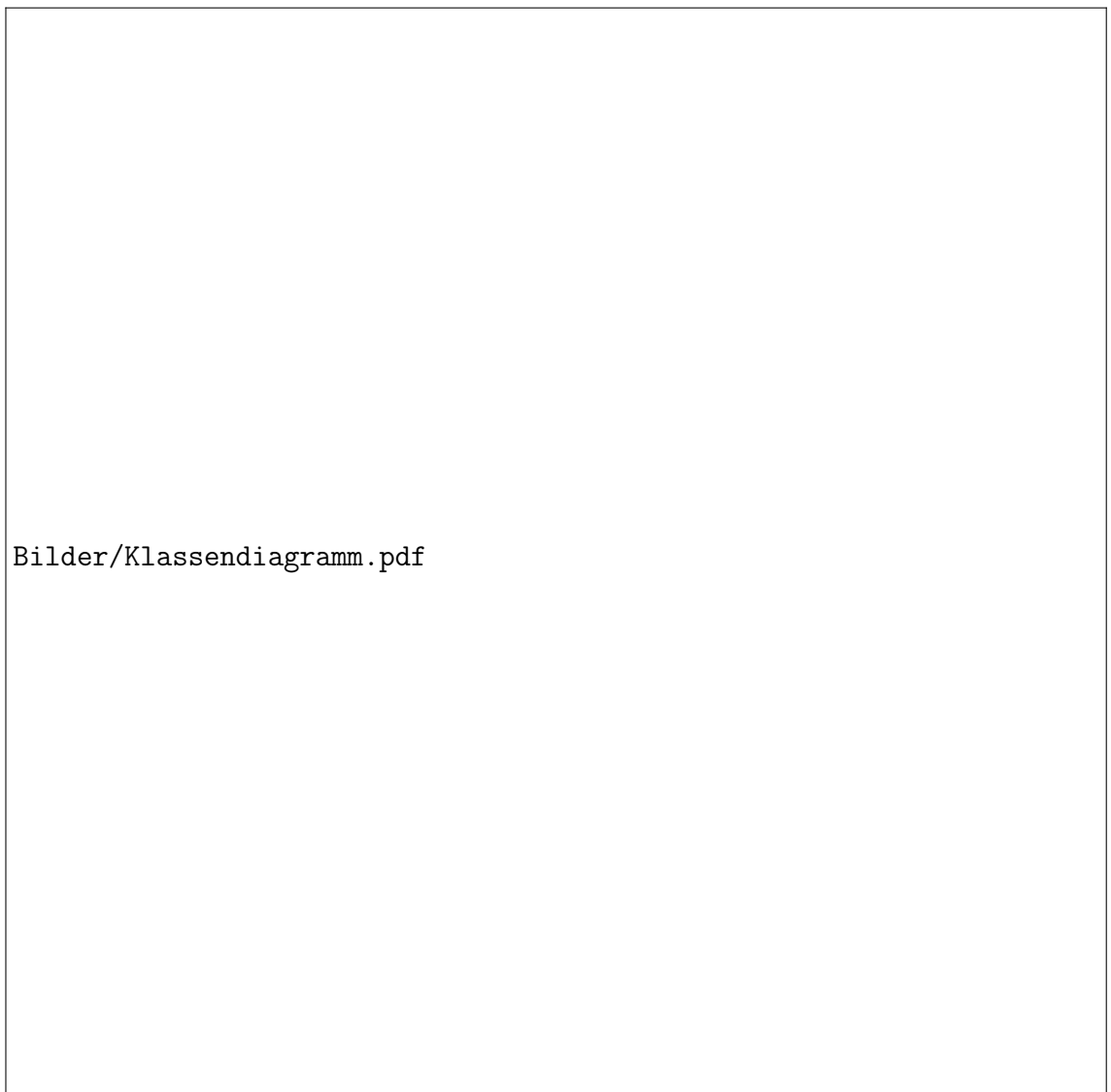
Das *R-Terminal* wurde mit der Entwicklungsumgebung *Eclipse* [?] entwickelt. Als Programmiersprache wurde *Java* [?] gewählt. Die graphische Benutzeroberfläche wurde mit *SWT* [?] realisiert, genauere Details hierzu werden in ?? erläutert. Um *R*-Kommandos auszuführen, startet das hier dokumentierte Programm *R-Terminal* einen *R*-Prozess der Standalone-Installation von *R* durch die *Java*-Library *ProcessBuilder* [?].

### 3.2 Architektur des R-Terminals

Die Software des *R-Terminal* ist in zwei Komponenten aufgeteilt, welche als getrennte Packages vorliegen. Das erste Package „org.deidentifizier.arx.gui“ beinhaltet die graphische Benutzeroberfläche, das Zweite „org.deidentifizier.arx.r“ beinhaltet die gesamte Integration von *R* in *Java*, es stellt also alle funktionalen Methoden zur Verfügung und verwaltet die Kommunikation zwischen *R* und dem *Java*-Programm.

Durch die Trennung von graphischer Benutzeroberfläche und der Integration von *R*, also dem funktionalen Kern, kann die *R*-Integration auch ohne graphische Benutzeroberfläche ausgeführt und verwendet werden. Hierzu wird nur das zweite Package benötigt, welches in ?? behandelt wird. Die Ausführung ohne GUI wird in ?? erläutert.

Alle Klassen der Software *R-Terminal*, sowie deren wichtigsten Methoden, werden in den Kapiteln ?? und ?? vorgestellt.



Bilder/Klassendiagramm.pdf

Abbildung 3.1: *R-Terminal*: Klassendiagramm des Programms

## 3.3 Graphische Benutzeroberfläche (GUI)

Das erste Package, „org.deidentifizier.arx.gui“, erzeugt und verwaltet die graphische Benutzeroberfläche. Es umfasst sieben Klassen:

- *RMain*
- *RTerminal*
- *RSetupTab*
- *RTerminalTab*
- *RBrowserWindow*
- *RLayout*
- *RCommandListener*

### 3.3.1 *RMain*

Um das Programm mit einer graphischen Benutzeroberfläche zu starten, muss die „main“-Methode der Klasse *RMain* ausgeführt werden.

Diese erzeugt ein neues Display sowie eine Shell. Im Anschluss wird ein neues Objekt der Klasse *RTerminal* erzeugt.

### 3.3.2 *RTerminal*

Die Erzeugung der einzelnen Komponenten der graphischen Benutzeroberfläche wird durch den Konstruktor der Klasse *RTerminal* realisiert.

Dieser erzeugt einen *TabFolder* mit zwei Tabs, der erste Tab beinhaltet ein Objekt der Klasse *RTerminalTab*, der zweite ein Objekt der Klasse *RSetupTab*. Außerdem erzeugt der Konstruktor noch einen Ring-Puffer der Klasse *RBuffer* „buffer“, welcher zur Speicherung des Std-Output von *R* verwendet wird, sowie einen *RListener* „listener“. Sowohl der Ring-Puffer, als auch der *RListener* wurden im Package „org.deidentifizier.arx.r“ implementiert. Im Konstruktor werden die abstrakten Methoden „bufferUpdated()“, „closed()“ und „setupUpdate()“ überschrieben, sodass diese die Inhalte der beiden Tabs aktualisieren.

Im Anschluss wird die Integration mit *R* durch den Aufruf der Methode „startRIntegration“ eingeleitet. Diese Methode „startRIntegration“

erzeugt ein neues Objekt der Klasse „*RIntegration*“, welches in ?? beschrieben wird und die Integration von *R* realisiert.

### 3.3.3 *RSetupTab*

Die Klasse *RSetupTab* erzeugt einen Tab, welcher Informationen zum verwendeten Betriebssystem sowie dem aktuellen Status der Integration von *R* anzeigt.

Die unterschiedlichen Informationen werden in SWT-Labels dargestellt, welche in einem *GridLayout* angeordnet sind. Die SWT-Labels werden durch die Methodenaufrufe „*showOS()*“, „*showRLocation()*“, „*showRVersion()*“ im Konstruktor erzeugt.

Um das verwendete Betriebssystem abzufragen, wird die Methode „*printOS()*“ der Klasse *OS*, siehe ??, von „*showOS()*“ aufgerufen. Diese gibt das verwendete Betriebssystem als String zurück.

Die Integration von *R* wird in der Klasse *RIntegration* des Package „*org.deidentifier.arx.r*“ umgesetzt. Diese wird beim Start des Programms in *RTerminal* aufgerufen. Um den Status der Integration zu prüfen, rufen „*showRLocation()*“ und „*showRVersion()*“ die Methode „*getR()*“ aus *OS* auf. Diese gibt den absoluten Pfad zur aktuell verwendeten *R-Executive* zurück, welcher im *RSetupTab* angezeigt wird. Wurde keine *R-Executive* gefunden oder konnte diese nicht erfolgreich gestartet werden, so wird von „*getR()*“ *null* zurückgegeben und im *RSetupTab* wird ausgegeben:

Location: „No valid R-exec found!“  
Version: „No valid R-Version selected!“

Wurde eine *R-Version* gefunden und erfolgreich ausgeführt, so wird der erzeugte Listener in *RTerminal* ausgelöst. Dieser ruft anschließend die Methode „*update()*“ auf, durch welche der Inhalt von Location und Version aktualisiert wird.

Außerdem ermöglicht der *RSetupTab* die manuelle Auswahl einer *R-Executive*. Im Konstruktor wird hierfür ein Knopf mit „*createManuellSearchWindow()*“ zum Öffnen eines Navigationsfensters sowie eine Kommandozeile „*createDirSearchLine()*“ erzeugt, sodass entweder der absolute Pfad zur Datei angegeben oder diese mittels des Navigationsfensters ausgewählt werden kann.

Die Kommandoleiste wird in der Methode „createDirSearchLine()“ erzeugt und erfasst die Eingabe durch einen *TraversalListener*. Der *TraversalListener* wird durch Drücken der Return-Taste ausgelöst und gibt den eingegeben Pfad an die Methode „updateSetup(*String* path)“ weiter. Wird die R-Executive mit dem Navigationsfenster gesucht, so wird der absolute Pfad der ausgewählten Datei ebenfalls an „updateSetup(*String* path)“ als Argument übergeben. Nähere Informationen zum Navigationsfenster finden sich in ??.

Die Methode „updateSetup(*String* path)“ startet eine neue Integration von R durch den Aufruf der Methode „startManuellRIntegration(*String* path)“ aus der Klasse *RTerminal*. Nähere Details hierzu in ??.

Falls die R-Executive erfolgreich gestartet wurde, wird durch den *RListener* (siehe oben) der Tab wieder aktualisiert. Andernfalls wird die selbe Ausgabe angezeigt, wie bei der automatischen Suche.

### 3.3.4 RTerminalTab

Im *RTerminalTab* werden R-Befehls-Eingaben des Nutzers erfasst sowie der Std-Output von R ausgegeben. Außerdem können mittels eines Knopfes fertige R-Skripte geladen und ausgeführt werden. Der *RTerminalTab* ist nur nutzbar, falls R erfolgreich gestartet und integriert wurde.

Der Tab hat als Layout ebenso wie der *RSetupTab* ein *GridLayout*. Dieses besteht aus einem *Composite* und einem Textfeld.

Das Kompositum „topline“ umfasst eine Kommandozeile, ein Dropdown-Menü um eingegebene Befehle erneut auszuführen sowie einen Knopf zum Aufrufen von R-Skripten. Als Layout wurde ebenfalls *GridLayout* gewählt, da es alle Komponenten möglichst kompakt darstellt.

Die Kommandozeile und das Dropdown-Menü „input“ wurden mit der importierten Klasse *swt.widget.Combo* erstellt. Die Kommandozeile erfasst die Eingaben des Nutzers beim Drücken der Enter-Taste durch einen *TraversalListener*. Die Eingabe wird anschließend an die Methode „command(*String* command)“ der Klasse *RCommandListener* übergeben, welche den Befehl an R übergibt. Dies wird in ?? genauer beschrieben.

Die letzten 10 eingegebenen Befehle werden im Dropdown-Menü angezeigt. Die Befehle werden in einem Ring-Puffer mit 10 Elementen gespeichert, welcher als String Array implementiert wurde.

Skripte können durch den Knopf „scriptButton“ ausgewählt und ausgeführt werden. Der Knopf wird durch einen *MouseListener* ausgelöst und erzeugt ein Navigationsfenster der Klasse *RBrowserWindow*. Weiter Informationen hierzu in ???. Durch dieses kann das Skript ausgewählt werden und es wird vom *RBrowserWindow* der absolute Pfad des Skriptes übergeben. Anschließend wird überprüft, ob es sich um ein R-Skript handelt, also die Datei auf „.r“ endet. Trifft dies zu, so wird durch die Methode „command(*String* command)“ der Klasse *RCommandListener* der Befehl das Skript zu öffnen an *R* übergeben. Dieser setzt sich zusammen aus:

```
source(" absoluter Pfad ")
```

Das Textfeld „output“ beinhaltet die Ausgabe von *R* und wurde durch die importierte Klasse *StyledText* realisiert. Das Textfeld ist als Ring-Puffer implementiert, sodass dieser nur die letzten 10000 Zeichen des *R*-Std-Output anzeigt. Die Größe des Ring-Puffers ist in der Klasse *RTerminal* als globale int Variable „BUFFER\_SIZE“ festgelegt und kann hier geändert werden. Der Std-Output von *R* wird durch das Auslösen des *R*Listener „listener“ in *RTerminal* aktualisiert. Dieser ruft hierzu die Methode „setOutput(*String* text)“ aus *RTerminalTab* auf. Der Listener wird nach erfolgreichem Starten von *R* durch die Klasse *RIntegration* übergeben. Diese ruft die Methode „setCommandListener(*RCommandListener* listener)“ im *RTerminalTab* auf und übergibt den Listener als Argument. Details zum Listener finden sich in ???.

Außerdem wird nach einem erfolgreich Start von *R* durch „startRIntegration(*String* path)“ aus der Klasse *RTerminal* die Methode „enableTab()“ in *RTerminalTab* aufgerufen, sodass der Tab nutzbar wird. Beim Beenden von *R* wird in der Methode „endR()“ in *RTerminal* die Methode „disableTab()“ in *RTerminalTab* aufgerufen, welche den Tab wieder für den Nutzer sperrt.

### 3.3.5 RBrowserWindow

Die Klasse *RBrowserWindow* implementiert ein Navigationsfenster des Standard-Dateimanagers. Dieses wurde durch die importierte SWT-Klasse *FileDialog* implementiert. Die Klasse beinhaltet nur eine Methode „openBrowser(Shell)“, durch welche ein neuer *FileDialog*, also ein Navigationsfenster, erzeugt wird, mit welchem eine Datei ausgewählt werden kann. Anschließend gibt die Methode den absoluten Pfad der ausgewählten Datei zurück. Mit dem Navigationsfenster kann sowohl manuell die *R*-Executive

im *RSetupTab* sowie ein ausführbares *R*-Skript im *RTerminalTab* ausgewählt werden. Anschließend wird der absolute Pfad per *return* übergeben, sodass der funktionale Kern, siehe ??, die jeweilige Aktion durchführen kann.

### 3.3.6 RLayout

Diese Klasse definiert das *SWT*-Layout der Klassen *RSetupTab* und *RTerminalTab*. Die Methoden der Klasse werden im Konstrukt der beiden genannten Klassen aufgerufen, um die *SWT*-Objekte der des jeweiligen Tabs mit den richtigen Parametern und Abständen zu erzeugen.

### 3.3.7 RCommandListener

Das *Java*-Interface *RCommandListener* implementiert einen Listener, welcher bei der Eingabe von Befehlen für *R* ausgeführt wird. Die abstrakte Methode „*command(String command)*“ des Interfaces wird in der Klasse *RTerminal* mit der Methode „*execute(command)*“ aus der Klasse *RIntegration* überschrieben. Somit wird die Trennung der graphischen Oberfläche und des funktionalen Kerns der *R*-Integration, welcher in ?? beschrieben wird, beibehalten.

## 3.4 Funktionaler Kern - Integration von R

Die Integration von *R* in *Java* ist in dem Package „*org.deidentifizier.arx.r*“ implementiert. Dieses beinhaltet alle Funktionen, durch welche die *R*-Executive automatisch gefunden, ausgeführt und das Betriebssystem erkannt wird. Außerdem beinhaltet es alle Methoden zur Übermittlung des Input- und Outputstreams. Zur Integration von *R* in *Java* wurde die Klasse *ProcessBuilder* [?] verwendet.

Das Package umfasst vier Klassen:

- *RIntegration*
- *OS*
- *RBuffer*
- *RListener*



### 3.4.1 RIntegration

Die Klasse *RIntegration* ist das Kernstück des *R-Terminals* und ist für die Integration von *R* verantwortlich. Der *R*-Prozess wird durch die Library *ProcessBuilder* [?] gestartet.

Wird das Programm mit der graphischen Benutzeroberfläche gestartet, so wird für jede gefundene ausführbare *R*-Executive durch die Methode „startRIntegration“ beziehungsweise „startManuellRIntegration“ ein neues Objekt dieser Klasse erzeugt. Hierbei werden dem Konstruktor folgende Argumente übergeben:

- *final String* path
- *final buffer* buffer
- *final RListener* listener

Der absolute Pfad zur Executive wird durch „path“ angegeben. Bei „buffer“ handelt es sich um den Ring-Puffer der Klasse *RBuffer*, welcher zur Ausgabe des Std-Output des *R*-Prozesses im *RTerminalTab* erzeugt wurde. Gibt es eine neue Ausgabe des Std-Output des Prozesses, so wird dieser im Ring-Puffer angehängt. Um die beiden Tabs *RSetupTab* und *RTerminalTab* bei Änderungen zu aktualisieren, wird der *RListener*, welcher in *RTerminal* erzeugt wurde, als Attribut „listener“ übergeben. Bei einer Ausgabe des Std-Output oder Start bzw. Beenden des *R*-Prozesses wird dieser ausgelöst.

Der Konstruktor erzeugt ein neues Objekt der Klasse *ProcessBuilder* mit den betriebssystemspezifischen Parametern. Die Parameter werden durch Aufrufen der Methode „getParameters(*String* path)“, siehe ??, aus der Klasse *OS* übergeben und dem Konstruktor des *ProcessBuilders* als Argumente übergeben.

Anschließend wird der Prozess gestartet.

Nach erfolgreichen Start des *R*-Prozesses wird ein Reader „reader“ erzeugt, welchem die Ausgabe des *R*-Prozesses übergeben wird. Dieser Reader übergibt die Ausgabe an den Ring-Puffer „listener“.

Direkt im Anschluss wird die Methode „getVersion(*Reader* reader)“ aufgerufen. Durch diese wird abgefragt, welche Version von *R* verwendet wird. Die Ausgabe der Version wird durch das Kommando „version“, welches an den laufenden *R*-Prozess als Input durch den Methodenaufruf „execute(“version”)“ übergeben wird, erreicht. Die Ausgabe dieses Kommandos soll nicht im Ausgabefenster der GUI, dem *RTerminalTab*, angezeigt

werden. Deshalb erzeugt die Methode „`getVersion(Reader reader)`“ einen neuen, temporären Ring-Puffer, in welchem die Ausgabe für diesen Befehl gespeichert wird.

Aus dem Ausgabe-String dieses „version“-Kommandos wird anschließend die genaue Version sowie der Nickname extrahiert und in der lokalen Variable „version“ gespeichert. Im Anschluss wird der Rlistener „listener“ ausgelöst, welcher die Methode „`setupUpdate()`“ ausführt. Diese Methode ruft die „update“-Methode im *RSetupTab* auf und aktualisiert den Tab.

Nach Abschluss dieses Kommandos wird die gesamte Ausgabe in den Ring-Puffer „buffer“ geschrieben und durch Auslösen des Rlistener „listener“ in dem Ausgabefenster des *RTerminalTabs* angezeigt.

Kommandos, welche in *R* ausgeführt werden sollen, werden als String der Methode „`execute(String command)`“ übergeben, welche diese an den *R*-Prozess weiterleitet. Hier wird ein *BufferedWriter*-Objekt erstellt. Mit der „`write(String command)`“ Methode dieses Elementes wird das Kommando in dem *BufferedWriter* geschrieben und anschließend mit „`newline()`“ noch ein Zeilenumbruch im *BufferedWriter* gespeichert. Um den Inhalt des *BufferedWriter* an den *R*-Prozess zu übergeben, diesen auszuführen und anschließend den *BufferedWriter* zu leeren, wird zum Abschluss die Methode „`flush()`“ aufgerufen. Mit dieser Methode werden alle Kommandos an den *R*-Prozess übergeben. Die „`execute(String command)`“ Methode wird bei der Nutzung der GUI von der Methode „`command(String command)`“ aufgerufen. Nähere Informationen zur Methode „`command(String command)`“ finden sich in ??.

### 3.4.2 OS

Die Klasse *OS* implementiert folgende Funktionalitäten: Erkennung des verwendeten Betriebssystems, die Anpassung der Pfade der Standard-Speicherorte der *R*-Executive je nach verwendetem Betriebssystem und die Überprüfung der übergebenen *R*-Executives.

Um die drei unterstützten Betriebssysteme Windows, Mac und Unix effizient zu unterscheiden, werden diese als Enum Konstanten „*OSType*“ erzeugt.

Die absoluten Pfade der Standard-Speicherorte der *R*-Executive werden

für jedes Betriebssystem in einem eigenen String Array gespeichert. Es ist wichtig hier zu beachten, dass Windows andere Separatoren verwendet als Linux und OS X. Bei Windows müssen die einzelnen Verzeichnisse durch „\\“ getrennt werden, bei beiden anderen Betriebssystemen durch „/“.

Falls neue Standard-Speicherorte für die R-Executive hinzugefügt werden sollen, kann das jeweilige Array einfach um den neuen Pfad erweitert werden. Beim Starten des *R-Terminals* durch die graphische Benutzeroberfläche werden alle Pfade des Arrays nach einer ausführbaren R-Executive durchsucht. Hierfür wird die Methode „getR()“ aufgerufen, welche im Folgenden noch vorgestellt wird.

Außerdem werden die Dateinamen der R-Executive für die unterschiedlichen Betriebssysteme in einem String-Array gespeichert, da sich diese je nach Betriebssystem unterscheiden. Außerdem wird das jeweilige Array benötigt, um bei der manuellen R-Auswahl zu prüfen, ob es sich um eine gültige R-Executive handelt. Dies wird in der Methode „isRExec(String path)“ in Abhängigkeit vom verwendeten Betriebssystem überprüft.

Die Methode „getOS()“ fragt den Namen des Betriebssystems ab und gibt dieses als Enum „OSType“ passend zurück.

Wie bereits beschrieben, werden nach dem Start des Programms die Standard-Speicherorte nach einer ausführbaren R-Executive durchsucht. Ob an einem dieser Speicherorte eine Executive liegt, wird mit der Methode „getPath(String[] locations, String[] executables)“ überprüft. Außerdem wird versucht die Executive mit *ProcessBuilder* zu starten, um zu prüfen, ob der Nutzer ausreichende Zugriffsberechtigungen besitzt.

Zusätzlich sind in der Klasse *OS* die Parameter zur Ausführung der R-Executive für den *ProcessBuilder* gespeichert. Diese werden beim Aufruf der Methode „getParameters(String path)“ mit dem Pfad „path“ der Executive zu einem String-Array konkateniert und anschließend als String-Array zurückgegeben.

### 3.4.3 RBuffer

Diese Klasse implementiert den Ring-Puffer, welcher für die Ausgabe des Std-Output von *R* im Ausgabefenster des *RTerminalTab* verwendet wird. Die Größe des Ring-Puffers wird bei der Erzeugung dem Konstruktor über-

geben und ist beim Starten des Programms mit graphischer Benutzeroberfläche in der Klasse *RTerminal* in der Variable „BUFFER\_SIZE“ festgelegt.

### 3.4.4 RListener

Die Klasse *RListener* implementiert einen Listener, welcher in den Klassen *RTerminal*, siehe ??, sowie *RIntegration* verwendet wird.

Der Listener aktualisiert die Inhalte von *RSetupTab* sowie *RTerminalTab*. Durch die abstrakte Methode „setupUpdate()“, welche in *RTerminal* überschrieben wird, wird bei Änderung des Status von *R*, also dem Beenden oder dem Starten eines neuen *R*-Prozesses der *RSetupTab* aktualisiert. Durch „bufferUpdate()“ wird das Ausgabefenster im *RTerminalTab* aktualisiert. Diese abstrakte Methode wird ebenfalls in *RTerminal* überschrieben.

## 3.5 Ausführung ohne graphische Benutzeroberfläche

Der funktionale Kern des *R-Terminals* kann auch ohne die graphische Benutzeroberfläche ausgeführt werden. Hierfür wird die Umsetzung anhand eines Beispiels in ?? erklärt. Eine Übersicht der Methoden zur Nutzung des *R-Terminals* ohne GUI befindet sich in ??.

### 3.5.1 Aufruf des R-Terminals ohne GUI

Um das *R-Terminal* ohne graphische Benutzeroberfläche zu nutzen müssen folgende Objekte erzeugt werden:

- ein Ring-Puffer der Klasse *RBuffer*, welcher der Std-Output von *R* übergeben wird
- ein Listener der Klasse *RListener*, welcher durch den Std-Output ausgelöst wird

Anschließend müssen die abstrakten Methoden des *RListeners* überschrieben werden:

- *bufferUpdated()*
- *closed()*

- *setupUpdate()*

Nun kann die Integration von *R* gestartet werden. Hierfür muss ein neues *RIntegration*-Objekt erzeugt werden. Diesem werden der Pfad „path“ zur *R*-Executive als String, der erzeugte *RBuffer* „buffer“ und der *RListener* „listener“ übergeben:

```
final RIntegration rProcess =  
    new RIntegration(path, buffer, listener)
```

Um dem *R*-Prozess nun Befehle zu übergeben, muss lediglich die Methode „execute“ des erzeugten *RIntegration*-Objekt aufgerufen werden und das Kommando als String übergeben werden.

Beispiel: Es wurde ein *RIntegration*-Objekt *rProcess* erzeugt und es soll „1+2+3+4+5“ als Kommando übergeben werden, dann muss

```
rProcess.execute("1+2+3+4+5")
```

ausgeführt werden. Die Ausgabe wird im Ring-Puffer gespeichert, welcher für den Std-Output erzeugt wurde.

Ein Code-Beispiel hierzu befindet sich im Package „ExampleWithoutGUI“.

## 3.5.2 API

### 3.5.3 Klasse: *RIntegration*

```
public RIntegration(final String path,  
                    final RBuffer buffer,  
                    final RListener listener)
```

Der Konstruktor dieser Klasse erzeugt den *R*-Prozess, startet diesen und verwaltet die Input- und Outputstreams.

- path: String des Pfades zur *R*-Executive
- buffer: Output-Buffer
- listener: Listener des Std-Outputstreams des *R*-Prozess

```
public void execute(String command)
```

- `command`: Das Kommando als String

Durch die Methode „`execute`“ werden die Befehle für den *R*-Prozess ausgeführt.

```
public boolean isAlive()
```

Mit der Methode „`isAlive`“ kann geprüft werden, ob der *R*-Prozess noch aktiv ist.

- returns `true`: *R*-Prozess ist noch aktiv
- returns `false`: *R*-Prozess ist nicht mehr

```
public void shutdown()
```

Die Methode „`shutdown`“ beendet den laufenden *R*-Prozess.

```
public String getVersion()
```

Die Methode „`shutdown`“ gibt die Version und den Nickname der letzten verwendeten *R*-Version konkateniert als String zurück.

### 3.5.4 Klasse: OS

```
public static OSType getOS()
```

Diese Methode gibt das verwendete Betriebssystem zurück:

- Enum OSType (WINDOWS,UNIX,MAC)

Wird das Betriebssystem nicht unterstützt, wird folgende Exception geworfen:

- `IllegalStateException("Unsupported operating system")`

```
public static String getR()
```

- returns *String* path: gibt den Pfad zur gefundenen *R*-Executive als String zurück
- returns *null*: falls keine Executive gefunden wurde

```
public static String getR(String folder)
```

Diese Methode durchsucht das übergebene Verzeichnis nach einer *R-Executive*

- returns *String* path: Falls eine Executive gefunden wurde wird der absolute Pfad zu dieser zurückgegeben
- returns *null*: Falls keine Executive gefunden wurde

```
public static String [] getParameters(String path)
```

Die Methode „getParameters“ gibt die Parameter für den *ProcessBuilder* aus.

- path: Der Pfad zur *R-Executive*
- MAC: returns *String[]* {path, -vanilla", -quiet", -interactive"};
- UNIX: returns *String[]* {path, -vanilla", -quiet", -interactive"};
- WINDOWS: returns *String[]* {path, -vanilla", -quiet", -ess"};

```
public static String printOS()
```

Diese Methode gibt das verwendete Betriebssystem als String zurück.

- returns: "macOS", "Unix", "Windows"
- *IllegalStateException*("Unknown operating system"): falls das OS nicht zu den drei oben genannten gehört

```
public static boolean isR_Exec(String path)
```

Diese Methode prüft, ob die übergebene Datei eine *R-Executive* ist.

- *String* path: der absolute Pfad der Datei
- returns *true*: Datei ist eine *R-Executive*
- returns *false*: Datei ist keine *R-Executive*

### 3.5.5 Klasse: RBuffer

```
public RBuffer(int size)
```

Dem Konstruktor wird die Größe des Ring-Puffers als Anzahl der Zeichen übergeben.

```
public void append(char c)
```

Fügt ein Zeichen an den Ring-Puffer an.

```
public void append(char[] buffer)
```

Fügt alle im Char-Array enthaltenen Zeichen an den Ring-Puffer an.

```
public String toString()
```

Diese Methode gibt den gesamten Inhalt des Ring-Puffers als String zurück.

### 3.5.6 Klasse: RListener

```
public RListener(int ticksPerSecond)
```

Dem Konstruktor wird die maximale Anzahl an Events pro Sekunde als `int` übergeben.

Die folgenden drei abstrakten Methoden müssen beim Erzeugen eines neuen *RListener*-Objektes überschrieben werden.

```
public abstract void bufferUpdated();
```

Diese Methode wird bei einem Update des Output-Buffers ausgelöst.

```
public abstract void closed();
```

Diese Methode wird beim Beenden des *R*-Prozesses ausgelöst.

```
public abstract void setupUpdate();
```

Diese Methode wird beim Starten eines neuen *R*-Prozesses ausgelöst.



## 3.6 Abhängigkeiten

Das Projekt beinhaltet bereits alle benötigten *Java*-Libraries zur Ausführung der Software. Da die graphische Benutzeroberfläche mit *SWT* realisiert wurde, müssen die *SWT*-Libraries je nach Betriebssystem ausgewählt werden. Dies wird genauer unter ?? erläutert.

Für die Integration von *R* in *Java*, welche durch das *R-Terminal* umgesetzt wurde, muss eine Standalone-Installation von *R* vorliegen. Nähere Details hierzu in

### 3.6.1 Java

Die für das *R-Terminal* verwendete Programmiersprache ist *Java*. Für die Entwicklung und Tests wurde *Java SE 8 (1.8.0\_91)* verwendet. [?]

### 3.6.2 R-Project

Um *R* in *Java* zu integrieren muss eine Standalone-Installation von *R*, für welche der aktuelle Benutzer ausreichende Zugriffsrechte besitzt, vorhanden sein. Alle aktuellen *R*-Versionen können von der Homepage des *R-Projects* [3] heruntergeladen werden.

### 3.6.3 SWT

Die graphische Benutzeroberfläche wurde mithilfe des *Standard Widget Toolkit (SWT)* realisiert [?]. *SWT* ist ein „open-source widget toolkit“ für *Java* mit Integration in die GUI des nativen Betriebssystems, sodass das Design der Benutzeroberfläche an das Design des Betriebssystems angepasst wird.

Für das Projekt *R-Terminal* wurde die *SWT*-Version 4.2.1 verwendet.

Wie die betriebssystemspezifischen *SWT*-Libraries auf den „Build Path“ hinzugefügt werden, kann unter ?? nachgeschlagen werden.

# Literaturverzeichnis

1. <https://www.apache.org/licenses/GPL-compatibility.html>
2. [https://biometrie.charite.de/fileadmin/user\\_upload/microsites/m\\_cc04/biometrie/Q1/SkriptQ1.pdf](https://biometrie.charite.de/fileadmin/user_upload/microsites/m_cc04/biometrie/Q1/SkriptQ1.pdf)
3. <http://www.r-tutor.com/r-introduction/data-frame>
4. <https://www.r-project.org/about.html>
5. <https://www.eclipse.org>
6. <https://www.eclipse.org/swt/>
7. <https://www.java.com/de/>
8. [http://www.ats.ucla.edu/stat/r/modules/factor\\_variables.html](http://www.ats.ucla.edu/stat/r/modules/factor_variables.html)
9. <https://docs.oracle.com/javase/7/docs/api/java/lang/ProcessBuilder.html>
10. <http://arx.deidentifier.org>