# Terminal zur Bearbeitung von ARX-Daten mit R

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

**Zusammenfassung**

Morbi tempor congue porta. Proin semper, leo vitae faucibus dictum, metus mauris lacinia lorem, ac congue leo felis eu turpis. Sed nec nunc pellentesque, gravida eros at, porttitor ipsum. Praesent consequat urna a lacus lobortis ultrices eget ac metus. In tempus hendrerit rhoncus. Mauris dignissim turpis id sollicitudin lacinia. Praesent libero tellus, fringilla nec ullamcorper at, ultrices id nulla. Phasellus placerat a tellus a malesuada.

# Inhaltsverzeichnis

# Kapitel 1

# Einführung

This statement requires citation [1]; this one does too [**?**]. Lorem ipsum dolor sit amet, consectetur adipiscing elit. Aenean dictum lacus sem, ut varius ante dignissim ac. Sed a mi quis lectus feugiat aliquam. Nunc sed vulputate velit. Sed commodo metus vel felis semper, quis rutrum odio vulputate. Donec a elit porttitor, facilisis nisl sit amet, dignissim arcu. Vivamus accumsan pellentesque nulla at euismod. Duis porta rutrum sem, eu facilisis mi varius sed. Suspendisse potenti. Mauris rhoncus neque nisi, ut laoreet augue pretium luctus. Vestibulum sit amet luctus sem, luctus ultrices leo. Aenean vitae sem leo.

Nullam semper quam at ante convallis posuere. Ut faucibus tellus ac massa luctus consectetur. Nulla pellentesque tortor et aliquam vehicula. Maecenas imperdiet euismod enim ut pharetra. Suspendisse pulvinar sapien vitae placerat pellentesque. Nulla facilisi. Aenean vitae nunc venenatis, vehicula neque in, congue ligula.

## 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, ein weiteres GNU-Projekt, welches weitgehend auch unter R läuft. [2]

R bietet standardmäßig alle Hauptfunktionen an 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.

## 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.

# 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 und Skripte auszuführen, und gleichzeitig die im Kapitel 1 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

In Abbildung 2.3 ist das R-Terminal unter OS X (hier Version 10.11.1) zu sehen. Das Terminal verfügt über die beiden Tabs *Terminal* und *Setup* (s. **??**. 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.

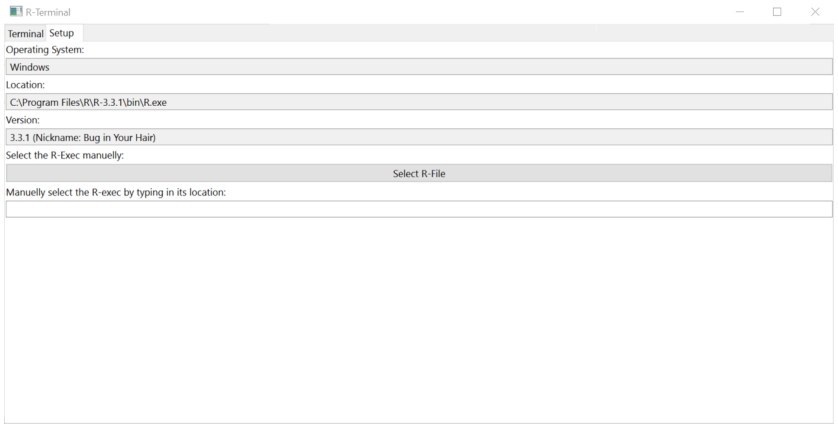


Abbildung 2.1: R-Terminal: *Terminal* unter Windows 10 Education (Version 1511)

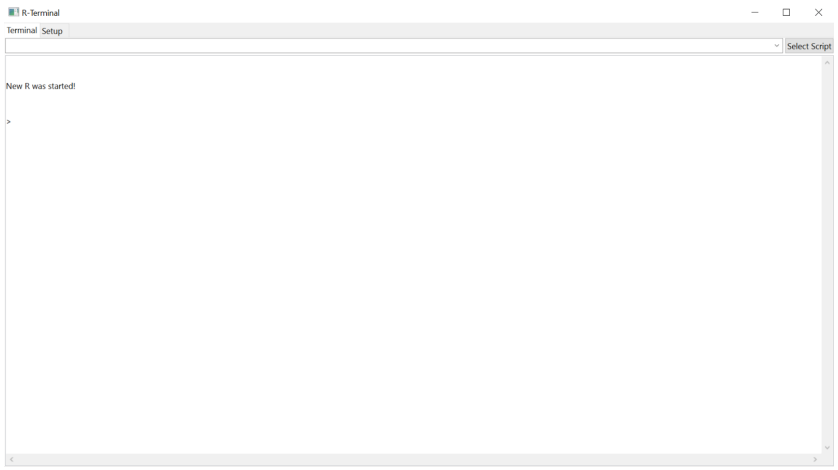


Abbildung 2.2: R-Terminal: *Setup* unter Windows 10 Education (Version 1511)

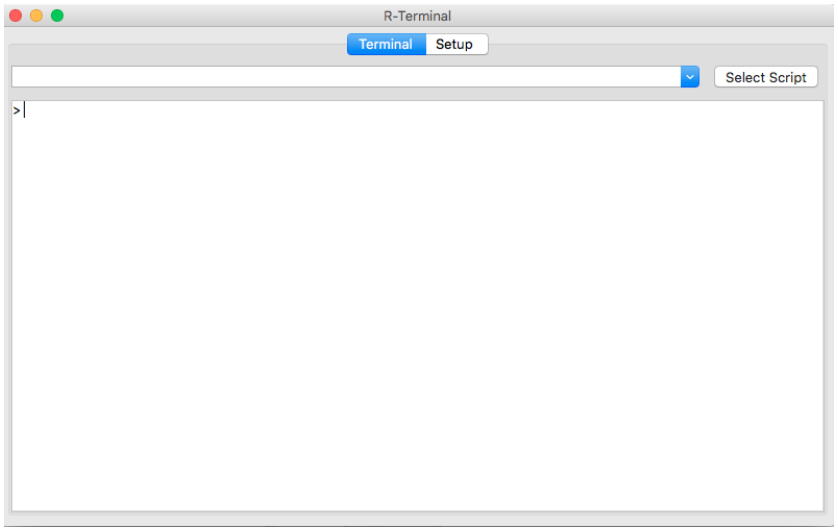


Abbildung 2.3: R-Terminal: *Terminal* unter OS X (Version 10.11.1)

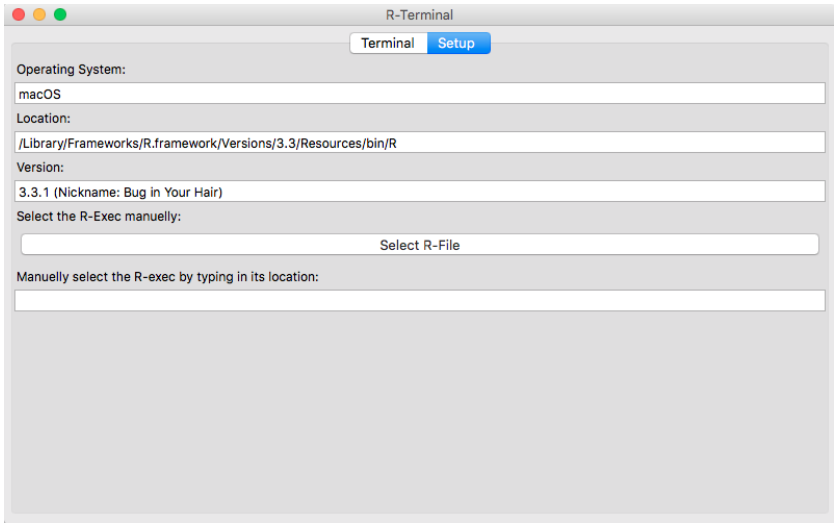


Abbildung 2.4: R-Terminal: *Setup* unter OS X (Version 10.11.1)

# Kapitel 3

# Entwicklerdokumentation

## 3.1 Verwendete Technologien

Das *R-Terminal* wurde mit der Entwicklungsumgebung *Eclipse* entwickelt. Als Programmiersprache wurde *Java* gewählt. Zur Realisierung der graphischen Benutzeroberfläche wurde *SWT* gewählt, genauere Details hierzu werden in 3.5.1 erläutert.

## 3.2 Architektur des R-Terminals

Die Software des *R-Terminal* ist in zwei Komponenten aufgeteilt, welche als getrennte Packages vorliegen. Das erste Package beinhaltet die graphische Benutzeroberfläche, das Zweite beinhaltet die gesamte Integration von *R* in *Java*, es stellt also alle Funktionen 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, können alle Funktionen auch ohne die GUI ausgeführt und verwendet werden.

## 3.3 Graphische Benutzeroberfläche (GUI)

Das erste Package, benannt „org.deidentifier.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 der graphischen Benutzeroberfläche zu starten, muss die Methode „main“ der Klasse *RMain* aufgerufen werden. Diese erzeugt ein neues Display sowie eine Shell und im Anschluss ein neues Objekt der Klasse *RTerminal*. Letzteres erzeugt die einzelnen Komponenten der Oberfläche.

### 3.3.2 RTerminal

Die Erzeugung der GUI wird durch den Konstruktor des neuen *RTerminal*-Objektes realisiert. Dieser erzeugt einen TabFolder mit zwei Tabs, welche mit den Klassen *RTerminalTab* und *RSetupTab* befüllt werden. Außerdem erzeugt der Konstruktor noch einen Ring-Puffer, welcher zur Speicherung des Std-Output von *R* verwendet wird, sowie einen Listener. Sowohl der Ring-Puffer, als auch der Listener wurden im zweiten Package implementiert. Im Anschluss wird die Integration mit *R* durch den Aufruf der Methode „startRIntergration“ eingeleitet.

Diese Methode erzeugt ein neues Objekt der Klasse „RIntegration“, welches in 3.4.1 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 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 mehrere Methodenaufrufe im Konstruktor erzeugt. Das aktuell verwendete Betriebssystem wird durch den Aufruf der Methode „printOS()“ der Klasse *OS* aus dem zweiten Package aufgerufen. Weitere Informationen hierzu finden sich in 3.4.2.

Die Integration von *R* wird in der Klasse *RIntegration* des zweiten Package durchgeführt. Diese wird beim Start des Programms aufgerufen und durchsucht die Standart-Installationspfade von *R* nach einer ausführbaren R-Executive. Um den Status der Integration zu prüfen, wird aus *OS* die Methode „getR()“ aufgerufen. 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 es wird im *RSetupTab* ausgegeben:

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

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

Außerdem ermöglicht der *RSetupTab* die manuelle Suche einer *R*-Executive. Hierfür kann entweder der absolute Pfad zur Datei angegeben werden oder diese mittels eines Navigationsfensters ausgewählt werden. Der RSetupTab beinhaltet hierfür eine Kommandozeile zur Pfadeingabe bzw. einen Knopf, durch welchen ein Navigationsfensters aufgerufen wird.

Die Kommandoleiste wird in der Methode „createDirSearchLine()“ erzeugt und erfasst die Eingabe durch einen *TraverseListener*. Der *TraverseListener* wird durch Drücken der Return-Taste ausgelöst und gibt den eingegeben Pfad an die Methode „updateSetup(path)“ weiter. Diese startet eine neue Integration von R durch den Aufruf der Methode „startManuellRIntegration(*String* path)“ aus der Klasse *RTerminal*. Nähere

Details hierzu in 3.3.4. 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

In dem *RTerminalTab* werden die *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 Button 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 *TraverseListener*. Die Eingabe wird anschließend an die Methode „command(command)“ der Klasse *RCommandListener* übergeben, welche den Befehl an R übergibt. Dies wird in 3.3.7 beschrieben.

Die letzten 10 Befehle werden im Dropdown-Menü angezeigt. Die Befehle werden in einem Ringpuffer mit 10 Elementen gespeichert, welcher als String Array implementiert ist.

Skripte können durch den Knopf „Select Script“ ausgewählt und ausgeführt werden. Der Knopf wird durch einen *MouseListener* ausgelöst und erzeugt ein Navigationsfensters der Klasse *RBrowserWindow*. Weiter Informationen hierzu in 3.3.5. Durch dieses kann das Skript ausgewählt werden und es wird 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(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 Ringpuffer implementiert, sodass dieser nur die letzten 10000 Zeichen des *R*-Std-Output anzeigt. Die Größe des Ringpuffers ist in der Klasse *RTerminal* in den globalen Variablen als int Variable „BUFFER_SIZE“ festgelegt und kann hier geändert werden. Der Std-Output von *R* wird durch das Auslösen des RListener „listener“ in *RTerminal* aktualisiert. Dieser ruft hierzu die Methode „setOutput(Text)“ aus *RTerminalTab* auf.

Falls *R* erfolgreich durch „startRIntegration(path)“ aus der Klasse *RTerminal* gestartet wurde, wird 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 Standart-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 muss der Pfad übergeben werden, damit der funktionale Kern, siehe 3.4, die Aktion durchführen kann.

### 3.3.6 RLayout

Diese Klasse definiert das SWT-Layout der Klassen *RSetupTab* und *RTerminalTab*. Die Methoden der Klasse werden im Konstruktur 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(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 3.4 beschrieben wird, beibehalten.

## 3.4 Funktionaler Kern - Integration von R

Die eigentliche Integration von R in Java ist in dem Package „org.deidentifier.arx.r“ implementiert. Dieses beinhaltet alle Funktionen, durch welche die *R*-Executive automatisch oder manuell gefunden, ausgeführt und das Betriebssystem erkannt wird. 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 Programms 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 jedes mal, wenn eine ausführbare *R*-Executive gefunden wurde ein neues Objekt dieser Klasse erzeugt. Hierbei werden dem Konstruktor folgende Attribute von der Methode „startRIntegration“ beziehungsweise „startManuellRIntegration“ *RTerminal* 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 Ringpuffer angehängt. Um die beiden Tabs *RSetupTab* und *RTerminalTab* immer 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 *ProzessBuilder* mit den passenden Parametern. Die Parameter werden durch Aufrufen der Methode „getParameters(*String* path)“ aus der Klasse *OS* übergeben und dem Konstruktor des ProzessBuilders als Attribut 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.

### 3.4.2 OS

Die Klasse *OS* beinhaltet die Erkennung des verwendeten Betriebsystems, die Anpassung der Pfade der Standart-Speicherorte der R-Executive je nach verwendeten Betriebssystem und die Überprüfung der übergebenen *R*-Executives.

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

Die absoluten Pfade der Standart-Speicherorte der *R*-Executive wird 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 Verzeignisse durch „\\“ getrennt werden, bei beiden anderen Betriebssystemen nur durch „/“.
Falls neue Standart-Speicherorte für die *R*-Executive hinzugefügt werden sollen, so kann das jeweilige Array einfach um den neuen Pfad erweitert werden. Beim Starten des Programms werden alle Pfade des Arrays nach einer ausführbaren *R*-Executive durchsucht.

Außerdem werden die Dateinamen der *R*-Executive für die unterschiedlichen Betriebssysteme in einem String-Array gespeichert, da diese je nach

Betriebssystem anders benannt sind. Außerdem wird das jeweilige Array benötigt, um bei der manuellen *R*-Auswahl zu prüfen, ob es sich um eine *R*-Executive handelt. Dies wird in der Methode „isRExec(*String* path)“ in Abhängigkeit des verwendeten Betriebssystems 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 Standart-Speicherorte nach einer ausführbaren R-Executive durchsucht. Ob an einem dieser Speicherorte eine Executive liegt, wird mit der Methode „String getPath(*String*[] locations, *String*[] executables)“ überprüft. Außerdem wird versucht die Executive mit einem 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 in *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 Ringpuffer, welcher für die Ausgabe des Std-Output von *R* im Textausgabefenster des *RTerminalTab* verwendet wird. Die Größe des Ringpuffers wird bei der Erzeugung dem Konstruktor übergeben 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* des ersten Package, siehe 3.3, sowie *RIntegration* verwendet wird.

Der Listener aktualisiert die Inhalte der *RSetupTabs* sowie des *RTerminalTabs*. Durch die abstrakte Methode „setupUpdate()“, welche in *RTerminal* überschrieben wird, wird bei einer Änderung der Integration von *R*, also dem Beenden oder dem Starten eines neuen *R*-Prozesses der *RSetupTab*

aktualisiert. Durch „bufferUpdate“ wird das Textausgabefenster im *RTerminalTab* aktualisiert. Diese abstrakte Methode wird ebenfalls in *RTerminal* überschrieben.

## 3.5 Abhängigkeiten

### 3.5.1 SWT

Die graphische Benutzeroberfläche wurde mithilfe des *Standard Widget Toolkit* (SWT) realisiert. SWT ist ein öpen-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.

## 3.6 Ausführung ohne graphische Benutzeroberfläche

## 3.7 Alternativen

### 3.7.1 rJava

### 3.7.2 JRI

# Literaturverzeichnis


**1.** Michel Goossens, Frank Mittelbach, and Alexander Samarin. *The LaTeX Companion*. Addison-Wesley, Reading, Massachusetts, 1993.

**2.** `https://www.r-project.org/about.html`