Implementierungsdokument

Hanselmann, Hecht, Klein, Schnell, Stapelbroek, Wohnig

13. Februar 2017
v0.4

Inhaltsverzeichnis

1 Einleitung **4**

2 Unterschiede zu den im Pflichtenheft gestellten Kriterien **5**
2.1 Sollkriterien . . . 5
2.1.1 Wahltemplate Gewichtete Wahl . . . 5
2.2 Kannkriterien . . . 5
2.2.1 Betrieb auf dem Betriebssytem MacOS . . . 5

3 Änderungen am Entwurf **6**
3.1 Package Highlevel . . . 6
3.1.1 CentralObjectProvider . . . 6
3.1.2 DisplaysStringsToUser . . . 6
3.1.3 start-/stopReacting . . . 6
3.1.4 ProjectSource . . . 7
3.1.5 isCorrect . . . 7
3.2 UserActions . . . 7
3.3 Package DataTypes . . . 7
3.4 Package SaverLoader . . . 8
3.4.1 FileChooser . . . 8
3.5 Package Codearea . . . 9
3.5.1 JTextPaneToolbox . . . 9
3.5.2 Errordisplayer . . . 9
3.5.3 SaveTextBeforeRemove . . . 10
3.5.4 TextLineNumber . . . 10
3.5.5 SquigglePainter . . . 10
3.5.6 Tabinserter . . . 10
3.5.7 LineBeginningTabsHandler . . . 10
3.5.8 UserActions . . . 11
3.5.9 NewlineInserter . . . 11
3.5.10 UserInsertToCode . . . 11
3.6 Package BooleanExpEditor . . . 12
3.7 Package CElectionDescriptionEditor . . . 12
3.7.1 ChangeElectionTypeUserAction . . . 13
3.7.2 Subpackage ElectionTemplates . . . 13
3.7.3 ErrorHandling . . . 13
3.8 Package PropertyChecker . . . 14
3.8.1 CBMCCodeGeneration . . . 14

3.8.2 PropertyChecker 14
3.9 Package PropertyList 14
3.9.1 Invarianten 15
3.9.2 Subpackage Model 15
3.9.3 Subpackage Controller 16
3.9.4 Subpackage View 18
3.9.5 Class ParameterEditor 18
3.9.6 UserActions 18
3.9.7 Handler 19
3.9.8 AboutWindow 19
3.10 Package Toolbox 19
3.10.1 SortedIntegerList 19
3.10.2 RepaintThread 19
3.10.3 CCodeHelper 19
3.10.4 ActionIdAndListener 19
3.10.5 Tuple 20

4 Zeitablauf Implementierungsphase **21**
4.1 Geplanter Ablauf 21
4.2 Tatsächlicher Ablauf 22

5 Statistiken **24**

1 Einleitung

Dieses Dokument beschreibt die Implementierungsphase einer Praxis der Softwareentwicklungsgruppe am Karlsruher Institut für Technologie.
Das implementierte Programm sollte folgende Module bereitstellen:

- Eine Möglichkeit zur Beschreibung eines Wahlverfahrens in der Programmiersprache C.
- Eine Möglichkeit zur Beschreibung von Eigenschaften, auf die das Wahlverfahren geprüft werden soll. Die Beschreibung erfolgt in einer von der Entwicklergruppe definierten Makrosprache. [1]
- Eine Möglichkeit zum Angeben der Parameter, für welche das angegebenen Wahlverfahren analysiert werden soll (Anzahl Wähler, Anzahl Kandidaten, Anzahl Sitze).
- Eine Möglichkeit, das beschriebene Wahlverfahren auf die eingegeben Eigenschaften zu überprüfen. Diese Überprüfung soll durch eine Schnittstelle mit CBMC [2] verwirklicht werden.
- Eine Ausgabe des Ergebnisses der Analyse: Eine Erfolgsmeldung, für jede Eigenschaft, die erfüllt wurde, sowie eine Darstellung eines Gegenbeispieles, für jede nicht erfüllte Eigenschaft.

[1] Makro: Ein Codeabschnitt, der durch den Präprozessor ersetzt wird.
[2] CBMC (C Bounded Model Checker): Ein Programm, welches C-Programme mittels bounded model checking auf Fehler und vom Benutzer definierte Assertions untersucht.

2 Unterschiede zu den im Pflichtenheft gestellten Kriterien

2.1 Sollkriterien

2.1.1 Wahltemplate Gewichtete Wahl

/FS1110/ Der Unterpunkt der gewichteten Wahl kann leider nicht komplett, wie im Pflichtenheft abgedeckt werden. So können Ober- und Untergrenze der gewichteten Wahl leider nicht vom Benutzer eingegeben werden, sondern müssen im Quellcode verändert werden. Die Umsetzung war leider nicht möglich, da diese Wahlart zu spät getestet wurde und somit der Fehler erst zu spät auffiel, um ihn noch zu beheben. Die Implementierung würde vermutlich circa 2 Stunden kosten.

2.2 Kannkriterien

2.2.1 Betrieb auf dem Betriebssytem MacOS

Da wir keinen MAC zum Testen hatten, haben wir keine Implementierung für MacOS vorgenommen. Da unser Entwurf Erweiterungen aber leicht zulässt, ist es für Benutzer, die dieses Betriebssystem besitzen mit einigen Programmierkenntnisse möglich sich diese Option selbst einzubauen. Dazu müsste die Funktionalität erstellt werden, Prozesse auf diesem Betriebssystem von Java aus zu starten. Mit dieser müssten dann CBMC sowie GCC angesprochen und deren Ausgabe zurückgegeben werden. Alle anderen Systeme sind unabhängig vom Betriebssystem.

3 Änderungen am Entwurf

3.1 Package Highlevel

Das Paket `highlevel` bildet den Kern von BEAST. Es enthält die `MainClass`, über die BEAST gestartet wird. Weiterhin enthält es Interfaces zu den anderen Paketen, wodurch diese Pakete unabhängiger von einander sind und damit leichter auszutauschen. Der `CentralObjectProvider` generiert Instanzen dieser Interfaces und stellt sie dem `BEASTCommunicator` bereit.

3.1.1 CentralObjectProvider

`AbstractFactory` in `highlevel` ist jetzt keine Abstrakte Fabrik mehr.
Dieses Entwurfsmuster konnte nicht verwendet werden, da die zu erstellenden Objekte teilweise voneinander abhängig sind. Weiterhin gibt es Objekte, die mehrere Rollen einnehmen, d.h. sie implementieren unterschiedliche Interfaces (wie etwa der Parametereditor, der sowohl `ParameterSource`, als auch `ProjectSource` und `MainNotifier` implementiert).
Deshalb wurde sie durch das Interface `CentralObjectProvider` und die Klasse `PSECentralObjectProvider`, die dieses implementiert, ersetzt. `CentralObjectProvider` verwirklicht die ursprüngliche Funktion der Abstrakten Fabrik, unabhängig von konkreten Implementierungen zu sein.
`PSECentralObjectProvider` erzeugt die konkreten Objekte für unsere Implementierung der `highlevel`-Interfaces und stellt diese dem `BEASTCommunicator` zur Verfügung. Dieser muss weiterhin nur von den Interfaces wissen.

3.1.2 DisplaysStringsToUser

Es wurde das Interface `DisplaysStringsToUser` hinzugefügt. Es wird von allen Elementen, die dem Nutzer Text anzeigen, implementiert. Damit wird die Einbindung anderer Sprachen vereinfacht.

3.1.3 start-/stopReacting

Allen Interfaces zu Paketen mit GUI wurden die Methoden `stopReacting` und `resumeReacting` hinzugefügt. Diese verhindern, dass der Nutzer während einer laufen-

den Analyse Änderungen an dafür benötigten Daten vornimmt.

3.1.4 ProjectSource

Es wurde das Interface `ProjectSource` hinzugefügt. Es wird von `ParameterEditor` implementiert. Damit wird es möglich, das Speichern und Laden von ganzen Projekten in zukünftigen Versionen leichter einem anderen Fenster als dem Parametereditor zu überlassen.

3.1.5 isCorrect

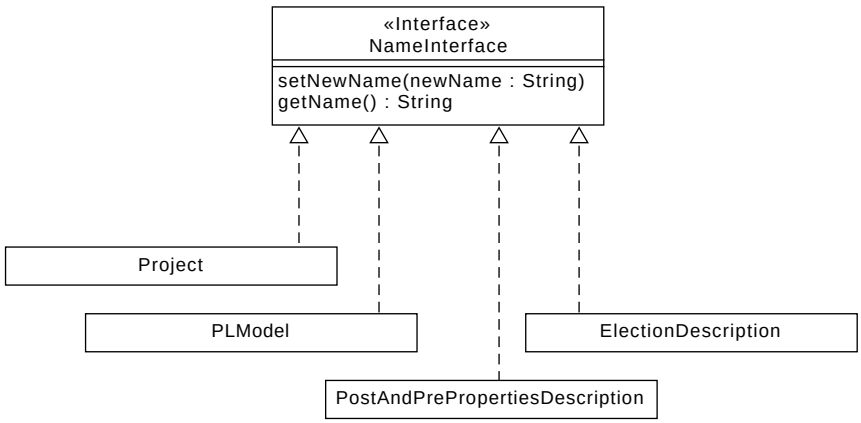
Interfaces zu Paketen, die Daten für die Analyse bereitstellen, wurde die Methode `isCorrect` hinzugefügt. Damit kann vor Start einer Analyse überprüft werden, ob die bereitgestellten Daten frei von Fehlern sind, die die Analyse beeinträchtigen würden.

3.2 UserActions

Alle `UserActions` der vier GUIs haben jetzt nur noch einen Verweis auf den ihnen zugehörigen Controller, und holen sich von diesem mit Gettern die von ihnen gebrauchten Klassen (`FileChooser`, `SaveBeforeChangeHandler`...).

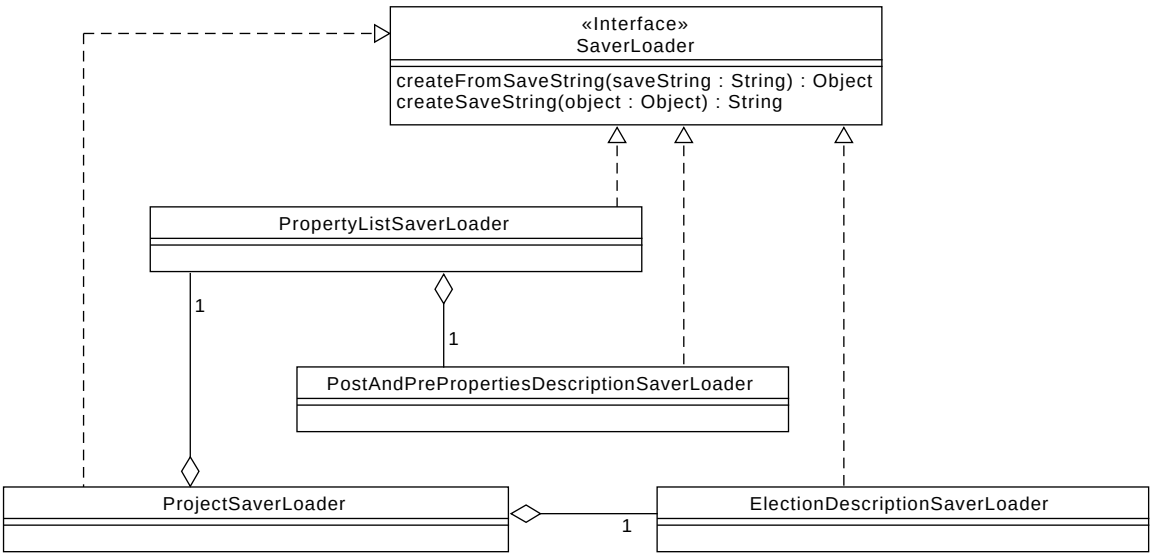
3.3 Package DataTypes

Die als Datei abspeicherbaren Datentypen `Project`, `PostAndPrePropertiesDescription`, `PLModel`, und `ElectrionDescription` implementieren nun alle das Interface `ChangeNameInterface`, das es dem `FileChooser` ermöglicht das `name`-Attribut dieser Klassen generisch zu verändern.



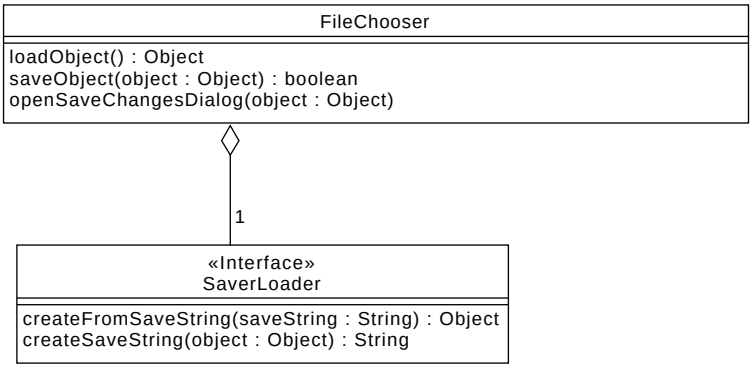
3.4 Package SaverLoader

`PostAndPrePropertiesDescriptionSaverLoader`, `ElectionDescriptionSaverLoader`, `ElectionCheckParameterSaverLoader` und `ProjectSaverLoader` implementieren nun das Interface `SaverLoader`. Dies ermöglicht es der Klasse `FileChooser` generisch zu seiner `SaverLoader`-Instanz passende Datentypen abzuspeichern und gegebene Dateien zu laden. Alle anderen `SaverLoader`-Klassen haben nur statische Methoden die von den schon genannten aufgerufen werden. Zudem gibt es noch eine `StringSaverLoader` Klasse, die mit `createSaveString` aus allen vom Nutzer editierbaren Strings alle Vorkommen von ">"durch ">> " ersetzt, beziehungsweise dies mit `createFromSaveString` rückgängig macht. Dies vehindert die Erstellung von nicht ladbaren Dateien trotz valider Nutzer-Eingaben.



3.4.1 FileChooser

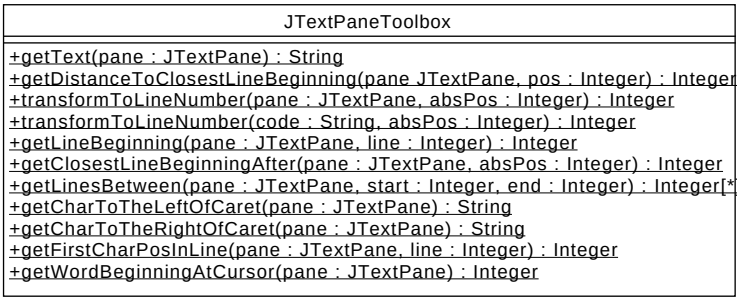
Diese Klasse kümmert sich um das Laden und Speichern der abspeicherbaren `datatypes`-Klassen. Sie besitzt eine SaverLoader Instanz mit der sie beim Aufrufen von `loadObject` und `saveObject` entsprechende Dateien generisch laden und Speichern kann.



3.5 Package Codearea

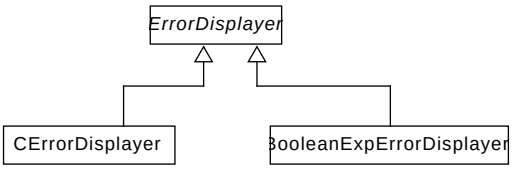
3.5.1 JTextPaneToolbox

Klasse `JTextPaneToolbox` wurde hinzugefügt. Diese enthält einige statische Methoden, welche oft benötigte, aber nicht zusammengehörige Funktionalität für `JTextPane` liefern. Dazu gehört unter anderem das Umwandeln absoluter Positionen in Zeilennummern.



3.5.2 Errordisplayer

`Errordisplayer` ist nun abstrakt. Von den erbenden Klassen müssen Fehlermeldungen generiert werden.



3.5.3 SaveTextBeforeRemove

DieKlasse `SaveTextBeforeRemove` wurde hinzugefügt. Diese speichert den Text einer `JTextPane`, sobald Text daraus entfernt wird. Dies ist nötig, da das `RemovedUpdate` des Styleddocuments keinen Zugriff auf den entfernten Text gewährt. Dieser wird jedoch benötigt, um Aktionen rückgängig zu machen. Implementiert wird die Funktionalität durch hören auf Keyevents.

3.5.4 TextLineNumber

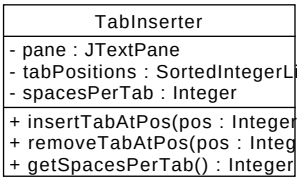
Klasse `TextLineNumber` wurde hinzugefügt, welche die Zeilennummer anzeigt. Diese Klasse wurde direkt aus
`https://tips4java.wordpress.com/2009/05/23/text-component-line-number/`
übernommen.

3.5.5 SquigglePainter

Klasse `SquigglePainter` wurde hinzugefügt. Diese unterstreicht Text in der `JTextPane` gezackt. Dies wird verwendet, um Fehler im Code anzuzeigen. Übernommen von
`https://tips4java.wordpress.com/2008/10/28/rectangle-painter/`

3.5.6 Tabinserter

`Tabinserter` wurde hinzugefügt. Dieser fügt Tabs in Form von Leerzeichen ein.



3.5.7 LineBeginningTabsHandler

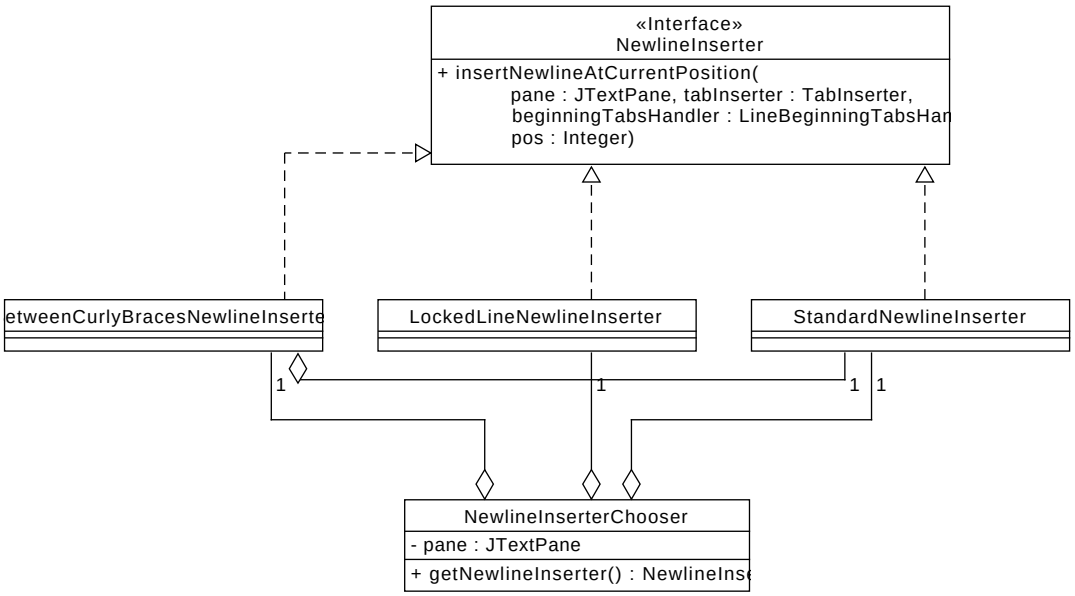
Interface `LineBeginningTabsHandler` und Implementierung
`CurlyBracesLineBeginningTabHandler` wurden hinzugefügt.
`LineBeginningTabsHandler` berechnet die benötigte Anzahl Tabs zu Beginn einer gegebenen Zeile. `CurlyBracesLineBeginningTabHandler` errechnet dies anhand der Anzahl geöffneter curly-Braces in vorangehenden Zeilen minus die Anzahl schließender curly-Braces am Ende der gegebenen Zeile

3.5.8 UserActions

Es gibt nun spezielle `UserActions` für Kopieren, Ausschneiden und Einfügen. Dies ist nötig um sicherzustellen, dass nicht editierbare Zeilen nicht durch diese Aktionen verändert werden.

3.5.9 NewlineInserter

`NewlineInserter` ist nun ein eigenes Paket. `BetweenCurlyBracesNewlineInserter` verwendet nun `StandardNewlineInserter`.



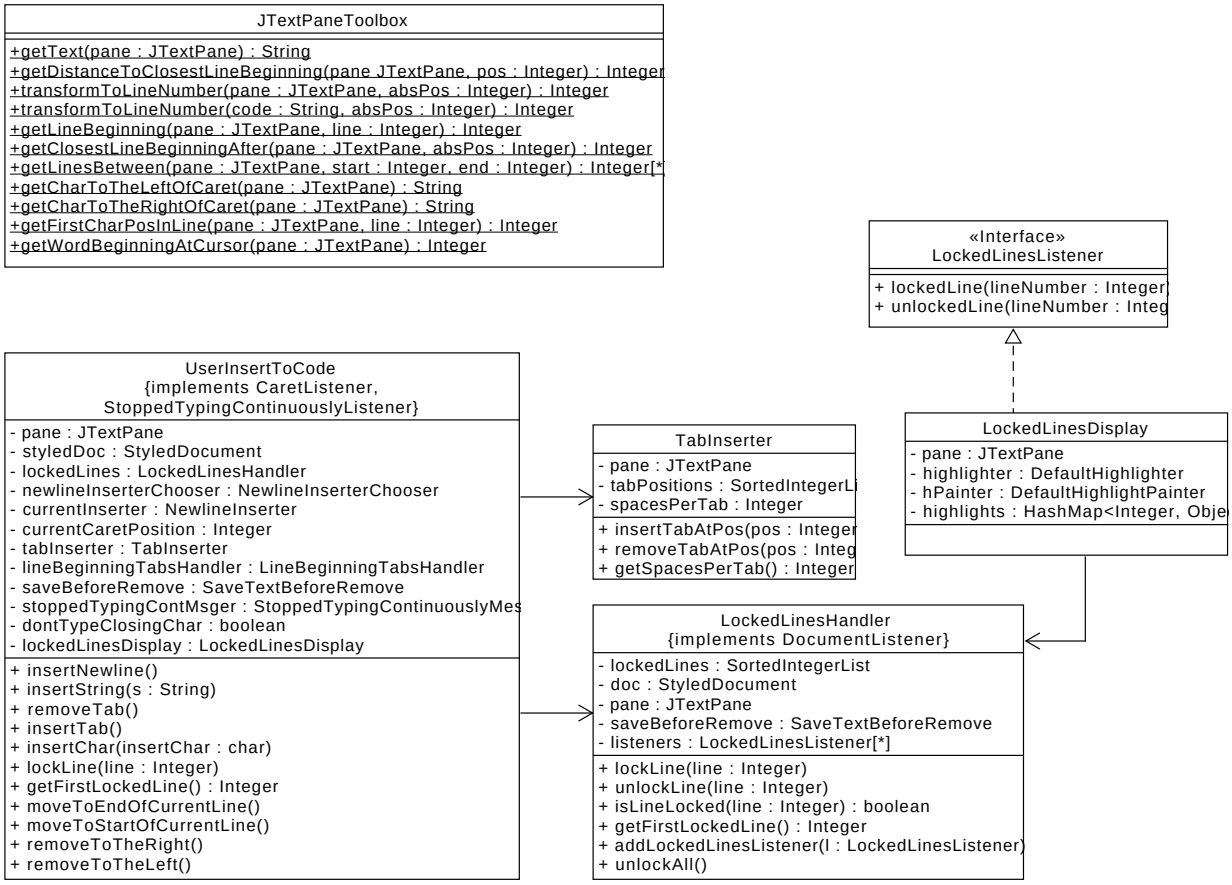
3.5.10 UserInsertToCode

Hinzu kommen folgende Funktionen:

- insertTab fügt an der Momentanen Position ein Tab ein
- insertChar Fügt das gegebene Zeichen an der momentanen Position ein
- getFirstLockedLine gibt die erste nicht editierbare Zeilennummer
- moveToEndOfCurrentLine Bewegt den Caret ans Ende der momentanen Zeile
- moveToStartOfCurrentLine Bewegt den Caret an den Start der momentanen Zeile
- removeToTheRight Entfernt das Zeichen rechts vom Caret
- removeToTheLeft Entfernt das Zeichen links vom Caret

Entfernt wurden folgende Funktionen:

- msgLockedLinesListeners: wird nun von `LockedLineHandler` übernommen



3.6 Package BooleanExpEditor

Der `BooleanExpEditor` besitzt jetzt eine Referenz auf die `CElectonDescriptionEditor`-Instanz, da dies zur Fehlerfindung durch den `BooleanExpEditorVariableErrorFinder` nötig ist.
Er bekommt vom Builder nun eine Referenz auf die `PropertyList`-Instanz, da so neu erstellte Eigenschaften in der Liste gespeichert werden können.

3.7 Package CElectionDescriptionEditor

Das Paket CElectionDescrptionEditor stellt die GUI bereit, mit der der Nutzer ein Wahlverfahren in C-Code eingeben kann.

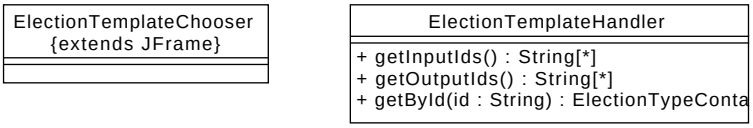
3.7.1 ChangeElectionTypeUserAction

Diese Klasse und der entsprechende Menüpunkt "Wahlart ändern" wurden entfernt.

3.7.2 Subpackage ElectionTemplates

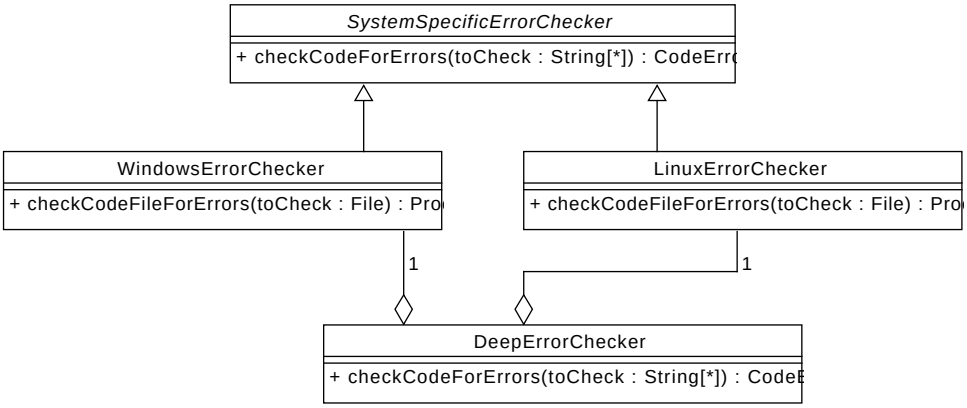
Das neue Paket `ElectionTemplates` zum Erstellen von `ElectionDescription`-Objekten kam hinzu. Dieses enthält folgende Klassen:

- `ElectionTemplateHandler`: Gibt alle Election Input und Output Datentypen und deren ids aus.
- `ElectionTemplateChooser`: Zeigt dem Benutzer einen Dialog, welcher es ermöglicht Input und Result eines neuen Wahlverfahrens zu wählen.



3.7.3 ErrorHandling

Die C-Fehlerfindung findet nun ausschließlich über Aufruf eines externen Compilers statt. Dieser Aufruf geschieht und das Parsen seiner Rückgabe findet in den Klassen `DeepErrorChecker`, `LinuxErrorChecker`, `WindowsErrorChecker` und `SystemSpecificErrorChecker` statt.



3.8 Package PropertyChecker

3.8.1 CBMCCodeGeneration

- Die Klasse `CBMCCodeGeneration` ist nicht mehr statisch.
  Sie wird in der Implementierung von der Klasse `CBMCProcessFactory` instantiiert. Diese Änderung war sinnvoll, da die Generierung des Codes jeweils von den Datentypen `ElectionDescription` und `PostAndPrePropertiesDescription` abhängt.

- Jede Instanz der Klasse `CBMCCodeGeneration` erstellt eine Instanz der Klasse `CBMCCodeGenerationVisitor`.
  Diese besitzen zwei neue Methoden, die einstellen, ob der Visitor zur Codegenerierung einer Vor- oder Nachbedingungen eines Wahlverfahrens verwendet wird.

3.8.2 PropertyChecker

- Die Klasse `CBMCResult` besitzt nun die Methode "createFailureExample" samt zugehöriger Untermethoden, welche zur Erstellung des Failurexamples genutzt werden. Deshalb besitzt die Klasse `Checker` diese Fähigkeit nicht mehr.

- Es wurden drei neue Klassen [1] erstellt, welche beim Parsen der Ergebnisse von CBMC mithelfen. Sie werden während des Parsens der Rückgabe von CBMC verwendet, um die Teilergebnisse in Listen zu speichern und sie am Ende als Array ausgeben zu können. Dies wurde auf diese Weise implementiert, da am Anfang des Parsens nicht bekannt sein kann, wie groß die Datentypen bei der Rückgabe werden würden und es die eigentliche Methode "createFailureExample" deutlich verkürzen konnte.

- Die Klasse `CheckerFactory` besitzt nun zwei neue Methoden:
  "getNewInstance(...)" wird dazu verwendet eine neue Instanz einer `CheckerFactory` zu erstellen, damit die `CheckerFactoryFactory` neue `CheckerFactorys` erstellen kann.
  Außerdem gibt es nun die Methode "getMatchingResult(int amount)", welche die gewünschte Anzahl an checkerspezifischen `Result` Objekten zurückgibt, sodass die `CheckerFactoryFactory` von diesen dann auf Wunsch so viele wie nötig erstellen kann.

3.9 Package PropertyList

Das Paket `PropertyList` hat sich im Unterpaket `Controller` geändert. Durch die Anbindung an die `highlevel`-Interfaces wurde es nötig, dass das Model der `PropertyList` kein Einzelstück mehr ist. Der Controller benötigt deshalb eine eigene Referenz auf das

[1] CBMCResultWrapper/long/singleArray/multiArray

Model, weil er nicht auf die einzelne Instanz zugreifen kann. Eine zentrale Controllerklasse übernimmt nun die Steuerung, anstatt wie vorgesehen einzelne Klassen.

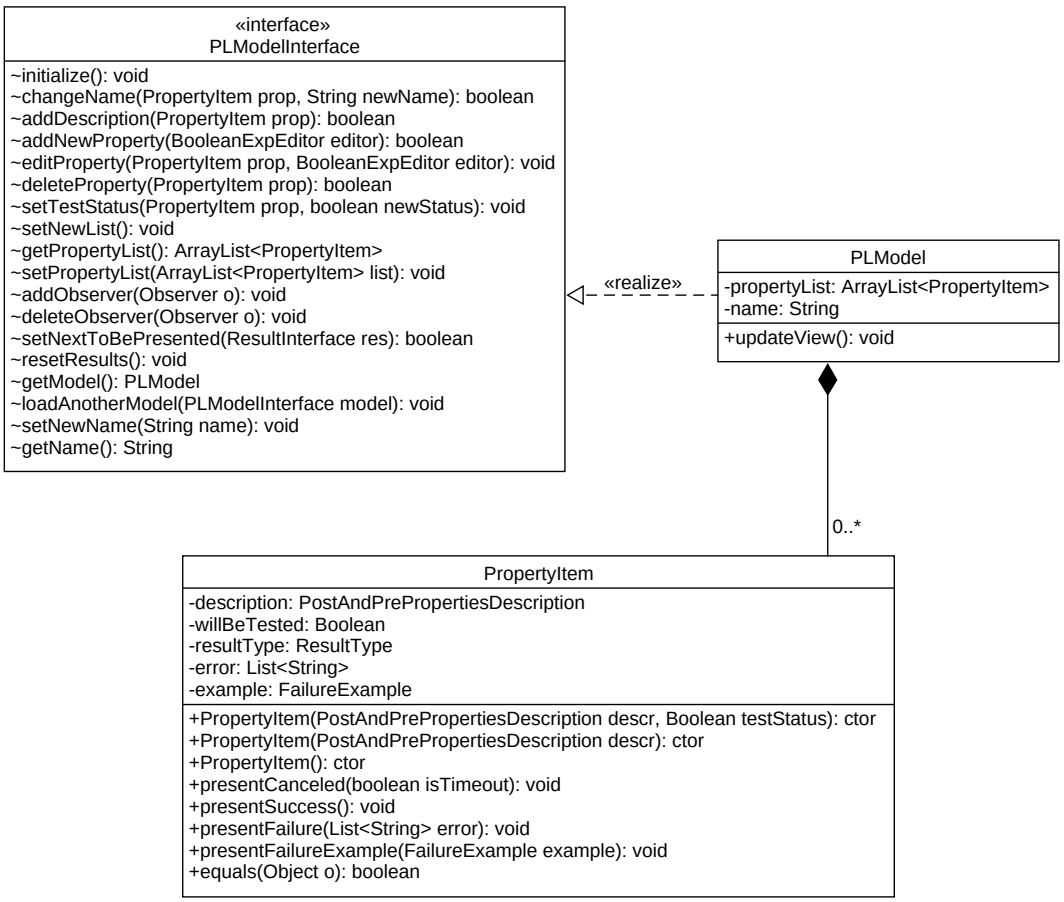
Die Methoden für Controller und Model wurden außerdem in eigenen Interfaces beschrieben, sodass ein schneller Überblick über die Methoden gegeben ist.

3.9.1 Invarianten

Invarianten für das Package ist das Funktionieren des BooleanExpEditor. Eine Referenz des Editors wird gebraucht, um die Liste dynamisch zu erweitern und Eigenschaften in den Editor laden zu können.

Weiter sind durch die Entscheidung, dass die Zählung für Kandidaten bei 1 beginnt, einige Stimmensummen im `ResultPresenterWindow` nur richtig, falls dies eingehalten wird. Stimmt etwa ein Wähler für Kandidat 0, so kann seine Stimme in der Auszählung nicht berücksichtigt werden. Die Zählung für Kandidaten soll bei 1 beginnen, weil das Programm auch für Nicht-Informatiker sein soll.

3.9.2 Subpackage Model

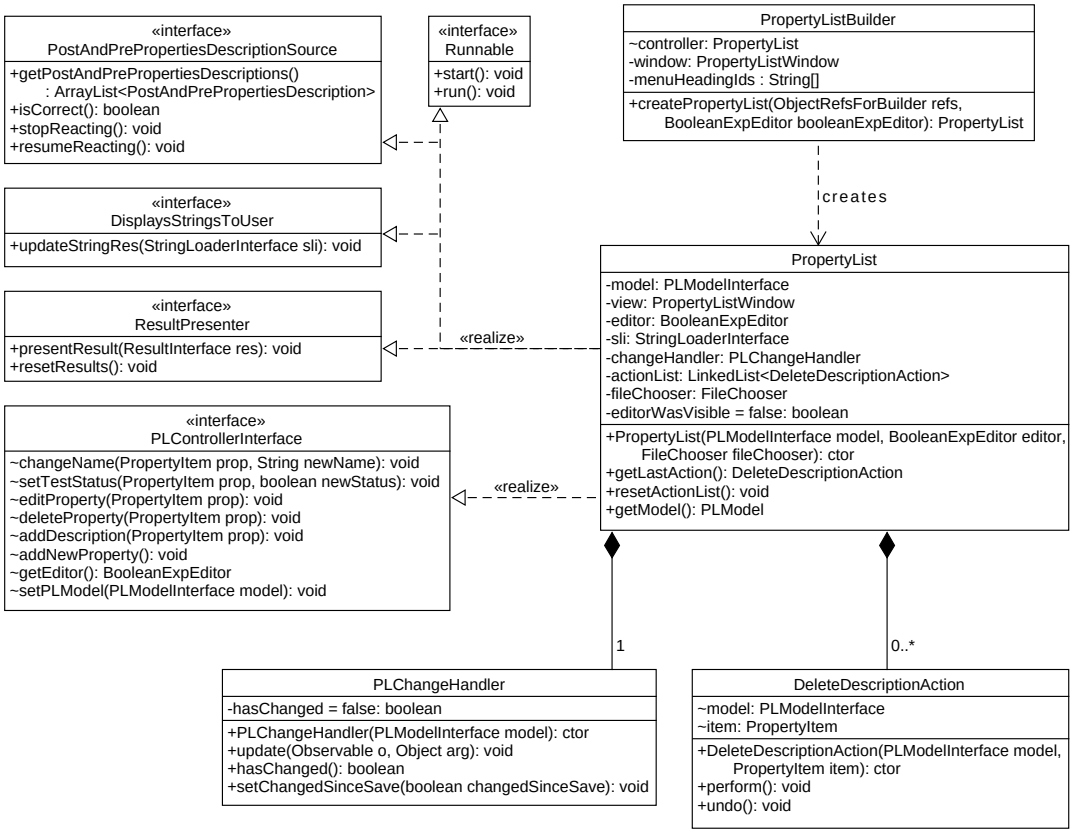


Die Klasse `PropertyList` wurde zu `PLModel` umbenannt. Sie hat keine Anbindung nach außerhalb des Pakets mehr. Das Interface `PostAndPrePropertiesDescriptionSource` wird nun vom Controller implementiert.

Die Klasse `PropertyItem` hält nun Daten zum Ergebnis der Analyse (zusätzlich zu `PropertiesDescription` und Teststatus).

Das Interface `PLModelInterface` beschreibt alle Manipulationen der Eigenschaftenliste.

3.9.3 Subpackage Controller



Die zentrale Klasse des Controllers heißt `PropertyList` und implementiert alle nötigen Interfaces nach außen. Das sind:

- `ResultPresenter` (im Entwurf nur vom View implementiert)
- `PostAndPrePropertiesDescriptionSource` (eigentlich im Entwurf im Model implementiert)
- `Runnable` (Klasse startet den View)
- `DisplaysStringsToUser` (zur Entgegennahme der Strings für den View)

- `PLControllerInterface` (alle möglichen Befehle für die `PropertyList`)

Die `Action-` und `ChangeListener` wurden aus dem Controller rausgezogen und direkt im View implementiert.

Die Klassen, die `ListChangeCommand` erweiterten (`ChangeDescription`, `ChangeDescriptionName`, `AddStandardDescription`, `AddNewDescription`, `ChangeTestedStatus`), sind nur noch Methoden im Interface des Controllers (void changeName(PropertyItem prop, String newName); usw.). Das geschah, weil sie so kleine Änderungen an der `PropertyList` bedeuten, dass sie nicht rückgängig gemacht werden müssen. Stattdessen kann der Nutzer neu erstellte Eigenschaften mit dem Klick auf den entsprechenden Button wieder löschen. Lediglich `DeleteDescriptionAction` ist den Undo wert und wurde deshalb in einer eigenen Klasse gekapselt.

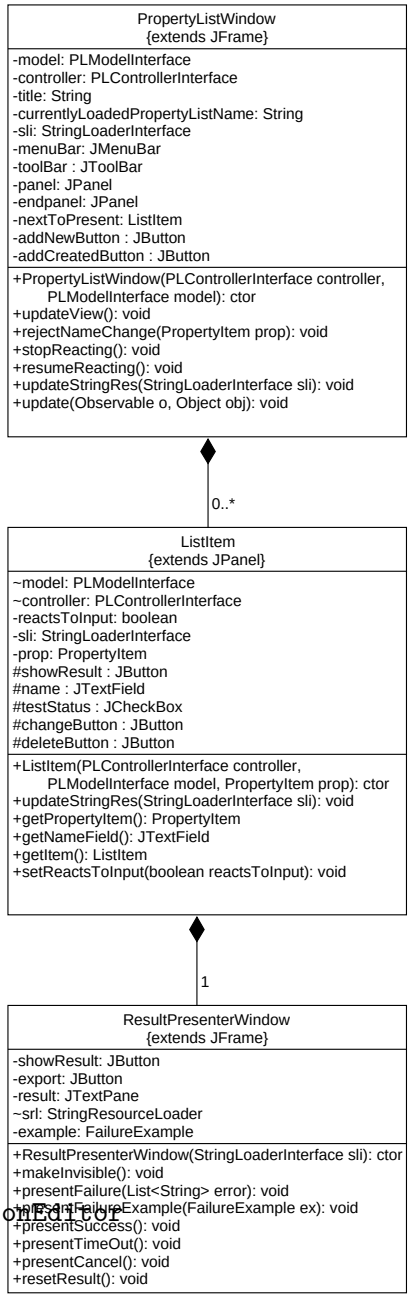
Der `SaveBeforeChangeHandler` war im Entwurf noch nicht beschrieben.

3.9.4 Subpackage View

Die Elemente des View sind großteils gleich geblieben. Sie wurden allerdings nicht mit einem GUI-Builder erstellt, da sonst nicht dynamisch Komponenten hinzugefügt werden könnten (Formulare mit fester Anzahl von Komponenten).

Im Entwurf war angedacht, dass das Model die View direkt von Änderungen benachrichtigt. Nun wird das Beobachtermuster benutzt, um der View Änderungen im Model mitzuteilen. In diesem Falle wird die Liste der Eigenschaften (ArrayList¡ListItem¿) neu aufgebaut.

Neu hinzugekommen ist die Klasse `ResultPresenterWindow`, die die Swing-Klasse `JFrame` erweitert. Dadurch war es nicht möglich, sie als Kinder des Hauptfensters zu deklarieren. Aber durch die Entscheidung gegen eine `JTextPane` konnte das Layout aus dem Pflichtenheft besser dargestellt werden.



3.9.5 Class ParameterEditor

Die Klasse `ParameterEditor` implementiert jetzt nicht mehr `BEASTCommunicator` aus `highlevel`. Damit wird `ParameterEditor` klar von der Kommunikation zwischen den einzelnen Teilen von BEAST getrennt.

3.9.6 UserActions

Es wurden neue `UserActions` hinzugefügt:

- `OptionsUserAction`, um es dem Nutzer zu ermöglichen, Einstellungen wie etwa die Sprache zu ändern.
- `ShowHideBooleanExpEditor`, `ShowHideCElectionEditor` und `ShowHidePropertyList`, um die anderen GUI-Fenster vom Parametereditor aus öffnen und schließen zu können.

3.9.7 Handler

Es wurden neue Handler hinzugefügt:

- `ArgumentHandler`, um die benutzerdefinierten Argumente für CBMC zu verarbeiten.
- `SaveBeforeChangeHandler`, um sicherzustellen, dass der Nutzer nicht versehentlich durch Laden oder Erzeugen eines neuen Projekts ein ungespeichertes Projekt verwirft.

3.9.8 AboutWindow

Es wurde die Klasse `AboutWindow` hinzugefügt, die dem Nutzer das Akronym BEAST erklären und die aktuelle Versionsnummer sowie das Build-Datum anzeigt.

3.10 Package Toolbox

Das Package Toolbox beinhaltet Hilfsklassen, die einzelne spezifische Aufgaben übernehmen, die keinem anderen Package klar zuteilbar ist.

3.10.1 SortedIntegerList

Eine einfache Wrapperklasse, welche eine stets sortierte Liste von Integern enthält. Diese wird verwendet von `LockedLinesHandler` und `TabInserter`.

3.10.2 RepaintThread

Implementiert `Runnable`. Bekommt einen `JFrame` für welchen er 60 Mal pro Sekunde die repaint-Funktion aufruft. Dies ist benötigt, da die Frames auf Windows sonst nicht korrekt rendern, falls zuvor ein anderes Fenster vor ihnen war.

3.10.3 CCodeHelper

Diese Klasse enthält Funktionalität, um aus den internen Datentypen C Code zu erstellen. Sie wird sowohl von der Codegenerierung als auch dem C-Editor verwendet.

3.10.4 ActionIdAndListener

Eine einfache Wrapper-Klasse welche Zugriff auf einen `ActionListener` und die dazugehörige String-Id bietet.

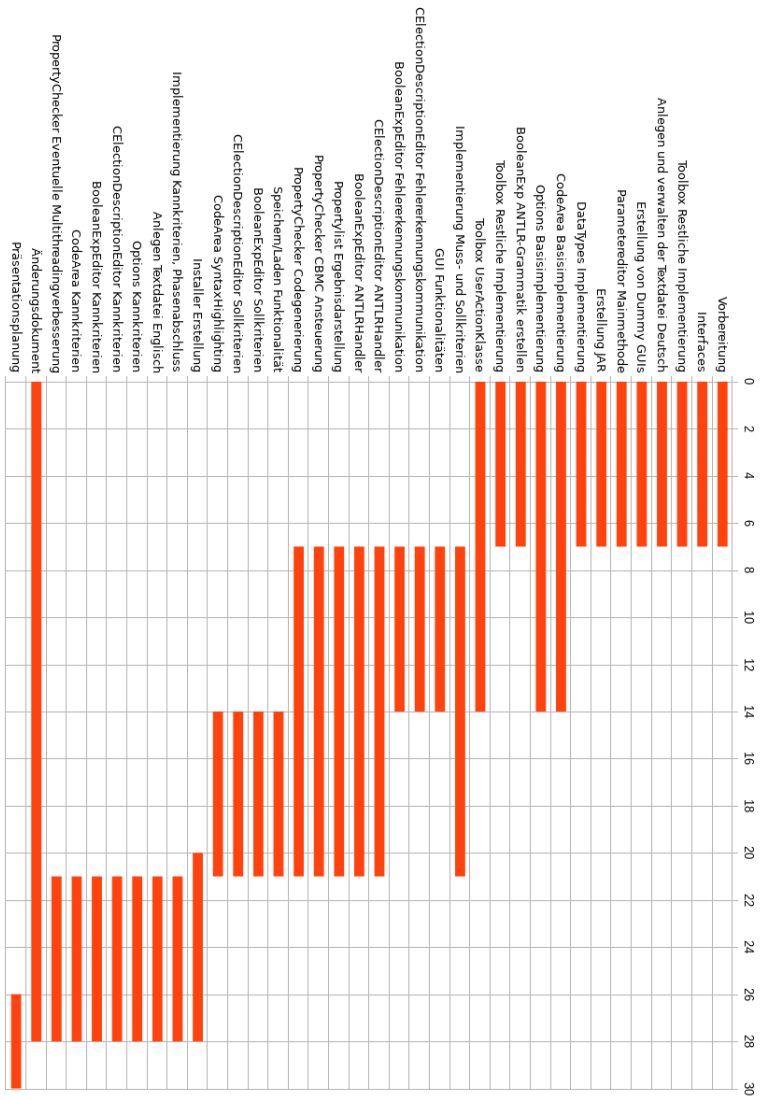
3.10.5 Tuple

Eine Klasse, welche zwei verschiedene Typen hält.

4 Zeitablauf Implementierungsphase

4.1 Geplanter Ablauf

Hier das ursprüngliche GANTT-Diagramm aus dem Entwurfsdokument das die Zeitplanung von Tag 0 bis Tag 30 der Implementierungsphase darstellt.



4.2 Tatsächlicher Ablauf

Im Großen und Ganzen wurde der ursprüngliche Plan bis auf kleinere Verzögerungen eingehalten.
Zu Anfang wurde die Vorbereitungsphase ein paar Tage verzögert abgeschlossen da die Zeit nicht reichte um die Antlr-Grammatik für die formalen Eigenschaften noch in der ersten Woche fertigzustellen. Dies geschah ein paar Tage später während die Muss- und Sollkriterienphase schon angelaufen war.
In dieser Phase gab es auch Abweichungen zum ursprünglichen Plan. So waren z.B. grundlegende GUI-Funktionalitäten nach Woche 2 noch nicht implementiert, da die Arbeitshaltung zweier Gruppenteilnehmer suboptimal war. Dies verzögerte die ganze Phase und führte schließlich zu einer Aufgabenumverteilung in Woche 3. Ziel dieser war es kurzfristig für den Rest der Implementierungsphase eine ausgeglichenere und zielführende Aufgabenverteilung zu schaffen, damit die vorletzte Artefakt-Abgabe am Ende der Woche 3 - trotz bisheriger Schwierigkeiten - erfolgreich erbracht werden konnte. Bis auf kleinere Bugs, und noch nicht komplett fertig gestellte Laden und Speichern-Funktionalitäten, erfolgt diese dann auch.
Die letzte Woche bestand danach hauptsächlich aus Debuggen, dem Implementieren von Kann-Kriterien und dem Erstellen dieses Dokuments.
Es folgt ein GANTT-Diagramm des tatsächlichen Zeitablauf.

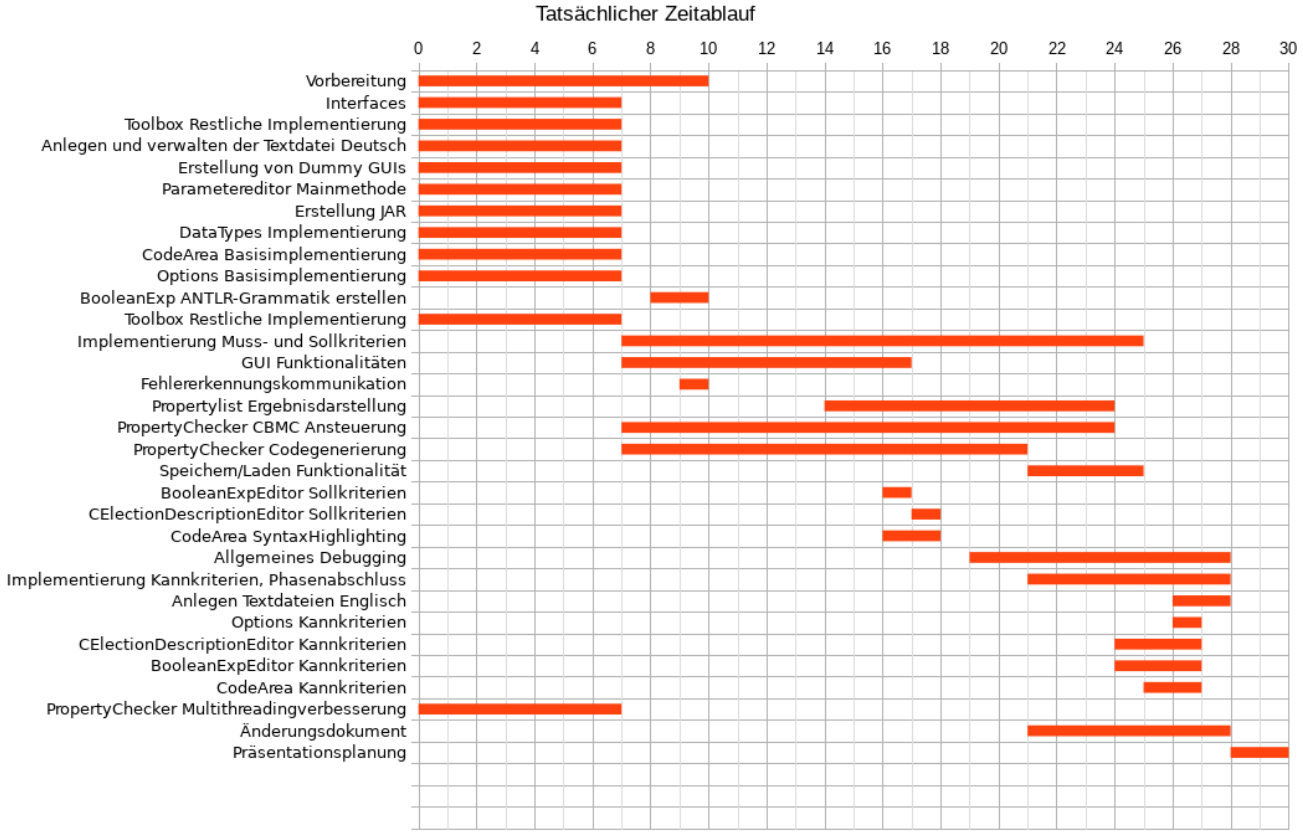


Abbildung 4.1: GANTT-Diagramm das den tatsächlichen Zeitablauf der Implementierungsphase von Tag 0 bis Tag 30 darstellt.

5 Statistiken

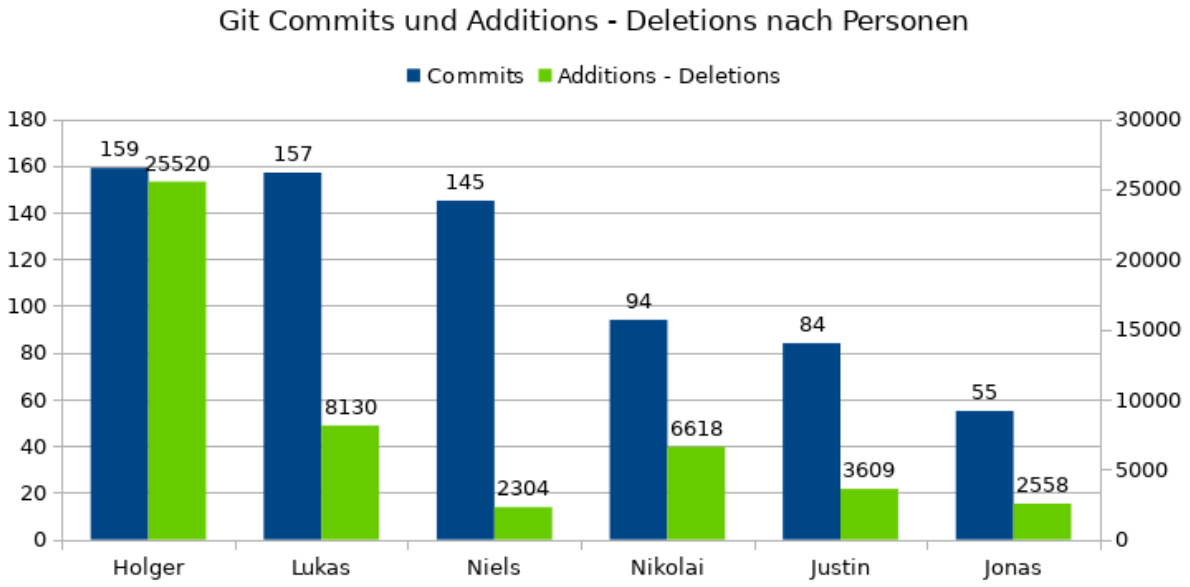


Abbildung 5.1: Graph der Git Commits und Differenz von Addtions und Deletions nach Personen aufgeteilt darstellt.

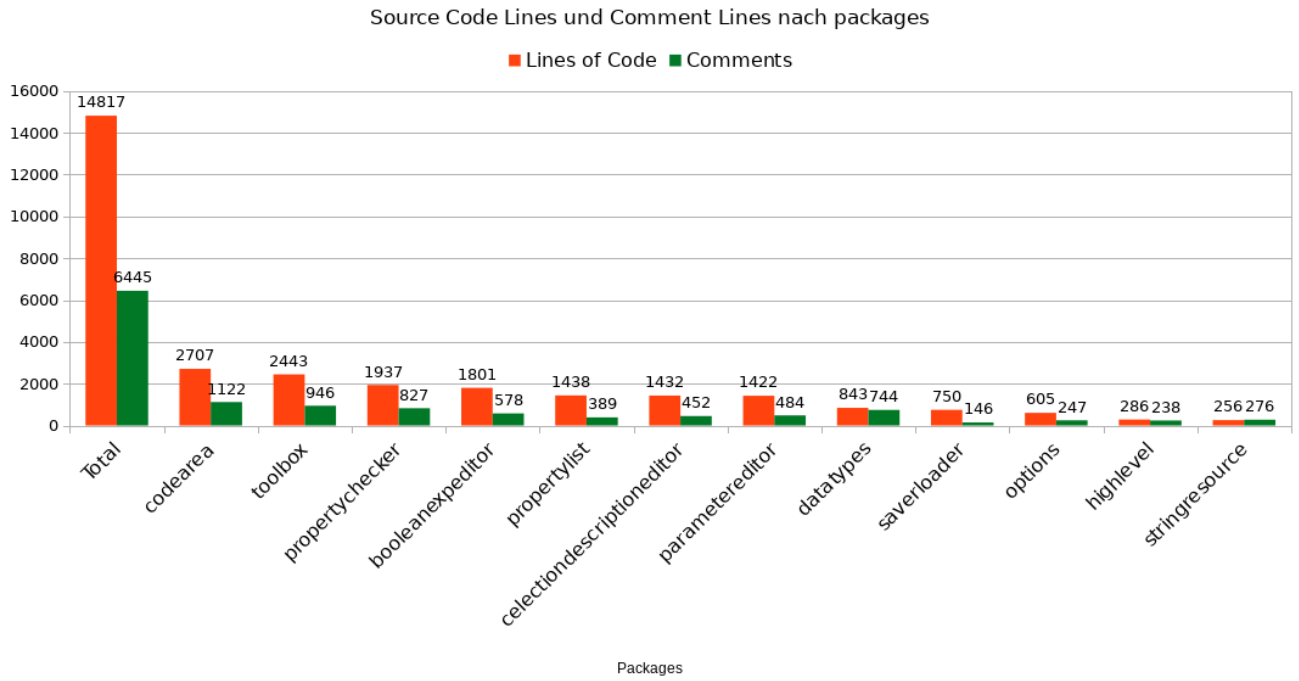


Abbildung 5.2: Graph der die Java Source Code Lines und Comment Lines nach packages aufgeteilt darstellt.