

# **Pflichtenheft**

Hanselmann, Hecht, Klein, Schnell, Stapelbroek, Wonig

4. Dezember 2016

# Inhaltsverzeichnis

# **Abbildungsverzeichnis**

# **Abkürzungsverzeichnis**

# 1 Produktübersicht

Wahlverfahren bilden den Grundstein unserer Demokratie. Dabei werden viele Anforderungen an sie gestellt, welche unsere intuitiven Ideen über Gerechtigkeit formalisieren: Proportionalität, Anonymität, etc. Moderne Wahlverfahren sind oft so komplex, dass sie viele überraschende und teils unerwünschte Eigenschaften haben. Ein Nachweis deren Abwesenheit ist nicht trivial. So wurde beispielsweise 2008 das Bundestagswahlrecht vom BVerfG für verfassungswidrig erklärt, da es unter anderem die Gleichheit der Wirkung verschiedener Stimmen verletzte. Auf der anderen Seite ist es auch sehr schwer, Wahlverfahren auf die Präsenz erwünschter Eigenschaften zu untersuchen.

**BMC!** (**BMC!**) wird normalerweise dazu verwendet zu überprüfen, ob ein gegebenes Programm gegebene Eigenschaften erfüllt. Da dieses Problem im Allgemeinen unentscheidbar ist, werden nur endliche Codepfade überprüft. Dadurch wird der Zustandsraum endlich und das Problem entscheidbar. Um dies zu bewerkstelligen, werden potentiell unendliche Codepfade - also Schleifen - bis zu einer vom Benutzer bestimmten Grenze aufgerollt. Danach wird eine SAT-Formel erstellt, die erfüllbar ist, genau dann wenn das Programm einen Zustand einnehmen kann, welcher die gegebene Eigenschaft nicht erfüllt. Dies ist vollautomatisch und gibt bei Nichterfüllung das Gegenbeispiel zurück.

In unserem Fall kann **BMC!** konkret dazu verwendet werden, ein C-Programm darauf zu untersuchen ob es im Falle gegebener Vorbedingungen gegebene Nachbedingungen erfüllt. Dies wird dazu verwendet, obige Problemstellung innerhalb einer bestimmten Genauigkeit zu lösen: so kann ein in C beschriebenes Wahlverfahren, wie z.B. die einfache Mehrheitswahl darauf geprüft werden, ob es bestimmte Eigenschaften erfüllt.

Unser Programm ist im Wesentlichen eine sehr umfangreiche Schnittstelle um mit **CBMC!** (**CBMC!**) zu kommunizieren. Es bietet dem Benutzer über eine **GUI!** (**GUI!**) die Möglichkeit, formale Eigenschaften für Wahlverfahren sowie diese Wahlverfahren selbst, anzugeben und zu editieren. Weiterhin liefert es Möglichkeiten, die Interaktion mit **CBMC!** zu gestalten: Für wie viele Wähler etc. die Eigenschaft überprüft werden soll. Nach erfolgreicher Überprüfung durch **CBMC!** bekommt der Benutzer schließlich eine Antwort des Programms, in der er bei Nichterfüllung der Eigenschaft ein Gegenbeispiel angezeigt bekommt. Wird kein Gegenbeispiel gefunden, so wird eine Erfolgsmeldung ausgegeben. All dies wird graphisch über die **GUI!** aufbereitet.

Die **GUI!** ist nach Funktionalität in vier Teilen angeordnet:

1. „C-Editor“: Code-Editor für Wahlverfahren in der Programmiersprache C
2. „Eigenschaftenliste“: Listenansicht aller Eigenschaften, die für dieses Wahlverfahren untersucht werden sollen, und Ausgabe der Analyse
3. „Eigenschafteneditor“: Editor für Spezifikation formaler Eigenschaften als boolsche Ausdrücke in nachfolgend beschriebener Grammatik

#### 4. „Parametereditor“: Eingabe von Parametern einer zu analysierenden Wahl

### 1.1 Die Syntax zur Angabe der formalen Eigenschaften

In diesem Abschnitt wird ein grober Überblick über die Sprache, welche der Eigenschafteneditor verwendet, gegeben. Es handelt sich um ein `D` der C-Sprache mit einigen Ergänzungen. Diese werden im Folgenden erläutert.

Formale Eigenschaften werden in Vor- und Nachbedingungen unterteilt. Diese wiederum werden vom User als eine Liste boolscher Ausdrücke angegeben. Die Sprache erlaubt folgende Konstrukte zur Darstellung boolscher Ausdrücke:

- Folgende, aus C übernommene, binäre Operatoren:

Operator	Symbol	Erwartet Argumente	Rückgabewert
Das logische Und	<code>&amp;&amp;</code>	zwei boolsche Ausdrücke	<code>true</code> oder <code>false</code>
Das logische Oder	<code>  </code>	zwei boolsche Ausdrücke	<code>true</code> oder <code>false</code>

- Folgende, zusätzlich hinzukommende binäre Operatoren:

Operator	Symbol	Erwartet Argumente	Rückgabewert
Die logische Implikation	<code>==&gt;</code>	zwei boolsche Ausdrücke	<code>true</code> oder <code>false</code>
Die logische Äquivalenz	<code>&lt;==&gt;</code>	zwei boolsche Ausdrücke	<code>true</code> oder <code>false</code>

Beispiel: `x > y <==> x + 1 > y + 1`

Bedeutung: `x` ist größer als `y` genau dann, wenn auch `x + 1` größer als `y + 1` ist.

- Folgende aus C bekannte Operatoren. „Vergleichbarer Typen“ bedeutet, dass zwei Variablen dieser Typen bereits in C mit demselben Operator vergleichbar sind.

Operator	Symbol	Erwartet Argumente	Rückgabewert
Gleichheit	<code>==</code>	zwei Variablen vergleichbarer Typen	<code>true</code> oder <code>false</code>
Ungleichheit	<code>!=</code>	zwei Variablen vergleichbarer Typen	<code>true</code> oder <code>false</code>
kleiner als	<code>&lt;</code>	zwei Variablen vergleichbarer Typen	<code>true</code> oder <code>false</code>
kleiner gleich	<code>&lt;=</code>	zwei Variablen vergleichbarer Typen	<code>true</code> oder <code>false</code>
größer als	<code>&gt;</code>	zwei Variablen vergleichbarer Typen	<code>true</code> oder <code>false</code>
größer gleich	<code>&gt;=</code>	zwei Variablen vergleichbarer Typen	<code>true</code> oder <code>false</code>

- Symbolische Variablen vom Typ Wähler, Kandidat oder Sitz. Für deren Benennung gelten dieselben Regeln wie für die Benennung von Variablen in C.

Beispiel: Wähler `v`, Kandidat `c`

- Quantoren für Wähler, Kandidaten und Sitze in der Form von Makros. Ein bisher ungenutzter Variablenname wird als Argument erwartet. Dieser kann in dem darauf folgenden Ausdruck als symbolische Variable entsprechenden Typs verwendet werden.

Beispiel: `FOR_ALL_VOTERS(v) : EXISTS_ONE_CANDIDATE(c) : v mag c`

Bedeutung: Für jeden Wähler (*v*) gibt es zumindest einen Kandidaten (*c*), sodass das Tupel die Eigenschaft “der Wähler *v* mag den Kandidaten *c*“ erfüllt (Dies ist nur ein Beispiel, der Editor wird die Eigenschaft ‘mögen’ nicht zu Verfügung stellen. An dessen Stelle könnte jeder andere binäre Operator stehen, welcher eine Variable vom Typ Wähler und eine Variable vom Typ Kandidat erwartet und `true` oder `false` zurückgibt).

- Ausgabe der Anzahl der Stimmen für einen Kandidaten in der Form eines Makros
- Viele Eigenschaften benötigen zu ihrer Überprüfung den Vergleich mehrerer Wahldurchläufe. Dies wird ermöglicht durch die Variablen `VOTESx()` und `ELECTx`. Dabei steht ‘x’ für die Nummer des Wahldurchgangs. `VOTESx()` erwartet als Argument eine symbolische Variable vom Typ Wähler. Die Methode gibt die Stimme zurück, welche *v* im Wahldurchgang *x* abgegeben hat. `ELECTx` erwartet kein Argument und gibt das Wahlergebnis im *x*-ten Durchgang zurück. Der Rückgabebetyp beider Makros hängt von der Kategorie des Wahlverfahrens ab. Gibt das Wahlverfahren zum Beispiel nur einen “Gewinner“ aus, so ist `ELECTx` vom Typ Kandidat.

Beispiel: `FOR_ALL_VOTERS(v) : VOTES1(v) == VOTES2(v)`

Bedeutung: Alle Wähler wählen in beiden Wahlen (`VOTES1` und `VOTES2`) gleich.

Beispiel: `ELECT1 == ELECT2`

Bedeutung: Das Ergebnis des ersten Wahldurchlaufs stimmt mit dem des zweiten Wahldurchlaufs überein.

- Folgende Konstanten: Anzahl Wähler (*V*), Anzahl Kandidaten (*C*) und Anzahl Sitze. (*S*)

Beendet wird ein boolscher Ausdruck durch ein Semikolon.

## 2 Zielbestimmung

Ziel des Programmes ist es eine Lösung zur Analyse von formalen Eigenschaften von Wahlverfahren zu präsentieren. Die Analyse solcher Eigenschaften ist ein nicht entscheidbares Problem und wird deshalb innerhalb von Grenzen mit Hilfe eines **BMC!** ausgeführt. Das Programm soll auch von Nicht-Informatikern mit minimalem Aufwand erlernt und eingesetzt werden können. Das Programm soll Folgendes bereitstellen:

- Eine Möglichkeit zur Beschreibung eines Wahlverfahrens, das in C-Code geschrieben ist
- Eine Möglichkeit zur Beschreibung von formalen Eigenschaften, welche das Wahlverfahren erfüllen soll, in der in ?? beschriebenen Syntax
- Eine Möglichkeit zum Angeben der Parameter des angegebenen Wahlverfahrens (Anzahl Wähler, Anzahl Kandidaten, Anzahl Sitze)
- Eine Möglichkeit, die Analyse auszuführen
- Eine Ausgabe des Ergebnisses der Analyse: eine Erfolgsmeldung falls alle Eigenschaften erfüllt werden und Präsentation eines Gegenbeispiels sonst

Die Analyse der gegebenen Eigenschaften wird durch **CBMC!** geschehen. Aufgabe des Programmes wird sein, die Eingaben des Benutzers aufzubereiten, so dass **CBMC!** aufgerufen werden kann. Die Ausgabe von **CBMC!** wird dann vom Programm interpretiert und dem Benutzer präsentiert.

Die Analyse der Eigenschaften lässt sich auch schon ohne Verwendung unseres Programms erledigen. Allerdings wäre der damit verbundene Lern- und Einarbeitungsaufwand sehr hoch. Weiterhin ist damit viel, sich jedes Mal wiederholender Aufwand, verbunden, der sich automatisieren lässt. Daher ist ein Schwerpunkt unseres Programms einfache Benutzbarkeit, besonders für Nicht-Informatiker. Dies soll erreicht werden über eine **GUI!**, welche oft benötigte Funktionalität bereitstellt. Einfache syntaktische Fehler im Code sollen während des Editierens erkannt werden. Dadurch soll die Untersuchung von Wahlverfahren leichter und schneller werden, was den Mehrwert unseres Programmes ausmacht.

### 2.1 Musskriterien

#### 2.1.1 Allgemein

- Das Programm kann auf 32-Bit Versionen von Windows und Linux-Betriebssystemen betrieben werden.



- Alle Abhängigkeiten werden mit dem Programm ausgeliefert.

### 2.1.2 C-Editor

Das Programm bietet einen Editor für das zu prüfende Wahlverfahren. Der Editor erwartet eine Eingabe in der Programmiersprache C.

Folgende Funktionalität bietet der C-Editor:

- Eingeegebener Code kann abgespeichert werden.
- Gespeicherter Code kann in den C-Editor geladen werden.
- Der Code-Editor zeigt Fehler im eingegebenen Code an.
- Aktionen können widerrufen und wiederhergestellt werden.

### 2.1.3 Eigenschafteneditor

Ein Editor wird bereitgestellt, in den formale Eigenschaften eingegeben werden können. Die Eingabe erfolgt in der in ?? beschriebenen Syntax. Die eingegebenen Eigenschaften sind Parameter bei der Ansteuerung von **CBMC!**.

Folgende Funktionalität bietet der Eigenschafteneditor:

- Eine formale Eigenschaft kann abgespeichert werden.
- Gespeicherte Eigenschaften können in den Eigenschafteneditor geladen werden.
- Ein Fehler wird angezeigt, wenn kein korrekter boolescher Ausdruck eingegeben wurde.
- Bereitgestellte Makros, symbolische Variablen und Operationen können verwendet werden.

### 2.1.4 Eigenschaftenliste

Das Ergebnis der Analyse wird vom Programm graphisch ausgegeben. Falls alle Eigenschaften erfüllt sind, wird eine Erfolgsmeldung ausgegeben, ansonsten wird ein Gegenbeispiel ausgegeben.

- Vordefinierte Standardeigenschaften (z.B. Anonymität) können einzeln in die Analyse mitaufgenommen werden.
- Das Ergebnis der Analyse für diese Standardeigenschaften wird für jede Eigenschaft einzeln ausgegeben.
- Eigenschaften können der Liste hinzugefügt werden und von ihr entfernt werden.

### 2.1.5 Parametereditor

Parameter für eine Wahl (Anzahl der Wähler, Kandidaten und Sitzen) können festgelegt werden. Bei der Analyse wird das Wahlverfahren nur innerhalb dieser Wahlparameter geprüft. Werden keine Parameter angegeben, wird **CBMC!** ohne sie aufgerufen.

Folgende Funktionalität bietet der Parametereditor:

- Die Anzahl der Wähler, Kandidaten und Sitze kann festgelegt werden.
- Die Analyse des Wahlverfahrens kann gestartet werden.
- Es kann ein Timeout festgelegt werden, nach dem die Analyse automatisch abgebrochen wird.
- Projektdaten (Wahlverfahren, Eigenschaften, Parameter) können abgespeichert werden und Projekte können geladen werden.

## 2.2 Sollkriterien

### 2.2.1 C-Editor

Folgende Funktionalität bietet der C-Editor zusätzlich:

- Syntax-Highlighting
- Tastatur-Shortcuts
- Zeilennummern
- Codevorlagen können in den Editor geladen werden.
- Beim Laden eines Wahlverfahrens wird das Format der geladenen Datei überprüft.
- Fehleranzeige bei Konstrukten, die der C-Grammatik widersprechen

### 2.2.2 Eigenschafteneditor

Folgende Funktionalität bietet der Eigenschafteneditor zusätzlich:

- Syntax-Highlighting
- Fehleranzeige bei syntaktischen Fehlern
- Tastatur-Shortcuts

### 2.2.3 Parametereditor

Folgende Funktionalität bietet der Parametereditor zusätzlich:

- Die Anzahl der Wähler, Kandidaten und Sitze kann als Intervall festgelegt werden (z.B. von  $x$  bis  $y$  Wähler,  $x < y$ ).
- Die Analyse von **CBMC!** kann jederzeit abgebrochen werden.

## **2.3 Wunschkriterien**

### **2.3.1 Allgemein**

- Das Programm kann auf einem Mac betrieben werden.

### **2.3.2 C-Editor**

Folgende Funktionalität bietet der C-Editor zusätzlich:

- Code-Completion
- Automatisches Einrücken
- Optionen für Individualisierung (z.B. Fontwahl)
- Fehleranzeige asynchron während der Eingabe

### **2.3.3 Eigenschafteneditor**

- Code-Completion

## **2.4 Abgrenzungskriterien**

- Das Programm kann keine Angabe darüber machen, wie lange die Überprüfung einer Eigenschaft dauern wird.

## 3 Produkteinsatz

Das Programm dient dazu, Wahlverfahren auf ihre formalen Eigenschaften hin zu analysieren. Es richtet sich an Benutzer, die ein Interesse an der Erforschung oder Entwicklung solcher Wahlverfahren haben. Benutzer kommen aus dem Bereich der theoretischen Informatik oder der Politikwissenschaft. Deshalb soll das Programm auch für Nicht-Informatiker zugänglich sein. Für die Bedienung des Programms ist jedoch Kenntnis der Programmiersprache C und der Aussagen- und Prädikatenlogik nötig.

Das Erstellen von neuen Wahlverfahren gestaltet sich ohne Hilfsmittel als schwierig. Besonders aber die Analyse solcher Wahlverfahren auf die Einhaltung von gegebenen Vor- und Nachbedingungen ist für viele Menschen eine Unmöglichkeit. Das Produkt soll diese Analyse stark vereinfachen.

### 3.1 Anwendungsbereiche

- Universität
- Forschung

### 3.2 Zielgruppen

- Wahlforscher
- Mitarbeiter in Prüfstellen
- Softwareentwickler
- Hobbyisten

### 3.3 Betriebsbedingungen

Das Produkt kommt in einer Büroumgebung zum Einsatz.

### 3.4 Szenarien

#### Szenario 1

Ein Wahlforscher wird von der Regierung eines Landes beauftragt ein neues Wahlverfahren zu entwerfen. Dieses soll bestimmte, vorgegebene formale Eigenschaften erfüllen.

Er installiert das Tool und entwickelt mit diesem eine erste Version des Wahlverfahrens. Zusätzlich gibt er die vorgegebenen Eigenschaften als formale Eigenschaften in das Tool ein.

Nun gibt er Parameter zum Prüfen an (Wähler, Stimmen, Sitze, Timeout) und lässt das Wahlverfahren auf die formalen Eigenschaften prüfen.

Das Ergebnis des Prüfens zeigt, dass nicht alle Eigenschaften vom Wahlverfahren erfüllt werden. Mithilfe der Informationen aus der Ergebnisausgabe analysiert und modifiziert der Entwickler das Wahlverfahren so, dass es alle Eigenschaften erfüllt.

Er prüft es nochmals auf alle Eigenschaften und nun werden alle erfüllt.

Zuletzt speichert er das Gesamtprojekt und einzeln das Wahlverfahren ab. Letzteres sendet er dann an eine Prüfstelle der Regierung.

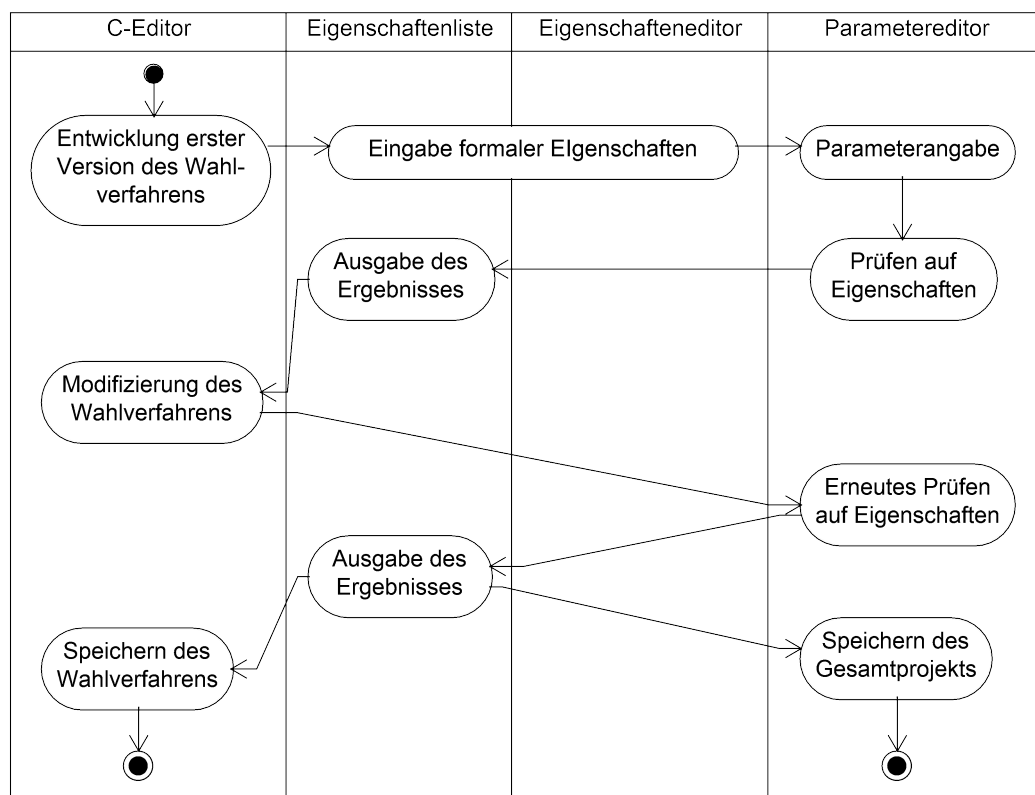


Abbildung 3.1: Aktivitätsdiagramm zu Szenario 1

## Szenario 2

Eine Prüfstelle bekommt ein Wahlverfahren übergeben welches sie auf bestimmte formale Eigenschaften testen soll.

Das Tool wird installiert und das Wahlverfahren im C-Editor geladen und es werden formale Eigenschaften erstellt.

Um das Wahlverfahren auf die Eigenschaften zu prüfen werden Parameter zu Kandidaten-, Stimmen- und Wähleranzahl, sowie einem Timeout als obere Zeitgrenze für das Prüfen angegeben.

Wenn das Tool mit der Überprüfung fertig ist, werden erfüllte und nicht erfüllte Eigenschaften angezeigt. Diese Informationen benutzt die Prüfstelle dann in ihrem Bericht über das Wahlverfahren.

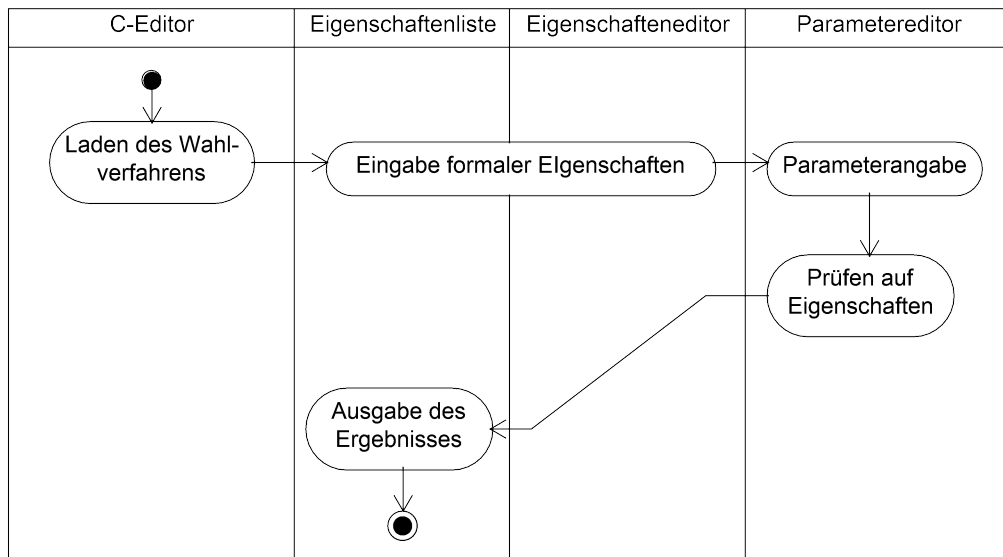


Abbildung 3.2: Aktivitätsdiagramm zu Szenario 2

## 4 Produktumgebung

### 4.1 Software

- Das Betriebssystem ist entweder Microsoft Windows (7,8 oder 10) oder eine der Linux-Distributionen Arch oder Ubuntu.
- Java SE Runtime Environment 8 ist installiert.
- Ist das Betriebssystem Microsoft Windows 7, 8 oder 10 benötigt man zusätzlich
  - Die Developer Command Prompt für Visual Studio
  - cl.exe, ein tool welches die Microsoft Compiler und Linker kontrolliert. Es wird ebenfalls mit Visual Studio geliefert.

### 4.2 Hardware

- Arbeitsspeicher: 4 GB DDR3 RAM oder besser
- Prozessor: Intel i3 dual core mit 2 \* 1,7 GHz oder äquivalent
- Grafik: Intel HD3000 oder besser
- Maus, Tastatur und Bildschirm

### 4.3 Produkt-Schnittstellen

- ANTLR zur Echtzeit-Überprüfung des Codes
- CBMC zur Überprüfung der formalen Eigenschaften
- GNU C-Compiler zur statischen Analyse des C-Codes

# 5 Funktionale Anforderungen

## 5.1 Allgemein

### 5.1.1 Muss-Kriterien

/FM0010/ Bereitstellen von Editoren zur Beschreibung des Wahlverfahrens sowie zur Beschreibung zu erfüllender formaler Eigenschaften

/FM0020/ Kommunikation und Überprüfung dieser Eigenschaften via **CBMC!**

/FM0030/ Bereitstellen von Kommunikationsschnittstellen mit **CBMC!** sowohl für Eingabe von Parametern als auch zur Ausgabe der Ergebnisse einer Überprüfung

/FM0031/ Darstellung der Ergebnisse einer Überprüfung in für Nicht-Informatiker verständlicher Form:

- Zahlen in Dezimaldarstellung
- Ausgabe der Eingabeparameter des Wahldurchlaufs bzw. der Wahldurchläufe:
  - Ausgabe des Stimmenarrays (Anzahl Stimmen und einzelne Stimmen)
  - Ausgabe der Kandidaten (Anzahl)
  - Ausgabe der Sitze (Anzahl)
- Ausgabe des Ergebnisses des Wahldurchlaufs bzw. der Wahldurchläufe

/FM0040/ Möglichkeit des Speicherns von Code, formalen Anforderungen und Eingabeparametern als ein Projekt

/FM0050/ Möglichkeit des Öffnens und Editierens der Projekte aus /F0040/

## 5.2 C-Editor für Wahlverfahren

### 5.2.1 Muss-Kriterien

/FM1010/ Darstellung aller für das Programmieren in C benötigten Zeichen

/FM1020/ Veränderung des dargestellten Textes durch Eingabe über Tastatur und Maus wie in Notepad

/FM1030/ Speichern von erstelltem Code als Datei auf der Festplatte an vom Benutzer angegebenen Ort

/FM1040/ Laden und Darstellen von Dateien korrekten Formats

/FM1050/ Anzeigen syntaktischer Fehler im Fehlerfenster (siehe ??). Nicht asynchron während Eingabe des Codes, sondern synchron nach Eingabe des entsprechenden Befehls.



### 5.2.2 Soll-Kriterien

/FS1060/ Überprüfen des Formats beim Ladevorgang. Falls falsches Format: Ausgabe einer Fehlermeldung

/FS1070/ Syntax-Highlighting: Darstellung diverser Schlüsselwörter in anderen Farben als den Rest des Codes. Dies beinhaltet, ist jedoch nicht beschränkt auf:

- Typendeklaration (int, float, ...)
- Kontrollflow-Konstrukte (if, else, while...)
- Variablennamen
- Kommentare

/FS1080/ Anzeigen der Zeilennummern am linken Zeilenrand

/FS1090/ Anzeigen von syntaktischen Fehlern im Code, welche durch einen Lexer oder Parser erkannt werden können:

- Verwendung von Schlüsselwörtern als Variablennamen
- Vergessene Semikolons am Ende von Anweisungen
- Nicht geschlossene Klammern und Anführungszeichen
- Andere Konstrukte, welche der Grammatik der C-Sprache widersprechen

Wie /FM1050/ ist dies von /FK1150/ abgegrenzt durch synchronen Aufruf der Funktionalität.

/FS1100/ Reaktion auf typische Tastenkürzel

Tabelle 5.1: Hotkeys und verbundene Operationen

Kürzel	Operation
Strg + c	Kopieren
Strg + x	Auschneiden
Strg + v	Einfügen
Strg + z	Zuletzt ausgeführte Aktion rückgängig machen
Strg + r	Zuletzt rückgängig gemachte Aktion erneut ausführen
Strg + s	Speichern
Strg + o	Öffnen
Strg + Leer	Anzeigen der Code-Completion-Vorschläge

/FS1110/ Bereitstellen von Wahl-Templates

- Jeder Wähler wählt genau einen Kandidaten.
- Jeder Wähler ordnet Kandidaten nach Präferenz in absteigender Reihenfolge.

- Jeder Wähler ordnet Kandidaten eine Nummer zwischen MAX (maximale Zustimmung) und MIN (maximale Abneigung) zu. MAX und MIN sind dabei vom Benutzer konfigurierbar.
- Jeder Wähler gibt zu jedem Kandidaten entweder ein Ja oder Nein ab. Zustimmungswahl.

### 5.2.3 Kann-Kriterien

/FK1120/ Automatisches Einrücken des Codes in Schleifen und if-Statements

/FK1130/ Code-Completion

- Automatisches Schließen von Klammern und Anführungszeichen
- Primitiv: Vorschlagen bereits im Code vorgekommener Wörter
- Intelligent: Durch Analysieren eines ASTs nur Vorschlagen der Wörter welche im Kontext Sinn ergeben

/FK1140/ Durch den User konfigurierbares Verhalten:

- Festlegen der Farben, welche beim Syntax-Highlighting verwendet werden
- Festlegen des verwendeten Fonts
- An- und Ausschalten der angezeigten Zeilennummern
- Festlegen, wie vielen Leerzeichen ein Tab entspricht

/FK1150/ Anzeige syntaktischer Fehler im Code Editor asynchron während Eingabe des Codes

## 5.3 Eigenschafteneditor

### 5.3.1 Muss-Kriterien

/FM2010/ Darstellung aller für das Programmieren in C benötigten Zeichen

/FM2020/ Veränderung des dargestellten Textes durch Eingabe über Tastatur und Maus wie in Notepad

/FM2030/ Beschreibung von formalen Eigenschaften als Vor- und Nachbedingung

/FM2040/ Beschreibung der Vor- und Nachbedingungen als Auflistung boolscher Ausdrücke (siehe ??)

/FM2050/ Bereitstellung von Makros zur Beschreibung der Eigenschaften (siehe ??)

Tabelle 5.2: Makros zur Beschreibung formaler Eigenschaften

Makro	Effekt
FOR_ALL_VOTERS(i)	In dem darauf folgenden boolschen Ausdruck kann i als symbolische Variable von Typ Wähler verwendet werden. Der Gesamtausdruck ist wahr, falls der folgende Ausdruck für alle Wähler gilt.
FOR_ALL_CANDIDATES(i)	In dem darauf folgenden boolschen Ausdruck kann i als symbolische Variable von Typ Kandidat verwendet werden. Der Gesamtausdruck ist wahr, falls der folgende Ausdruck für alle Kandidaten gilt.
FOR_ALL_SEATS(i)	In dem darauf folgenden boolschen Ausdruck kann i als symbolische Variable von Typ Sitz verwendet werden. Der Gesamtausdruck ist wahr, falls der folgende Ausdruck für alle Sitze gilt.
EXISTS_ONE_VOTER(i)	In der darauf folgenden Eigenschaft kann i als symbolische Variable von Typ Wähler verwendet werden. Der Gesamtausdruck ist wahr, falls der folgende Ausdruck für mindestens einen Wähler gilt.
EXISTS_ONE_CANDIDATE(i)	In der darauf folgenden Eigenschaft kann i als symbolische Variable von Typ Kandidat verwendet werden. Der Gesamtausdruck ist wahr, falls der folgende Ausdruck für mindestens einen Kandidat gilt.
EXISTS_ONE_SEAT(i)	In der darauf folgenden Eigenschaft kann i als symbolische Variable von Typ Sitz verwendet werden. Der Gesamtausdruck ist wahr, falls der folgende Ausdruck für mindestens einen Sitz gilt.
VOTE_SUM_FOR_CANDIDATE(c)	Gibt die Anzahl Stimmen für Kandidaten c zurück

/FM2060/ Bereitstellen symbolischer Variablen für Wähler, Kandidaten und Sitzes  
 /FM2070/ Bereitstellen von Operatoren für Implikation und Äquivalenz  
 /FM2071/ Bereitstellen von Operatoren für logisches UND und ODER  
 /FM2072/ Bereitstellen von Vergleichen: Kleiner, kleiner gleich, größer, größer gleich, Gleichheit und Ungleichheit zwischen Typen, die diese Vergleiche auch in C zulassen  
 /FM2073/ Möglichkeit zur Abfrage der Stimme eines Wählers v in Wahldurchlauf Nummer x durch VOTESx(v)  
 /FM2080/ Möglichkeit zur Abfrage des Ergebnisses von Wahldurchlauf Nummer x durch ELECTx  
 /FM2090/ Beliebige tiefe, lediglich von Hardware begrenzte Schachtelung dieser Konstrukte  
 /FM2100/ Möglichkeit zum Speichern von Eigenschaften  
 /FM2110/ Möglichkeit zum Laden und Bearbeiten von Eigenschaften in korrektem Format

/FM2120/ Sanity-Checks: Nur syntaktisch korrekte boolsche Ausdrücke werden zur Überprüfung zugelassen.

### 5.3.2 Soll-Kriterien

/FS2120/ Syntax-Highlighting

/FS2130/ Anzeigen von syntaktischen Fehlern im Code

/FS2131/ Reaktion auf typische Tastenkürzel wie in /FS1100/ (siehe ??)

### 5.3.3 Kann-Kriterien

/FK2140/ Code-Completion

- Auto-Vervollständigung der Makros
- Analyse des Codes und Anzeigen relevanter, bereits definierter Eigenschaften und symbolischer Variablen

## 5.4 Eigenschaftenliste

### 5.4.1 Muss-Kriterien

/FM3010/ Darstellung der zu analysierenden Eigenschaften in Listenform

/FM3020/ Möglichkeit zum Erstellen neuer Eigenschaften

/FM3030/ Möglichkeit zum Hinzufügen bereits vorhandener Eigenschaften

/FM3040/ Möglichkeit zum Entfernen von Eigenschaften

/FM3050/ Möglichkeit das Überprüfen einzelner Eigenschaften an- und auszuschalten

/FM3060/ Möglichkeit Listen zu speichern

/FM3070/ Möglichkeit Listen in korrektem Format zu laden und zu editieren

## 5.5 Editor für Eingabeparameter

### 5.5.1 Muss-Kriterien

/MF4010/ Möglichkeit zur Angabe der zu analysierenden Anzahl von Wählern, Kandidaten und Sitzen

/MF4011/ Möglichkeit die Parameter in /F4010/ als Minimum und Maximum anzugeben, welche nacheinander abgearbeitet werden

/FM4020/ Sanity-Checks der eingegebenen Parameter: Alle größer 0, Minimum kleiner gleich Maximum

/FM4030/ Möglichkeit zum Eingeben einer Zeitspanne nach welcher die Berechnung abgebrochen wird

/FM4040/ Möglichkeit direkter Eingabe von Argumenten, welche dem Aufruf von CBMC

mitgegeben werden

/FM4050/ Möglichkeiten das momentan geöffnete Wahlverfahren, die momentan eingestellten Parameter und die momentan geöffnete Eigenschaftenliste zusammen als Projekt zu speichern

/FM4060/ Möglichkeit die Projekte aus /F4050/ zu öffnen und zu bearbeiten

/FM4070/ Möglichkeit Überprüfungen zu starten

/FM4070/ Möglichkeit Überprüfungen zu beenden

## 6 Produktdaten

### 6.1 Code-Editor Wahlverfahren

/D10/ Das Wahlverfahren wird vom Benutzer als C-Code definiert und in einer Datei mit vom Benutzer gegebenen Namen und der Endung .c gespeichert

/D11/ Die verfügbaren Kategorien an Wahlverfahren werden in einer Datei namens electionTemplates gespeichert

### 6.2 Editor von formalen Eigenschaften

/D20/ Eine vom Benutzer definierte formale Eigenschaft wird in einer Datei mit vom Benutzer gegebenen Namen und der Endung .eig gespeichert

### 6.3 Parameter

/D30/ Angegebene Parameter für Wahlen werden in einer Textdatei gespeichert

### 6.4 Projektdaten

/D40/ Ein Projekt besteht aus Parametern, Eigenschaften und Wahlverfahren. Diese werden gemeinsam in einer Textdatei gespeichert

### 6.5 Eigenschaftenliste

/D50/ Die in der Eigenschaftenliste erstellten Listen werden in Textdateien gespeichert

## 7 Nichtfunktionale Anforderungen

- /NF10/ Nicht mehr als 0,5 Sekunden Verzögerung bei Erfragen der Code-Completion
- /NF20/ Maximal 10000 Kandidaten, Wähler oder Sitze pro Überprüfung
- /NF30/ Maximal 10000 Zeilen C-Code
- /NF40/ Maximal 100 Vor- und Nachbedingungen pro formaler Eigenschaft
- /NF50/ Maximal 50 symbolische Variablen pro formaler Eigenschaften
- /NF60/ Maximal 10 Jahre währender Timeout

## 8 Globale Testfälle und Testszenarien

Generell testet ein Testfall nicht alle Funktionen einer aufgelisteten funktionalen Eigenschaft. Somit müssen alle Tests in denen eine funktionale Eigenschaft aufgelistet ist gelingen, damit eine funktionale Eigenschaft als erfüllt gilt.

Innerhalb des kompletten Kapitels 8 ist mit dem Ausdruck **Dokument** je nachdem welches **GUI!**-Fenster betrachtet wird etwas anderes gemeint. Außerdem werden Im Text **GUI!**-spezifische Sonderfälle mit einem Kürzel markiert.

Im Falle des **C-Editors** handelt es sich um den C-Code und das Kürzel **C**.

Im Falle des **Eingeschafteneditors** handelt es sich um eine spezifische Eigenschaft von Wahlverfahren und das Kürzel **E**.

Im Falle der **Eigenschaftenliste** handelt es sich um die Liste der Eigenschaften von Wahlverfahren und das Kürzel **L**.

Im Falle des **Parametereditors** handelt es sich um ein komplettes Projekt. Genauer gesagt also den C-Code, die Parameter und die Eigenschaften für Wahlverfahren inklusive der kompletten Eigenschaftenliste. Der Kürzel für den Parametereditor lautet **P**.

### 8.1 Testfälle für die Datenverwaltung

Die Testfälle innerhalb dieses Abschnittes betreffen alle 4 GUI-Elemente.

/T010/

**Prozess:** Erstellen eines neuen Dokuments

**Abgedeckte Funktionale Anforderungen:**

C: /FS1030/ /FS1100/ /FS1110/

E: /FM2100/

L: /F3020/

P: /F4050/

**Ziel:** Es kann ein neues Dokument angelegt werden

**Kategorie:** muss

**Vorbedingung:** Das Programm wartet auf eine Eingabe. Der Benutzer wählt einen korrekten Speicherort und einen gültigen Dateinamen.

**Nachbedingung (Erfolg):** Das Programm zeigt ein neues, leeres Dokument an, das einen Speicherplatz und einen Namen hat.

C: Der für das gewählte Wahl-Template benötigte Funktionsrumpf wird generiert. **Nach-**

**bedingung (Fehlschlag):** Es wurde kein neues, leeres Dokument angelegt und dargestellt

**Akteur:** Benutzer



**Auslösendes Ereignis:** Der Benutzer will ein neues Dokument anlegen

**Beschreibung:**

1. Der Nutzer verwendet die Tastenkombination Strg + N (hierbei muss das GUI-Element ausgewählt sein, welches getestet werden soll) oder der Benutzer betätigt den Button Neu des jeweiligen Fensters.

2. Das Programm öffnet einen Dateibrowser.

3. Der User gibt den Speicherort und den Name des neuen Dokuments an.

C: Der Benutzer wählt das zu verwendende Wahl-Template aus.

4. Das Programm zeigt das neue leere Dokument an.

C: Das Programm generiert den für das gewählte Wahl-Template benötigten Funktionsrumpf.

**Erweiterung:** Falls es im zuvor geöffneten Dokument noch nicht gespeicherte Änderungen gibt, wird der Benutzer gefragt ob er den Vorgang nach 1. abbrechen will oder ob er fortfahren will. Beim Fortfahren werden jegliche Änderungen verworfen.

**Alternativen:** -keine-

/T020/

**Prozess:** Speichern der Dokumente

**Abgedeckte Funktionale Anforderungen:**

C: /FS1030/ /FS1100/

E: /FM2100/

L: /F3060/

P: /F4050/

/T021/

**Prozess:** Speichern unter

**Ziel:** Das gerade geladene Dokument des Benutzers wird im Dateisystem des Computers gespeichert. Ort und Name bestimmt der Benutzer

**Kategorie:** muss

**Vorbedingung:** Das Programm wartet auf eine Eingabe. Der Name und der gewählte Speicherort sind gültig

**Nachbedingung (Erfolg):** Das Dokument wurde korrekt an den gewählten Speicherort mit dem gewählten Namen gespeichert.

**Nachbedingung (Fehlschlag):** Das Dokument wurde nicht gespeichert

**Auslösendes Ereignis:** Der Benutzer will das Dokument an einem anderen Ort oder unter einem anderen Name speichern

**Beschreibung:**

1. Der Benutzer betätigt die Taste Speichern unter innerhalb des gewünschten Fensters

2. Das Programm öffnet einen Dateibrowser

3. Der Benutzer gibt Name und Speicherort des Dokuments im Dateibrowser ein. 4. Das Programm speichert das Dokument an der genannten Stelle mit dem eingegebenen Namen.

**Erweiterung:** -keine-

**Alternativen:** -keine-

### /T022/

**Prozess:** Speichern

**Ziel:** Das Dokument des Benutzers wird im Dateisystem des Computers mit zuvor gewählten Ort und Name gespeichert

**Kategorie:** muss

**Vorbedingung:** Das Programm wartet auf eine Eingabe.

**Nachbedingung (Erfolg):** Das Dokument wurde korrekt gespeichert. Ort und Name wurden nicht verändert.

**Nachbedingung (Fehlschlag):** Das Dokument wurde nicht gespeichert.

**Akteur:** Benutzer

**Auslösendes Ereignis:** Der Benutzer will das Dokument speichern

**Beschreibung:**

1. Der Nutzer verwendet die Tastenkombination Strg + s (hierbei muss das GUI-Element ausgewählt sein, welches verwendet werden soll) oder der Benutzer betätigt den Button Speichern des jeweiligen Fensters. 2. Das Programm speichert das Dokument an der zuvor genannten Stelle mit dem zuvor eingegebenen Namen

**Erweiterung:** -keine- **Alternativen:** Mit Speichern unter die Datei an anderer Stelle speichern.

### /T030/

**Prozess:** Laden eines Dokuments aus einer Datei

**Abgedeckte Funktionale Anforderungen:**

C: /FS1040/ /FS1060/ /FS1100/

E: /FM2110/

L: /F3070/

P: /F4060/

**Ziel:** Ein zuvor gespeichertes Dokument wird geladen

**Kategorie:** muss

**Vorbedingung:** Das Programm wartet auf eine Eingabe. Es existiert ein zu ladendes Dokument, welches korrekt gespeichert wurde

**Nachbedingung (Erfolg):** Das Dokument wurde korrekt in den Editor geladen.

**Nachbedingung (Fehlschlag):** Das zuvor geöffnete Dokument ist geladen.

**Akteur:** Benutzer

**Auslösendes Ereignis:** Der Benutzer möchte ein Dokument laden

**Beschreibung:**

1. Der Nutzer verwendet die Tastenkombination Strg + O (hierbei muss das GUI-Element ausgewählt sein, welches verwendet werden soll) oder der Benutzer betätigt den Button Öffnen des jeweiligen Fensters.

2. Das Programm öffnet einen Dateibrowser

3. Der Benutzer wählt im Dateibrowser das zu ladende Dokument

4. Das Programm lädt das gewählte Dokument

**Erweiterung:** Allgemein: Falls es im zuvor geöffneten Dokument noch nicht gespeicherte Änderungen gibt, wird der Benutzer gefragt ob er den Vorgang nach 1. abbrechen will

oder ob er fortfahren will. Beim Fortfahren werden jegliche Änderungen verworfen.  
5. Das Programm überprüft beim Ladevorgang das Format des zu ladenden Dokuments. Ist das Format falsch wird eine Fehlermeldung ausgegeben und das Dokument nicht geladen.

**Alternativen:** -keine-

## 8.2 Testfälle für Rückgängig und Wiederherstellen

Die 2 Nachfolgende Testfälle /T100/ und /T110/ betreffen die 3 Fenster C-Editor, Eigenschaftenliste und Eigenschafteneditor.

**/T100/**

**Prozess:** Rückgängig

**Abgedeckte Funktionale Anforderungen:**

C: /FS1100/

Allgemein: /F0010/ /F0050/

**Ziel:** Einzelne Änderungsschritte innerhalb der Bearbeitung rückgängig machen können

**Kategorie:** soll

**Vorbedingung:** Das Programm wartet auf eine Eingabe. Der Benutzer hat zuvor Bearbeitungsschritte ausgeführt.

**Nachbedingung (Erfolg):** Der letzte Bearbeitungsschritt wurde rückgängig gemacht.

**Nachbedingung (Fehlschlag):** Das geöffnete Dokument bleibt unverändert

**Akteur:** Benutzer

**Auslösendes Ereignis:** Der Nutzer will einen oder mehrere Bearbeitungsschritte rückgängig machen.

**Beschreibung:**

1. Der Nutzer verwendet die Tastenkombination Strg + z (hierbei muss das GUI-Fenster ausgewählt sein, welches getestet werden soll) oder der Benutzer betätigt den Button Rückgängig des jeweiligen Fensters.

2. Das Programm lädt den vorherigen Zustand des Dokuments in das gewählte Fenster. Somit wird der letzte Bearbeitungsschritt rückgängig gemacht.

**Erweiterung:** -keine-

**Alternativen:** -keine-

**/T110/**

**Prozess:** Wiederherstellen

**Abgedeckte Funktionale Anforderungen:**

C: /FS1100/

Allgemein: /F0010/ /F0050/

**Ziel:** Einzelne Änderungsschritte innerhalb der Bearbeitung, die zuvor mithilfe des Rückgängig-Prozesses rückgängig gemacht wurden, können wiederhergestellt werden.

**Kategorie:** soll

**Vorbedingung:** Das Programm wartet auf eine Eingabe. Der Benutzer hat zuvor einen

oder mehrere Bearbeitungsschritte rückgängig gemacht

**Nachbedingung (Erfolg):** Der zuletzt rückgängig gemachte Stand des Dokuments, der noch nicht wiederhergestellt wurde, wird wiederhergestellt. **Nachbedingung (Fehlschlag):** Der Zustand des Dokuments bleibt unverändert.

**Akteur:** Benutzer

**Auslösendes Ereignis:** Der Nutzer will einen oder mehrere Bearbeitungsschritte, die er zuvor rückgängig gemacht hat, wiederherstellen

**Beschreibung:**

1. Der Nutzer verwendet die Tastenkombination Strg + r (hierbei muss das GUI-Fensters ausgewählt sein, welches verwendet werden soll) oder der Benutzer betätigt den Button Wiederherstellen des jeweiligen Fensters.

2. Das Programm lädt den zuvor Rückgängig gemachten Zustand des Dokuments in das gewählte Fenster. Somit wird der letzte Bearbeitungsschritt, welcher rückgängig gemacht wurde, wiederhergestellt

**Erweiterung:** -keine-

**Alternativen:** -keine-

## 8.3 Testfälle für die Editoren

Die Nachfolgenden Testfälle betreffen nur die 2 GUI-Elemente C-Editor und Eigenschafteneditor.

/T200/

**Prozess:** Kopieren, Ausschneiden und Einfügen

**Abgedeckte Funktionale Anforderungen:**

Allgemein: /F0010/

C: /FS1100/

**Ziel:** Bearbeiten des Textes mit den aus anderen Editoren bekannten Funktionen Kopieren Ausschneiden und Rückgängig

**Kategorie:** soll

**Vorbedingung:** Das Programm wartet auf eine Eingabe

**Nachbedingung (Erfolg):** Der Text wurde wie gewünscht bearbeitet

**Nachbedingung (Fehlschlag):** Der Text wurde nicht wie gewünscht bearbeitet

**Akteur:** Benutzer

**Auslösendes Ereignis:** Der Benutzer will eine der Funktionen Kopieren, Ausschneiden oder Einfügen verwenden

**Beschreibung:**

1. Der Nutzer verwendet entweder die Buttons in der GUI oder die Tastenkürzel:

Strg + x für Ausschneiden, Strg + c für Kopieren und Strg + v für Einfügen.

2. Das Programm führt die gewählte Aktion im Text aus.

**Erweiterung:** -keine-

**Alternativen:** -keine-

/T210/

**Prozess:** Bearbeiten des Codes

**Abgedeckte Funktionale Anforderungen:**

Allgemein: /F0010/

C: /FM1010/ /FM1020/ /FS1070/ /FS1080/ /FS1090/ /FS1120/ /FS1130/

E: /FM2010/ /FM2020/ /FS2120 /FS2130/ /FS2140/

**Ziel:** Der angezeigte Text in den Editoren kann bearbeitet werden. Die Bearbeitung wird mit der Bereitstellung von unterstützenden Funktionen weiterhin vereinfacht

**Kategorie:** muss bzw. soll oder kann bei den Erweiterungen

**Vorbedingung:** Das Programm wartet auf eine Eingabe

**Nachbedingung (Erfolg):** Der Text kann bearbeitet werden. Die unterstützenden Funktionen funktionieren wie beschrieben

**Nachbedingung (Fehlschlag):** Der Text kann nicht bearbeitet werden

**Akteur:** Benutzer

**Auslösendes Ereignis:** Der Benutzer will seinen Code bearbeiten

**Beschreibung:**

1. Der Nutzer verwendet die Tastatur um neuen Text einzugeben bzw. zu verändern
2. Das Programm stellt die Änderung im Text dar. Der gerade bearbeitete Text wird durch ein blinkenden senkrechten Balken angezeigt. Es ist darauf zu achten, dass alle Zeichen, die benötigt werden um C-Code zu schreiben zur Verfügung stehen.

**Erweiterungen:**

Anzeige von Syntax-Highlighting

**Beschreibung der Erweiterungen:**

1. Der Benutzer verändert seinen Code.
2. Während der Nutzer seinen Code verändert werden diverse Schlüsselwörter im Code farbig markiert. Dies beinhaltet unter anderem Typendeklarationen, Kontrollflow-Konstrukte Variablennamen und Kommentare

Anzeige von syntaktischen Fehlern im Code

**Beschreibung der Erweiterungen:**

1. Der Benutzer verändert seinen Code.
2. Während der Nutzer seinen Code verändert werden syntaktische Fehler wie vergessene Semicolon, nicht geschlossene Klammern, Verwendung von Schlüsselwörtern als Variablennamen und andere Konstrukte, welche nicht der (C) C-Grammatik oder der (E) Grammatik für die Beschreibung der Wahlverfahren entsprechen.

C: Automatisches Einrücken des Codes in Schleifen und if-Statements

**Beschreibung der Erweiterungen:**

1. Der Benutzer verändert seinen Code.
2. Während der Nutzer seinen Code verändert werden die Strukturen von Schleifen und if-Statements erkannt und automatisch eingerückt.

C: Die Zeilennummern werden angezeigt

**Alternativen:** -keine-

## 8.4 Testfälle für den C-Editor

/T310/

**Prozess:** Auswahl des zu verwendenden Wahl-Templates.

**Abgedeckte Funktionale Anforderungen:** /FS1110/

**Ziel:** Der Benutzer kann anhand der Wahl-Templates unterschiedliche Wahlverfahren implementieren.

**Kategorie:** Soll

**Vorbedingung:** Das Programm wartet auf eine Eingabe.

**Nachbedingung (Erfolg):** Der Funktionskörper wird entsprechend des gewählten Wahlverfahrens geändert.

**Nachbedingung (Fehlschlag):** Der Funktionskörper bleibt derselbe.

**Akteur:** Benutzer

**Auslösendes Ereignis:** Der Benutzer will ein anderes Wahl-Template verwenden.

**Beschreibung:**

1. Der Benutzer klickt auf Bearbeiten -> Wahlart ändern und wählt dort ein neues Wahl-Template.
2. Der Funktionskörper wird durch den Funktionskörper des neuen Wahl-Templates ersetzt.

**Erweiterung:** -keine-

**Alternativen:** -keine-

/T320/

**Prozess:** Statische Analyse des Codes.

**Abgedeckte Funktionale Anforderungen:** /FM1050/

**Ziel:** Der Benutzer kann den geschriebenen Code auf syntaktische Fehler untersuchen lassen.

**Kategorie:** Muss

**Vorbedingung:** Der Editor enthält Code des Benutzers.

**Nachbedingung (Erfolg):** Alle durch Lexer oder Parser erkennbaren Fehler werden im Fehlerfenster ausgegeben.

**Nachbedingung (Fehlschlag):** Fehler werden nicht ausgegeben.

**Akteur:** Benutzer

**Auslösendes Ereignis:** Der Benutzer will Fehler in seinem Code finden.

**Beschreibung:**

1. Der Benutzer klickt auf Code -> Statische Analyse.
2. Syntaktische Fehler werden im Fehlerfenster ausgegeben.

**Erweiterung:** -keine-  
**Alternativen:** -keine-

## 8.5 Testfälle für den Eigenschafteneditor

/T410/

**Prozess:** Erstellen einer korrekten Eigenschaft.

**Abgedeckte Funktionale Anforderungen:** /FM2040/ /FM2050/ /FM2060/ /FM2070/  
/FM2071/ /FM2072/ /FM2073/ /FM2080/ /FM2090/ /FM2100/ /FM2120/

**Ziel:** Der Benutzer kann durch korrektes Anwenden der in ?? beschriebenen Syntax Eigenschaften erstellen.

**Kategorie:** Muss

**Vorbedingung:** Das Programm wartet auf eine Eingabe

**Nachbedingung (Erfolg):** Die Eingabe wird korrekt erkannt und als Eigenschaft gespeichert.

**Nachbedingung (Fehlenschlag):** Einzelne Elemente der Syntax werden nicht erkannt oder ein inkorrekt boolescher Ausdruck wird akzeptiert.

**Akteur:** Benutzer

**Auslösendes Ereignis:** Der Benutzer will eine neue Eigenschaft erstellen.

**Beschreibung:**

1. Der Benutzer erstellt eine neue Eigenschaft unter Verwendung beliebiger Elemente der Syntax aus ??.
2. Der boolesche Ausdruck der Eigenschaft wird korrekt erkannt und gespeichert oder als fehlerhaft erkannt, woraufhin eine Fehlermeldung ausgegeben wird.

**Erweiterung:** -keine-

**Alternativen:** -keine-

## 8.6 Testfälle für die Eigenschaftenliste

/T410/

**Prozess:** Ablesen des Ergebnisses der Überprüfung der formalen Eigenschaft.

**Abgedeckte Funktionale Anforderungen:** /FM0010/ /FM0020/ /FM0030/ /FM0031/

**Ziel:** Der Benutzer kann die Ergebnisse der Überprüfung der formalen Eigenschaften ablesen.

**Kategorie:** Muss

**Vorbedingung:** Das Programm wartet auf den Druck des Knopfes, um mit der Überprüfung einer vorher eingegebenen Formalen Eigenschaft für ein ebenso vorher eingegebenes Wahlverfahren zu beginnen.

**Nachbedingung (Erfolg):** Das Programm hat die Überprüfung abgeschlossen und zeigt das Ergebnis dieser korrekt an.

**Nachbedingung (Fehlenschlag):** Es wird kein oder ein falsches Ergebnis angezeigt.

**Akteur:** Benutzer

**Auslösendes Ereignis:** Der Nutzer startet die Überprüfung und wartet auf die Beendigung.

**Beschreibung:**

1. Der Nutzer gibt mindestens eine formale Eigenschaft ein, die er überprüft haben will, und wählt sie aus.
2. Der Nutzer startet die Überprüfung.
3. Das Program überprüft die Eigenschaft.
4. Nach der Überprüfung zeigt das Program das Ergebnis dem Benutzer an.

**Erweiterung:** -keine-

**Alternativen:** -keine-

/T420/

**Prozess:** Einstellen ob eine formale Eigenschaft, die in der Liste geladen ist, in der nächsten Überprüfung verwendet werden soll.

**Abgedeckte Funktionale Anforderungen:** /FM3010/ /FM3050/

**Ziel:** Der Benutzer kann die Überprüfung einzelner formaler Eigenschaften an und ausstellen.

**Kategorie:** Muss

**Vorbedingung:** Es sind mehrere formale Eigenschaften in der Liste geladen. Der Benutzer wählt nun die formalen Eigenschaften aus, die er überprüft haben wird, indem der Benutzer für jede dieser Eigenschaft auf die Checkbox, neben der „Überprüfen“ steht, klickt.

**Nachbedingung (Erfolg):** Nachdem die Überprüfung abgeschlossen sind wurden alle ausgewählten Eigenschaften überprüft

**Nachbedingung (Fehlschlag):** Es wurden Eigenschaften nicht überprüft, die überprüft werden sollten und/oder es wurden Eigenschaften überprüft, die nicht überprüft werden sollen.

**Akteur:** Benutzer

**Auslösendes Ereignis:** Der Nutzer wählt die Eigenschaften, startet die Überprüfung, und wartet auf die Beendigung

**Beschreibung:**

1. Der Nutzer wählt die Eigenschaften über die „Überprüfen“-Checkboxen aus, die er vom Programm überprüft haben möchte.
2. Der Nutzer startet das Program.
3. Das Program startet die Überprüfung der ausgewählten Elemente.
4. Das Programm beendet die Überprüfung und zeigt bei allen ausgewählten Eigenschaften an, ob diese erfolgreich waren, oder fehlschlagen.

**Erweiterung:** -keine-

**Alternativen:** -keine-

/T430/



**Prozess:** Eine neue formale Eigenschaft in die Liste aufnehmen und die Darstellung in Listenform.

**Abgedeckte Funktionale Anforderungen:** /FM3010/ /FM3020/

**Ziel:** Der Benutzer kann neue formale Eigenschaften in die Liste aufnehmen.

**Kategorie:** Muss

**Vorbedingung:** Der Nutzer will eine neue formale Eigenschaft aufnehmen.

**Nachbedingung (Erfolg):** Eine neue formale Eigenschaft wurde hinzugefügt.

**Nachbedingung (Fehlschlag):** Die Eigenschaft wurde nicht hinzugefügt.

**Akteur:** Benutzer

**Auslösendes Ereignis:** Der Nutzer drückt auf das grüne Plus.

**Beschreibung:**

1. Der Nutzer drückt das grüne Plus, und erstellt damit eine neue formale Eigenschaft und gibt ihr, wenn gewünscht, einen Name.
2. Die Eigenschaft taucht nach dem Abschluss der Erstellung in der Liste auf.

**Erweiterung:** -keine-

**Alternativen:** -keine-

## 8.7 Testfälle für den Parametereditor

/T510/

**Prozess:** Eingabe der Anzahl von Wählern, Kandidaten und Sitzen.

**Abgedeckte Funktionale Anforderungen:** /FM4010/ /FM4020/ /FM4070/

**Ziel:** Der Benutzer kann die Anzahl der Wähler, Kandidaten und zu vergebenden Sitze angeben.

**Kategorie:** Muss

**Vorbedingung:** Der Nutzer will die Parameter der Wahl angeben.

**Nachbedingung (Erfolg):** Die Parameter der Wahl wurden angegeben.

**Nachbedingung (Fehlschlag):** Die Eigenschaften konnten nicht angegeben werden.

**Akteur:** Benutzer

**Auslösendes Ereignis:** Der Nutzer wählt die gewünschten Eigenschaften aus.

**Beschreibung:**

1. Der Nutzer gibt die Eigenschaften der Wahl in den dafür vorgesehenen Feldern ein.
2. Die Eigenschaften der Wahl werden bei der nächsten Überprüfung angewendet.

**Erweiterung:** Es werden zusätzlich zu den Eigenschaften Intervalle angegeben.

**Alternativen:** -keine-

/T520/

**Prozess:** Eingabe einer maximalen Zeitspanne.

**Abgedeckte Funktionale Anforderungen:** /FM4020/ /FM4030/

**Ziel:** Der Benutzer kann die maximale Länge einer Überprüfung angeben.

**Kategorie:** Muss

**Vorbedingung:** Der Nutzer will, dass das Program beim überschreiten einer Zeit, wenn es noch nicht fertig ist, die Überprüfung beendet.

**Nachbedingung (Erfolg):** Das Program beendet die Überprüfung von selbst , wenn das Program länger als die eingegebene Zeit braucht.

**Nachbedingung (Fehlschlag):** Das Program stellt die Berechnung nicht aus, wenn die Zeit überschritten wurde, oder es stellt sie zu früh aus.

**Akteur:** Benutzer

**Auslösendes Ereignis:** Der Nutzer wählt die Dauer aus, die das Program maximal laufen darf.

**Beschreibung:**

1. Der Nutzer wählt eine Dauer aus.
2. Der Nutzer startet die Berechnung.
3. Das Program überschreitet die gegebene Zeitdauer und beendet die Überprüfung.
4. Dem Benutzer wird angezeigt, dass die Zeit überschritten wurde.

**Erweiterung:** -keine-

**Alternativen:** -keine-

**/T530/**

**Prozess:** Starten und stoppen der Überprüfung des Wahlverfahren auf die gewählte formale Eigenschaft(en).

**Abgedeckte Funktionale Anforderungen:** /FM4070/

**Ziel:** Der Benutzer kann die Überprüfung einschalten, und auch wieder ausschalten.

**Kategorie:** Muss

**Vorbedingung:** Der Nutzer kann die Überprüfung und an aus schalten.

**Nachbedingung (Erfolg):** Der Nutzer startet die Überprüfung, oder beendet eine schon laufende Überprüfung.

**Nachbedingung (Fehlschlag):** Die Überprüfung lässt sich nicht starten oder nicht beenden.

**Akteur:** Benutzer

**Auslösendes Ereignis:** Der Nutzer startet die Überprüfung.

**Beschreibung:**

1. Der Nutzer drückt auf den Start Knopf und startet damit die Überprüfung.
2. Das Program startet die Überprüfung.
3. Der Nutzer drückt den Beenden Knopf.
4. Dem Benutzer wird angezeigt, dass die Zeit überschritten wurde.

**Erweiterung:** -keine-

**Alternativen:** -keine-

**/T540/**

**Prozess:** Einstellungen von CBMC.

**Abgedeckte Funktionale Anforderungen:** /FM4040/

**Ziel:** Der Benutzer kann die Argumente die CBMC beim Aufruf mitgegeben werden eingeben.

**Kategorie:** Muss

**Vorbedingung:** Der Nutzer will Argumente an CBMC übergeben.

**Nachbedingung (Erfolg):** CBMC wird mit den gegebenen Argumenten aufgerufen.

**Nachbedingung (Fehlschlag):** CBMC wird nicht mit den gegebenen Argumenten gestartet.

**Akteur:** Benutzer

**Auslösendes Ereignis:** Der Nutzer klickt auf „Erweitert“ um CBMC Einstellungen zu verändern .

**Beschreibung:**

1. Der Nutzer drückt „Erweitert“.
2. Er gibt die gewünschten Argumente ein und speichert sie.
3. Beim nächsten Aufruf werden die Argumente an CBMC mit übergeben.

**Erweiterung:** -keine-

**Alternativen:** -keine-

## 8.8 Allgemeine Testfälle

/T610/

**Prozess:** Prüfung formaler Eigenschaften von Wahlverfahren über CBMC.

**Abgedeckte Funktionale Anforderungen:** /FM0020/ /FM0030/ /FM0031/

**Ziel:** Der Benutzer kann Wahlverfahren auf formale Eigenschaften innerhalb gewählter Grenzen testen.

**Kategorie:** Muss

**Vorbedingung:** Das Programm wartet auf eine Eingabe. Das eingegebene Wahlverfahren ist in richtiger Form (d.h. ausführbar und dem gewählten Template entsprechend). Die zu prüfende Eigenschaft ist syntaktisch korrekt. Es wurden gültige Werte für die Parameter eingegeben. Eine oder mehrere Eigenschaften wurden in der Eigenschaftenliste zur Überprüfung in der nächsten Überprüfung ausgewählt.

**Nachbedingung (Erfolg):** In der Eigenschaftenliste wird die Eigenschaft entweder als erfüllt angezeigt, oder es wird ein Gegenbeispiel angezeigt.

**Nachbedingung (Fehlschlag):** Es wird kein Ergebnis angezeigt.

**Akteur:** Benutzer

**Auslösendes Ereignis:** Der Benutzer will ein Wahlverfahren auf eine formale Eigenschaft innerhalb fester Grenzen überprüfen.

**Beschreibung:**

1. Der Benutzer betätigt den Button Teste Eigenschaften in dem Parametereditor.
2. Das Programm testen mithilfe von CBMC das Wahlverfahren auf die formale Eigenschaft(en).
3. In der Eigenschaftenliste sind die gerade getesteten Wahlverfahren entweder grün hin-

terlegt (Die Eigenschaft trifft zu) oder rot hinterlegt und ausklappbar (Die Eigenschaft trifft nicht zu). Klappt der Benutzer nun eine dieser nicht zutreffenden Eigenschaft auf, sieht er ein Gegenbeispiel.

**Erweiterung:** -keine-

**Alternativen:** -keine-

# 9 Systemmodelle

## 9.1 Anwendungsfälle

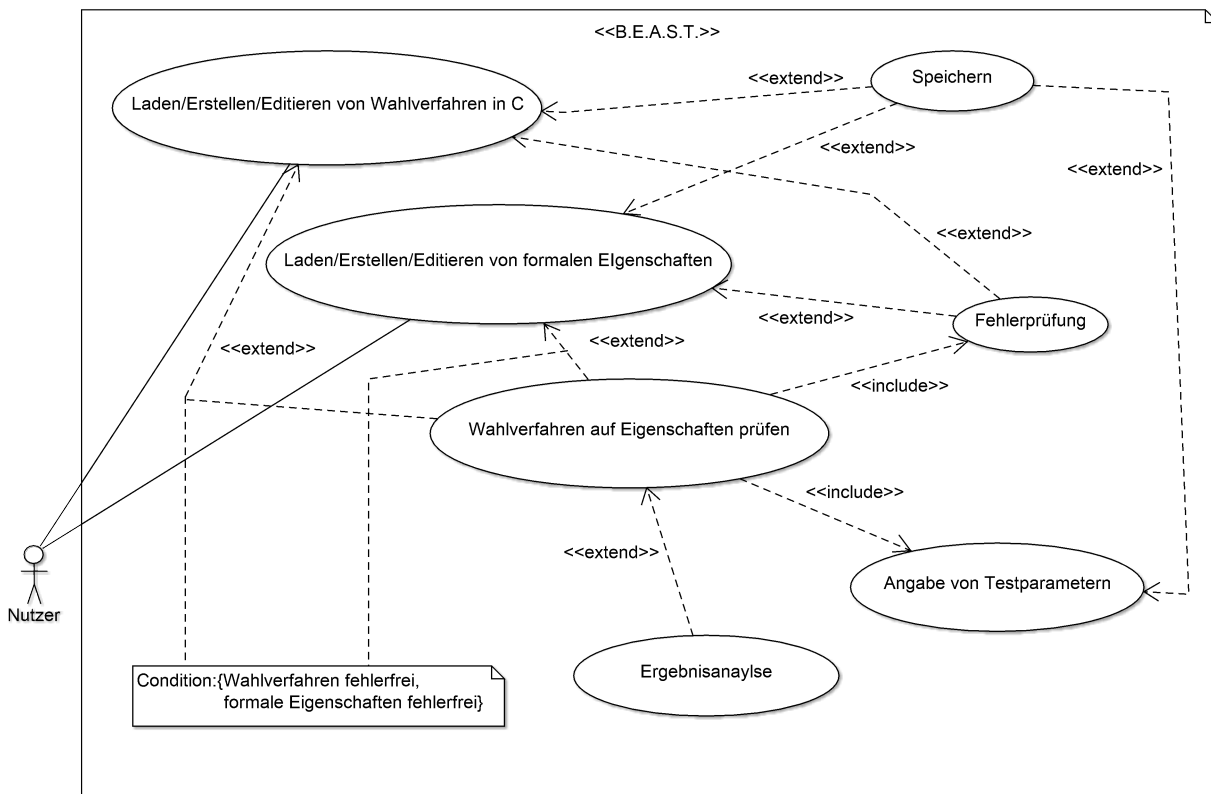


Abbildung 9.1: Anwendungsfalldiagramm von B.E.A.S.T.

Der Nutzer (Entwickler, Wahlforscher, Prüfstelle...) kann Wahlverfahren und formale Eigenschaften in den jeweiligen Editoren des Tools Laden bzw. neu entwickeln und editieren. Beide Editoren verfügen über die Option zum Speichern, sowie dem Anzeigen von syntaktischen Fehlern.

Es kann nach Eingabe von Wahlverfahren und Eigenschaften geprüft werden, ob das Wahlverfahren die formalen Eigenschaften erfüllt. Voraussetzung dafür sind ein syn-

taktisch fehlerfreies Wahlverfahren und mindestens eine syntaktisch fehlerfreie formale Eigenschaft. Vor dem Prüfen kann der Nutzer Testparameter angeben (Anzahl von Kandidaten, Wählern, Sitzen und einem Timeout, welches dem Prüfen eine zeitliche Obergrenze gibt). Die Testparameter können gespeichert werden.

Nach dem Prüfen wird für jede Eigenschaft angezeigt, ob sie vom Wahlverfahren - innerhalb der eingestellten Grenzen - erfüllt wird oder nicht. Das Ergebnis wird nur angezeigt wenn kein Abbruch (Durch den Timeout-Parameter oder einem manuellen Abbruch) eintrifft. Werden formale Eigenschaften als nicht erfüllt erkannt, werden ein oder mehrere Gegenbeispiele angezeigt, welche die Nichterfüllbarkeit beweisen. Anhand dieser Informationen kann der Nutzer dann das Wahlverfahren und die Eigenschaften analysieren, falls nötig modifizieren und erneut testen.

Zudem kann der Nutzer das Wahlverfahren, Eigenschaften und die Parameter gebündelt als ein Projekt speichern.

## 9.2 High-Level-Beschreibung der Architektur

Es folgt eine skizzenhafte Beschreibung der High-Level-Architektur des Softwaresystems.

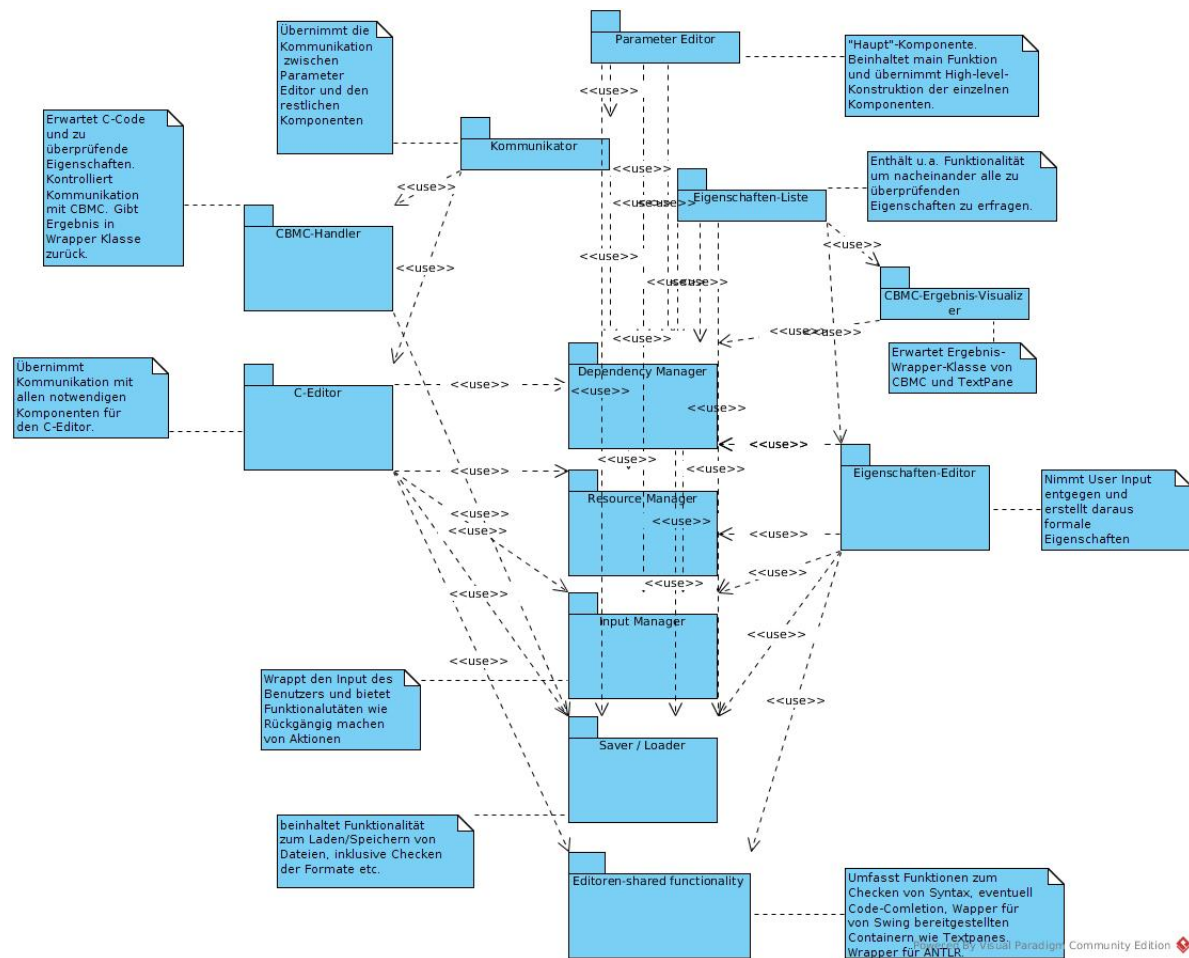


Abbildung 9.2: Skizzenhafte Beschreibung der Pakete und ihrer Abhängigkeiten

?? zeigt einen ersten Entwurf der Pakete und ihrer Abhängigkeiten untereinander. Diese einzelnen Pakete sollen ähnliche Funktionalität bündeln. Zum momentanen Zeitpunkt liegen nur grobe Schätzungen vor, wie aufwendig die Implementierung der verschiedenen Funktionalitäten sein wird. Daher ist es möglich, dass manche Pakete im Verlauf des Entwurfs als Klassen realisiert werden. Andererseits ist es auch möglich, dass weitere Pakete hinzukommen. Abhängigkeiten zu Java-Bibliotheken, wie zum Beispiel Swing, sind nicht dargestellt.

Ziel ist es, möglichst viel gemeinsame Funktionalität zu kapseln und wiederzuverwerten. Beispielsweise haben C- und Eigenschafteneditor viele analoge Aufgaben. Beide müssen Text darstellen und es dem Benutzer erlauben, diesen zu bearbeiten. Beide müs-

sen eingegebenen Code auf syntaktische Fehler untersuchen und die Ergebnisse dieser Untersuchung dem Benutzer signalisieren. Unterschiedlich wird es in den Regeln der einzuhaltenden Syntax sowie der **GUI** geben. Daher wird diese Funktionalität in einem eigenen Paket gekapselt.

Diese Kapselung setzt voraus, dass die verschiedenen Komponenten mit den korrekten Abhängigkeiten konstruiert werden. Funktionalität hierzu befindet sich in dem Dependency-Manager. Dieser stellt eine Schnittstelle dar, über welcher man Instanzen der benötigten Komponenten beziehen kann. In dem Diagramm werden dennoch die konkreten "benutzen"-Beziehungen visualisiert. In dem konkreten Entwurf werden diese jedoch über den Dependency-Manager geregelt.

Das Kommunikator-Paket übernimmt die Kommunikation zwischen dem Parameter-Editor und den Komponenten. So wird es unter anderem der CBMC-Schnittstelle die korrekten Daten zukommen lassen und deren Ergebnisse an die Eigenschaftenliste weiterleiten.

Alle Haupt-Komponenten müssen Dateien speichern und laden können. Ebenso müssen alle Haupt-Komponenten geladene Dateien auf korrektes Format hin überprüfen und das Ergebnis dieser Überprüfung mit dem Benutzer kommunizieren. Das mit Laden/Speichern beschriftete Paket wird Funktionalität zu diesem Zweck bereitstellen.

Die CBMC-Schnittstelle erwartet die Beschreibung des Wahlverfahrens sowie die zu überprüfenden Eigenschaften (jeweils in Wrapper-Klassen). Sie erzeugt den Input für CBMC, ruft CBMC mit diesem Input auf und gibt das Ergebnis (in einer Wrapper-Klasse) zurück.

Diese Wrapper-Klassen erleichtern den Umgang mit den enthaltenen Daten. Die Wrapper-Klasse für CBMC-Ergebnisse wird Funktionen zur Abfrage der Wahleingaben und Wahlergebnisse bereitstellen. Die Wrapper-Klassen für Eigenschaften und C-Code werden es erleichtern, den für CBMC notwendigen Code zu generieren. Zusätzlich erleichtern sie eine Trennung der internen (Model) und externen (View) Darstellung der Daten. So kann die Visualisierung der CBMC-Ergebnisse von einer weiteren Klasse übernommen werden. Diese erwartet lediglich die Ergebnis-Wrapper-Klasse sowie eine Swing-Komponente auf welcher das Ergebnis darzustellen ist. Allgemein sorgt diese zusätzliche Abstraktion für eine flexible Architektur. Sollte das Programm in Zukunft um weitere Programmiersprachen oder einen graphischen Editor zur Beschreibung von Wahlverfahren erweitert werden, so muss lediglich die entsprechende Wrapper-Klasse korrekt erzeugt werden.

Alle Editoren benötigen Funktionalität zum Wiederholen sowie zum Rückgängig machen von Aktionen. Dazu müssen Eingaben des Benutzers entsprechend gekapselt werden. Funktionalität hierzu befindet sich im Input-Paket.

Sämtliche Ressourcen wie Strings und durch den Benutzer konfigurierbare Eigenschaften, wie der verwendete Font, werden über den Resource-Manager bereitgestellt. Dieser kann die vom Benutzer gegebenen Einstellungen auch speichern und laden.



# 10 GUI

Die hier vorgestellte **GUI!** erfüllt alle Muss-, Soll- und Kann-Kriterien. Das endgültige Produkt kann daher davon abweichen. Im Folgenden wird stets explizit darauf hingewiesen, falls es sich bei einem Feature um ein Kann-Kriterium handelt. Die **GUI!** besteht aus 4 verschiedenen Fenstern:

- Ein Editor für C-Code, in welchem die Wahlverfahren editiert werden können
- Eine Liste, in welcher alle für dieses Wahlverfahren zu überprüfenden Eigenschaften angezeigt werden
- Ein Editor in welchem Eigenschaften editiert werden können
- Das Hauptfenster, dessen Schließen ein Beenden des kompletten Tools nach sich zieht. Darin können Parameter für Überprüfungen eingestellt und Überprüfungen gestartet bzw. beendet werden

Jedes dieser Elemente verfügt auch über weitere Eigenschaften, die im Folgenden beschrieben werden.

## 10.1 C-Editor

Der C-Editor verfügt über dieselbe Funktionalität, welche andere Texteditoren wie zum Beispiel Notepad aufweisen. Ziel ist es, das Eingeben von Funktionen, welche ein Wahlverfahren implementiert, zu ermöglichen. Dazu bietet er die Möglichkeit, C-Code zu schreiben und zu bearbeiten. Ein angemessener Funktionskörper, welcher die auswählbare Art der Wahl repräsentiert, wird dabei automatisch generiert (siehe ??). Es wird nicht möglich sein, diese Funktion zu editieren. Während des Eingebens des Codes wird dieser automatisch analysiert, um Schlüsselwörter sowie syntaktische Fehler zu markieren. Der C-Editor teilt sich in vier Untereinheiten auf: Dem Menüstreifen, der Tool-Leiste, dem Textfeld und dem Fehlerfeld. Der Menü-Streifen ist unterteilt in Datei, Bearbeiten, Editor (Kann) und Code. Bilder aller geöffneten Untermenüs befinden sich im Anhang. Sie beinhalten folgende Funktionalität:

Menüpunkt	Bedeutung
Neu	Öffnet ein neues Dokument, wobei die Art der Wahl vom User angegeben wird
Speichern	Speichert das Dokument unter bereits gegebenem Namen
Speichern unter	Speichert das Dokument unter neuem Namen an neuem Ort, beide durch User angegeben
Öffnen	Öffnet neues Dokument des richtigen Formats

Tabelle 10.1: Unterpunkte des Datei-Menüs

Menüpunkt	Bedeutung
Rückgängig	Falls möglich: Macht die letzte ausgeführte Aktion rückgängig
Wiederholen	Wiederholt die zuletzt rückgängig gemachte Aktion
Kopieren	Fügt markierten Text in die Zwischenablage ein
Ausschneiden	Fügt markierten Text in die Zwischenablage ein und entfernt ihn aus dem Textfeld
Einfügen	Fügt Text aus der Zwischenablage an der Stelle des Cursors ein
Wahlart ändern	Ändert den Funktionskörper zu dem der vom User ausgewählten Art

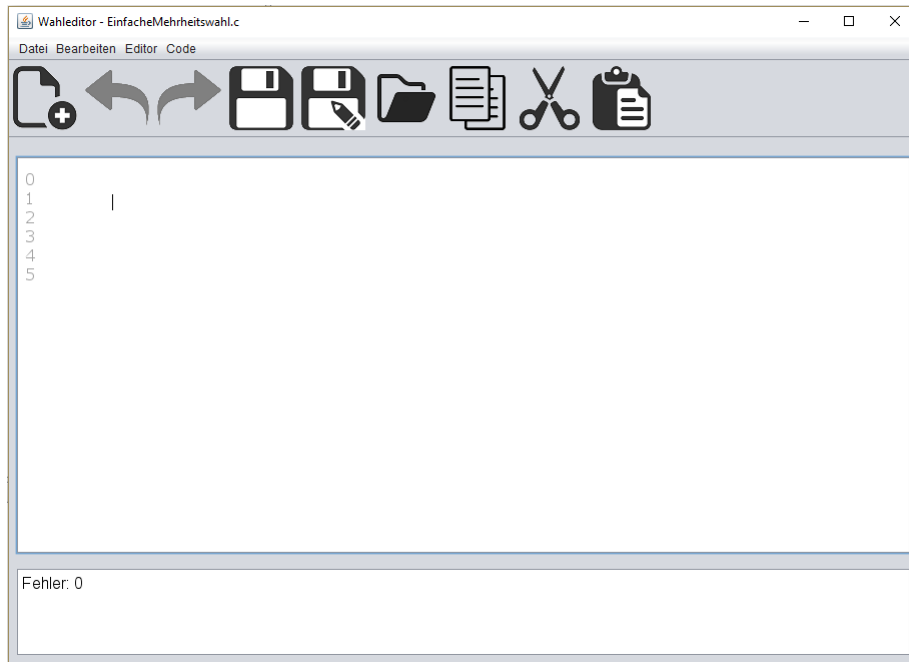
Tabelle 10.2: Unterpunkte des Bearbeiten-Menüs

Menüpunkt	Bedeutung
Einstellungen	Öffnet den Einstellungen-Dialog. Dies ist Teil der Kann-Kriterien. Falls implementiert, wird es Möglichkeiten zur Einstellung des Fonts und Syntax-Highlighting geben.

Tabelle 10.3: Unterpunkte des Editor-Menüs

Menüpunkt	Bedeutung
Statische Analyse	Startet eine statische Analyse des Codes, welche ihn auf von Lexer oder Parser erkennbare Fehler untersucht. Gefundene Fehler werden in dem Fehlerfeld angezeigt. Zusätzliches Kann-Kriterium: Die Zeile, in welcher der Fehler ist, wird zusätzlich durch einen roten Punkt markiert (siehe ??)

Tabelle 10.4: Unterpunkte des Code-Menüs



Über den Tool-Streifen (siehe ??) lassen sich einige dieser Aktionen ohne Öffnen eines Menüs ausführen. Von Links nach Rechts: Neu, Rückgängig, Wiederholen, Speichern, Speichern unter, Öffnen, Kopieren, Ausschneiden, Einfügen. Aktivieren der Aktion “Neu“ öffnet einen Dialog, welcher vom Benutzer die Eingabe folgender Eigenschaften erwartet: Speicherort, Name und Kategorie des Wahlverfahrens.

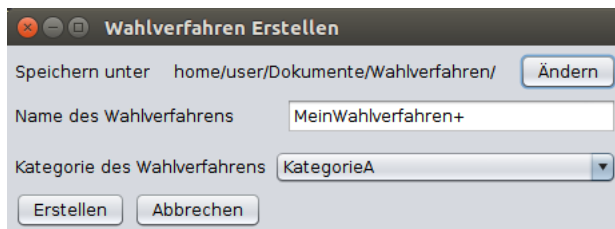


Abbildung 10.2: Der Dialog welcher dem User das Erstellen neuer Wahlverfahren ermöglicht

Bei erfolgreicher Eingabe wird eine neue Datei geöffnet, welche der Nutzer editieren kann. Die Kategorie des Wahlverfahrens wird durch die Funktionssignatur der Wahlfunktion (`voting`) dargestellt, welche nicht vom Nutzer verändert werden kann.

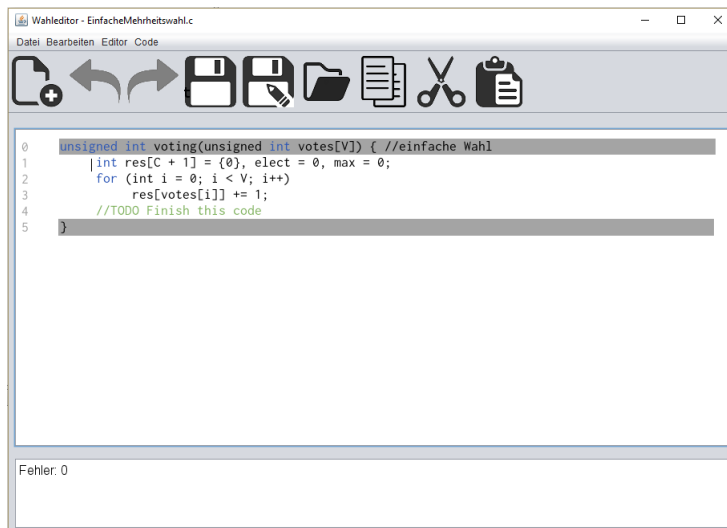


Abbildung 10.3: Der C-Editor mit Code und Anzeige der Wahlart

In ?? sieht man den Editor nach Eingabe diverser Elemente der C-Sprache. Anzeige der Zeilennummern ist Soll-Kriterium. Die Möglichkeit, diese Funktion zu deaktivieren, ist ein Kann-Kriterium. Die grauen Balken zeigen an, dass man den vorgegebenen Funktionskörper nicht editieren kann. Der Kommentar rechts der Funktionsdefinition gibt die Kategorie des Wahlverfahrens an.

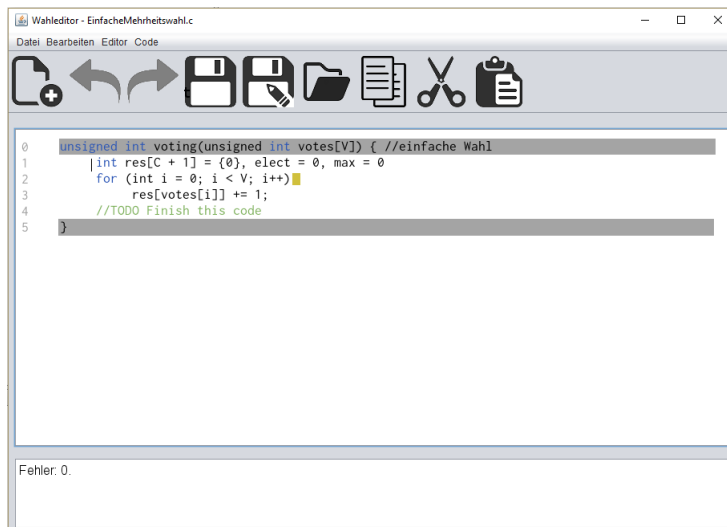


Abbildung 10.4: Fehleranzeige bei syntaktischem Fehler ohne Maus-Hover (Kann-Kriterium)



Abbildung 10.5: Fehleranzeige bei syntaktischem Fehler mit Maus-Hover (Kann-Kriterium).

In ?? sieht man wie der gefundene syntaktische Fehler, während des Editierrens des Codes, angezeigt wird. Dies ist ein Kann-Kriterium. Sobald man die Maus über die markierte Stelle bewegt, wird in einem neuen Fenster nahe der Maus eine Beschreibung des Fehlers angezeigt. Dies ist ebenfalls ein Kann-Kriterium (siehe ??).



Abbildung 10.6: Fehleranzeige nach statischer Analyse.

?? Zeigt die Anzeige der Fehler nach Durchführung einer statischen Code Analyse. Markierung der Zeile ist Kann-Kriterium.

## 10.2 Eigenschaftenliste

Die GUI trennt das Editieren der zu überprüfenden Eigenschaften (in eigens zu diesem Zweck erstellter Syntax, siehe ??) und das Zuordnen dieser Eigenschaften zu Wahlverfahren. Dadurch können diese Eigenschaften einzeln abgespeichert und flexibel wiederverwertet und kombiniert werden. Das Zuordnen der Eigenschaften zu Wahlverfahren geschieht in der Eigenschaftenliste. Darin werden die einzelnen Eigenschaften namentlich aufgelistet (siehe ??).

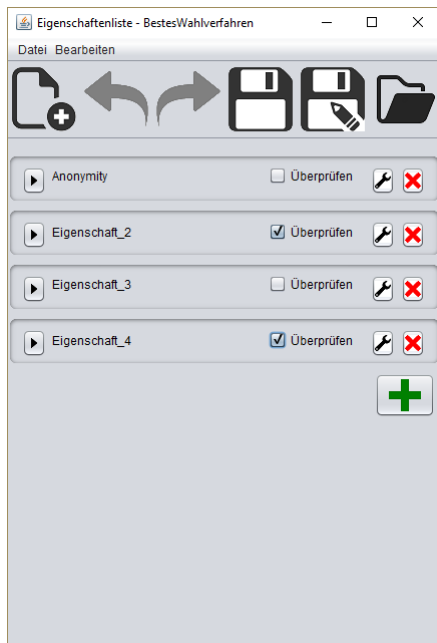


Abbildung 10.7: Eigenschaftenliste vor einer Überprüfung  
Abbildung 10.8: Eigenschaftenliste während einer Überprüfung

Im Folgenden werden die einzelnen **GUI**-Bestandteile und deren Funktionalität erläutert.

Menüpunkt	Bedeutung
Neu	Erstellt eine neue Liste.
Speichern	Speichert die Liste.
Speichern unter	Speichert die Liste unter neuem Namen an neuem Ort.
Öffnen	Öffnet Liste (Ort von User angegeben).

Tabelle 10.5: Unterpunkte des Datei-Menüs.

Menüpunkt	Bedeutung
Rückgängig	Falls möglich: Macht die letzte ausgeführte Aktion rückgängig.
Wiederholen	Wiederholt die zuletzt rückgängig gemachte Aktion.

Tabelle 10.6: Unterpunkte des Bearbeiten-Menüs.

Der Tool-Streifen hat den selben Zweck wie der des C-Editors, jedoch ohne die Funktionen Kopieren, Ausschneiden und Einfügen.



Abbildung 10.9: Liste nach Überprüfung.



Abbildung 10.10: Anzeige des Gegenbeispiels.

Es folgt eine Beschreibung der Icons, welche in den Listenelementen zu sehen sind.

Icon	Bedeutung
Pfeil nach rechts	Falls Gegenbeispiel gefunden: Durch Klicken öffnet sich unter dem Listenelement ein Textfeld in welchen das Gegenbeispiel dargestellt wird.
Checkbox	Nur falls aktiviert, wird das Wahlverfahren auf die Eigenschaft getestet.
Maulschlüssel	Öffnet den Eigenschafteneditor für die Eigenschaft.
Rotes Kreuz	Entfernt die Eigenschaft aus der Liste.
Grünes Plus	Fügt neue, leere Eigenschaft oder bereits gespeicherte Eigenschaft der Liste hinzu.

Tabelle 10.7: Icons der Eigenschaftensliste.

## 10.3 Eigenschafteneditor

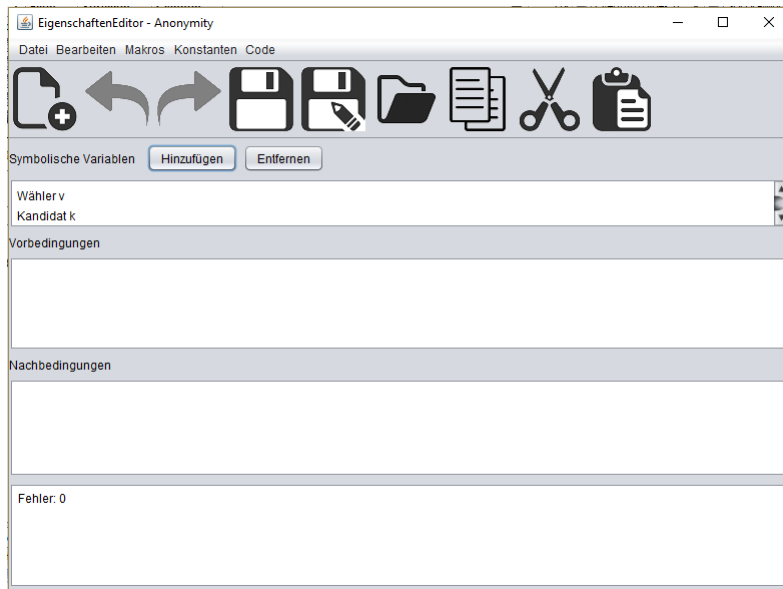


Abbildung 10.11: Eigenschafteneditor ohne Code mit symbolischen Variablen.

Der Eigenschafteneditor hat den Zweck, das Bearbeiten der zu überprüfenden Eigenschaften zu ermöglichen. Eigenschaften werden aufgeteilt in Vor- und Nachbedingungen. Möglich ist die Verwendung sowohl von Quantoren in der Form von Makros als auch symbolischer Variablen.

Die Untermenüs Datei, Bearbeiten und Code sowie der Tool-Streifen sind analog zu denen des C-Editors. Hinzu kommen jedoch Konstanten und Makros. Bereitgestellte Konstanten sind:

- Die Anzahl der Wähler (V)
- Die Anzahl der Kandidaten (C)
- Die Anzahl vorhandener Sitze (S)

Bereitgestellte Makros sind:

- `FOR_ALL_VOTERS()`
- `FOR_ALL_CANDIDATES()`
- `FOR_ALL_SEATS()`
- `EXISTS_ONE_VOTER()`



- EXISTS\_ONE\_CANDIDATES()
- EXISTS\_ONE\_SEAT()
- SUM\_VOTES\_FOR\_CANDIDATE()

Jedes dieser Makros bis auf das Letzte nimmt eine bisher ungenutzte Variable als Argument. Diese kann im darauf Folgenden boolschen Ausdruck verwendet werden (siehe ??). Das letzte Makro nimmt eine bereits definierte symbolische Variable vom Typ Kandidaten und gibt berechnet die Anzahl Stimmen für diesen Kandidaten.

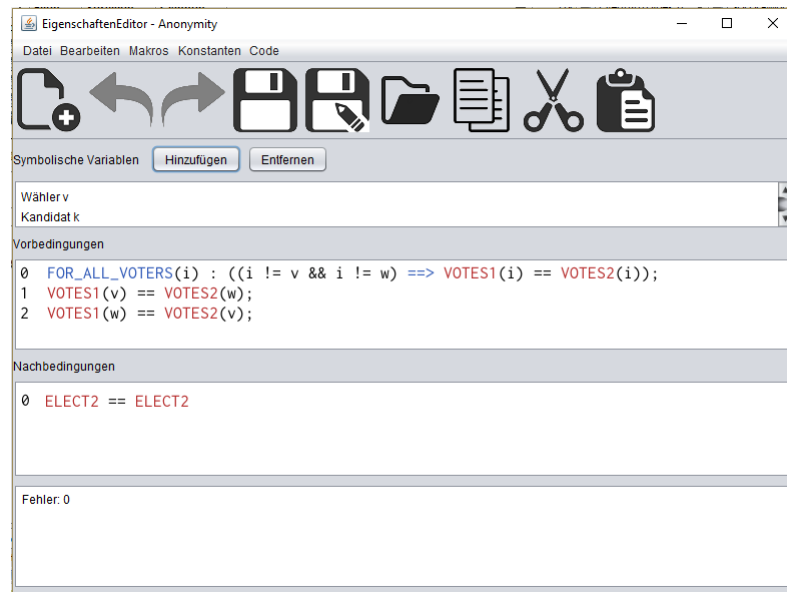


Abbildung 10.12: Eigenschafteneditor mit Beispielhafter Eigenschaft Anonymität und beispielhaft dargestelltem Syntax-Highlighting (Kann-Kriterium).

Wie man sehen kann, sind auch Implikationen ( $\implies$ ) möglich. Auch das logische Und ( $\&\&$ ), Oder ( $\|$ ) und die Äquivalenz ( $\iff$ ) werden zur Verfügung stehen. Die Anzeige erkannter Fehler wird analog zum C-Editor geschehen.

## 10.4 Parametereditor

Der Parametereditor stellt das Hauptfenster der Anwendung dar. Es ermöglicht das Starten und Beenden von Tests. Hier lassen sich auch komplette Projekte Speichern. Code, Eigenschaften und Parameter werden dabei in einer Datei gebündelt.

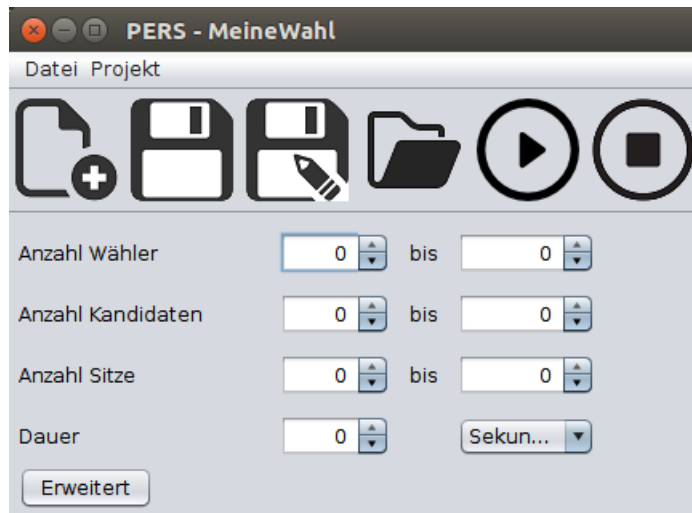


Abbildung 10.13: Der Parametereditor.

Der Menüpunkt Datei ist analog zu dem des C-Editors. Der Menüpunkt Projekt wird im Folgenden erläutert.

Menüpunkt	Bedeutung
Teste Eigenschaften	Startet Tests mit den angegebenen Parametern
Stop Test	Unterbricht momentan laufenden Test

Tabelle 10.8: Menüunterpunkte vom Menüpunkt Projekt”.

Der Pfeil und das Stopp-Zeichen des Tool-Streifens haben ebenfalls die Wirkungen Teste Eigenschaften und Stop Test. Als Mögliche Zeitangaben stehen Sekunden, Minuten, Stunden und Tage zur Verfügung.

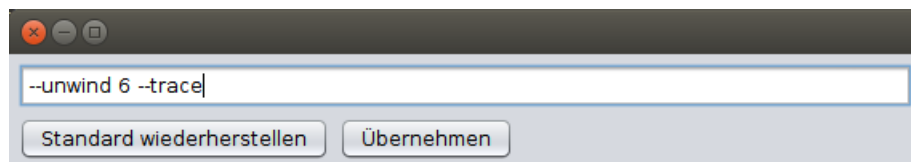


Abbildung 10.14: Das Fenster, welches dem Benutzer erlaubt die an CBMC weitergegebenen Argumente zu spezifizieren.

Klickt der Benutzer auf das Feld ”Erweitert”, so öffnet sich Dialog ???. Darin kann der Benutzer direkt die an **CBMC!** gereichten Argumente zu spezifizieren. Vom Benutzer gegebene Argumente unterliegen keiner Überprüfung durch das Programm können daher zu unerwünschtem Verhalten führen.

# 11 Zeit- und Ressourcenplanung

Für den Rest des Projektes veranschlagen wir 1500 Mannstunden. Die Verteilung der Stunden sieht wie folgt aus.

## 11.1 Zeitplan

Phase	Zeitraum	Stunden
Entwurf	05.12.16 - 16.01.17	480 Mann- stunden
Implementierung	18.01.17 - 13.02.17	560 Mann- stunden
Qualitätssicherung	01.03.17 - 20.03.17	480 Mann- stunden

Tabelle 11.1: Zeiträume und Arbeitsanteile.

## 11.2 Unteraufteilung der Phase

Nun unterteilen wir die einzelnen Phasen in ihre Unterbestandteile, zusammen mit den Veranschlagungen für den veranschlagten Anteil am Gesamtaufwand der Phase.

### 11.2.1 Entwurfsphase

Sub-Phase	Anteil
GUI	35%
C-Code Analyse	30%
Dateiverwaltung	10%
Rest	25%

Tabelle 11.2: Unteraufteilung der Entwurfsphase.

### 11.2.2 Implementierungsphase

Sub-Phase	Anteil
GUI	35%
C-Code Analyse	30%
Dateiverwaltung	10%
Rest	25%

Tabelle 11.3: Unteraufteilung der Implementierungsphase.

### 11.2.3 Qualitätssicherung

Sub-Phase	Anteil
Test der GUI	25%
Testfälle für die Codeanalyse	25%
Testfälle für CBMC	15%
Test des Dateisystems	10%
Rest	25%

Tabelle 11.4: Unteraufteilung der Entwurfsphase.

# **12 Phasenverantwortliche**

## **12.1 Pflichtenheft**

Justin Hecht

## **12.2 Entwurf**

Holger Klein

## **12.3 Implementierung**

Niels Hanselmann, Nikolai Schnell

## **12.4 Qualitätssicherung**

Lukas Stapelbroek

## **12.5 Abschlusspräsentation**

Jonas Wohnig

## 13 Qualitätsanforderungen

Name	sehr relevant	relevant	weniger relevant	nicht relevant
Fehlertoleranz	x			
Sicherheit				x
Bedienbarkeit	x			
Zeitverhalten Editor		x		
Zeitverhalten Analyse				x
Erlernbarkeit		x		
Modifizierbarkeit	x			

Tabelle 13.1: Qualitätsanforderungen.