

# **Qualitätssicherungsdokument**

Hanselmann, Hecht, Klein, Schnell, Stapelbroek, Wohnig

19. März 2017

v0.4

# Inhaltsverzeichnis

<b>1</b>	<b>Einleitung</b>	<b>3</b>
<b>2</b>	<b>Codereviews</b>	<b>4</b>
2.1	Planung . . . . .	4
2.2	Ergebnis . . . . .	4
<b>3</b>	<b>Unit-Tests</b>	<b>5</b>
3.1	Planung . . . . .	5
3.2	Übersicht über gefundene Fehler . . . . .	5
3.3	Testüberdeckung . . . . .	5
3.4	Unit-Tests für AST- und Codegenerierung . . . . .	6
<b>4</b>	<b>Performance und Verbrauch:</b>	<b>7</b>
<b>5</b>	<b>Fehlerbehebungen</b>	<b>11</b>
<b>6</b>	<b>Verbesserungen in der Phase</b>	<b>12</b>
<b>7</b>	<b>Anhang</b>	<b>13</b>
7.1	Testprotokolle . . . . .	13

# **1 Einleitung**

## 2 Codereviews

### 2.1 Planung

Wir haben die Qualitätssicherungsphase mit Codereviews angefangen. Hierfür wurde unsere Gruppe in Gruppen zu je zwei Leuten unterteilt, wobei darauf Wert gelegt wurde, dass diese, die sich gegenseitig ihren Code erklären müssen, möglichst wenig über den Code des anderen wissen.

Wir haben hiermit angefangen, um möglichst schnell die größten Fehler im Code zu finden, sodass wir uns im weiteren Verlauf der Qualitätssicherung auf versteckter liegende Fehler konzentrieren konnten. Ein weiterer wichtiger Aspekt dieser Codereviews war, den Code zu refactorn, um die Lesbarkeit, Wartbarkeit und spätere Testbarkeit zu erhöhen.

### 2.2 Ergebnis

Das Ergebnis der Codereviews ist nicht ganz eindeutig. Manchen Teammitgliedern haben sie geholfen Fehler zu finden, die beim späteren Testen wahrscheinlich nicht entdeckt worden wären, und den Code an sich etwas robuster zu machen. Andere haben eigentlich nur ein paar Style-Fehler gefunden, und angegeben, dass ihnen die Codereviews eigentlich nicht geholfen hätten. Dies kann aber auch daran gelegen haben, dass die Leute zu schnell über den Code gegangen sind, und die andere Person nicht genügend tiefgründige Fragen gestellt hat.

## 3 Unit-Tests

### 3.1 Planung

Neben den Codereviews haben wir anfangs parallel (zum Beispiel weil ein Gruppenmitglied einer Zweiergruppe keine Zeit hat und sein Partner etwas zu tun braucht) und später auch verstärkt darauf hingearbeitet, Testfälle für den Code zu schreiben.

Zum einen werden wir alle Testfälle, welche im Pflichtenheft genannt wurden, implementieren. Sollte der Testfall GUI Bezug haben, oder sich überhaupt nicht mit JUnit realisieren lassen, wird er dann von Hand ausgeführt. Dabei ist es jedoch wichtig alle Schritte genau zu dokumentieren, damit der Test, im Falle einer Änderung, auch später noch reproduzierbar ist.

### 3.2 Übersicht über gefundene Fehler

Dank der Unit-Tests und dem Testen von Hand konnten in dieser Phase viele Fehler gefunden werden, sodass wir hier eine Übersicht über einige Fehler geben können:

- Es gab einen Fehler in der Codegenerierung, sodass zum Beispiel Voting-Arrays, die gleich sein sollten, unterschiedlich waren.
- In manchen Fälle ließ sich die Analyse nicht starten.
- Ein paar Nullpointer-Exceptions.
- Die Ausgabe von CBMC konnte nicht immer richtig geparsed werden.
- Ein Fehler in der Codegenerierung, wenn man „EXISTSONE“ verwendet.
- Fehler bei der Präferenz Wahl, bei der Wähler Kandidaten dieselbe Position geben konnten.

### 3.3 Testüberdeckung

Zur Bewertung unserer Tests setzten wir als Metrik auf die „Instruktionsüberdeckung“, da sich diese am leichtesten messen lässt, und für so ein komplexes Programm gut anzeigt, welche Bereiche noch weiterer Tests bedürfen.

Weiterhin wird aber auch darauf geachtet, dass in den Methoden der einzelnen Klassen eine möglichst hohe Pfadüberdeckung gegeben ist. Da die Metrik-Werkzeuge, welche wir verwenden, zwar nicht überdeckte Pfade anzeigen, sich daraus aber keine ausdrucksvolle

Metrik ergibt, fließt sie nicht in die Metrik an sich mit ein, auch wenn darauf geachtet wurde.

Momentan erreichen wir eine Testabdeckung von ca 77 % (Stand 15.3.17 am Abend). Für das fertige Dokument kommt hier ein Graph hin, in dem man die Testabdeckung im Laufe der Zeit (mindestens zu jedem Meilenstein) erkennen kann.

### **3.4 Unit-Tests für AST- und Codegenerierung**

Da die theoretische Anzahl möglicher korrekter boolscher Ausdrücke abzählbar unendlich ist, ist es unmöglich jeden möglichen Ausdruck auf korrekte Übersetzung in AST und C-Code zu überprüfen. Daher wird stattdessen die AST- und Codegenerierung jedes Sprachkonstrukts einmal auf Korrektheit überprüft.

Sprachkonstrukte sind im Pflichtenheft in “1.1 Die Syntax zur Angabe der formalen Eigenschaften“ beschrieben. Zusätzlich werden einige gängige komplexere Ausdrücke überprüft (Beispiele in <https://formal.iti.kit.edu/teaching/pse/201617/voting/kickOff.pdf>, Folie 22).

Zur Überprüfung der ASTs wurde Funktionalität zur Darstellung eines ASTs in textueller Form implementiert. Diese Repräsentation wird auf Korrektheit überprüft. Die Codegenerierung wird so getestet, dass ein gegebener boolscher Ausdruck übersetzt wird. Dadurch wird bei der Überprüfung der Codegenerierung erneut die Erstellung der ASTs überprüft.

## 4 Performance und Verbrauch:

Über die Phase hinweg haben wir unser Programm stetig in einem Profiler betrachtet, um schnell reagieren zu können, sollte eine Änderung in dieser Phase die Lauffähigkeit unseres Programmes stärker als erwartet beeinflussen.

Das war jedoch nicht der Fall, sodass der Ressourcenverbrauch vor und nach der Qualitätssicherungsphase relativ konstant geblieben ist.

Wie man in 4.1 und 4.2 erkennen kann, ist der Verlauf des Speicherverbrauches so gut wie identisch mit ca. 30MB, bevor der „garbage collector“ es wieder auf ca. 10 MB herunterbringt. Anscheinend haben viele unserer Objekte nur eine kurze Lebensdauer, woraus sich auch schließen ließe, dass unser Programm im „Leerlauf“ einen insignifikanten Speicherverbrauch hat, der Computersysteme von heute vor keine große Aufgabe stellen sollte.

Vergleicht man nun 4.3 mit 4.4 sieht man, dass sich die Unterschiede der Versionen, während eine Eigenschaft überprüft wird, schon stärker unterscheiden. Während der Arbeitsspeicherverbrauch zwar noch relativ ähnlich zwischen den beiden Versionen ist, sieht man, dass die Auslastung des Prozessors schon deutliche Unterschiede aufweist. Diese Unterschiede sind jedoch vor allem darauf zurückzuführen, dass nicht die exakt gleichen Wahlverfahren verglichen wurden, weil sich im Laufe der Qualitätssicherungsphase etwas am System zum Speichern der Wahlverfahren geändert hatte.

Der Grund für den höheren Ressourcenverbrauch bei der Überprüfung ist, dass in dieser Phase zum einen der Code, welcher an CBMC gesendet werden muss, für jede Eigenschaft einzeln erzeugt wird. Auch müssen mehrere Threads konstant die Ausgabe von CBMC auffangen. Ist die Überprüfung jedoch abgeschlossen normalisiert sich der Ressourcenverbrauch wieder relativ schnell.

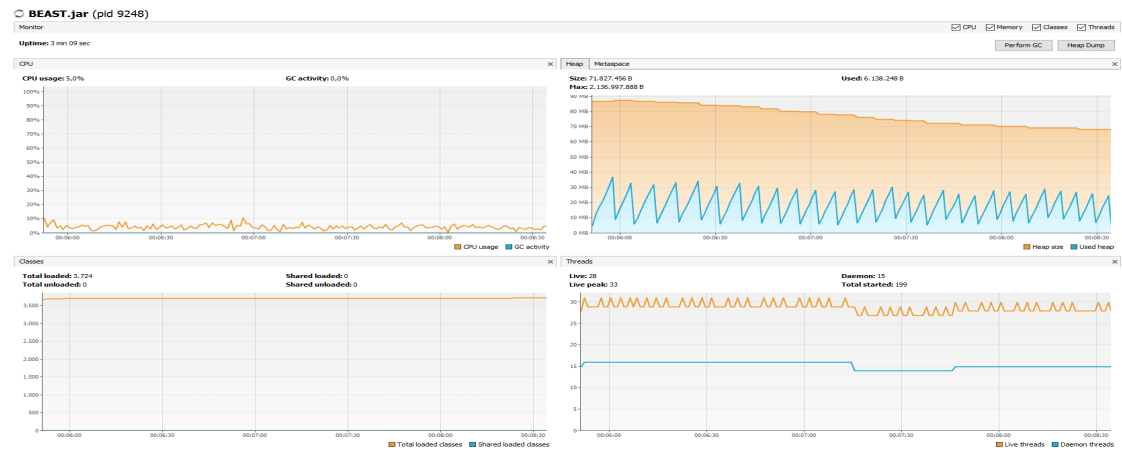


Abbildung 4.1: Dies ist der Ressourcenverbrauch des Programmes, während es auf eine Eingabe vom Nutzer wartet und momentan keine Verifikation durchführt

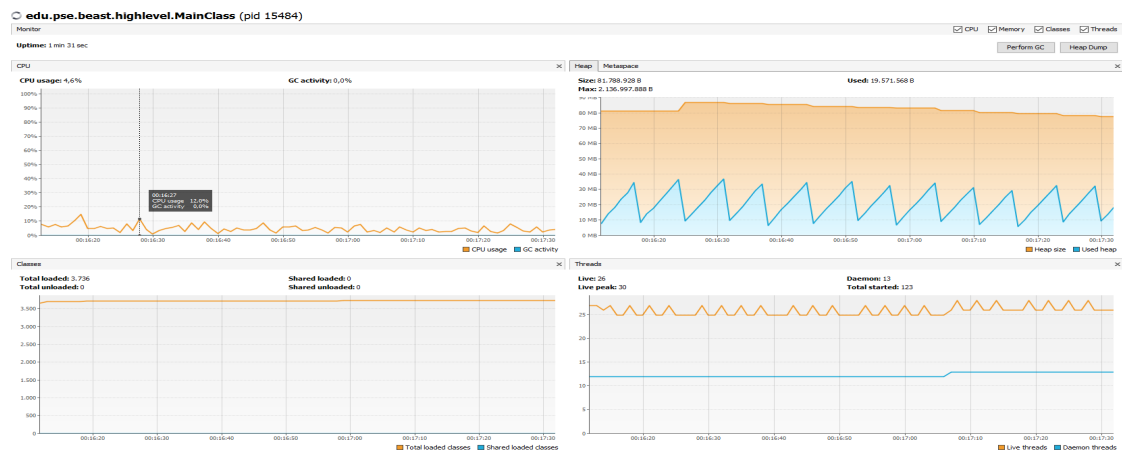


Abbildung 4.2: Der Ressourcenverbrauch der momentanen Version des Programmes, während keine Überprüfung durchgeführt wird



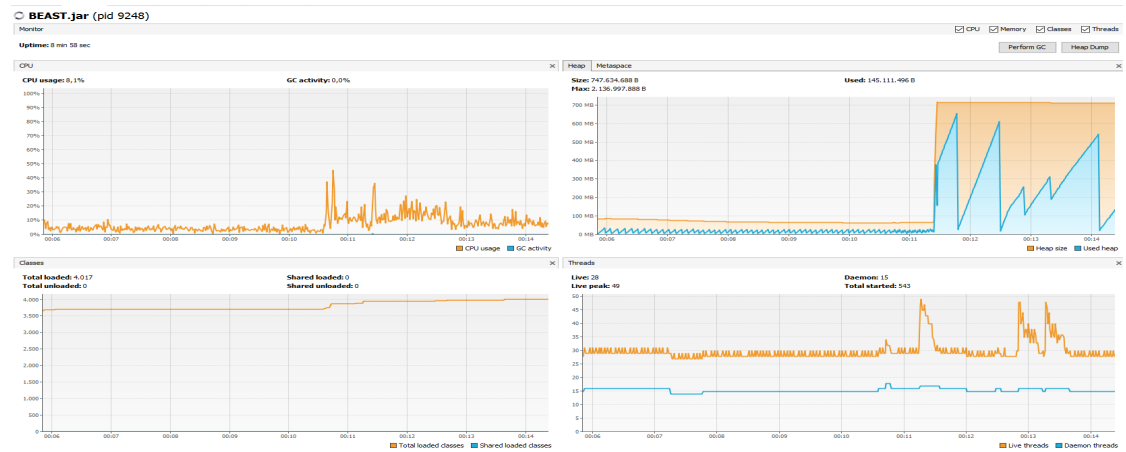


Abbildung 4.3: Der Ressourcenverbrauch der originalen Version von BEAST, während Eigenschaften überprüft werden

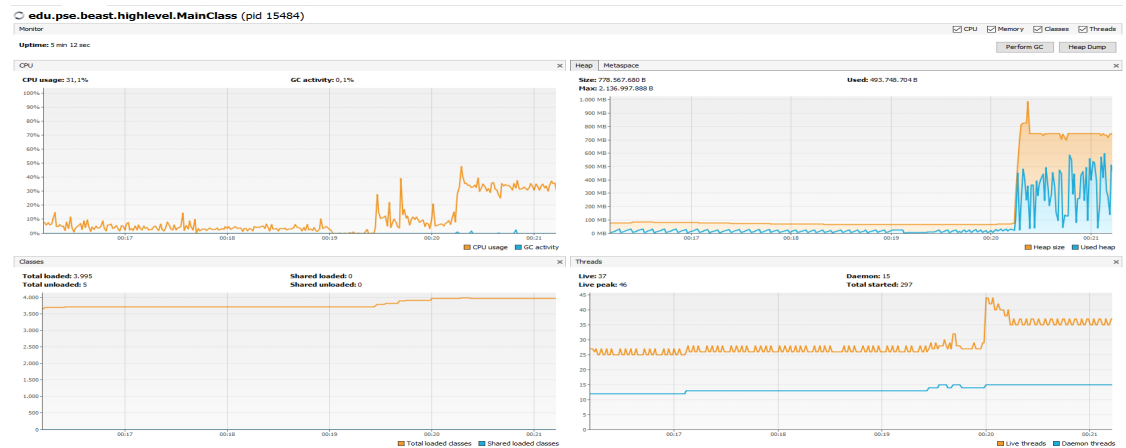


Abbildung 4.4: Der Ressourcenverbrauch der momentanen Version des Programmes, während Eigenschaften überprüft werden

Betrachtet man die Verteilung der Prozessorzeit der neusten BEAST-Version während einer Analyse (siehe 4.5) fällt auf, dass die Methoden, welche die meiste Zeit in Anspruch nehmen, die sind, die dafür sorgen, dass das Programm so angenehm wie möglich läuft. Würde man zum Beispiel die konstante Überprüfung auf Fehler weniger häufig ausführen, müsste der Nutzer länger auf eine Rückmeldung warten, was er noch ändern müsste.

Ähnlich verhält es sich zu den „ThreadedBufferedReader“-Instanzen, die auch noch einen großen Anteil an der Prozessorzeit haben. Dies liegt daran, dass sie die gesamte Kommunikation zu außerhalb laufenden Prozessen übernehmen, und deshalb die gesamte Zeit ohne Unterbrechung laufen müssen, solange der Prozess, den sie überwachen, auch noch läuft.

Hot Spots - Method	Self Time [%] ▾	Self Time	Total Time	Invocations
edu.pse.beast.codearea.ErrorHandling.ErrorFinderThread.run ()		176.096 ms (31.1%)	176.181 ms	4
edu.pse.beast.codearea.AutoComplete.FindWordsConcurrently.run ()		160.039 ms (28.3%)	160.041 ms	4
edu.pse.beast.highlevel.BEASTCommunicator\$1.run ()		79.022 ms (14%)	79.171 ms	1
edu.pse.beast.propertychecker.CheckerFactory.run ()		76.151 ms (13.4%)	76.446 ms	2
edu.pse.beast.toolbox.ThreadedBufferedReader.run ()		34.087 ms (6%)	34.087 ms	15
javax.swing.RepaintManager\$ProcessingRunnable.run ()		29.220 ms (5.2%)	29.232 ms	11.197
edu.pse.beast.propertychecker.Checker.run ()		9.008 ms (1.6%)	9.025 ms	2
edu.pse.beast.highlevel.BEASTCommunicator.checkForErrors (edu.pse.beast.highlevel.CentralObjectProvider)		1.832 ms (0.3%)	1.878 ms	1
edu.pse.beast.highlevel.BEASTCommunicator.createTimeString (double)		144 ms (0%)	144 ms	1.560
edu.pse.beast.propertychecker.CBMCRResult.mergeLinesToOne (java.util.Iterator, String)		92,4 ms (0%)	92,4 ms	1.869
edu.pse.beast.highlevel.BEASTCommunicator.startCheck ()		85,2 ms (0%)	1.964 ms	1
edu.pse.beast.propertychecker.CBMCCodeGenerator.generateAST (String)		68,0 ms (0%)	79,6 ms	2
edu.pse.beast.propertychecker.CBMCRResult.readLongs (String, java.util.List)		33,8 ms (0%)	63,1 ms	1
edu.pse.beast.propertychecker.CBMCRResult.readTwoDimVar (String, java.util.List)		27,6 ms (0%)	55,8 ms	1
sun.awt.GlobalCursorManager\$NativeUpdater.run ()		16,0 ms (0%)	16,0 ms	52
edu.pse.beast.highlevel.BEASTCommunicator.stopReacting (edu.pse.beast.highlevel.CentralObjectProvider)		12,9 ms (0%)	12,9 ms	1
edu.pse.beast.toolbox.FileSaver.writeStringLinesToFile (java.util.List, java.io.File)		12,0 ms (0%)	12,0 ms	2
edu.pse.beast.propertychecker.WindowsProcess.createProcess (java.io.File, int, int, int, String)		11,4 ms (0%)	16,2 ms	2
sun.awt.windows.WComponentPeer\$2.run ()		11,2 ms (0%)	11,2 ms	28

Abbildung 4.5: Der prozentuale Anteil einzelner Methoden an der gesamten genutzten Prozessorzeit

## 5 Fehlerbehebungen

Nummer des Problems — Ursache — Lösung

### Issue 16 -

Beschreibung: Es war möglich, Zeilen zu verändern, welche als nicht editierbar angezeigt und festgelegt wurden. Dies geschah, wenn man unterhalb einer solchen Zeile etwas schrieb. Durch Entfernen des Zeilentrennungszeichens wurde dieses Zeichen dann in die nicht editierbare Zeile angehoben.

Lösung: In die `removeToTheLeft` Methode in `UserInsertToCode` wurde ein zusätzlicher Check eingefügt. Es überprüft nun, ob die Zeile über der, in welcher etwas gelöscht wird, nicht editierbar ist. Falls ja, und das zu löschende Zeichen ist ein Zeichentrennungszeichen, wird nur gelöscht, falls die Zeile leer ist, auf welcher sich der Cursor befindet.

### Issue 27 -

Beschreibung: Obwohl als Voraussetzung angegeben war, dass beide vote-Arrays gleich sein sollten (`VOTES1==VOTES2`) wurden in beiden Wahlvorgängen verschiedene Stimmen abgegeben.

Lösung: Der Bug stammte daher, dass der generierte Code die abgegebenen Stimmen nur bis zur Anzahl der Wähler verglich. Bei Wahlverfahren, bei welchen jeder Wähler eine Liste mit Länge der Anzahl von Kandidaten abgibt, wurden daher nur die ersten Stimmen verglichen. Dies führte zu dem Bug, sobald es mehr Kandidaten als Wähler gab.

### Issue 28 -

Beschreibung: Bei einem Wahlverfahren, welches Preference-Voting als Input verwendet, dauerte es enorm lange eine Eigenschaft zu testen, wenn es mehr Kandidaten als Wähler gab. Bei 6 Wählern und Kandidaten dauerte eine Überprüfung wenige Sekunden. Bei 5 Wählern und 6 Kandidaten war die Überprüfung nach 2 Minuten noch nicht fertig.

Lösung: Die Ursache war, dass bei Preference-voting als zusätzliche Voraussetzung alle von einem Wähler abgegebenen Stimmen verschieden sein müssen. Dies liegt daran, dass diese Platzierungen von Wählern repräsentieren. Der Code, welcher produziert wurde, um diese Eigenschaft sicherzustellen, war fehlerhaft.

### Issue 42 -

Beschreibung: Bei der Codeerzeugung für Vergleiche wurden linke und rechte Seite des Vergleichs vertauscht.

Lösung: An der Stelle, an welcher der String für den Vergleich generiert wird, wurde „lhs“ und „rhs“ vertauscht.

## 6 Verbesserungen in der Phase

Neben Fehlerbehebungen haben wir BEAST in dieser Phase auch in einigen Punkten verbessert:

- Im Eigenschafteneditor gibt es nun einen Knopf, welcher eine Erklärung über die BooleanExpressionLanguage gibt, mit der der Nutzer seine Befehle schreiben kann.
- Der Nutzer kann nun Wähler, Kandidaten und Sitze via ihrer Position in den entsprechenden Arrays angeben. Dazu wurden die Sprachkonstrukte `VOTER_AT_POS`, `CAND_AT_POS` und `SEAT_AT_POS` implementiert.
- Der Nutzer kann nun beliebige mathematische Terme angeben, welche `*`, `/`, `+` und `-` unterstützen. Diese binären mathematischen Operationen können auf sämtliche Ausdrücke angewendet werden, welche einen ganzzahligen Wert liefern.

## **7 Anhang**

### **7.1 Testprotokolle**

Tabelle 7.1: Testfall 8.1 (Testfälle für die Datenverwaltung)

Sub-Testfall	Abgedeckte Funktionalitäten	Beschreibung	Ergebnis	Lukas (Windows 10) Version ???	Justin (Lubuntu 16.1 Version 1.4.19)
/T010/ (C-Editor)	/FS1030/ /FS1100/ /FS1110/	Man gibt ein Wahlverfahren ein. Man wählt in der Toolbar den Button „Neu“ aus. In einen Dialog gibt man das gewünschte Wahlverfahren und die Anzahl der Sitze ein. In ein Textfeld wird der Name eingegeben. Man drückt auf den Button „Erstellen“.	Ein neuer vorgefertigter C-Code erscheint im C-Editor. Ausgegraut sind die Argumente des Wahlverfahrens.	.	✓
/T010/ (Eigenschafteneditor)	/FM2100/ /FS2150/	Man gibt formale Eigenschaften ein. Man wählt in der Toolbar den Button „Neu“ aus.	Die Felder für „Symbolische Variablen“, „Vorbedingungen“ und „Nachbedingungen“ leeren sich. In der Titelleiste erscheint der Name „Eigenschaft 0“.	,	✓
/T010/ (Eigenschaftentaste)	/FM3020/	Man fügt Eigenschaften zur Liste hinzu. Man wählt in der Toolbar den Button „Neu“ aus. Die Nachfrage, ob man speichern will, wird verneint.	Die Liste der Eigenschaften leert sich.	.	✓
/T010/ (Parametereditor)	/FM4050/	Man ändert die Parameter. Man wählt in der Toolbar den Button „Neu“ aus.	Es existiert kein Button für das Neu erstellen.	.	X

Tabelle 7.2: Testfall 8.1 (Testfälle für die Datenverwaltung)

<b>Sub-Testfall</b>	<b>Abgedeckte Funktionalitäten</b>	<b>Beschreibung</b>	<b>Ergebnis</b>	<b>Lukas</b> (Windows 10) Version ???	<b>Justin</b> Lubuntu 16.1 Version 1.4.19)
/T020/ /T030/ (C-Editor)	/FM1030/ /FS1100/ /FS1040/ /FS1060/	Man gibt ein Wahlverfahren ein. Man wählt in der Toolbar den Button „Speichern“ aus. In einen Dialog gibt man den gewünschten Speicherort ein. Man drückt auf den Button „Speichern“. Man wählt in der Toolbar den Button „Öffnen“ aus. In einem Dialog wählt man das gespeicherte Wahlverfahren aus.	Das Wahlverfahren wurde gespeichert. Das Laden des Wahlverfahrens schlug fehl. Das Format wurde nicht erkannt.	.	X
/T020/ /T030/ (Eigenschafteneditor)	/FM2100/ /FS2110/	Man gibt formale Eigenschaften ein. Man wählt in der Toolbar den Button „Speichern“ aus. In einen Dialog gibt man den gewünschten Speicherort ein. Man drückt auf den Button „Speichern“. Man wählt in der Toolbar den Button „Öffnen“ aus. In einem Dialog wählt man die gespeicherten formalen Eigenschaften aus.	Die Eigenschaft wurde gespeichert. Das Laden schlägt fehl. Das Format wurde nicht erkannt.	.	X

Tabelle 7.3: Testfall 8.1 (Testfälle für die Datenverwaltung)

<b>Sub-Testfall</b>	<b>Abgedeckte Funktionalitäten</b>	<b>Beschreibung</b>	<b>Ergebnis</b>	<b>Lukas</b> (Win-dows 10) Version ???	<b>Justin</b> Lubun-tu 16.1 Version 1.4.19)
/T020/ /T030/ (Eigen-schaf-tenliste)	/FM3060/ /FM3070/	Man fügt Eigenschaften zur Liste hinzu. Man wählt in der Toolbar den Button „Speichern“ aus. In einen Dialog gibt man den gewünschten Speicherort ein. Man drückt auf den Button „Speichern“. Man wählt in der Toolbar den Button „Öffnen“ aus. In einem Dialog wählt man die gespeicherte Eigenschaftenliste aus.	Die Liste der Eigenschaften wurde gespeichert. Die Liste wird wieder geladen.	.	✓
/T020/ /T030/ (Para-metere-ditor)	/FM4050/ /FM4060/	Man ändert die Parameter. Man wählt in der Toolbar den Button „Speichern“ aus. In einen Dialog gibt man den gewünschten Speicherort ein. Man drückt auf den Button „Speichern“. Man wählt in der Toolbar den Button „Öffnen“ aus. In einem Dialog wählt man die gespeicherte Eigenschaftenliste aus.	Das Projekt wird gespeichert. Das Projekt kann wieder geladen werden.	.	✓



Tabelle 7.4: Testfall 8.2 (Testfall für Rückgängig machen und Wiederherstellen)

Sub-Testfall	Abgedeckte Funktionalitäten	Beschreibung	Ergebnis	Lukas (Windows 10) Version 1.4.13	Jemand anderes (Ubuntu 14.0 Version 1.4.13)
/T100/	/FS1100/ /FS2150/ /F0010/ /F0050/	Man startet das Programm ganz normal. Nun gibt man in jedes Feld, das die „Rückgängig machen“ Funktionalität unterstützt, einen kleinen Text ein, und drückt dann, während der Fokus auf dem zu testendem Feld liegt „Strg + z“	Der zuletzt eingegebene Buchstabe oder Textblock (im Falle des Einfügens mit „Strg + c“) wird gelöscht	✓	X
/T110/	/FS1100/ /FS2150/ /F0010/ /F0050	Man startet das Programm ganz normal. Nun gibt man in jedes Feld, das die „Rückgängig machen“ Funktionalität unterstützt, einen kleinen Text ein, und drückt dann, während der Fokus auf dem zu testendem Feld liegt „Strg + z“. Nun drückt man „Strg + r“	Der vorher durch das rückgängig machen verschwundene Buchstabe oder Textblock erscheint wieder	✓	X

Tabelle 7.5: Testfälle 8.4 (Testfälle für den C-Editor)

Sub-Testfall	Abgedeckte Funktionalitäten	Beschreibung	Ergebnis	Holger Windows 7	Holger Ubuntu (16.04 LTS))
/T310/	/FS1110/	Man startet das Programm und öffnet den C-Editor. Dort wählt man eine der Möglichkeiten eine neue Wahlbeschreibung zu entwerfen			
	/FS1100/	Per Shortcut: Strg + n	Der Dialog zum Erstellen einer neuen Wahlverfahrensbeschreibung wird angezeigt	✓	✓
		Per Menü: Datei, dann Neu	Der Dialog zum Erstellen einer neuen Wahlverfahrensbeschreibung wird angezeigt	✓	✓
		Per Toolbar: Erster Button der Toolbar	Der Dialog zum Erstellen einer neuen Wahlverfahrensbeschreibung wird angezeigt	✓	✓

Tabelle 7.6: Testfälle 8.4 (Testfälle für den C-Editor)

Sub-Testfall	Abgedeckte Funktionalitäten	Beschreibung	Ergebnis	Holger Windows 7	Holger Ubuntu (16.04 LTS))
	/FS1110/	Auswahl eines Input- und Resulttypen sowie eines Namens. Klicken des ErstellenButtons	Der Funktionskörper wird entsprechend aktualisiert		
		Single-chice	Input: unsigned int votes[V]	✓	✓
		Preference	Input: unsigned int votes[V][C]	✓	✓
		Approval	Input: unsigned int votes[V][C]	✓	✓
		Weighted Approval	Input: unsigned int votes[V][C]	✓	✓
		Candidate or not determined	Result: unsigned int	✓	✓
		Seats per party	Result: unsigned int *	✓	✓

Tabelle 7.7: Testfälle 8.4 (Testfälle für den C-Editor)

Sub-Testfall	Abgedeckte Funktionalitäten	Beschreibung	Ergebnis	Holger Windows 7	Holger Ubuntu (16.04 LTS))
/T320/	/FM1050/	Man startet das Programm und öffnet den C-Editor. Dort gibt man ein Programm ein welches mehrere Fehler enthält. Danach wählt man im Menü Code → statische Analyse aus. Fehler: Fehlendes return, Zugriff auf nicht deklarierte Variable, Verwendung nicht deklarierter Funktion, Funktionsaufruf mit falschen Parametern, fehlendes Semikolon, Fehlende schließende geschweifte Klammer nach for-Schleife	Es werden alle Fehler im Code angezeigt	✓	✓

Tabelle 7.8: Testfälle 8.4 (Testfälle für den C-Editor)

Sub-Testfall	Abgedeckte Funktionalitäten	Beschreibung	Ergebnis	Holger Windows 7	Holger Ubuntu (16.04 LTS))
	/FM1030/, /FM1040/	Man startet das Programm und öffnet den C-Editor. Dort speichert man die geöffnete Wahlverfahrensbeschreibung an einem beliebigen Ort. Danach klickt man auf Öffnen, navigiert an den Ort an dem die Datei gerade gespeichert wurde, und öffnet sie	Die gespeicherte Datei wird angezeigt	✓	✓

Tabelle 7.9: Testfälle 8.4 (Testfälle für den C-Editor)

Sub-Testfall	Abgedeckte Funktionalitäten	Beschreibung	Ergebnis	Holger Windows 7	Holger Ubuntu (16.04 LTS))
	/FK1130/	Man startet das Programm und öffnet den C-Editor. Dort geht man in den Körper der voting-Funktion und beginnt return zu tippen. Nach den ersten zwei Buchstaben betätigt man den Shortcut Strg - Leer. In dem erschienenen Menü wählt man return aus und drückt Enter.	Das Wort return wird in den Funktionskörper geschrieben	✓	✓
	/FK1130/	Man gibt in den Funktionskörper der voting-Funktion den text int asdasdasd ein und wartet 10 Sekunden. Danach geht man auf eine neue Zeile und tippt a. Dann betätigt man den Shortcut Strg-Leer	In dem erschienenen Menü wird nun asdasdasd als Option angezeigt	✓	✓

Tabelle 7.10: Testfälle 8.4 (Testfälle für den C-Editor)

Sub-Testfall	Abgedeckte Funktionalitäten	Beschreibung	Ergebnis	Holger Windows 7	Holger Ubuntu (16.04 LTS))
	/FK1140/	Man startet das Programm und öffnet den C-Editor. Dort geht man auf den Menüpunkt Editor -> Eigenschaften. In dem erschienenen Dialog wählt man einen anderen Font und Schriftgröße aus	Der verwendete Font und Schriftgröße werden zu der gewählten aktualisiert. Diese Änderung bleibt auch nach Neustart des Programmes	✓	✓

Tabelle 7.11: Testfall 8.5 (Testfall für das Erstellen einer Eigenschaft im Eigenschafteneditor)

Sub-Testfall	Abgedeckte Funktionalitäten	Beschreibung	Ergebnis	Lukas (Windows 10) Version 1.4.13	Nikolai Arch Linux (4.10.3-1-ARCH))
/T410/	/FM2040/ /FM2050/ /FM2070/ /FM2071/ /FM2072/ /FM2073/ /FM2080/ /FM2100/ /FM2120/	Man startet das Programm ganz normal. Nun gibt man im Eigenschafteneditor in den Vorbedingungen 'VOTES1 == VOTES2;', und in den Nachbedingungen 'ELECT1 != ELECT2;' ein. Durch auswählen von SStatische Fehlersuchettestet man die Eigenschaft auf Korrektheit und kann diese anschließend mit dem entsprechenden Menüpunkt oder Toolbar-Button speichern.	Es wird 'Fehler: 0' im Fehlerfenster angezeigt und die Eigenschaft hat sich ohne Fehlermeldung speichern lassen.	X	✓



Tabelle 7.12: Testfall 8.6 (Testfälle für die Eigenschaftenliste)

Sub-Testfall	Abgedeckte Funktionalitäten	Beschreibung	Ergebnis	Lukas (Windows 10) Version ???	Justin (Lubuntu 16.1 Version 1.4.19)
/T510/	/FM0010/ /FM0020/ /FM0030/ /FM0031/	Man gibt ein einfaches Wahlverfahren ein, das eine gewählte Person zurückgibt. Man erstellt eine erste Eigenschaft, die erfüllt ist, und eine zweite Eigenschaft, die nicht erfüllt ist. Man wählt im Parametereditor den Start der Analyse in der Toolbar aus.	Die erste Eigenschaft erscheint grün. Die zweite Eigenschaft erscheint rot. Beim Klick auf das Augensymbol der zweiten Eigenschaft öffnet sich ein Fenster mit einem Gegenbeispiel.	.	✓
/T520/	/FM3010/ /FM3050/	Man fügt der Eigenschaftenliste eine Eigenschaft hinzu, indem man auf den Button mit dem Pluszeichen und der Beschriftung „Neu“ drückt. Die Checkbox mit der Beschriftung „Analyse“ klickt man an. Man wählt im Parametereditor den Start der Analyse in der Toolbar aus.	Die Eigenschaft erscheint grün. Die Eigenschaft wurde von CBMC überprüft.	.	✓
/T530/	/FM3010/ FM3020/	Man drückt auf den Button mit dem Pluszeichen und der Beschriftung „Neu“.	Eine neue Eigenschaft mit dem Name „Eigenschaft 0“ erscheint in der Liste.	.	✓

Tabelle 7.13: Testfall 8.7 (Testfälle für den Parametereditor)

Sub-Testfall	Abgedeckte Funktionalitäten	Beschreibung	Ergebnis	Jonas (Windows 10 Version 1607) BEAST v1.4.18	??? ???)
/T610/	/FM4010/ /FM4020/ /FM4070/	Man versucht zunächst negative Zahlen oder 0 als Wähler, Kandidaten und Sitze anzugeben. Dann gibt man als Minimum größere Zahlen als das jeweilige Maximum und dann als Maximum kleinere Zahlen als das jeweilige Minimum an. Zuletzt gibt man sinnvolle Zahlen (alle größer als 0 und Minimum kleiner als Maximum an.	Der Parametereditor setzt nach Eingabe der negativen Zahlen das entsprechende Feld auf den letzten validen Wert zurück. Nach Eingabe der größeren Minima und der kleineren Maxima wird der jeweilige andere Wert angepasst. Sinnvolle Zahlen werden angenommen.	✓	.
/T620/	/FM4020/ /FM4030/	Man hat ein Wahlverfahren und Eigenschaften geladen, sowie Parameter angegeben, deren Analyse länger als der zu testende Timeout dauert. Man gibt den Timeout im Parametereditor an. Man startet die Analyse.	Die Überprüfung wird nach Ablauf der angegebenen Dauer abgebrochen.	✓	.

Tabelle 7.14: Testfall 8.7 (Testfälle für den Parametereditor)

Sub-Testfall	Abgedeckte Funktionalitäten	Beschreibung	Ergebnis	Jonas (Windows 10 Version 1607) BEAST v1.4.18	??? ???)
/T630/	/FM4070/ /FM4080/	Man hat ein korrektes Wahlverfahren und korrekte Eigenschaften geladen. Man startet die Analyse im Parametereditor. Man stoppt die Analyse im Parametereditor manuell.	Die Analyse wird abgebrochen.	✓	.
/T640/	/FM4040/	Man hat ein korrektes Wahlverfahren und korrekte Eigenschaften geladen. Man öffnet das “Erweitert“-Fenster des Parametereditors. Man gibt dort zusätzliche Argumente zur Ausführung von CBMC an. Man startet die Analyse.	Die Analyse wird unter Berücksichtigung der angegebenen Argumente ausgeführt.	✓	.

Tabelle 7.15: Testfall 8.1 (Testfälle für die Datenverwaltung)

Testfall	Abgedeckte Funktionalitäten	Beschreibung	Ergebnis	Jonas (Windows 10 Version 1607) BEAST v1.4.18	??? ???)
/T710/	/FM0020/ /FM0030/ /FM0031/	Man lädt ein korrektes Wahlverfahren und korrekte Eigenschaften. Man gibt sinnvolle Parameter an. Man startet die Analyse.	Nach Abschluss der Analyse werden die Eigenschaften rot markiert, wenn sie nicht auf das Wahlverfahren zutreffen und grün, falls sie es tun.	✓	.