Qualitätssicherungsdokument

Hanselmann, Hecht, Klein, Schnell, Stapelbroek, Wohnig

20. März 2017
v0.4

Inhaltsverzeichnis

1 Einleitung **3**

2 Codereviews **4**
2.1 Planung 4
2.2 Ergebnis 4

3 Unit-Tests **5**
3.1 Planung 5
3.2 Übersicht über gefundene Fehler 5
3.3 Testüberdeckung 5
3.4 Unit-Tests für AST- und Codegenerierung 6

4 Performance und Verbrauch: **8**

5 Fehlerbehebungen **12**

6 Verbesserungen in der Phase **14**

7 Anhang **15**
7.1 Testprotokolle 15

1 Einleitung

Dieses Dokument ist dafür gedacht einen Überlick über die Qualitätssicherungsphase unseres Projektes „BEAST“ zu geben. Obwohl unser Programm schon bei der Abgabe funktionierte, fanden wir erst in dieser Phase viele Bereiche, vor allem Randfälle, in denen es noch nicht funktionierte. Wir wissen jedoch auch, dass man durch Teseten nicht die Abwesenheit von Fehler zeigen kann, sondern nur deren Anwesenheit. Wir hoffen trotzdem, dass wir so viele Fälle der Benutzung, sei es automatisch oder von Hand, getestet haben, dass die späteren Nutzer unseres Programmes es ohne Probleme nutzen können.
Das Dokument wird im weiteren eine Übersicht über den Ablauf unserer Qualitätssicherungsphase geben, und die Vorgehensweise der von uns eingesetzten Methoden beschreiben.

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 gröbsten 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. So fiel Beispielsweise auf, dass bei ein paar Methoden ein „synchronized“ gefunden wurde, oder aber, dass ein „Result“-Objekt zu früh auf „finished“ gesetzt wurde, was dazu führen könnte, dass ein noch nicht fertig bearbeitetes Objekt angezeigt werden würde. 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 ausgefürt. 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.

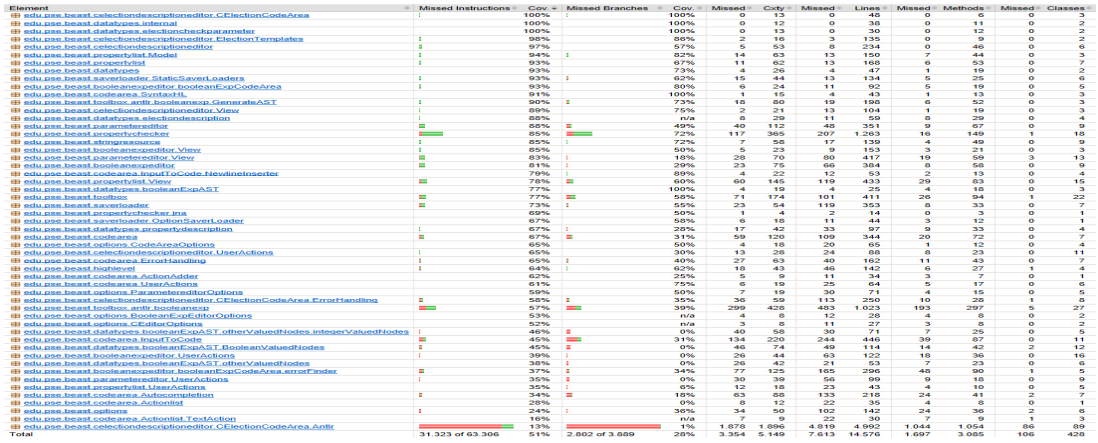
3.3 Testüberdeckung

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.

Wie man im Bild 3.1sehen kann gibt es in unserem Paket große Unterschiede, was die Coverage[1] der verschiedenen Pakete anbelangt. Während Beispielsweise die Pakete der Datentypen ziemlich leicht eine sehr hohe Coverage erreichen können, haben vor allem Pakete die einen höheren GUI Bezug haben deutlich geringere Werte.
Außerdem muss man bedenken, dass einige Klassen aus Paketen betriebssytemabhänging sind, sodass diese Pakete nie eine 100%ige Coverage erreichen können. Auch von AntLR erstellte Klassen haben eine sehr geringe Coverage, da wir diese nicht testen.
Berechnet man nun die Coverage ohne die oben genannten Teile, kommen wir auf ca 76%[2]. Zusammen mit den vielen GUI-Tests, und auch dem normalen Benutzen mit BEAST sind wir aber zuversichtlich, dass unere Testfallüberdeckung ausreicht um eine angenehme Benutzung von BEAST möglich zu machen.



Element	Missed Instructions	Cov.	Missed Branches	Cov.	Missed	Cxty	Missed	Lines	Missed	Methods	Missed	Classes
edu.pse.beast.celectiondescriptioneditor.CElectionCodeArea		100%		100%	0	13	0	48	0	5	0	3
edu.pse.beast.datatypes.internal		100%		100%	0	12	0	38	0	11	0	2
edu.pse.beast.datatypes.electioncheckparameter		100%		100%	0	13	0	30	0	12	0	2
edu.pse.beast.celectiondescriptioneditor.ElectionTemplates		98%		86%	2	16	3	135	0	9	0	2
edu.pse.beast.celectiondescriptioneditor		97%		57%	5	53	8	234	0	48	0	6
edu.pse.beast.propertylist.Model		94%		82%	14	63	13	150	7	44	0	3
edu.pse.beast.datatypes		93%		67%	11	62	13	168	6	53	0	7
edu.pse.beast.datatypes		93%		73%	4	26	4	47	1	19	0	2
edu.pse.beast.saveloader.StaticSaverLoaders		93%		62%	15	44	13	134	5	25	0	6
edu.pse.beast.booleanexpeditor.booleanExpCodeArea		93%		80%	6	24	11	92	5	19	0	5
edu.pse.beast.codearea.SyntaxHL		91%		100%	1	15	4	43	1	13	0	3
edu.pse.beast.toolbox.antlr.booleanexp.GenerateAST		90%		73%	16	80	19	198	6	52	0	3
edu.pse.beast.celectiondescriptioneditor.View		89%		75%	2	21	13	104	1	19	0	3
edu.pse.beast.datatypes.electiondescription		88%		n/a	8	29	11	59	8	29	0	4
edu.pse.beast.parameditor		88%		49%	40	112	48	351	9	67	0	9
edu.pse.beast.propertychecker		85%		72%	117	365	207	1.263	16	149	1	18
edu.pse.beast.stringsource		85%		72%	7	58	17	139	4	49	0	9
edu.pse.beast.booleanexpeditor.View		85%		50%	5	23	9	153	3	21	0	3
edu.pse.beast.parameditor.View		83%		18%	26	70	80	417	19	59	3	13
edu.pse.beast.booleanexpeditor		81%		29%	23	75	66	384	8	58	0	9
edu.pse.beast.codearea.InputToCode.NewlineInserter		79%		89%	4	22	12	53	2	13	0	4
edu.pse.beast.propertylist.View		78%		60%	60	145	119	433	29	83	0	15
edu.pse.beast.datatypes.booleanExpAST		77%		100%	4	19	4	25	4	18	0	3
edu.pse.beast.toolbox		77%		58%	71	174	101	411	26	94	1	22
edu.pse.beast.saveloader		73%		55%	23	54	119	353	8	33	0	7
edu.pse.beast.propertychecker.ina				50%	1	4	2	14	0	3	0	1
edu.pse.beast.saveloader.OptionSaverLoader		67%		58%	8	18	11	44	3	12	0	1
edu.pse.beast.datatypes.propertydescription		67%		28%	17	42	33	97	9	33	0	4
edu.pse.beast.codearea		67%		31%	59	120	109	344	20	72	0	7
edu.pse.beast.options.CodeAreaOptions		65%		50%	4	18	20	65	1	12	0	4
edu.pse.beast.celectiondescriptioneditor.UserActions		65%		30%	13	28	24	88	8	23	0	11
edu.pse.beast.codearea.ErrorHandling		65%		40%	27	63	40	162	11	43	0	7
edu.pse.beast.highlevel		64%		62%	18	43	46	142	6	27	1	4
edu.pse.beast.codearea.ActionAdder		62%		25%	5	9	11	34	3	7	0	1
edu.pse.beast.codearea.UserActions		61%		75%	6	19	25	64	5	17	0	6
edu.pse.beast.options.ParametereditorOptions		59%		50%	7	19	30	71	4	15	0	5
edu.pse.beast.celectiondescriptioneditor.CElectionCodeArea.ErrorHandling		58%		35%	36	59	113	250	10	28	1	8
edu.pse.beast.toolbox.antlr.booleanexp		57%		39%	299	428	483	1.023	193	297	5	27
edu.pse.beast.options.BooleanExpEditorOptions		53%		n/a	4	8	12	28	4	8	0	2
edu.pse.beast.options.CEditorOptions		52%		n/a	3	8	11	27	3	8	0	2
edu.pse.beast.datatypes.booleanExpAST.otherValuedNodes.integerValuedNodes		46%		0%	40	58	30	71	7	25	0	5
edu.pse.beast.codearea.InputToCode		45%		31%	134	220	244	446	39	67	0	11
edu.pse.beast.datatypes.booleanExpAST.BooleanValuedNodes		45%		0%	46	74	49	114	14	42	2	12
edu.pse.beast.booleanexpeditor.UserActions		39%		0%	29	44	63	122	18	36	0	16
edu.pse.beast.datatypes.booleanExpAST.otherValuedNodes		38%		0%	26	42	21	53	7	23	0	6
edu.pse.beast.booleanexpeditor.booleanExpCodeArea.errorFinder		37%		34%	77	125	165	296	48	90	1	5
edu.pse.beast.parameditor.UserActions		35%		0%	30	39	56	99	9	18	0	9
edu.pse.beast.propertylist.UserActions		35%		6%	12	18	23	43	4	10	0	5
edu.pse.beast.codearea.Autocompletion		34%		18%	63	88	133	218	24	41	2	7
edu.pse.beast.codearea.Actionlist		28%		0%	8	12	22	35	4	8	0	1
edu.pse.beast.options		24%		36%	34	59	102	142	24	36	2	6
edu.pse.beast.codearea.Actionlist.TextAction		16%		n/a	7	9	22	30	7	9	1	3
edu.pse.beast.celectiondescriptioneditor.CElectionCodeArea.Antlr		13%		1%	1.678	1.896	4.819	4.992	1.044	1.054	86	89
Total	31.323 of 63.306	51%	2.802 of 3.889	28%	3.354	5.149	7.613	14.576	1.697	3.085	106	428

Abbildung 3.1: Eine Übersicht über die in den einzelnen Paketen erreichte CodeCoverage (eine größere Version des Bildes befindet sich im Anhang)

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.

[1] Die „Coverage“ einer Klasse beschreibt wie viele der Befehle (Zuweisungen, Methodenaufrufe, ...) , wobei es davon auch mehrere pro Zeile code geben kann, von mindestens einem Testfall abgelaufen werden.

[2] Berechnet durch: $32000(coveredinstructions)/63300(alllines) - 19500(CElectionCodeArea.Antlr) - 1800(booleanexp) - 200(LinuxProcess)$

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 Grundfür 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

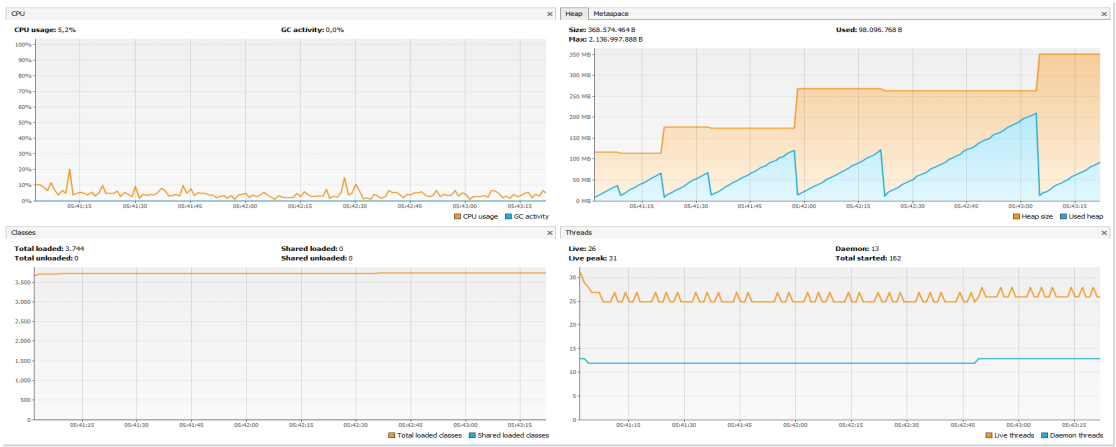


Abbildung 4.2: Der Ressourcenverbrauch der momentanen Version des Programmes, während keine Überprüfung durchgeführt wird

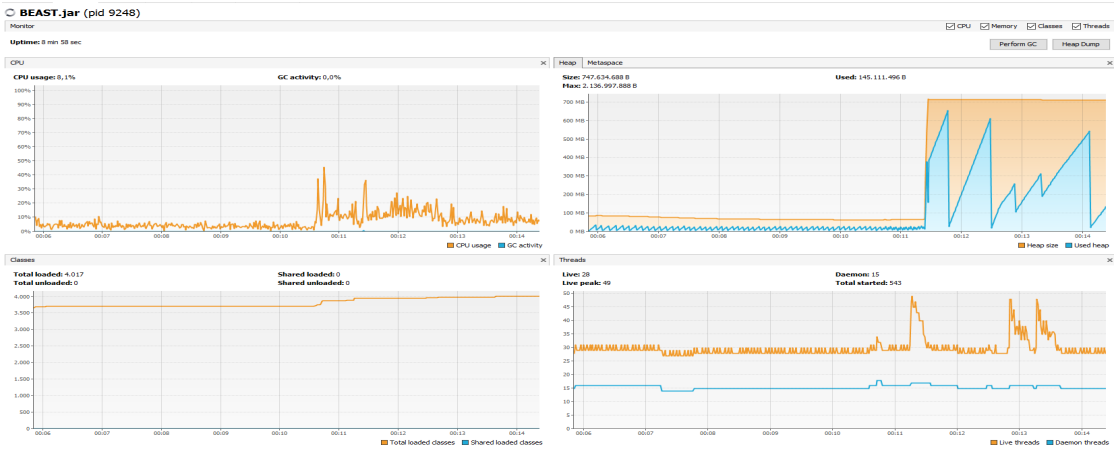


Abbildung 4.3: Der Ressourcenverbrauch der originalen Version von BEAST, während Eigenschaften überprüft werden

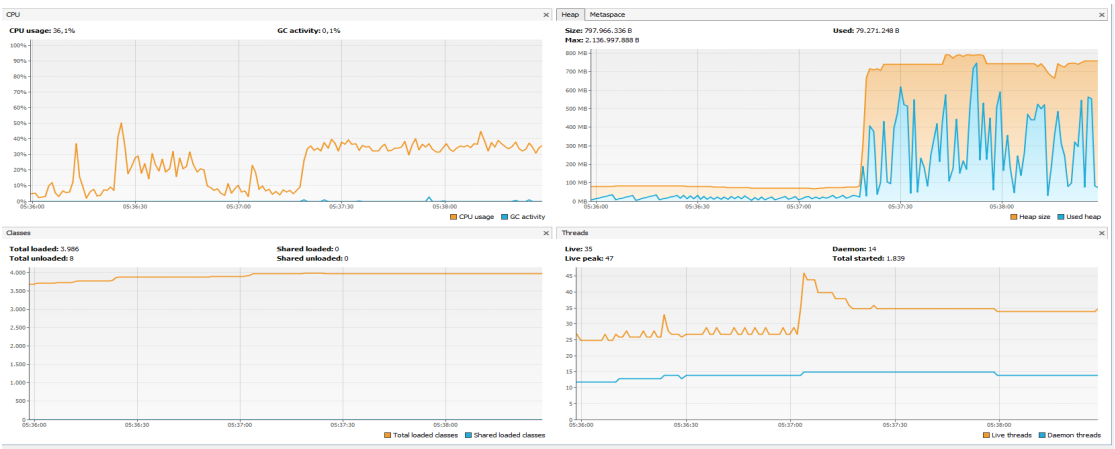
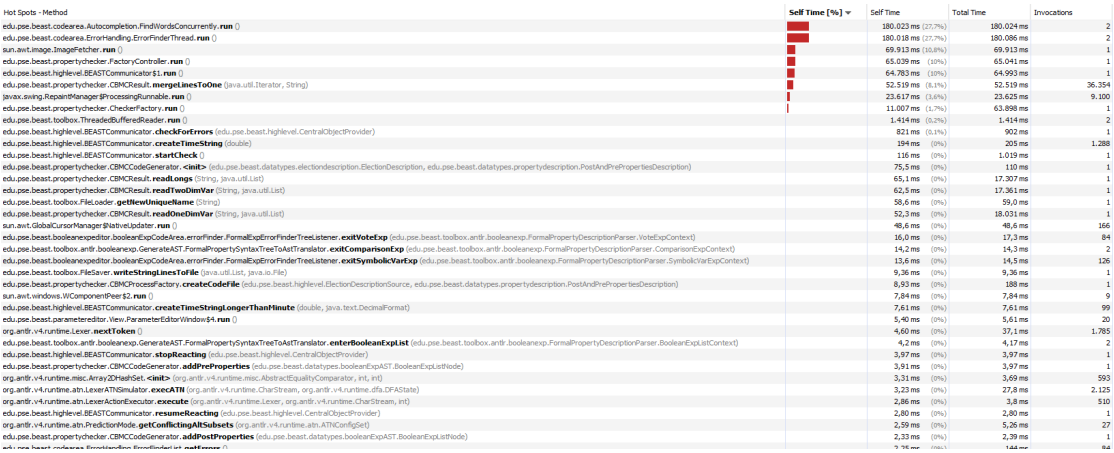


Abbildung 4.4: Der Ressourcenverbrauch der momentanen Version des Programmes, während Eigenschaften überprüft werden

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.Autocompletion.FindWordsConcurrently.**run** ()		180.023 ms (27,7%)	180.024 ms	2
edu.pse.beast.codearea.ErrorHandling.ErrorFinderThread.**run** ()		180.018 ms (27,7%)	180.086 ms	2
sun.awt.image.ImageFetcher.**run** ()		69.913 ms (10,8%)	69.913 ms	1
edu.pse.beast.propertychecker.FactoryController.**run** ()		65.039 ms (10%)	65.041 ms	1
edu.pse.beast.highlevel.BEASTCommunicator$1.**run** ()		64.783 ms (10%)	64.993 ms	1
edu.pse.beast.propertychecker.CBMCResult.**mergeLinesToOne** (java.util.Iterator, String)		52.519 ms (8,1%)	52.519 ms	36.354
javax.swing.RepaintManager$ProcessingRunnable.**run** ()		23.617 ms (3,6%)	23.625 ms	9.100
edu.pse.beast.propertychecker.CheckerFactory.**run** ()		11.007 ms (1,7%)	63.898 ms	1
edu.pse.beast.toolbox.ThreadedBufferedReader.**run** ()		1.414 ms (0,2%)	1.414 ms	2
edu.pse.beast.highlevel.BEASTCommunicator.**checkForErrors** (edu.pse.beast.highlevel.CentralObjectProvider)		821 ms (0,1%)	902 ms	1
edu.pse.beast.highlevel.BEASTCommunicator.**createTimeString** (double)		194 ms (0%)	205 ms	1.288
edu.pse.beast.highlevel.BEASTCommunicator.**startCheck** ()		116 ms (0%)	1.019 ms	1
edu.pse.beast.propertychecker.CBMCCodeGenerator.**<init>** (edu.pse.beast.datatypes.electiondescription.ElectionDescription, edu.pse.beast.datatypes.propertydescription.PostAndPrePropertiesDescription)		75,5 ms (0%)	110 ms	1
edu.pse.beast.propertychecker.CBMCResult.**readLongs** (String, java.util.List)		65,1 ms (0%)	17.307 ms	1
edu.pse.beast.propertychecker.CBMCResult.**readTwoDimVar** (String, java.util.List)		62,5 ms (0%)	17.361 ms	1
edu.pse.beast.toolbox.FileLoader.**getNextUniqueName** (String)		58,6 ms (0%)	59,0 ms	1
edu.pse.beast.propertychecker.CBMCResult.**readOneDimVar** (String, java.util.List)		52,3 ms (0%)	18.031 ms	1
sun.awt.GlobalCursorManager$NativeUpdater.**run** ()		48,6 ms (0%)	48,6 ms	166
edu.pse.beast.booleanexpeditor.booleanExpCodeArea.errorFinder.FormalExpErrorFinderTreeListener.**exitVoteExp** (edu.pse.beast.toolbox.antlr.booleanexp.FormalPropertyDescriptionParser.VoteExpContext)		16,0 ms (0%)	17,3 ms	84
edu.pse.beast.toolbox.antlr.booleanexp.GenerateAST.FormalPropertySyntaxTreeToAstTranslator.**exitComparisonExp** (edu.pse.beast.toolbox.antlr.booleanexp.FormalPropertyDescriptionParser.ComparisonExpContext)		14,2 ms (0%)	14,3 ms	2
edu.pse.beast.booleanexpeditor.booleanExpCodeArea.errorFinder.FormalExpErrorFinderTreeListener.**exitSymbolicVarExp** (edu.pse.beast.toolbox.antlr.booleanexp.FormalPropertyDescriptionParser.SymbolicVarExpContext)		13,6 ms (0%)	14,5 ms	126
edu.pse.beast.toolbox.FileSaver.**writeStringLinesToFile** (java.util.List, java.io.File)		9,36 ms (0%)	9,36 ms	1
edu.pse.beast.propertychecker.CBMCProcessFactory.**createCodeFile** (edu.pse.beast.highlevel.ElectionDescriptionSource, edu.pse.beast.datatypes.propertydescription.PostAndPrePropertiesDescription)		8,93 ms (0%)	188 ms	1
sun.awt.windows.WComponentPeer$2.**run** ()		7,84 ms (0%)	7,84 ms	9
edu.pse.beast.highlevel.BEASTCommunicator.**createTimeStringLongerThanMinute** (double, java.text.DecimalFormat)		7,61 ms (0%)	7,61 ms	99
edu.pse.beast.parametereditor.View.ParameterEditorWindow$4.**run** ()		5,40 ms (0%)	5,61 ms	20
org.antlr.v4.runtime.Lexer.**nextToken** ()		4,60 ms (0%)	37,1 ms	1.785
edu.pse.beast.toolbox.antlr.booleanexp.GenerateAST.FormalPropertySyntaxTreeToAstTranslator.**enterBooleanExpList** (edu.pse.beast.toolbox.antlr.booleanexp.FormalPropertyDescriptionParser.BooleanExpListContext)		4,2 ms (0%)	4,17 ms	2
edu.pse.beast.highlevel.BEASTCommunicator.**stopReacting** (edu.pse.beast.highlevel.CentralObjectProvider)		3,97 ms (0%)	3,97 ms	1
edu.pse.beast.propertychecker.CBMCCodeGenerator.**addPreProperties** (edu.pse.beast.datatypes.booleanExpAST.BooleanExpListNode)		3,91 ms (0%)	3,97 ms	1
org.antlr.v4.runtime.misc.Array2DHashSet.**<init>** (org.antlr.v4.runtime.misc.AbstractEqualityComparator, int, int)		3,31 ms (0%)	3,69 ms	593
org.antlr.v4.runtime.atn.LexerATNSimulator.**execATN** (org.antlr.v4.runtime.CharStream, org.antlr.v4.runtime.dfa.DFAState)		3,23 ms (0%)	27,8 ms	2.125
org.antlr.v4.runtime.atn.LexerActionExecutor.**execute** (org.antlr.v4.runtime.Lexer, org.antlr.v4.runtime.CharStream, int)		2,86 ms (0%)	3,8 ms	510
edu.pse.beast.highlevel.BEASTCommunicator.**resumeReacting** (edu.pse.beast.highlevel.CentralObjectProvider)		2,80 ms (0%)	2,80 ms	1
org.antlr.v4.runtime.atn.PredictionMode.**getConflictingAltSubsets** (org.antlr.v4.runtime.atn.ATNConfigSet)		2,59 ms (0%)	5,26 ms	27
edu.pse.beast.propertychecker.CBMCCodeGenerator.**addPostProperties** (edu.pse.beast.datatypes.booleanExpAST.BooleanExpListNode)		2,33 ms (0%)	2,39 ms	1

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

5 Fehlerbehebungen

Im Laufe der Phase haben wir einige Fehler gefunden, welche die Benutzung von BEAST stark beeinträchtigt haben. Eine komplette Liste aller „Issues“ kann auf der BEAST GitHub-Seite[1] angeschaut werden. Trotzdem werden wir hier einen kleinen Überblick über ein paar der gefundenen Fehler und deren Behebung geben:

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 `UserInserToCode` 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 17 -
Beschreibung: Öffnet man eine falsch formatierte Datei in BEAST führte dies zu einer NullpointerException
Lösung: Gelöst durch zwingende Namensgebung der form „list$_o f_c andidates_p er_v oter$“0

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.

[1] https://github.com/NikolaiLMS/PSE-Wahlverfahren-Implementierung/issues

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.

- Gibt der Nutzer nun Dateien, welche in das C-Programm eingebunden werden sollen, an, wird automatisch überprüft, ob diese einem standart C-Include entsprechen, oder aber in dem speziellen Ordner $\backslash core\backslash user_{i}ncludes\backslash liegen.Siewerdendannauchautomatischancbmcweitergegeben$,

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 1.4.22	**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/ (Eigenschaftenliste)	/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	X

Tabelle 7.2: Testfall 8.1 (Testfälle für die Datenverwaltung)

Sub-Testfall	**Abgedeckte Funktionalitäten**	**Beschreibung**	**Ergebnis**	**Lukas** (Windows 10) Version 1.4.22	**Justin** 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. Man kann das vorher gespeicherte Wahlverfahren öffnen	✓	✓ /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. Man kann die vorher gespeicherte Eigenschaft öffnen	✓	✓	

Tabelle 7.3: Testfall 8.1 (Testfälle für die Datenverwaltung)

Sub-Testfall	**Abgedeckte Funktionalitäten**	**Beschreibung**	**Ergebnis**	**Lukas** (Windows 10) Version 1.4.22	**Justin** Lubuntu 16.1 Version 1.4.19
/T020/ /T030/ (Eigenschaftenliste)	/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/ (Parametereditor)	/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: Testfall 8.3 (Testfall für Kopieren, Einfügen und Ausschneiden in den Editoren)

Sub-Testfall	**Abgedeckte Funktionalitäten**	**Beschreibung**	**Ergebnis**	**Niels** (Windows 10) Version 1.4.22	**Niels** (Linux Mint Cinnamon 3.0.7) Version 1.4.22
/T200/	/F0010/ /FS1100/ /FS2150/	Man startet das Programm ganz normal. Nun öffnet man den den jeweiligen Editor (C-Editor und Eigenschafteneditor) und gibt einen kleinen Text ein. Man markiert den kleinen Text und drückt dann, während der Fokus auf dem Editor liegt, „Strg + x“ oder betätigt den Button für Ausschneiden.	Der markierte Text wird gelöscht und in den Zwischenspeicher gespeichert.	✓	✓
/T200/	/F0010/ /FS1100/ /FS2150/	Man startet das Programm ganz normal. Nun öffnet man den den jeweiligen Editor (C-Editor und Eigenschafteneditor) und gibt einen kleinen Text ein. Man markiert den kleinen Text und drückt dann, während der Fokus auf dem Editor liegt, „Strg + c“ oder betätigt den Button für Kopieren	Der markierte Text wird in den Zwischenspeicher gespeichert.	✓	✓
/T200/	/F0010/ /FS1100/ /FS2150/	Man startet das Programm ganz normal. Nun öffnet man den den jeweiligen Editor (C-Editor und Eigenschafteneditor) Man drückt „Strg + c“ oder betätigt den Button für Einfügen.	Falls ein Text im Zwischenspeicher gespeichert ist, wird er im Editor eingefügt.	✓	✓

Tabelle 7.6: Testfall 7 (Nichtfunktionale Anforderungen)

Sub-Testfall	**Abgedeckte Funktionalitäten**	**Beschreibung**	**Ergebnis**	**Lukas** (Windows 10) Version 1.4.22	**Niels** (Linux Mint Cinnamon 3.0.7) Version 1.4.22
	/NF10/	Man startet das Programm ganz normal. Nun öffnet man den C-Editor und gibt in die Mitte der Voting Methode „for“ ein. Nun drückt man „strg“ + „leer“	In weniger als 0.5 Sekunden öffnet sich ein Fenster, welches die Code-Completion anzeigt	✓	✓
	/NF30/	Man startet das Programm ganz normal und öffnet den C-Editor. Hier gibt man nun einen Code ein, der über 10000 Zeilen lang ist. Im ParameterEditor wählt man alle Variablen kleiner als 10 und stellt den TimeOut aus. Im Eigenschafteneditor öffnet man „FalseProperty.props“. Nun startet man die Überprüfung	Nach kurzer Zeit beendet sich die Überprüfung, und man kan das Ergebnis im Eigenschafteneditor ablesen	✓	✓
	/NF20/ /NF40/ /NF50/ /NF60/	Man startet das Programm ganz normal und öffnet den C-Editor. Hier gibt man nur einen sehr einfachen Code ein. Im Eigenschafteneditor erstellt man eine neue Eigenschaft, welche 10 Vor- und Nachbedingungen enthält. Im Parametereditor stellt man alle Werte auf 10000.	CBMC startet mit der Überprüfung der Eigenschaft. Nach 15 Minuten hört es mit der Überprüfung auf.	✓	✓

Tabelle 7.7: Testfall 8.3 (Bearbeiten des Codes in den Editoren)

Sub-Testfall	**Abgedeckte Funktionalitäten**	**Beschreibung**	**Ergebnis**	**Niels** (Windows 10) Version 1.4.22	**Niels** (Linux Mint Cinnamon 3.0.7) Version 1.4.22
/T210/	/FS1130/ /FK2140/	Man startet das Programm ganz normal. Nun öffnet man den den jeweiligen Editor (CEditor und Eigenschafteneditor) und gibt ein Wort innerhalb des Editors teilweise ein, dass der richtigen Syntax entspricht. Nun betätigt man „Strg + Leertaste“	Es öffnet sich ein Fenster für die Autocompletion, aus der man die gewünschte Eingabe wählen kann, welche nach Auswahl hinzugefügt wird.	✓	✓
/T210/	/FS1120/ /FS1130/	Man startet das Programm ganz normal. Man öffnet den C-Editor. Man gibt C-Code ein.	Klammern und Anführungszeichen werden automatisch geschlossen. Code in Schleifen und if-Statements wird automatisch eingerückt.	✓	✓

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))
/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 7.9: 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 ËrstellenButtons	Der Funktionskörper wird entsprechend aktualisiert		
		Single-choice	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 7.10: 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 ßtatische Analyseäus. 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 7.11: 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 7.12: 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 7.13: 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 7.14: Testfall 8.5 (Testfall für das Erstellen einer Eigenschaft im Eigenschafteneditor)

Sub-Testfall	**Abgedeckte Funktionalitäten**	**Beschreibung**	**Ergebnis**	**Lukas** (Windows 10) Version 1.4.22	**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 Fehlersuchett-estet 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.	✓	✓

Tabelle 7.15: 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 7.16: Testfall 8.7 (Testfälle für den Parametereditor)

Sub-Testfall	**Abgedeckte Funktionalitäten**	**Beschreibung**	**Ergebnis**	**Jonas** (Windows 10 Version 1607) BEAST v1.4.18	**Niels** (Linux Mint Cinnamon 3.0.7) Version 1.4.22
/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 7.17: Testfall 8.7 (Testfälle für den Parametereditor)

Sub-Testfall	**Abgedeckte Funktionalitäten**	**Beschreibung**	**Ergebnis**	**Jonas** (Windows 10 Version 1607) BEAST v1.4.18	**Niels** (Linux Mint Cinnamon 3.0.7) Version 1.4.22
/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 7.18: Testfall 8.8 (Testfälle für die Datenverwaltung)

Testfall	**Abgedeckte Funktionalitäten**	**Beschreibung**	**Ergebnis**	**Jonas** (Windows 10 Version 1607) BEAST v1.4.18	**Niels** (Linux Mint Cinnamon 3.0.7) Version 1.4.22
/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.	✓	✓

Tabelle 7.19: Testfall 8.8 (Testfälle für die Datenverwaltung)

Testfall	**Abgedeckte Funktionalitäten**	**Beschreibung**	**Ergebnis**	**Lukas** (Windows 10) Version 1.4.22	**Niels** (Linux Mint Cinnamon 3.0.7) Version 1.4.22
	/NF10/	Man startet das Programm normal und öffnet den CEditor. Hier tippt man nun in die Mitte der voting Methode und schreibt „for“. Nun drückt man „strg + leer“.	Nach weniger als 0.5 Sekunden öffnet sich ein Fenster, welches alle Autovervollständigungen anzeigt	✓	✓
	/NF30/	Man startet das Programm normal und öffnet den CEditor. Hier tippt man nun 10000 Zeilen richtigen C-Code ein (Beispielsweise 1 Zeile: int i = 1; 1000 Zeilen: i++; Am Ende: return i;). Im Paramtereditor stellt man alle Parameter moderat ein (alles unter 10, Timeout ausgestellt). Im Eigenschafteneditor läd man die Eigenschaft „FalseProperty.props“ und startet die Analyse.	Nach einiger Zeit schließt die Analys ab und man kann das Ergebnis angucken	✓	✓

Tabelle 7.20: Testfall 8.8 (Testfälle für die Datenverwaltung)

Testfall	**Abgedeckte Funktionalitäten**	**Beschreibung**	**Ergebnis**	**Lukas** (Windows 10) Version 1.4.22	**Niels** (Linux Mint Cinnamon 3.0.7) Version 1.4.22
	/NF20/ /NF40/ /NF50/ /NF60/	Man startet das Programm normal und öffnet den CEditor. Hier tippt man nun 10000 Zeilen richtigen C-Code ein (Beispielsweise 1 Zeile: int i = 1; 1000 Zeilen: i++; Am Ende: return i;). Im Paramtereditor stellt man alle Parameter auf 10000 ein und den TimeOut auf 15 Minuten (der TimeOut ist sehr linear, wenn er 15 Minuten schafft, schafft er auch mehrere Tage / Jahre). Im Eigenschafteneditor läd man die Eigenschaft „FalseProperty.props“ und startet die Analyse.	Nach ziemlich genau 15 Minuten hört die Überprüfung auf, und die Eigenschaft wird als durch einen Timeout abgebrochen angezeigt	✓	✓