

# Pflichtenheft

Simon Bischof

Jan Haag

Adrian Herrmann  
Matthias Schnetz

Lin Jin

Tobias Schlumberger

15. November 2011

# Inhaltsverzeichnis

<b>1</b>	<b>Produktübersicht</b>	<b>4</b>
<b>2</b>	<b>Zielbestimmung</b>	<b>4</b>
2.1	Interpreter / Run-time Checker . . . . .	4
2.1.1	Musskriterien . . . . .	4
2.1.2	Sollkriterien . . . . .	4
2.1.3	Kannkriterien . . . . .	4
2.2	Grafische Benutzeroberfläche . . . . .	4
2.2.1	Musskriterien . . . . .	4
2.2.2	Kannkriterien . . . . .	5
2.2.3	Abgrenzungskriterien . . . . .	5
2.3	Verifikation . . . . .	5
2.3.1	Musskriterien . . . . .	5
2.4	While-Sprache . . . . .	5
2.4.1	Musskriterien . . . . .	5
2.4.2	Kannkriterien . . . . .	5
2.4.3	Abgrenzungskriterien . . . . .	5
2.5	Annotationssprache . . . . .	5
2.5.1	Musskriterien . . . . .	5
2.5.2	Sollkriterien . . . . .	5
<b>3</b>	<b>Produkteinsatz</b>	<b>6</b>
3.1	Anwendungsbereiche . . . . .	6
3.2	Zielgruppen . . . . .	6
3.3	Betriebsbedingungen . . . . .	6
<b>4</b>	<b>Produktumgebung</b>	<b>6</b>
4.1	Software . . . . .	6
4.2	Hardware . . . . .	6
4.3	Schnittstellen . . . . .	6
<b>5</b>	<b>Funktionale Anforderungen</b>	<b>6</b>
5.1	Details . . . . .	6
<b>6</b>	<b>Produktdaten</b>	<b>9</b>
<b>7</b>	<b>Produktleistungen</b>	<b>9</b>
<b>8</b>	<b>Weitere nichtfunktionale Anforderungen</b>	<b>9</b>
<b>9</b>	<b>Qualitätsanforderungen</b>	<b>9</b>
<b>10</b>	<b>Globale Testfälle und Testszenarien</b>	<b>9</b>
<b>11</b>	<b>Systemmodelle</b>	<b>12</b>
11.1	Komponentenübersicht . . . . .	12
11.2	Workflow . . . . .	13
<b>12</b>	<b>Benutzeroberfläche</b>	<b>13</b>

12.1 Übersicht . . . . .	13
12.2 Menüpunkte . . . . .	13
<b>13 Spezielle Anforderungen an die Entwicklungsumgebung</b>	<b>19</b>
<b>14 Zeit- und Ressourcenplanung</b>	<b>19</b>
14.1 Projektplan . . . . .	19
14.2 Phasenverantwortliche . . . . .	19
<b>15 Ergänzungen</b>	<b>19</b>
15.1 While-Sprache . . . . .	19
15.1.1 Variablentypen . . . . .	19
15.1.2 Sprachkonstrukte . . . . .	20
15.1.3 Operatoren . . . . .	20
15.1.4 Sonstige Spracheigenschaften . . . . .	21
15.2 Annotationssprache . . . . .	21
<b>16 Glossar</b>	<b>21</b>

# 1 Produktübersicht

Das Produkt ist ein Werkzeug zur Analyse und Korrektheitsprüfung von Programmen.

Der Benutzer interagiert über eine grafische Oberfläche, die neben einem Editor für die Erstellung und Bearbeitung von Programmen auch detaillierte Rückmeldungen zum aktuellen Programmzustand sowie weiterführende Hilfen zur Verfügung stellt.

Nach dem Erstellen eines Programms mit der im Anhang spezifizierten While-Sprache wird dieses zunächst übersetzt. über das Ergebnis dieser Analyse sowie eventuelle Fehler wird der Benutzer über die grafische Oberfläche informiert.

Im Anschluss entsteht nach einer Erweiterung des Quelltextes um Annahmen in Form einer definierten Annotationssprache durch den Benutzer ein beweisbares Programm, welches mit den zur Verfügung stehenden Methoden zur Analyse des Programms untersucht werden kann.

Für diese Zwecke bietet das Produkt sowohl eine statische als auch dynamische Prüfung des Programmes an. Bei der dynamischen Prüfung kann der Benutzer Eingabeparameter frei wählen und das Programm mit diesen starten. Während der Ausführung werden die angegebenen Annotationen überprüft. Im Fehlerfall wird das Programm im Fehlerzustand eingefroren und der Benutzer über die grafische Oberfläche informiert.

Bei der statischen Überprüfung wird versucht, die Annotationen als korrekt zu beweisen. Wenn der Beweis gelingt, gibt das Programm dem Benutzer dies über die grafische Oberfläche bekannt; anderenfalls liefert das Programm ein Gegenbeispiel oder signalisiert, dass es keine Entscheidung treffen kann.

## 2 Zielbestimmung

### 2.1 Interpreter / Run-time Checker

#### 2.1.1 Musskriterien

- schrittweise Ausführung eines Programms
- Inspektion des aktuellen Programmzustands durch den Benutzer
- Prüfung von Zusicherungen in Programmen zur Laufzeit

#### 2.1.2 Sollkriterien

- Auswertung von Formeln mit Quantoren über eingeschränkten Bereich

#### 2.1.3 Kannkriterien

- Auswertung von benutzerdefinierten Ausdrücken über Programmzustand
- Auswertung von Formeln mit Quantoren mit Hilfe eines Beweisers
- Breakpoints zum Halten der Ausführung eines Programms an definierten Stellen
- Randomisiertes Testen in vorgegebenen Grenzen

### 2.2 Grafische Benutzeroberfläche

#### 2.2.1 Musskriterien

- englische Sprache
- Steuerung der einzelnen Komponenten des Systems
- Anzeige von fehlgeschlagenen Laufzeitprüfungen
- Speichern und Laden von Quelltexten

### **2.2.2 Kannkriterien**

- Syntax Highlighting
- Hilfe zur Syntax der While-Sprache und der Annotationssprache
- Markierung der aktuellen Zeile beim Halten der Ausführung

### **2.2.3 Abgrenzungskriterien**

- Optionen, die keinen echten Mehrwert bieten

## **2.3 Verifikation**

### **2.3.1 Musskriterien**

- formale Verifikation eines Programms mit Hilfe eines Beweisers

## **2.4 While-Sprache**

### **2.4.1 Musskriterien**

- Zuweisungen, Bedingten Anweisungen und Schleifen
- Turing-Vollständigkeit

### **2.4.2 Kannkriterien**

- Programmiersprache unterstützt Methodenaufrufe

### **2.4.3 Abgrenzungskriterien**

- Variablentypen außer Ganzzahl, Boolean und Arrays
- Implementierung eines Heaps
- Implementierung von Nebenläufigkeit
- überladen von Methoden

## **2.5 Annotationssprache**

### **2.5.1 Musskriterien**

- Ausdrücke in Annotationen ähnlich wie in der Programmiersprache, zusätzlich Quantoren
- Zusicherungen erlauben Aussagen der Prädikatenlogik
- Annotationen assert und assume

### **2.5.2 Sollkriterien**

- syntactic sugar für Vor- und Nachbedingungen, (In)Varianten und globale Annahmen

## 3 Produkteinsatz

Das Produkt ermöglicht dem Benutzer die Analyse eines Programmes, das in der im Anhang beschriebenen While-Sprache geschrieben ist. Dabei ist es sowohl möglich, die Anweisungen des Programms schrittweise oder komplett zu durchlaufen und dabei zur Laufzeit auf im Programmcode durch Annotationen eingebettete Zusicherungen zu überprüfen, als auch die formale Verifikation des Programms mithilfe des automatischen Theorembeweisers Z3 von Microsoft.

### 3.1 Anwendungsbereiche

- Formale Verifikation von Programmen
- Testen von Programmen der While-Sprache

### 3.2 Zielgruppen

- Softwareentwickler / Anwendungsentwickler

### 3.3 Betriebsbedingungen

- übliche Beweiserlaufzeit: 1 Stunde
- 

## 4 Produktumgebung

### 4.1 Software

- Java Runtime Environment SE 6 oder höher
- Windows NT 4.0 oder neuer oder Betriebssystem mit Wine

### 4.2 Hardware

- mindestens 512 MB Arbeitsspeicher

### 4.3 Schnittstellen

- Das Produkt erzeugt Dateien im SMT-LIB 2.0 Format als Schnittstelle zu Z3

## 5 Funktionale Anforderungen

### 5.1 Details

#### Funktionale Anforderung 1

Anzeige von Quelltext Der Benutzer sieht den Quelltext des aktuellen Programms in der grafischen Oberfläche.

#### Funktionale Anforderung 2

Teile des angezeigten Quelltextes werden abhängig von ihrer Bedeutung in unterschiedlichen Farben angezeigt. 1

#### Funktionale Anforderung 3

Der Benutzer kann den angezeigten Quelltext verändern oder einen neuen Quelltext eingeben.

#### **Funktionale Anforderung 4**

Der Benutzer kann den eingegebenen Quelltext in eine Datei speichern. Dabei gibt der Benutzer den gewünschten Speicherort und den Dateinamen über eine grafische Oberfläche an.

#### **Funktionale Anforderung 5**

Der Benutzer kann ein Quellcode eines Programms aus einer Datei laden. Hierbei gibt der Benutzer die zu ladende Datei über eine grafische Oberfläche an.

#### **Funktionale Anforderung 6**

Wenn der Interpreter Syntaxfehler erkannt hat (FA20), wird dies dem Benutzer angezeigt, wenn er die Programmausführung oder die Beweisführung anfordert.

#### **Funktionale Anforderung 7**

Wenn der Interpreter Typfehler erkannt hat (FA21), wird dies dem Benutzer angezeigt, wenn er die Programmausführung oder die Beweisführung anfordert.

#### **Funktionale Anforderung 8**

Wenn bei der Ausführung eines Programmes durch den Interpreter ein Fehler auftritt (FA23), wird dies dem Benutzer angezeigt.

#### **Funktionale Anforderung 9**

Der Benutzer kann Breakpoints zur Pausierung eines Programmdurchlaufs an Stellen des angezeigten Quellcodes setzen und wieder entfernen.

#### **Funktionale Anforderung 10**

Der Benutzer sieht während der Ausführung eines Programms den aktuellen Inhalt aller globalen und lokalen Variablen. Die Anzeige wird nur aktualisiert, wenn das Programm die Ausführung pausiert.

#### **Funktionale Anforderung 11**

Der Benutzer kann die Ausführung eines Programms anhalten. Dabei wird der aktuelle Programmzustand beibehalten.

#### **Funktionale Anforderung 12**

Der Benutzer kann die Ausführung eines Programms abbrechen. Dabei wird der Programmzustand verworfen.

#### **Funktionale Anforderung 13**

Meldet der Interpreter eine fehlgeschlagene Annotationsbedingung (FA25), so wird dies dem Benutzer angezeigt.

#### **Funktionale Anforderung 14**

Der Benutzer kann während der angehaltenen Ausführung eines Programms Ausdrücke eingeben, die vom Interpreter ausgewertet werden und deren Ergebnis dem Benutzer angezeigt wird.

**Funktionale Anforderung 15**

Der Benutzer kann die aktuellen Einstellungen (1) für die Ausführung des Beweisers (siehe FA0) anzeigen lassen. Diese werden vor der Anzeige aus einer Datei geladen (FA16) und bei einer Veränderung wieder in eine Datei geschrieben (FA17).

**Funktionale Anforderung 16**

Die Einstellungen für die Ausführung des Beweisers (siehe PD1) werden automatisch aus einer Datei gelesen.

**Funktionale Anforderung 17**

Die Einstellungen (siehe PD1) werden in eine Datei gespeichert.

**Funktionale Anforderung 18**

Der Benutzer kann den Beweisvorgang mit Hilfe des Beweisers unter den eingegebenen Einstellungen starten.

**Funktionale Anforderung 19**

Nach dem Durchlauf des Beweisers wird dem Benutzer das Ergebnis des Beweisvorgangs angezeigt.

**Funktionale Anforderung 20**

Der Interpreter erkennt Syntaxfehler des eingegebenen Quelltextes.

**Funktionale Anforderung 21**

Der Interpreter erkennt Typfehler des eingegebenen Quelltextes.

**Funktionale Anforderung 22**

Die Anweisungen im aktuell angezeigten Quelltext können ausgeführt werden. Dabei sind folgende Ausführungsmodi möglich:

- |                      |  |
|----------------------|--|
| Single Step          | Es wird auf eine Interaktion des Benutzers nur eine Anweisung ausgeführt.  |
| Run until Breakpoint | Es werden sequentiell alle Anweisungen ausgeführt, bis das Programm beendet ist oder ein Breakpoint (FA9) erreicht wird. |

**Funktionale Anforderung 23**

Tritt ein Fehler zur Laufzeit des Programms auf, wird die Ausführung abgebrochen und der Fehler dem Benutzer angezeigt.

**Funktionale Anforderung 24**

Der Benutzer kann Ausdrücke über den aktuellen Programzustand auswerten lassen.

**Funktionale Anforderung 25**

Während der Programmausführung werden die Bedingungen der Annotationssprache ausgewertet. Schlägt eine Überprüfung fehl, so wird die Ausführung des Programms angehalten und dem Benutzer angezeigt, welche Bedingung falsch ist.

**Funktionale Anforderung 26**

Der Beweisvorgang für das aktuelle Programm wird durchgeführt.



## 6 Produktdaten

Es sind folgende Daten des Benutzers zu Speichern:

### Produktdaten 1

Benutzereinstellungen

- Timeout in Sekunden für den Beweiser
- Speicherbegrenzung in Megabyte für den Beweiser

### Produktdaten 2

Zuletzt geöffnetes Verzeichnis

- Das zuletzt durch 5 geöffnete Verzeichnis.

## 7 Produktleistungen

- Verifikation- und Falsifikationsrückmeldungen vom Beweiser bezüglich der Spezifikation müssen korrekt sein, in anderen Fällen soll dem Benutzer mitgeteilt werden, dass der Beweiser das Problem nicht lösen kann
- Interpreter: Anweisungen mit weniger als 50 Operatoren werden in weniger als 0,1 Sekunde ausgeführt
- Parser: Programme mit 10.000 Zeilen werden in weniger als 1 Sekunde geparkt
- vollständige Dokumentation der Syntax der While- und Annotationssprache

## 8 Weitere nichtfunktionale Anforderungen

Die Lizenz von Z3 erlaubt ausschließlich eine nicht-kommerzielle Benutzung.

## 9 Qualitätsanforderungen

- Programmiersprache muss erweiterbar sein
- mögliche Anbindung anderer Beweiser, die das SMT-LIB 2.0 Format unterstützen
- unter (durch benutzerdefinierte Einstellungen vorgegebene, FA15) Normalbedingungen stürzt das Produkt nicht ab
- jede Aktion gibt in jedem Fall eine Rückmeldung an den Benutzer

## 10 Globale Testfälle und Testszenarien

Folgende Funktionssequenzen sind zu überprüfen:

### Globale Testfälle 1

Eingabe des leeren Programms → keine Syntaxfehler

### Globale Testfälle 2

Eingabe des leeren Programms ohne Annotationen, Beweisvorgang starten → Verifikation durch den Beweiser

### Globale Testfälle 3

Eingabe des leeren Programms, Annotation `ensure false`, Programmausführung → Annotationsbedingung schlägt fehl

### Globale Testfälle 4

Eingabe des leeren Programms, Annotation `ensure true`, Programmausführung → Annotationsbedingung schlägt nicht fehl

### Globale Testfälle 5

Eingabe eines beliebigen Programms, Annotation `require false`, Programmausführung → Annotationsbedingung schlägt fehl, Programmausführung wird abgebrochen

### Globale Testfälle 6

Eingabe eines Programms mit Zuweisung `x = 1`, Annotation `require x > 0`, Programmausführung → Annotationsbedingung schlägt nicht fehl

### Globale Testfälle 7

Eingabe eines Programms mit Syntaxfehler (z.B. fehlerhafte Klammerung, fehlendes Semikolon, Methodenname ist Keyword) → Anzeige des Syntaxfehlers

### Globale Testfälle 8

Eingabe eines Programms mit while-Schleife oder if-Konstrukt, deren Bedingung kein boolean ist → Anzeige des Typfehlers

### Globale Testfälle 9

Eingabe eines Programms mit einem Operator, der nur auf Ganzzahlen definiert ist und auf einen boolean angewendet wird → Anzeige des Typfehlers

### Globale Testfälle 10

Eingabe eines Programms mit einem Operator, der nur auf booleans definiert ist und auf Ganzzahlen angewendet wird → Anzeige des Typfehlers

### Globale Testfälle 11

Eingabe eines Programms mit einer Zuweisung eines boolean an eine Ganzzahl → Anzeige des Typfehlers

### Globale Testfälle 12

Eingabe eines Programms mit einer Zuweisung einer Ganzzahl an einen boolean → Anzeige des Typfehlers

### Globale Testfälle 13

Eingabe eines Programms mit einer Zuweisung einer Ganzzahl oder eines boolean an ein Array → Anzeige des Typfehlers

### Globale Testfälle 14

Eingabe eines Programms mit einem Vergleichsoperator, der auf zwei unterschiedliche Typen ausgeführt wird → Anzeige des Typfehlers

#### Globale Testfälle 15

Eingabe eines Programms mit Zuweisung  $y = 2x$ , Annotation `ensure y==2x`, Beweisvorgang starten → Verifizierung durch den Beweiser

#### Globale Testfälle 16

Eingabe eines Programms mit Zuweisung  $y = 2x$ , Annotation `ensure y%2==0`, Beweisvorgang starten → Verifizierung durch den Beweiser

#### Globale Testfälle 17

Eingabe eines längeren Programms ohne Schleifen und ohne if-Konstrukt, Eingabe von Annotationen, die in jedem Fall wahr sind, Beweisvorgang starten → Verifizierung durch den Beweiser

#### Globale Testfälle 18

Eingabe eines längeren Programms ohne Schleifen und mit if-Konstrukt, Eingabe von Annotationen, die in jedem Fall wahr sind, Beweisvorgang starten → Verifizierung durch den Beweiser

#### Globale Testfälle 19

Eingabe eines längeren Programms mit Schleifen, Eingabe von Annotationen, die in jedem Fall wahr sind, Beweisvorgang starten → Verifizierung durch den Beweiser

#### Globale Testfälle 20

Eingabe eines Programms mit Annotation `ensure x!=x`, Beweisvorgang starten → Falsifikation durch den Beweiser

#### Globale Testfälle 21

Eingabe eines längeren Programms mit Schleife (z.B. russische Multiplikation, Maximum der Zahlen in einem Array), Eingabe von Annotationen, die in mindestens einem Fall falsch sind, Beweisvorgang starten → Falsifikation durch den Beweiser

#### Globale Testfälle 22

Eingabe eines Programms mit Annotation `require x%2==0`, Zuweisung  $y = 2*(x/2)$  und Annotation `ensure (x==y)`, Beweisvorgang starten → Verifikation durch den Beweiser

#### Globale Testfälle 23

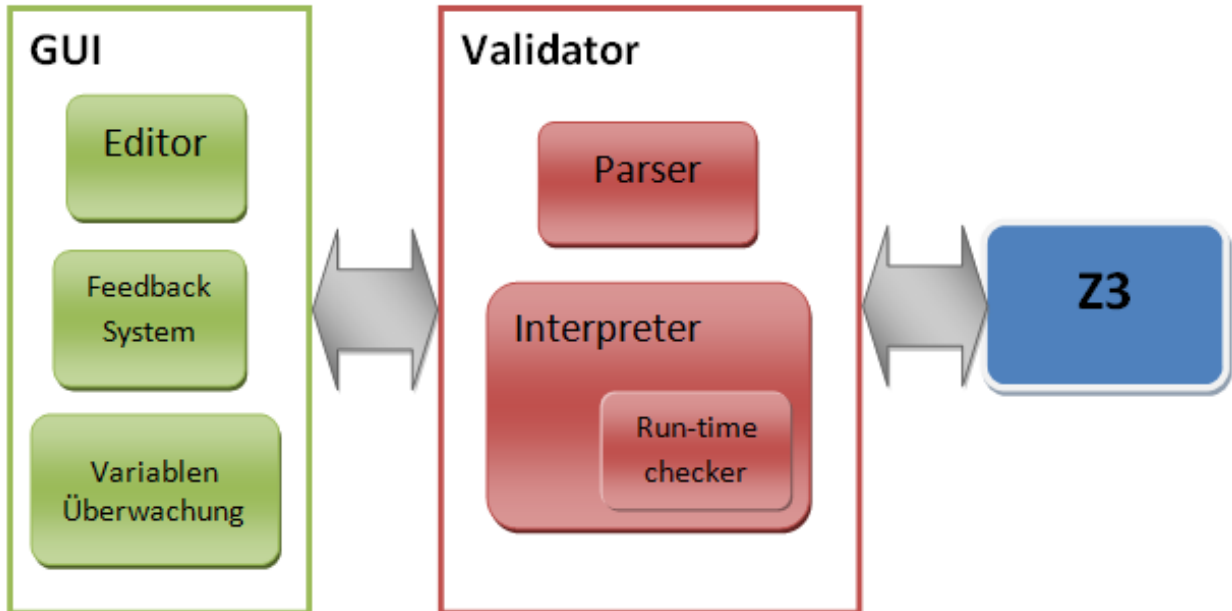
Eingabe eines Programms mit Division durch 0, Ausführung des Programms → Anzeige des Laufzeitfehlers

#### Globale Testfälle 24

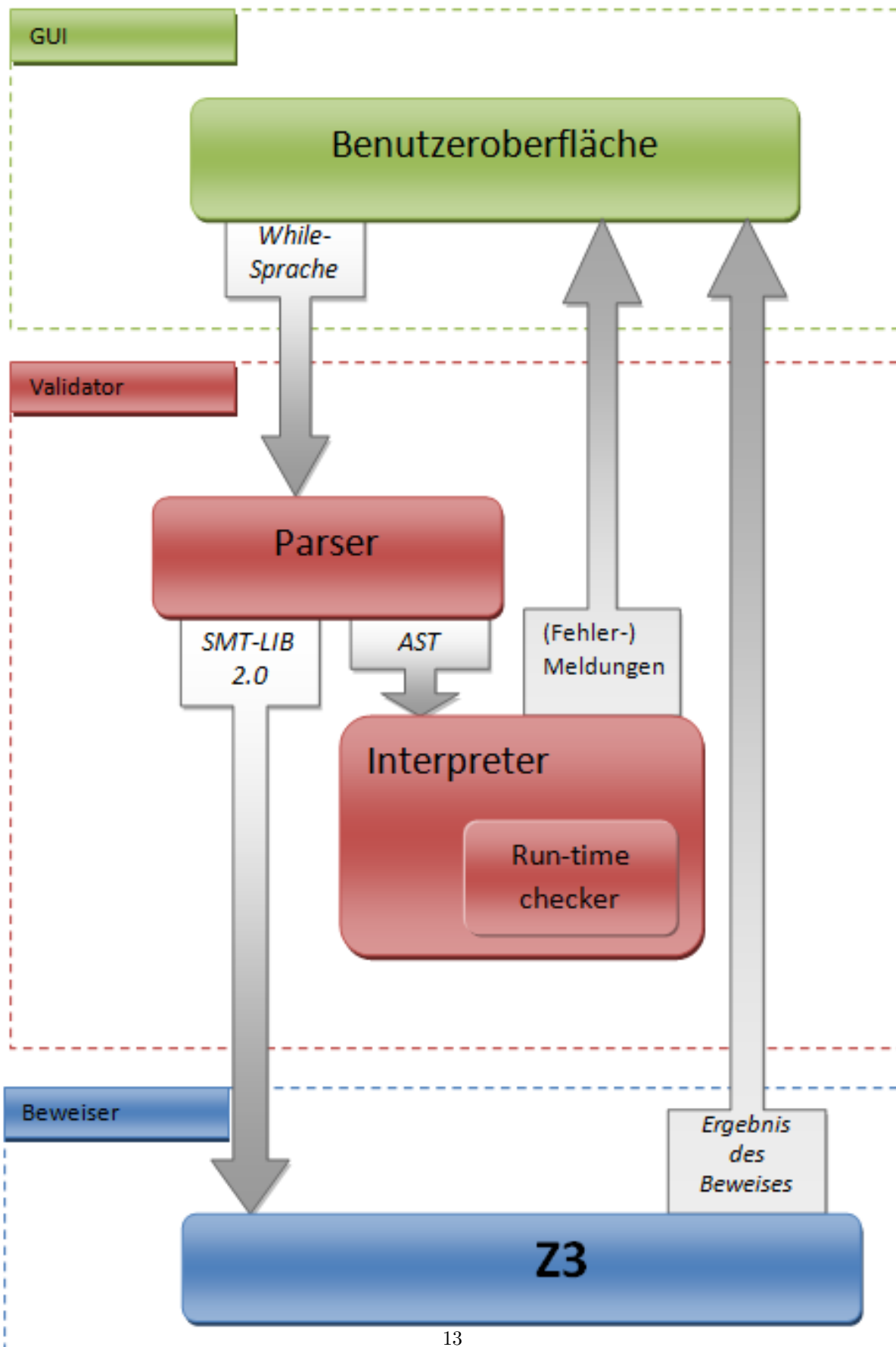
Eingabe eines Programms mit Deklaration und Initialisierung eines Arrays, fehlerhafter Arrayzugriff, Ausführung des Programms → Anzeige des Laufzeitfehlers

## 11 Systemmodelle

### 11.1 Komponentenübersicht

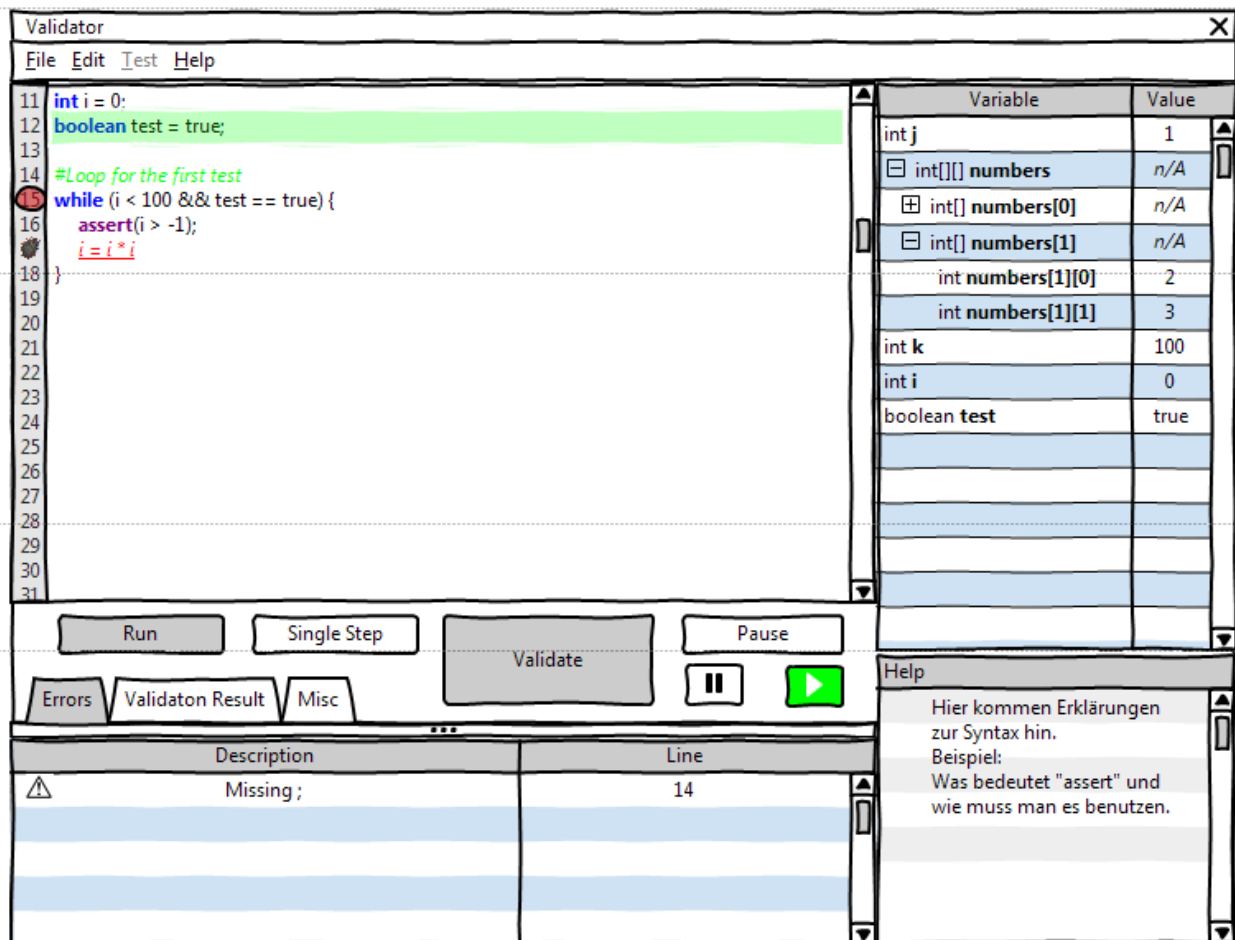


## 11.2 Workflow

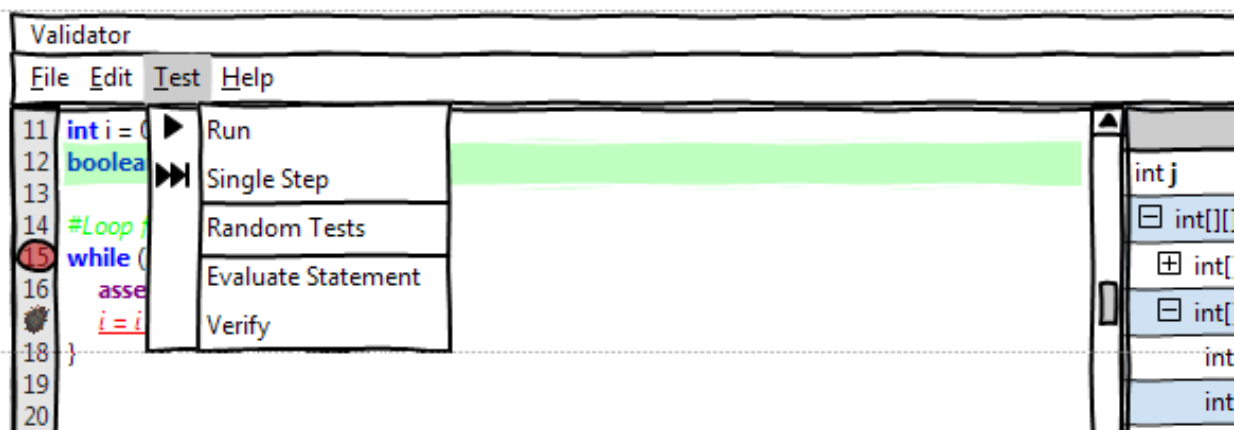
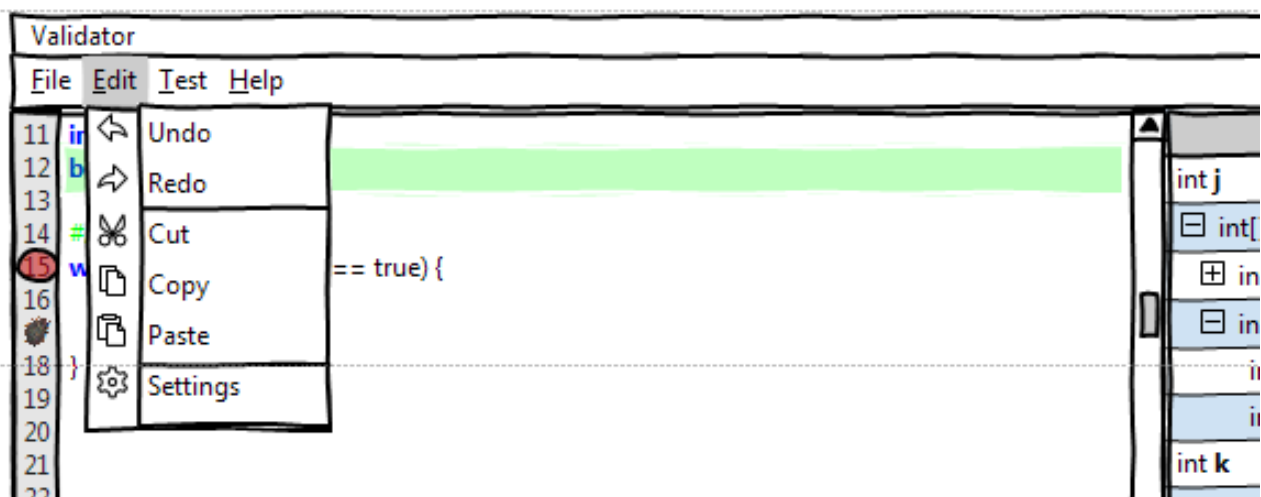
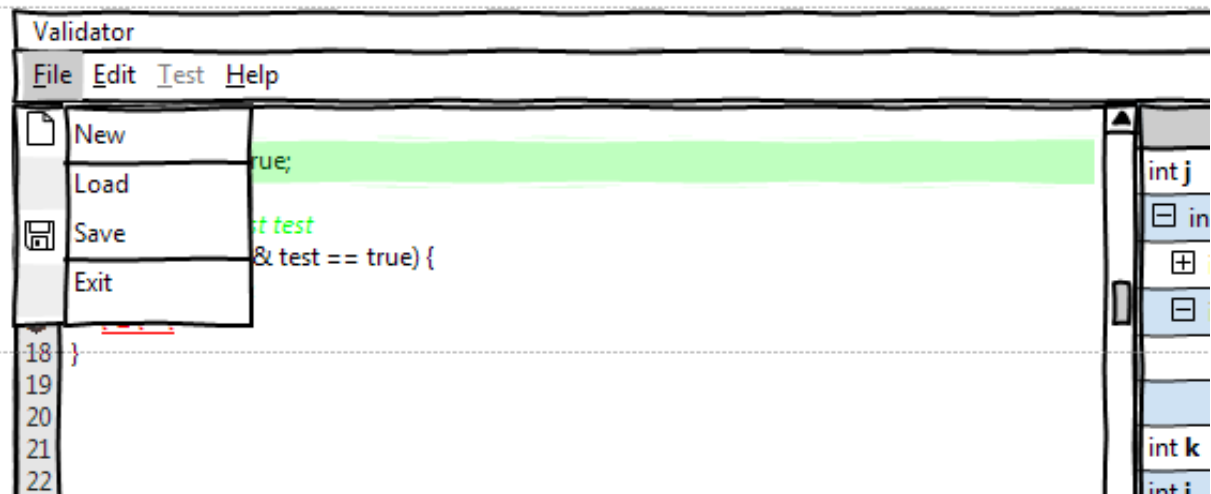


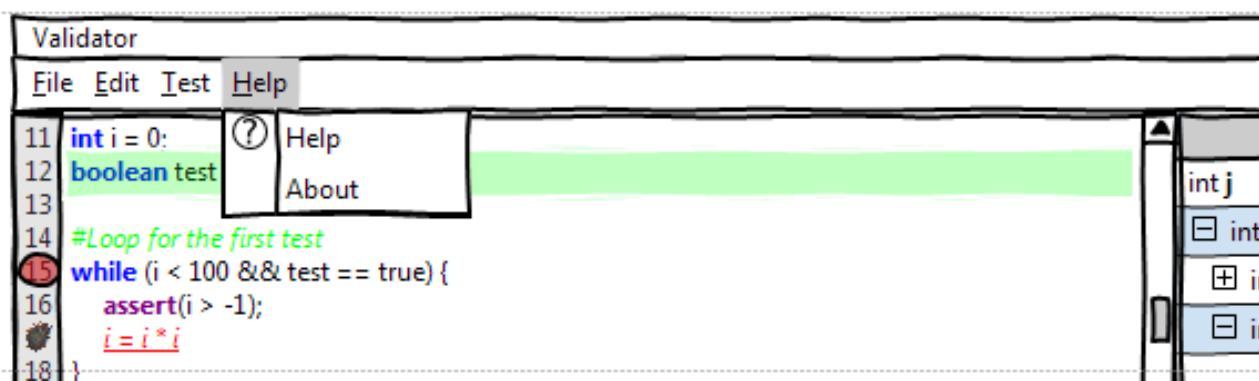
## 12 Benutzeroberfläche

### 12.1 Übersicht



### 12.2 Menüpunkte







## 13 Spezielle Anforderungen an die Entwicklungsumgebung

**Versionsverwaltung** Git (GitHub)

**Kommunikation** E-Mail, GitHub Wiki

**Dokumente** L<sup>A</sup>T<sub>E</sub>X 2<sub>ε</sub>

**Programmiersprache** Java

**UML-Diagramme** Dia, SD Edit (für Sequenzdiagramme)

**Entwicklungsumgebung** keine einheitliche, Ausschluss entwicklungsumgebungsspezifischer Dateien durch .gitignore

## 14 Zeit- und Ressourcenplanung

### 14.1 Projektplan

Komponente	Zeit
GUI	75 Stunden
Parser	75 Stunden
Interpreter	75 Stunden
Run-time Checker	100 Stunden
Typchecker	75 Stunden
Beweiseranbindung	100 Stunden

### 14.2 Phasenverantwortliche

Phase	Verantwortliche(r)
Pflichtenheft	Tobias
Entwurf	Adrian
Implementierung	Simon, Jan
Validierung	Lin
Abschlusspräsentation	Matthias

## 15 Ergänzungen

### 15.1 While-Sprache

#### 15.1.1 Variablentypen

**Ganzzahl** ganze Zahl beliebiger Genauigkeit.

**Boolean** Wahrheitswert, entweder wahr oder falsch.

**Array** eine Zusammenfassung mehrerer Variablen eines Variablentyps. Die maximale Anzahl möglicher Variablen in einem Array ist nach erster Festlegung konstant.

### 15.1.2 Sprachkonstrukte

Konstrukt	Bedeutung
sequentielle Komposition	Hintereinanderausführung von Anweisungen.
if-else	Ausführung bestimmter Anweisungen, falls eine Bedingung erfüllt ist. Ausführung anderer Anweisungen, falls die Bedingung nicht erfüllt ist.
while	Wiederholte Ausführung von bestimmten Anweisungen, wenn eine Bedingung erfüllt ist.
Methoden	Definition einer Folge von Anweisungen mit Parametern, die zu einem anderen Zeitpunkt im Programmablauf aufgerufen werden können, und einen Wert zurückgeben.
einzeilige Kommentare	Annotation, die keinen Einfluss auf die Quelltextverarbeitung haben.
Deklaration	Festlegung von Eigenschaften (Variablentyp, Bezeichner, ...) von Variablen und Methoden.

### 15.1.3 Operatoren

Operator	Eingabetypen	Ergebnistyp	Funktion
==	2 Variablen desselben Typs	Boolean	Prüft T1 und T2 auf Gleichheit, falls sie vom selben Variablentyp sind. Arrays sind gleich, wenn alle Elemente der tiefsten Ebene des Arrays gleich sind.
!=	2 Variablen desselben Typs	Boolean	Prüft T1 und T2 auf Ungleichheit, falls sie vom selben Variablentyp sind. Liefert denselben Wert wie die Negation von ==.
=	1 Variable und ein Ausdruck	?	Weist der Variable den Wert des Ausdrucks zu, wenn dieser Wert vom selben Variablentyp ist, wie die Variable.
<	2 Ganzzahlen, Ganzzahl1 und Ganzzahl2	Boolean	Prüft, ob Ganzzahl1 kleiner als Ganzzahl2 ist.
>	2 Ganzzahlen, Ganzzahl1 und Ganzzahl2	Boolean	Prüft, ob Ganzzahl1 größer als Ganzzahl2 ist.
<=	2 Ganzzahlen, Ganzzahl1 und Ganzzahl2	Boolean	Prüft, ob Ganzzahl1 kleiner oder gleich Ganzzahl2 ist.
>=	2 Ganzzahlen, Ganzzahl1 und Ganzzahl2	Boolean	Prüft, ob Ganzzahl1 größer oder gleich Ganzzahl2 ist.
+	1 Ganzzahl, Ganzzahl	Ganzzahl	Gibt Ganzzahl zurück.
-	1 Ganzzahl, Ganzzahl	Ganzzahl	Gibt -Ganzzahl zurück.
+	2 Ganzzahlen, Ganzzahl1 und Ganzzahl2	Ganzzahl	Addiert Ganzzahl1 und Ganzzahl2.
-	2 Ganzzahlen, Ganzzahl1 und Ganzzahl2	Ganzzahl	Subtrahiert Ganzzahl2 von Ganzzahl1.
*	2 Ganzzahlen, Ganzzahl1 und Ganzzahl2	Ganzzahl	Multipliziert Ganzzahl1 mit Ganzzahl2.
/	2 Ganzzahlen, Ganzzahl1 und Ganzzahl2	Ganzzahl	Dividiert Ganzzahl1 durch Ganzzahl2. Das Ergebnis wird zur Null hin gerundet.
%	2 Ganzzahlen, Ganzzahl1 und Ganzzahl2	Ganzzahl	Liefert den Rest der Ganzzahldivision von Ganzzahl1 durch Ganzzahl2.
!	1 Boolean, Boolean	Boolean	Negiert den Wert von Boolean.
&	2 Boolean, Boolean1 und Boolean2	Boolean	Liefert die Konjunktion von Boolean1 und Boolean2.
	2 Boolean, Boolean1 und Boolean2	Boolean	Liefert die Disjunktion von Boolean1 und Boolean2.

#### 15.1.4 Sonstige Spracheigenschaften

- Lexical Scoping
- Kurzauswertung von logischen Operatoren

### 15.2 Annotationssprache

Die Annotationssprache besteht aus Ausdrücken der While-Sprache mit Quantoren und zusätzlich den folgenden Konstrukten:

<b>require</b>	Stellt eine Vorbedingung vor Schleifen oder Methoden dar.
<b>ensure</b>	Stellt eine Nachbedingung nach Schleifen oder Methoden dar.
<b>assert</b>	Stellt eine Bedingung an der aktuellen Stelle im Programmablauf dar.
<b>assume</b>	Stellt eine globale, als wahr angenommene Aussage dar.
<b>invariant</b>	Stellt eine Invariante einer Schleife dar.
<b>thereis</b>	Existenzquantor $\exists$
<b>forall</b>	Allquantor $\forall$

## 16 Glossar

**Anweisung (Programm)** Eine syntaktisch korrekte Vorschrift, die bei der Abarbeitung des Programms auszuführen ist.

**Beweiser** Ein Programm, das versucht die Gültigkeit eines Theorems zu überprüfen. Ein Beispiel für ein solches Programm ist  $\rightarrow Z3$ .

**Kurzauswertung** Die Auswertung eines Ausdrucks wird abgebrochen, sobald der Wert des Ausdrucks bekannt ist.

**Lexical Scoping** Auf eine lokale Variable kann nur in ihrem lexikalischen Sichtbarkeitsbereich Bezug genommen werden.

**sequentielle Komposition** Hintereinanderausführung von mehreren Anweisungen.

**SMT-LIB 2.0** Ein Standardformat zur Beschreibung von Satisfiability Modulo Theories Problemen, d.h. Formeln der Prädikatenlogik erster Stufe.

**Z3** Ein  $\rightarrow$ Beweiser von Microsoft.

# Funktionale Anforderung

## Beweiser

- Beweisvorgang durchführen, 9

## GUI

- Abbrechen des Programmablaufs, 8
- Anhalten des Programmablaufs, 8
- Anzeige der Beweiseinstellungen, 8
- Anzeige des Ergebnis des Beweisers, 8
- Anzeige des Programmzustands, 8
- Anzeige von fehlgeschlagenen Annotationsbedingungen, 8
- Anzeige von Quelltext, 7
- Anzeigen von Laufzeitfehlern, 8
- Anzeigen von Syntaxfehlern, 7
- Anzeigen von Typfehlern, 7
- Eingabe von benutzerdefinierten Ausdrücken, 8
- Laden der Beweiseinstellungen aus einer Datei, 8
- Laden eines Programms aus einer Datei, 7
- Setzen und Entfernen von Breakpoints, 8
- Speichern der Beweiseinstellungen in eine Datei, 8
- Speichern des Quelltextes in eine Datei, 7
- Starten des Beweisvorgangs, 8
- Syntax Highlighting, 7
- Veränderung des Quelltextes, 7

## Interpreter

- Anhalten des Programmablaufs bei einem Fehler, 9
- Ausführen eines Programms, 9
- Auswertung der Ausdrücke der Annotationsprache, 9
- Auswertung von benutzerdefinierten Ausdrücken, 9
- Erkennen von Syntaxfehlern, 9
- Erkennen von Typfehlern, 9