

# Pflichtenheft

Simon Bischof, Jan Haag, Adrian Herrmann, Lin Jin, Tobias Schlumberger, Matthias Schnetz

Praxis der Softwareentwicklung

Projekt 4:

Automatisches Prüfen der Korrektheit von Programmen

Gruppe 1



WS 2011/2012

# 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>7</b>
5.1	Übersicht . . . . .	7
5.2	Details . . . . .	7
5.2.1	GUI . . . . .	7
5.2.2	Parser / Interpreter . . . . .	9
5.2.3	Beweiserschnittstelle . . . . .	10
<b>6</b>	<b>Produktdaten</b>	<b>10</b>
<b>7</b>	<b>Produktleistungen</b>	<b>10</b>
<b>8</b>	<b>Weitere nichtfunktionale Anforderungen</b>	<b>10</b>
<b>9</b>	<b>Qualitätsanforderungen</b>	<b>11</b>
<b>10</b>	<b>Globale Testfälle und Testszenarien</b>	<b>11</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>14</b>
12.1	Übersicht . . . . .	14
12.2	Menüpunkte . . . . .	14
12.3	Eingabe von benutzerdefinierten Ausdrücken . . . . .	15
<b>13</b>	<b>Spezielle Anforderungen an die Entwicklungsumgebung</b>	<b>16</b>
<b>14</b>	<b>Zeit- und Ressourcenplanung</b>	<b>17</b>
14.1	Phasenplan . . . . .	17
14.2	Projektplan . . . . .	17
14.3	Phasenverantwortliche . . . . .	17
<b>15</b>	<b>Ergänzungen</b>	<b>17</b>
15.1	While-Sprache . . . . .	17
15.1.1	Variablentypen . . . . .	17
15.1.2	Sprachkonstrukte . . . . .	18
15.1.3	Operatoren . . . . .	18
15.1.4	Sonstige Spracheigenschaften . . . . .	19
15.2	Annotationssprache . . . . .	19
<b>16</b>	<b>Glossar</b>	<b>19</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**

- Einstellungen bezüglich Schriftart, Schriftfarbe und Schriftgröße
- Projektmanagement

## **2.3 Verifikation**

### **2.3.1 Musskriterien**

- formale Verifikation eines Programms mit Hilfe eines Beweisers

## **2.4 While-Sprache**

### **2.4.1 Musskriterien**

- Zuweisungen, Bedingte Anweisungen und Schleifen
- Turing-Vollständigkeit

### **2.4.2 Kannkriterien**

- Unterstützung von Methodenaufrufen

### **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 und Ausführen 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 Übersicht

GUI	
/F1010/	Anzeige von Quelltext
/F1020/	Syntax Highlighting
/F1030/	Veränderung des Quelltextes
/F1040/	Speichern des Quelltextes in eine Datei
/F1050/	Laden eines Programms aus einer Datei
/F1060/	Anzeigen von Syntaxfehlern
/F1070/	Anzeigen von Typfehlern
/F1080/	Anzeigen von Laufzeitfehlern
/F1090/	Setzen und Entfernen von Breakpoints
/F1100/	Anzeige des Programmszustands
/F1110/	Anhalten des Programmablaufs
/F1120/	Fortsetzen des Programmablaufs
/F1130/	Abbrechen des Programmablaufs
/F1140/	Eingabe verschiedener Eingangskonfigurationen für Eingabeparameter
/F1150/	Anzeige von fehlgeschlagenen Annotationsbedingungen
/F1160/	Eingabe von benutzerdefinierten Ausdrücken
/F1170/	Anzeige der Beweiseinstellungen
/F1180/	Laden der Beweiseinstellungen aus einer Datei
/F1190/	Speichern der Beweiseinstellungen in eine Datei
/F1200/	Starten des Beweisvorgangs
/F1210/	Anzeige des Ergebnis des Beweisers
Parser / Interpreter	
/F2010/	Parsen des Quelltextes und Erzeugung eines AST
/F2020/	Erkennen von Syntaxfehlern
/F2030/	Erkennen von Typfehlern
/F2040/	Ausführen eines Programms
/F2050/	Anhalten der Ausführung eines Programms
/F2060/	Fortsetzung der Ausführung eines Programms
/F2070/	Abbrechen der Ausführung eines Programms
/F2080/	Ausführen einer einzelner Anweisung (Single Step)
/F2090/	Anhalten des Programmablaufs bei einem Breakpoint
/F2100/	Ausführung eines Programms mit verschiedenen Belegungen für Eingabeparameter
/F2110/	Anhalten des Programmablaufs bei einem Fehler
/F2120/	Auswertung von benutzerdefinierten Ausdrücken
/F2130/	Auswertung der Ausdrücke der Annotationssprache
Beweiserschnittstelle	
/F3010/	Quellcode in SMT-LIB 2.0 Format umwandeln
/F3020/	Übergabe der benutzerdefinierten Einstellungen
/F3030/	Beweiservorgang durchführen

### 5.2 Details

#### 5.2.1 GUI

##### /F1010/ Anzeige von Quelltext

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

##### /F1020/ Syntax Highlighting

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

- /F1030/ Veränderung des Quelltextes  
Der Benutzer kann den angezeigten Quelltext verändern oder einen neuen Quelltext eingeben.
- /F1040/ Speichern des Quelltextes in eine Datei  
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.
- /F1050/ Laden eines Programms aus einer Datei  
Der Benutzer kann ein Quellcode eines Programms aus einer Datei laden. Hierbei gibt der Benutzer die zu ladende Datei über eine grafische Oberfläche an.
- /F1060/ Anzeigen von Syntaxfehlern  
Wenn der Interpreter Syntaxfehler erkannt hat ( $\rightarrow$ /F2020/), wird dies dem Benutzer angezeigt, wenn er die Programmausführung oder die Beweisführung anfordert.
- /F1070/ Anzeigen von Typfehlern  
Wenn der Interpreter Typfehler erkannt hat ( $\rightarrow$ /F2030/), wird dies dem Benutzer angezeigt, wenn er die Programmausführung oder die Beweisführung anfordert.
- /F1080/ Anzeigen von Laufzeitfehlern  
Wenn bei der Ausführung eines Programmes durch den Interpreter ein Fehler auftritt ( $\rightarrow$ /F2050/), wird dies dem Benutzer angezeigt.
- /F1090/ Setzen und Entfernen von Breakpoints  
Der Benutzer kann Breakpoints zur Pausierung eines Programmdurchlaufs an Stellen des angezeigten Quellcodes setzen und wieder entfernen.
- /F1100/ Anzeige des Programmzustands  
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.
- /F1110/ Anhalten des Programmablaufs  
Der Benutzer kann die Ausführung eines Programms anhalten. Dabei wird der aktuelle Programmzustand beibehalten.
- /F1120/ Fortsetzen des Programmablaufs  
Hat der Benutzer das Programm angehalten ( $\rightarrow$ /F1110/), so kann er die Ausführung des Programms wieder aufnehmen.
- /F1130/ Abbrechen des Programmablaufs  
Der Benutzer kann die Ausführung eines Programms abbrechen. Dabei wird der Programmzustand verworfen.
- /F1140/ Eingabe verschiedener Eingangskonfigurationen für Eingabeparameter  
Der Benutzer kann einen Zahlenbereich sowie eine Anzahl an Konfigurationen angeben. Das Programm wird anschließend mit diesen Konfigurationen automatisch mehrfach ausgeführt ( $\rightarrow$ /F2070/). Bei fehlerfreier Ausführung wird dem Benutzer eine Auswertung der Durchläufe angezeigt, tritt bei einer Ausführung ein Fehler auf, so wird die Ausführung angehalten.
- /F1150/ Anzeige von fehlgeschlagenen Annotationsbedingungen  
Meldet der Interpreter eine fehlgeschlagene Annotationsbedingung ( $\rightarrow$ /F2060/), so wird dies dem Benutzer angezeigt.
- /F1160/ Eingabe von benutzerdefinierten Ausdrücken  
Der Benutzer kann während der angehaltenen Ausführung eines Programms Ausdrücke eingeben, die vom Interpreter ausgewertet werden und deren Ergebnis dem Benutzer angezeigt wird.
- /F1170/ Anzeige der Beweiseinstellungen  
Der Benutzer kann die aktuellen Einstellungen ( $\rightarrow$ /D10/) für die Ausführung des Beweisers ( $\rightarrow$ /F1180/) anzeigen lassen. Diese werden vor der Anzeige aus einer Datei geladen ( $\rightarrow$ /F1160/) und bei einer Veränderung wieder in eine Datei geschrieben ( $\rightarrow$ /F1170/).



- /F1180/ Laden der Beweiseinstellungen aus einer Datei  
Die Einstellungen für die Ausführung des Beweisers ( $\rightarrow$ /D10/) werden automatisch aus einer Datei gelesen.
- /F1190/ Speichern der Beweiseinstellungen in eine Datei  
Die Einstellungen ( $\rightarrow$ /D10/) werden in eine Datei gespeichert.
- /F1200/ Starten des Beweisvorgangs  
Der Benutzer kann den Beweisvorgang mit Hilfe des Beweisers unter den eingegebenen Einstellungen starten.
- /F1210/ Anzeige des Ergebnis des Beweisers  
Nach dem Durchlauf des Beweisers wird dem Benutzer das Ergebnis des Beweisvorgangs angezeigt.

### 5.2.2 Parser / Interpreter

- /F2010/ Parsen des Quelltextes und Erzeugung eines AST  
Der Parser analysiert den eingegebenen Quelltext und erzeugt daraus einen abstrakten Syntaxbaum zur Weiterverwendung für andere Programmkomponenten.
- /F2020/ Erkennen von Syntaxfehlern  
Der Interpreter erkennt Syntaxfehler des eingegebenen Quelltextes.
- /F2030/ Erkennen von Typfehlern  
Der Interpreter erkennt Typfehler des eingegebenen Quelltextes.
- /F2040/ Ausführen eines Programms  
Ist der Quellcode frei von Syntaxfehlern ( $\rightarrow$ /F2020/) und von Typfehlern ( $\rightarrow$ /F2030/) Anweisungen im aktuell angezeigten Quelltext werden sequentiell ausgeführt.
- /F2050/ Anhalten der Ausführung eines Programms  
Die laufende Ausführung eines Programms kann angehalten werden ( $\rightarrow$ /F1110/), wobei keine Anweisungen mehr ausgeführt werden und die Ausführung im aktuellen Zustand stehen bleibt.
- /F2060/ Fortsetzung der Ausführung eines Programms  
Wurde ein Programm angehalten ( $\rightarrow$ /F2050/), so kann die Ausführung mit dem Zustand zum Zeitpunkt der Pausierung fortgesetzt werden.
- /F2070/ Abbrechen der Ausführung eines Programms  
Die Ausführung eines Programms kann angehalten werden ( $\rightarrow$ /F1120/), wobei der aktuelle Programmzustand verworfen wird.
- /F2080/ Ausführen einer einzelner Anweisung (Single Step)  
Ist das Programm gestartet und angehalten, so wird die nächste Anweisung ausgeführt. Ist das Programm nicht gestartet, so wird es gestartet ( $\rightarrow$ /F2040/).
- /F2090/ Anhalten des Programmablaufs bei einem Breakpoint  
Wird bei der Ausführung eines Programms ( $\rightarrow$ /F2040/) ein Breakpoint ( $\rightarrow$ /F1090/) erreicht, hält die Ausführung an und der aktuelle Zustand wird zurückgegeben.
- /F2100/ Ausführung eines Programms mit verschiedenen Belegungen für Eingabeparameter  
Das Programm wird mehrmals nacheinander mit den Werten für die Eingabeparameter ausgeführt. Bei diesen Ausführungen werden Breakpoints ( $\rightarrow$ /F1090/) ignoriert. Tritt bei einer Ausführung ein Fehler auf, so wird die weitere Ausführung angehalten. Tritt kein Fehler auf, so wird eine Auswertung der Ausführungen erstellt.
- /F2110/ Anhalten des Programmablaufs bei einem Fehler  
Tritt ein Fehler zur Laufzeit des Programms auf, wird die Ausführung abgebrochen und der Fehler dem Benutzer angezeigt.
- /F2120/ Auswertung von benutzerdefinierten Ausdrücken  
Der Benutzer kann Ausdrücke über den aktuellen Programmzustand auswerten lassen ( $\rightarrow$ /F1140/).

/F2130/ Auswertung der Ausdrücke der Annotationssprache

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

### 5.2.3 Beweiserschnittstelle

/F3010/ Quellcode in SMT-LIB 2.0 Format umwandeln

Das Produkt wandelt den aktuellen Quellcode zur Eingabe in den Beweiser in das SMT-LIB 2.0 Format um. Dies geschieht implizit durch das Starten des Beweisvorgangs in der grafischen Oberfläche ( $\rightarrow$ /F1180/).

/F3020/ Übergabe der benutzerdefinierten Einstellungen

Die vom Benutzer eingegebenen Einstellungen für den Beweisvorgang ( $\rightarrow$ /F1170/) werden dem Beweiser übergeben.

/F3030/ Beweiservorgang durchführen

Nach der Umwandlung des Quellcodes in das Eingabeformat des Beweisers ( $\rightarrow$ /F3010/) wird der Beweisvorgang für das aktuelle Programm durchgeführt und nach Abschluss das Ergebnis zurückgemeldet.

## 6 Produktdaten

Es sind folgende Daten des Benutzers zu Speichern:

/D10/ Benutzereinstellungen

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

/D20/ Zuletzt geöffnetes Verzeichnis

- Das zuletzt durch /F1050/ 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. Die Benutzung von Z3 als Beweiser schränkt die Plattformunabhängigkeit auf Windows NT 4.0 und höher und Betriebssysteme mit Wine ein. Durch Anbindung eines anderen Beweisers reicht unabhängig von den Anforderungen des neuen Beweisers eine aktuelle Java Runtime Environment.

## 9 Qualitätsanforderungen

- Programmiersprache muss erweiterbar sein
- mögliche Anbindung anderer Beweiser, die das SMT-LIB 2.0 Format unterstützen
- unter (durch benutzerdefinierte Einstellungen vorgegebene,  $\rightarrow$ /F1150/) 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:

/T10/ Eingabe des leeren Programms  $\rightarrow$  keine Syntaxfehler

/T20/ Eingabe des leeren Programms ohne Annotationen, Beweisvorgang starten  $\rightarrow$  Verifikation durch den Beweiser

/T30/ Eingabe des leeren Programms, Annotation **ensure false**, Programmausführung  $\rightarrow$  Annotationsbedingung schlägt fehl

/T40/ Eingabe des leeren Programms, Annotation **ensure true**, Programmausführung  $\rightarrow$  Annotationsbedingung schlägt nicht fehl

/T50/ Eingabe eines beliebigen Programms, Annotation **require false**, Programmausführung  $\rightarrow$  Annotationsbedingung schlägt fehl, Programmausführung wird abgebrochen

/T60/ Eingabe eines Programms mit Zuweisung  $x = 1$ , Annotation **require  $x > 0$** , Programmausführung  $\rightarrow$  Annotationsbedingung schlägt nicht fehl

/T70/ Eingabe eines Programms mit Syntaxfehler (z.B. fehlerhafte Klammerung, fehlendes Semikolon, Methodename ist Keyword)  $\rightarrow$  Anzeige des Syntaxfehlers

/T80/ Eingabe eines Programms mit while-Schleife oder if-Konstrukt, deren Bedingung kein boolean ist  $\rightarrow$  Anzeige des Typfehlers

/T90/ Eingabe eines Programms mit einem Operator, der nur auf Ganzzahlen definiert ist und auf einen boolean angewendet wird  $\rightarrow$  Anzeige des Typfehlers

/T100/ Eingabe eines Programms mit einem Operator, der nur auf booleans definiert ist und auf Ganzzahlen angewendet wird  $\rightarrow$  Anzeige des Typfehlers

/T110/ Eingabe eines Programms mit einer Zuweisung eines boolean an eine Ganzzahl  $\rightarrow$  Anzeige des Typfehlers

/T120/ Eingabe eines Programms mit einer Zuweisung einer Ganzzahl an einen boolean  $\rightarrow$  Anzeige des Typfehlers

/T130/ Eingabe eines Programms mit einer Zuweisung einer Ganzzahl oder eines boolean an ein Array  $\rightarrow$  Anzeige des Typfehlers

/T140/ Eingabe eines Programms mit einem Vergleichsoperator, der auf zwei unterschiedliche Typen ausgeführt wird  $\rightarrow$  Anzeige des Typfehlers

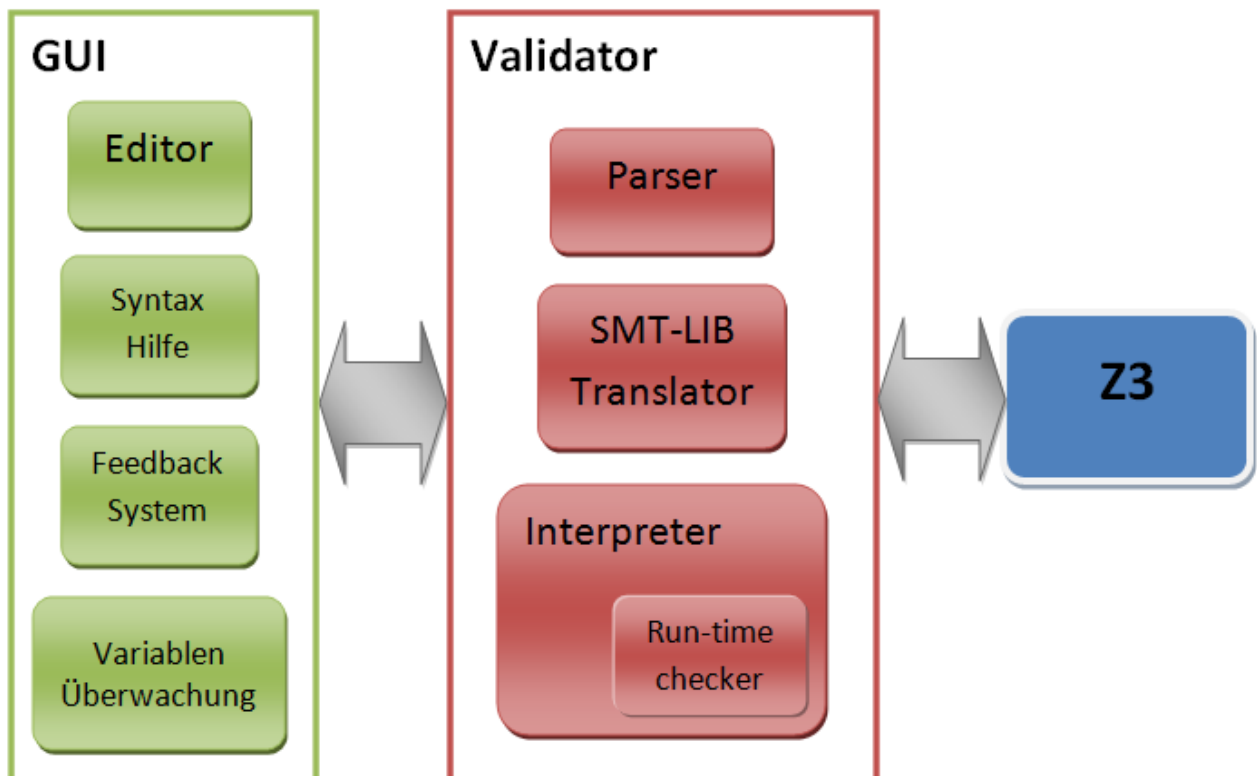
/T150/ Eingabe eines Programms mit Zuweisung  $y = 2x$ , Annotation **ensure  $y == 2x$** , Beweisvorgang starten  $\rightarrow$  Verifizierung durch den Beweiser

/T160/ Eingabe eines Programms mit Zuweisung  $y = 2x$ , Annotation **ensure  $y \% 2 == 0$** , Beweisvorgang starten  $\rightarrow$  Verifizierung durch den Beweiser

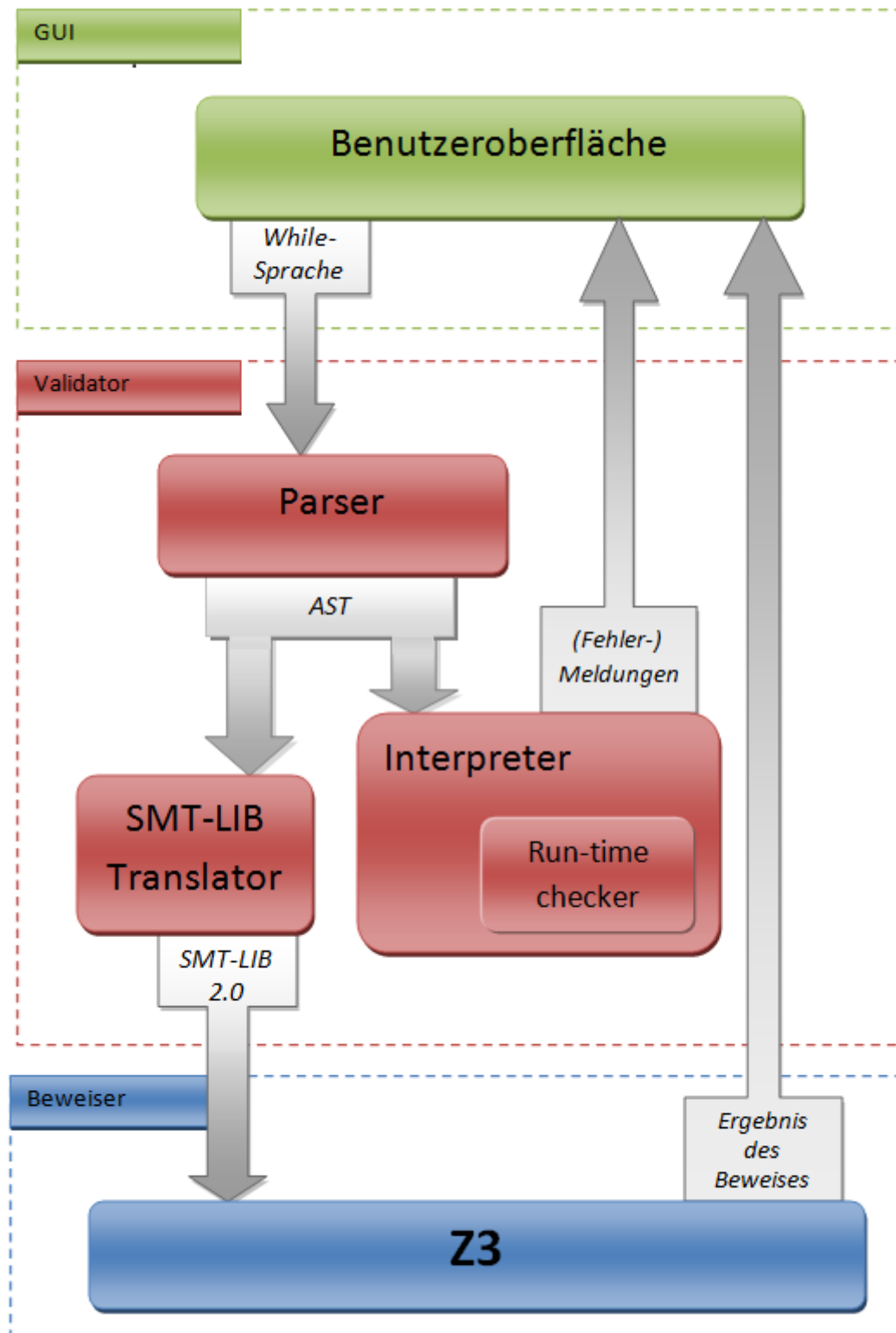
- /T170/ Eingabe eines längeren Programms ohne Schleifen und ohne if-Konstrukt, Eingabe von Annotationen, die in jedem Fall wahr sind, Beweisvorgang starten → Verifizierung durch den Beweiser
- /T180/ Eingabe eines längeren Programms ohne Schleifen und mit if-Konstrukt, Eingabe von Annotationen, die in jedem Fall wahr sind, Beweisvorgang starten → Verifizierung durch den Beweiser
- /T190/ Eingabe eines längeren Programms mit Schleifen (z.B. russische Multiplikation, Maximum der Zahlen in einem Array), Eingabe von Annotationen, die in jedem Fall wahr sind, Beweisvorgang starten → Verifizierung durch den Beweiser
- /T200/ Eingabe eines Programms mit Annotation **ensure**  $x \neq x$ , Beweisvorgang starten → Falsifikation durch den Beweiser
- /T210/ 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
- /T220/ 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
- /T230/ Eingabe eines Programms mit Division durch 0, Ausführung des Programms → Anzeige des Laufzeitfehlers
- /T240/ Eingabe eines Programms mit Deklaration und Initialisierung eines Arrays, fehlerhafter Arrayzugriff, Ausführung des Programms → Anzeige des Laufzeitfehlers

## 11 Systemmodelle

### 11.1 Komponentenübersicht

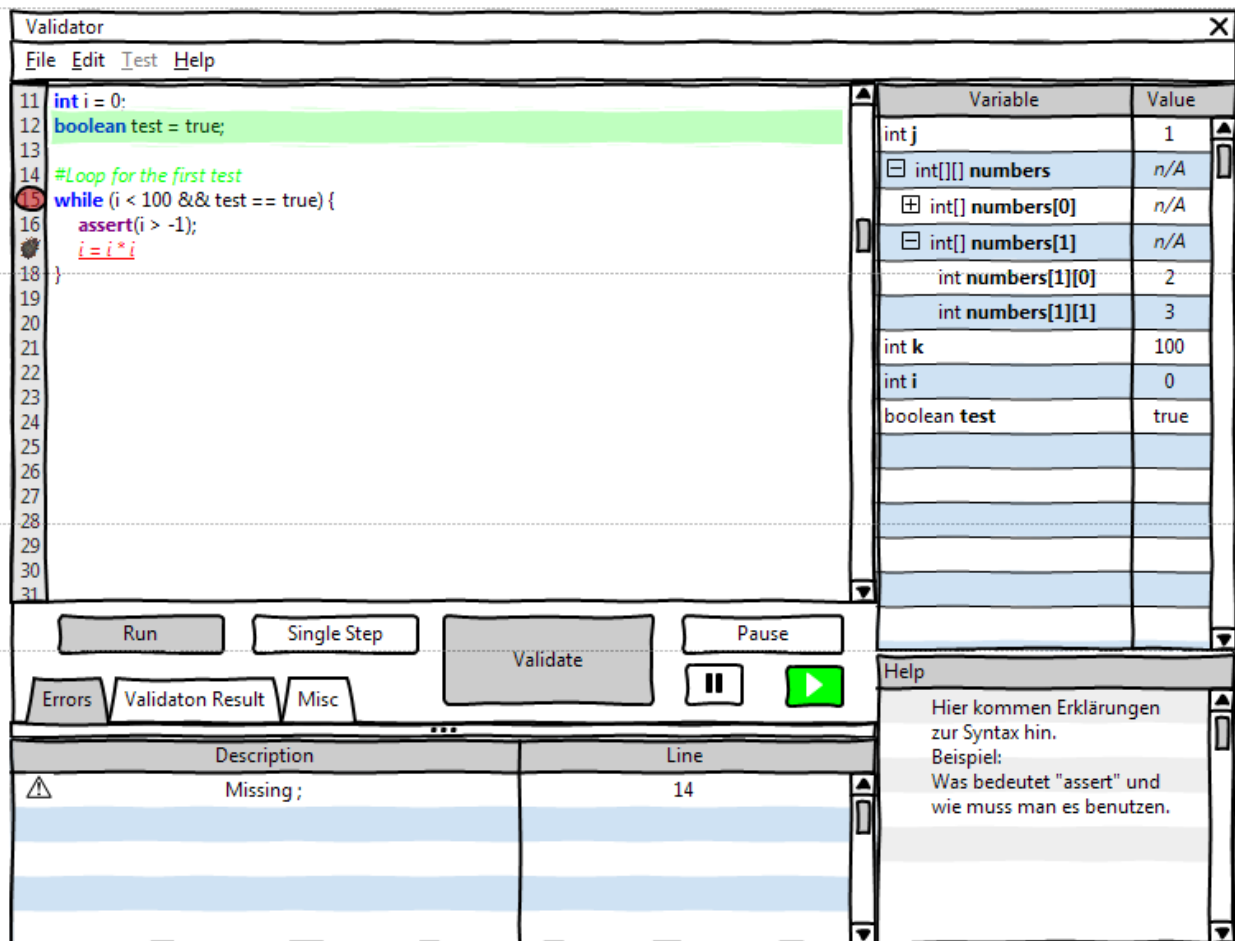


## 11.2 Workflow

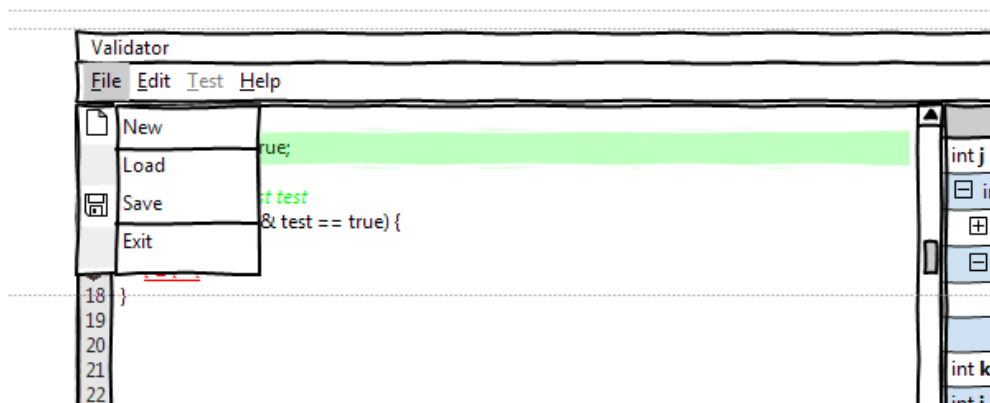


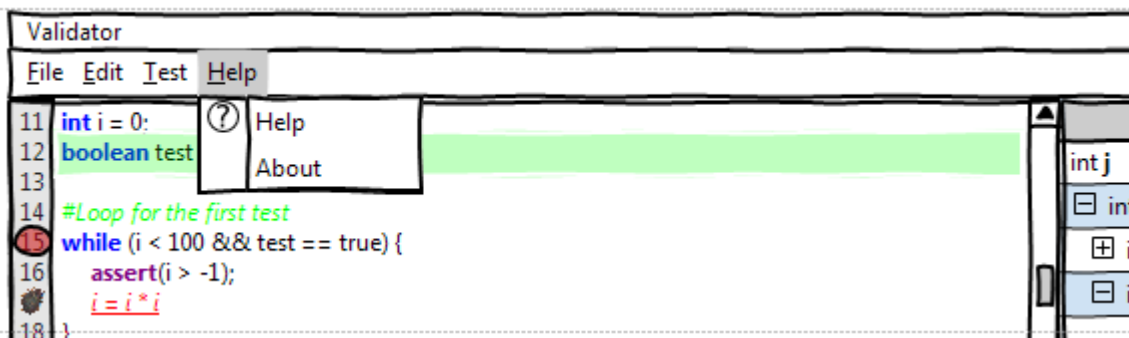
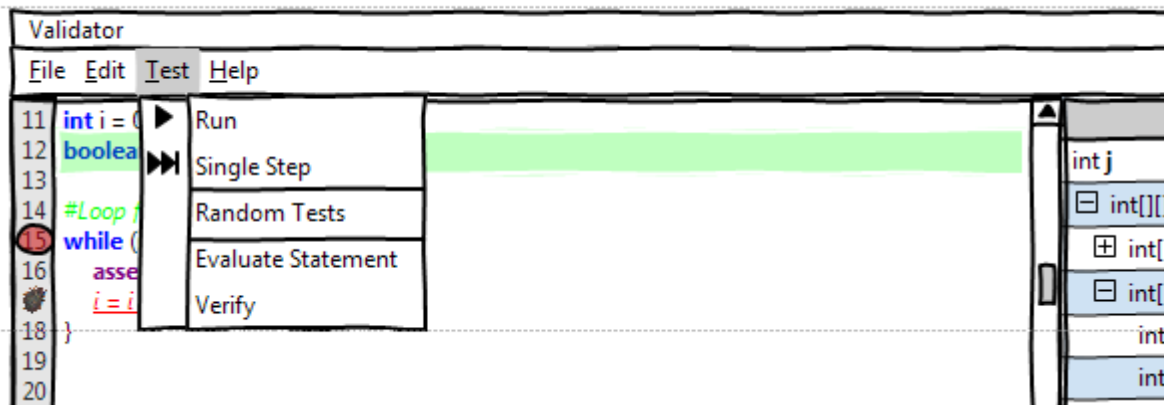
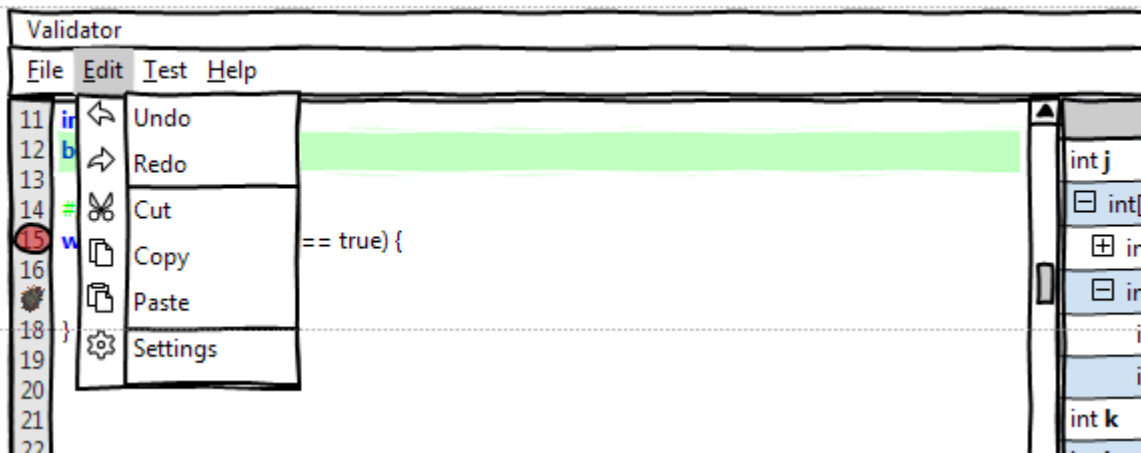
## 12 Benutzeroberfläche

### 12.1 Übersicht

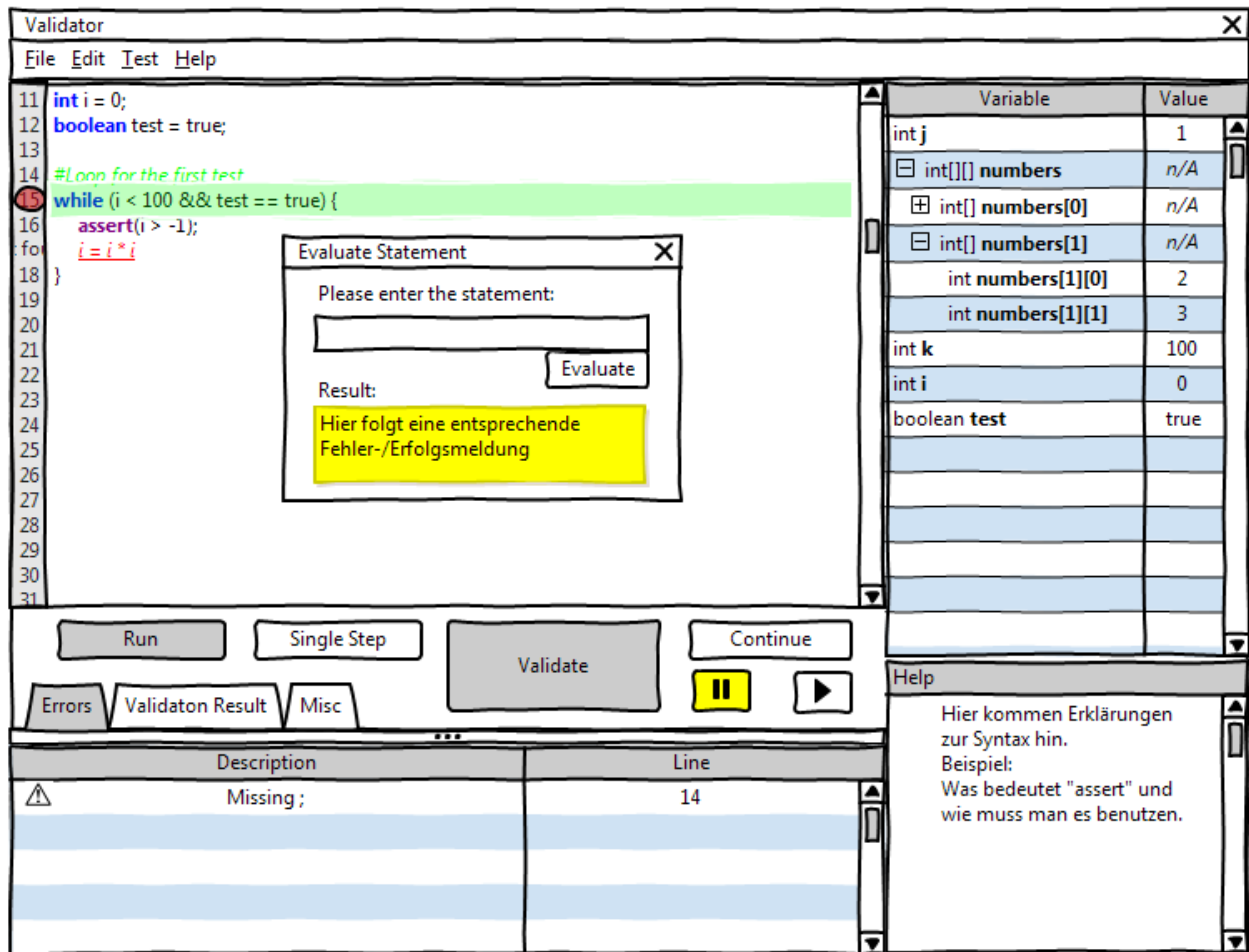


### 12.2 Menüpunkte





### 12.3 Eingabe von benutzerdefinierten Ausdrücken



## 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

**Office-Paket** Microsoft Office

**Bildbearbeitung** Paint.NET



## 14 Zeit- und Ressourcenplanung

### 14.1 Phasenplan

Phase	Zeit pro Person
Pflichtenheft	30 Stunden
Entwurf	60 Stunden
Implementierung	75 Stunden
Validierung	45 Stunden
Abschlusspräsentation	30 Stunden

### 14.2 Projektplan

Komponente	Zeit
GUI	70 Stunden
Parser	50 Stunden
Interpreter	80 Stunden
Run-time Checker	30 Stunden
Typchecker	50 Stunden
Beweiseranbindung	75 Stunden

### 14.3 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. Ein Array kann mehrdimensional sein, dann ist die Größe jeder Dimension 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 als wahr angenommene Aussage dar, die ab dem Punkt der Programmausführung gilt, an der die Annotation steht.
<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.

**AST** Abstract Syntax Tree, eine Datenstruktur zur Darstellung der syntaktischen Struktur eines Programmcodes.

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

**Breakpoint** Ein Haltepunkt im Programm, bei der die Ausführung eines Programms angehalten wird, sobald diese Stelle erreicht wird.

**GUI** Graphical User Interface, grafische Benutzeroberfläche.

**Interpreter** Ein Programm, das Quellcode ausführen kann.

**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.

**Parser** Ein Programm, das einen Quelltext in eine für die Weiterverarbeitung günstiges Format (z.B.  $\rightarrow$ AST) umwandelt.

**Prädikatenlogik** Ein logisches System, das es ermöglicht Argumente zu formalisieren und auf ihre Gültigkeit zu prüfen.

**Run-time checker** Programmkomponente, die Bedingungen zur Laufzeit eines Programms überprüft.

**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.

**syntactic sugar** Syntaxelemente einer Programmiersprache, die nur der Komfortabilität dienen. Diese Syntaxelemente können aus der Sprache entfernt werden, ohne die Funktionalität einzuschränken.

**While-Sprache** Eine Programmiersprache, die sich durch einfache und wenige Kontrollstrukturen auszeichnet.

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