Institut für Softwaretechnologie

Universität Stuttgart
Universitätsstraße 38
D–70569 Stuttgart

Diplomarbeit Nr. 3388

Dynamische Pruefung von Spreadsheets

Manuel Lemcke

Studiengang:	Softwaretechnik
Prüfer/in:	Prof. Dr. Jochen Ludewig
Betreuer/in:	M. Sc. Daniel Kulesz
Beginn am:	27. August 2012
Beendet am:	26. Februar 2013
CR-Nummer:	H.4.1, D.2.5

Kurzfassung

Spreadsheets sind weit verbreitet. Gleichzeitig weisen sie laut verschiedenen Studien Fehlerquoten um die 90% auf. Aufgrund dessen wurde SIF (Spreadsheet Inspection Framework) geschaffen. Auf Basis dieses Rahmenwerks können Prüfzentren für Spreadsheets entwickelt werden. SIF wird im Rahmen dieser Arbeit um dynamische Prüfungen erweitert.

Inhaltsverzeichnis

Abbildungsverzeichnis

Tabellenverzeichnis

Verzeichnis der Listings

1. Einleitung

1.1. Motivation

Spreadsheets sind in Unternehmen und Verwaltung in hohem Maße im Einsatz. Sie werden sowohl für Tätigkeiten wie die Erstellung von Finanzberichten, als auch zur Entscheidungsfindung verwendet [Pano8] und erfreuen sich immer größerer Beliebtheit: 1989 gaben 10% der amerikanischen Endanwender an Arbeitsplätzen an, Spreadsheets zu verwenden. Bis 2001 stieg der Anteil auf 60% [SSM05]. Verschiedene Studien haben jedoch ergeben, dass 88% aller Spreadsheets fehlerbehaftet sind [Pano8]. Deswegen wird versucht Maßnahmen des Software-Qualitätsmanagements auf die Spreadsheetdomäne zu übertragen. Einer dieser Versuche ist SIF, das Spreadsheet Inspection Framework. Dieses in Java entwickelte Open-Source Framework kann als Basis für Anwendungen zur Prüfung von Spreadsheets dienen. Bisher unterstützt es lediglich statische Prüfungen. Diese allein können jedoch nicht alle Fehlerarten erkennen.

1.2. Ziel

Ziel dieser Arbeit ist es, eine Erweiterung um dynamische Prüfungen für das Java Rahmenwerk SIF zu konzipieren und zu implementieren. Außerdem soll eine Möglichkeit geschaffen werden, Ein- und Ausgabezellen zu erkennen. Dabei sollen soweit möglich bestehende Konzepte und Paradigmen von SIF beibehalten werden. Es sollen sowohl Standardimplementierungen für dynamische Vorschriften geschaffen werden, als auch die Möglichkeit, einfach weitere Vorschriften für dynamische Prüfungen in das Rahmenwerk einzubinden.

1.3. Gliederung

Die Arbeit ist in folgender Weise gegliedert:

Kapitel 2 – Grundlagen: In diesem Kapitel werden Grundlagen, auf die später zurückgegriffen wird, beschrieben und verwendete Begriffe werden definiert.

Kapitel 3 – Konzept: In diesem Kapitel werden geplante Abläufe und Strukturen und Entscheidungen, die nicht die konkrete Implementierung betreffen beschrieben.

Kapitel 4 – Die Implementierung: Die etwas technischere Beschreibung der Umsetzung des Konzepts.

Kapitel 5 – Evaluation: Die Beschreibung der Benutzerevaluation der zweiten Ausbaustufe von SIF.

2. Grundlagen

2.1. Statische und dynamische Prüfungen

Da es das Ziel dieser Arbeit ist, ein Rahmenwerk, das aktuell auf statische Prüfungen beschränkt ist, um dynamische Prüfungen zu erweitern, wird geklärt, was mit diesen Begriffen im Kontext von Spreadsheets gemeint ist. Dazu wird in diesem Kapitel zunächst untersucht, was die Begriffe traditionell bedeuten und anschließend, wie diese Bedeutungen auf die Spreadsheetdomäne angewandt werden können.

2.1.1. Statische Prüfungen

Statische Prüfungen stellen im Software Qualitätsmanagement ein großes Feld dar. Sie können in am Rechner durchgeführte Prüfungen (mechanisch) und ohne Hilfe des Rechners durchgeführte Prüfungen (nichtmechanisch) unterteilt werden [FLS04, LL07]. Nichtmechanische Prüfungen spielen im Software Qualitätsmanagement zwar eine große Rolle z.B. in der Form von Reviews, da es in dieser Arbeit jedoch um eine Software geht, die Prüfungen automatisch durchführen soll, wird auf diese nicht weiter eingegangen.

Am Rechner durchgeführte statische Prüfungen können genutzt werden, um Daten darüber zu generieren, ob und wie stark ein Prüfling von gewissen Normen und Vorschriften abweicht, ob eine formal definierte Spezifikation eingehalten wird oder um Metriken über den Prüfling zu erheben (vgl. [FLS04]). Übertragen auf Spreadsheets lassen sich also beispielsweise die durchschnittliche Formelkomplexität oder ob eine Vorschrift wie „keine Konstenten in Formeln“ eingehalten wird.

2.1.2. Dynamische Prüfungen

Unter dynamischen Prüfungen versteht man im Software Qualitätsmanagement solche Prüfungen, die den Prüfling ausführen. [FLS04] führt als Unterarten dynamischer Prüfungen den Test und die Leistungsmessung auf. Da ich vermute, dass die Leistung bei Spreadsheets nur eine geringe Rolle spielt konzentriere ich mich hier auf den Test. In [LL07] wird der Test folgendermaßen definiert:

Testen ist die - auch mehrfache - Ausführung eines Programms auf einem Rechner mit dem Ziel, Fehler zu finden.

Das Ausführen ist also neben dem Ziel Fehler zu finden eine zentrale Eigenschaft des Tests. In dieser Arbeit wird unter dem Ausführen eines Spreadsheets das Auswerten der im Spreadsheet gespeicherten Formeln verstanden. Der Begriff des Ausführens im Zusammhang mit Spreadsheets leuchtet nicht unbedingt sofort ein. Das hat unter anderem den Grund, dass die Ausführung der Formeln in Spreadsheets in den meisten Spreadsheetumgebungen standardmäßig automatisch erfolgt. Das automatische Ausführen lässt sich allerdings meist auch deaktivieren, sodass eine Änderung der Eingabedaten nicht unmittelbar die Auswertung der Formeln zur Folge hat. Diese Auswertung der Formeln kann nur mit Hilfe einer geeigneten Umgebung erfolgen. Dies sind meist Spreadsheetumgebungen wie z.B. Microsoft Excel, können aber auch APIs wie z.B. Apache POI sein. Von den Ergebnissen dieser Formeln abhängig sind:

- Der dargestellte Wert der Formelzelle (die Zelle, in der die Formel enthalten ist). Die Zelle stellt das Ergebnis je nach Formatierung dar
- Alle Zellen, die auf die Formelzelle direkt oder indirekt verweisen
- Diagramme, die den Zellwert verwenden
- Bedingte Eigenschaften von Zellen wie z.B. die Hintergrundfarbe

2.2. Endanwender und Programmierung

Laut [KAB+11] ist der größte Unterschied zwischen professionellen Programmierern und Endanwendern, die programmieren ihr Ziel: Endanwender programmieren, um eine Aufgabe in ihrem Fachgebiet zu lösen. Professionelle Programmierer hingegen haben das Ziel, Programme für andere zu entwickeln. Die Gründe dafür können vielfältig sein und sind für diese Definition nicht relevant.

2.2.1. Gründe Spreadsheets zu testen

Will man die Qualität von Spreadsheets verbessern stellt sich die Frage, warum man primär den Test dazu verwenden sollte. Im Software-Qualitätsmanagement stellt der Test zwar eine gängige Methode dar, im Vordergrund steht jedoch systematisches Vorgehen nach bewährten Techniken der Software-Konstruktion [FLS04]. Viele Endanwender haben jedoch keine Ausbildung auf diesem Gebiet, sehen keinerlei Vorteile in einem Entwicklungsprozess [KAB+11] und erwarten wahrscheinlich auch keine langfristige Nutzung ihrer Spreadsheets. Gleichzeitig haben sie oft sehr großes Vertrauen in ihre Fähigkeit fehlerfreie Spreadsheets zu erstellen [Pan08], was die Bereitschaft Mühen auf sich zu nehmen um Fehler zu verhindern weiter schmälert. Wie in [FLS04] festgestellt wird ist das Testen ein „natürliches“Prüfverfahren, da es der Intuition der meisten Menschen entspricht und somit eine Technik, von der erwartet werden kann, von Endanwendern akzeptiert zu werden. Beim Test von Spreadsheets stelle ich mir zwei Hauptszenarien vor: Ein Test nach dem Erstellen der (vorerst) finalen Version des Spreadsheets und ein Test, der stattfindet nachdem das Spreadsheet erweitert wurde.

Der Test nach dem Erstellen soll vor allem überprüfen, ob das mentale Modell, das der Autor hatte als er das Spreadsheet erstellt hat korrekt übertragen wurde. Der Test nach Veränderungen ist vor allem dann interessant, wenn eine Vorlage durch einen Experten erstellt wird, die später durch andere Mitarbeiter ausgefüllt und dabei evtl. erweitert wird.

2.3. Das SIF Rahmenwerk

Rahmenwerk (Framework) Ein Rahmenwerk ist eine generische Lösung, die die Basis für sehr ähnliche Anwendungen bildet. Im Gegensatz zu einer Klassenbibliothek wird durch ein Rahmenwerk auch der Kontrollfluss der Anwendung vorgegeben (vgl. [LL07]).

Das in Java entwickelte Rahmenwerk SIF (Spreadsheet Inspection Framework) wurde im Rahmen der Diplomarbeit [Zit12] erstellt. Es unterstützt bereits drei statische Vorschriften:

- *Konstanten in Formeln* Mit dieser Vorschrift lässt sich überprüfen, ob Formeln konstante Werte enthalten. Bestimmte Werte lassen sich hiervon ausnehmen.
- *Leserichtung* Hiermit wird untersucht, ob nur Zellen, die sich rechts und unterhalb der betrachteten Zelle befinden diese referenzieren.
- *Formelkomplexität* Mit dieser Vorschrift lässt sich spezifzieren, wie viele Operationen maximel in Formeln enthalten sein dürfen und wie tief die maximale Schachtelungstiefe ist.

Der Umfang der unterstützten Prüfungen beschränkt sich zwar auf statische Prüfungen, bei der Konzeption wurde jedoch berücksichtigt, dass es um dynamische und nichtmechanische Prüfungen erweiterbar sein soll.

2.3.1. Die Metapher

Als Basis für die Struktur wurde die Metapher eines Kfz-Prüfzentrums verwendet. Dazu wurden in [Zit12] Bestandteile eines solchen Prüfzentrums ausgemacht und beschrieben. Die im Rahmen dieser Arbeit entwickelte Erweiterung fußt auf diesen Bestandteilen. Deswegen werden diese Beschreibungen hier im Kontext des in der ersten Ausbaustufe entstandenen Frameworks, als auch der zweiten Ausbaustufe zusammengefasst.

Prüfung: Eine Prüfung von Spreadsheets hat zum Ziel die Konformität von Spreadsheets mit Richtlinien zu überprüfen. Diese Richtlinien werden im Unterschied zum Kfz-Prüfzentrum vom Nutzer festgelegt.

Inspektions-Werkstatt: Stellt die technischen Mittel bereit, um die Prüfung von Spreadsheets durchzuführen. Dazu gehören sowohl allgemeine Werkzeuge um die Eigenschaften von Spreadsheets zu untersuchen als auch spezielle Werkzeuge um Bestandteile von Spreadsheets zu untersuchen.

Vorschrift: Eine Vorschrift definiert anpassbare Prüfkriterien, denen bestimmte Bestandteile eines Spreadsheets oder das Spreadsheet selbst genügen müssen.

Richtlinie: Eine Richtlinie besteht aus einer oder mehrerer Vorschriften.

Verstoß: Bei Abweichungen von einer Vorschrift wird ein Verstoß gemeldet. Ein solcher Verstoß kann je nach Vorschrift gegen die verstoßen wird unterschiedliche Daten enthalten.

Inspektionsauftrag: Ein Nutzer der Inspekions-Werkstatt kann Inspektionsaufträge aufgeben. Er gibt dazu einen Namen sowie das zu prüfende Spreadsheet an und wählt eine Richtlinie aus.

Spreadsheet-Element: Spreadsheet-Element sind Bestandteile von Spreadsheets wie z.B. Zellen, Referenzen zwischen Zellen. Dabei wird zwischen Basis-Elementen und Spezial-Elementen unterschieden. Basis-Elemente sind Elemente, die sich allein durch gängige Spreadsheet Formate beschreiben lassen. Spezial-Elemente hingegen können nicht in einem Spreadsheet Format als solche gekennzeichnet werden, sondern werden erst durch die Struktur, ihren Inhalt o.ä. zu dieser Art von Element.

Element-Scanner: Element-Scanner sind Module, die Spreadsheet-Elemente erkennen und dem Spredsheet-Inventar hinzufügen können. SIF bietet in der ersten Ausbaustufe bereits Möglichkeiten, weitere Element-Scanner einzubinden.

Spreadsheet-Inventar: Das Spreadsheet beinhaltet alle Spreadsheet-Elemente, die beim Einlesen des Spreadsheets oder durch Element-Scanner erstellt wurden.

Prüfstände: Mit Prüfständen kann die Einhaltung von Vorschriften überprüft werden. Ein Prüfstand kann entweder aus mehreren Prüfkwerzeugen zusammengestellt werden oder monolithisch sein.

Prüfwerkzeuge: Prüfwerkzeuge sind modulare Bestandteile von zusammengestellten Prüfständen. Sie überprüfen jeweils genau eine Klasse von Spreadsheet Elementen. In der ersten Ausbaustufe von SIF existiert zwar die Struktur um Prüfwerkzeuge einzubinden, allerdings gibt es keine konkreten Umsetzungen.

2.3.2. Ablauf einer Inspektion

Im Folgenden werden die Schritte erläutert, die nötig sind um SIF aus einer eigenen Anwendung heraus zu verwenden. Die Darstellung und Granularität sind bewusst anders gewählt als in [Zit12] um die tatsächlichen Abläufe deutlicher darzustellen.

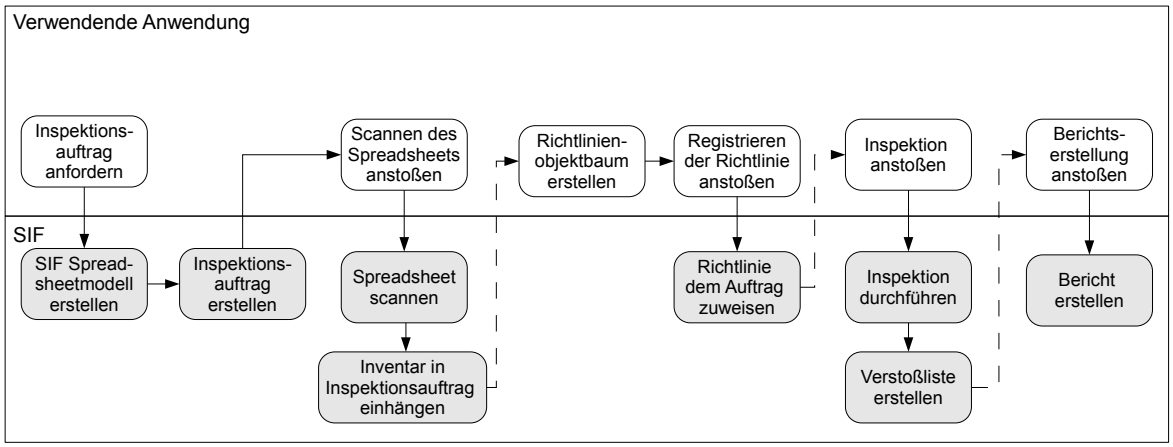

Abbildung 2.1.: Verwendung von SIF durch eine Anwendung

Inspektionsauftrag erstellen

Um SIF für eine Inspektion zu verwenden, muss zunächst ein Inspektionsauftrag angefordert werden. Dadurch liest SIF das Spreadsheet mit Hilfe einer externen Klassenbibliothek ein. Danach wird es aus dem Format dieser externen Klassenbibliothek in das SIF Objektmodell transfomiert. Abschließend wird ein Inspektionsauftrag erstellt, der dieses Objektmodell enthält.

Scannen des Spreadsheets anstoßen

In diesem Schritt wird das Spreadsheet durch alle registrierten Elementscanner untersucht. Auf diese Weise wird das Spreadsheetinventar erstellt, das dem Inspektionsauftrag zugewiesen wird.

Richtlinie erstellen

In [Zit12] wird beschrieben, dass der Nutzer zu diesem Zeitpunkt eine Richtlinie auswählt. Allerdings ist in der vorliegenden Version nur eine Richtlinie enthalten. Diese Richtlinie enthält alle Vorschriften in ihrer Standardkonfiguration. Deswegen werden Anwendungen wahrscheinlich an dieser Stelle Vorschriften erstellen und konfigurieren und sie zu einer Richtlinie zusammenstellen. Alternativ können Anwendungen schon vor der eigentlichen Inspektion eigene Richtlinien zusammenstellen und diese bei SIF registrieren.

Richtlinie registrieren

Nach der Erstellung muss die Richtlinie noch registriert werden. Durch das Registrieren der Richtlinie weist SIF die Richtlinie dem Inspektionsauftrag zu.

Inspektion

Durch das Anstoßen der Inspektion übergibt SIF den Inspektionsauftrag an die Inspektions-Werkstatt. Diese übergibt das Spreadsheet anschließend an alle benötigten Prüfwerkstätten. Die von den Prüfwerkstätten gefundenen Verstöße werden im Inspektionsauftrag vermerkt.

Berichtserstellung

Die Berichtserstellung ist nicht zwingend notwendig. Es ist durchaus denkbar, dass die Anwendung die Verstöße aus dem Inspektionsauftrag ausliest und in einer eigenen Darstellung präsentiert.

3. Konzept

3.1. Anforderungen

Vorgaben

Das Rahmenwerk SIF soll im Rahmen dieser Arbeit in erster Linie um die Funktionalität erweitert werden, dynamische Vorschriften erstellen und prüfen zu können. Außerdem soll es möglich sein, Ein- und Ausgabezellen automatisch zu identifizieren und Verstöße gegen Vorschriften auszugeben.

Eigene Anforderungen

Beibehalten von Struktur und Metapher

Um die Wartung und weitere Erweiterungen zu erleichtern sollen die vorhandenen Prinzipien, Strukturen und die Metapher „Kfz-Prüfwerkstatt“ beibehalten werden. Eine Erweiterung der bestehenden Struktur um neue Teilstrukturen ist also einer Veränderung der bereits existierenden Teilstrukturen vorzuziehen.

Austauschformat für Prüfregeln

In der ersten Ausbaustufe von SIF können Richtlinien ausschließlich per Java Programmcode erstellt werden. Vieles spricht jedoch dafür, die Spezifikation von Richtlinien nicht per Programmcode durchzuführen, sondern ein Austauschformat zu verwenden. Bei einem für Menschen lesbaren Format ist es so auch Benutzern, die keine Programmierkenntnisse besitzen möglich, Richtlinien für SIF zu spezifizieren. Wird zusätzlich die Möglichkeit geschaffen SIF eine solche Spezifikation (oder den Pfad zu ihr) als Kommandozeilenparameter zu übergeben wird so eine Schnittstelle geschaffen, mit der externe Programme Vorschriften spezifizieren können ohne Details über das interne Datenmodell von SIF kennen oder kompatible Technologien verwenden zu müssen. So könnte beispielsweise ein in VBA entwickeltes Microsoft Excel AddIn das SIF Framework mit ähnlich hohem Aufwand verwenden um Berichte zu erzeugen, wie eine in Java oder C++ entwickeltes LibreOffice Calc Extension. Außerdem ist es vorstellbar in zukünftigen Ausbaustufen auch ein Austauschformat für Berichte zu erstellen, sodass externe Programme in ihrer Art die Ergebnisse von SIF zu verwenden nicht eingeschränkt wären. Beispiele für solche Programme reichen von einer

einfachen graphischen Benutzungsoberfläche zur Spezifikation von Vorschriften über eine automatische Hintergrundüberwachung bei der Bearbeitung von Spreadsheets bis hin zur Integration in ein Document Management System.

Berührungsfreiheit

Das geprüfte Spreadsheet soll durch die Prüfung nicht verändert werden, denn jede Veränderung am Prüfling führt unter Umständen zu Seiteneffekten, die die Resultate verfälschen.

Überprüfbare SIF Spreadsheet-Element Eigenschaften

Es sollen nicht nur Zellwerte sondern auch andere Eigenschaften referenzierbarer Spreadsheet-Elemente überprüft werden können.

Umfang der angebotenen Vorschriften

Um die Fähigkeit der zweiten Ausbaustufe, dynamische Prüfungen durchzuführen, demonstrieren und testen zu können sollen drei Vorschriften angeboten werden:

Einfacher Vergleich von Zellwerten Die wohl einfachste Vorschrift. Sie schreibt vor, dass der u.U. durch eine Formel berechnete Zellwert einen bestimmten Wert hat.

Übereinstimmung mit Intervall Hierbei soll überprüft werden, ob der u.U. durch eine Formel berechnete Zellwert innerhalb eines Intervalls liegt. Das Intervall soll als offen, geschlossen oder nur auf einer Seite offen angegeben werden können.

Überprüfung der Anzahl von Spreadsheet-Elementen eines Typs Diese Prüfung ist auf den ersten Blick eine statische Prüfung. Allerdings kann sich durch das Eintragen von Testeingaben die Anzahl von Spreadsheet-Elementen eines bestimmten Typs verändern. So kann bei einem nicht gesperrten Spreadsheet das Überschreiben einer Formel dazu führen, dass bestimmte Zellen keine Eingabe-, Ausgabe- oder Zwischenzellen sind. Ein weiteres Szenario stellen zukünftige Element-Scanner dar, die z.B. Zellen mit Fehlern oder einer bestimmten Formatierung erkennen. Da eine Formatierung abhängig vom Zellwert sein kann, kann sich diese durch das Eintragen von Werten und berechnen der Formeln ändern.

Erweiterbarkeit um neue dynamische Vorschriften

Wie auch in der ersten Ausbaustufe soll es möglich sein, dem Rahmenwerk neue Vorschriften hinzuzufügen. Es soll dabei vor allem erleichtert werden, Vorschriften, die eine dynamische Prüfung voraussetzen, hinzuzufügen.

3.2. Notation für Vorschriften

3.2.1. Bewertungskriterien

Um ein Austauschformat für dynamische Vorschriften zu erstellen wird eine Notation bzw. Sprache benötigt, die geeignet ist Spreadsheet-Bedingungen zu beschreiben und von Mensch und Maschine lesbar ist. Diese gewünschten Eigenschaften werden im Folgenden genauer formuliert. Sie sind allerdings nicht als harte Anforderungen wie in einer Softwarespezifikation zu sehen, sondern als Bewertungskriterien um den geeignetsten Kandidaten zu bestimmen.

Sprachumfang

Die Sprache sollte geeignet sein um Bedingungen zu beschreiben und diese Spreadsheet-Elementen zuzuweisen. Dazu muss sie mindestens die gängigen Vergleichsoperationen gleich, kleiner, größer, kleiner-gleich und größer-gleich anbieten. Außerdem müssen Wertebereiche darstellbar sein. Gleichzeitig sollte die Sprache nicht zu umfangreich sein, da mit zunehmendem Umfang der Implementierungsaufwand steigt und es u.U. zu Verwirrung führt, wenn Sprachelemente ausgelassen werden oder ihre Verwendung zu Fehlern führt. Aus dem gleichen Grund sollte die Anzahl von Sprachelementen, die keine Anwendung in der Spreadsheet Domäne finden möglichst gering sein.

Dokumentation

Sollte die Sprache komplex oder umfangreich sein, wird eine (gute) Dokumentation gefordert. Diese hilft sowohl dem Entwickler, den Parser zu implementieren, als auch dem Endanwender Richtlinien zu spezifizieren.

Endbenutzertauglichkeit

Auch wenn die Sprache selbst in zukünftigen Ausbaustufen wahrscheinlich durch eine graphische Benutzungsschnittstelle abgelöst wird, sollte die Sprache auch von Endbenutzern verwendet werden können. Es wäre zwar schön aber es ist nicht nötig, dass völlig unerfahrene Benutzer sie benutzen können.

Implementierungsaufwand

Da die Zeit zum Implementieren der Deserialisierung nur einen Bruchteil des gesamten Aufwands darstellen soll und somit stark begrenzt ist, muss der hierfür zu erwartende Aufwand so gering wie möglich sein. Es ist also ein großer Vorteil, falls für eine Sprache bereits Java Klassenbibliotheken oder andere Hilfsmittel existieren.

Erweiterbarkeit

Da es möglich sein soll, neue Typen dynamischer Vorschriften in SIF einzubinden, muss auch das Austauschformat erweiterbar sein ohne dabei zu viel Aufwand zu erzeugen.

3.2.2. Kandidaten

Um eine Notation zu finden, wurden verschiedene existierende Sprachen zur Beschreibung von Prüfregeln verschiedener Art ermittelt und unter Berücksichtigung der im vorigen Kapitel beschriebenen Bewertungskriterien auf ihre Tauglichkeit hin untersucht. Außerdem wurde ein Zeitfenster festgelegt innerhalb dessen eine geeignete Sprache gefunden sein musste. Andernfalls sollte eine eigene Sprache erstellt werden.

Bei der Suche nach passenden Sprachen ergaben sich zunächst folgende Kandidaten:

- Eine namenlose, in [BCP+03] definierte
- Assertion Definition Language (ADL)
- Oracle CDL
- UML OCL (Object Constraint Language)

ADL und CDL

Diese beiden Kandidaten konnten schon früh ausgeschlossen werden, da sie ein anderes Zielpublikum besitzen und andere Einsatzzwecke verfolgen. Oracle CDL ist Teil von Oracle Configurator Developer, das verwendet wird um Werkzeuge für den Vertrieb zu entwickeln. Die ADL ist eine Zwischensprache zwischen natürlicher (englischer) Sprache und formalen Testspezifikationen und wird verwendet, um Testsysteme zu generieren. Beide Sprachen sind sehr umfangreich und es konnten keine öffentlich zugängliche Parser oder Klassenbibliotheken gefunden werden.

Burnett2003

Hierbei handelt es sich um eine Notation, die an der Oregon State University geschaffen wurde, um Regeln für Spreadsheets zu beschreiben. Sie wurde zunächst in [BCP+03] und später auch in [Cre05] verwendet.

Eine Bedingung wird in dieser Notation „Assertion“ genannt. Sie besteht aus Blöcken von Ausdrücken, die durch ein logisches Und verbunden sind. Die Ausdrücke innerhalb dieser Blöcke sind durch ein logisches Oder verbunden. Die Ausdrücke können entweder einen Vergleich zu einem Wert oder ein Intervall definieren. Die Definition der Notation aus dem Artikel [BCP+03] ist in 3.1 dargestellt. Allerdings wird in diesem Artikel auch erwähnt, dass Ausdrücke in der Spreadsheet Sprache zum Erscheinungszeitpunkt noch nicht implementiert

Listing 3.1 Definition der Notation in [BCP+03]

```
(N, {and-assertions}), where:
each and-assertion is a set of or-assertions,
each or-assertion is a set of (unary-relation, valueexpression) and (binary-relation,
    value-expression-pair) tuples,
each unary-relation ∈ {=, <, <=, >, >=},
each binary-relation ∈ {to-closed, to-open, to-openleft, to-openright},
each value-expression is a valid formula expression in the spreadsheet language,
each value-expression-pair is two value-expressions.
```

waren. Die Tatsache, dass das logische Ündünd das logische Öderïn dieser Grammatik nicht explizit vorkommen wird auf [PRM01] zurückgeführt. Dort wird beschrieben, dass Endbenutzer große Probleme beim Benutzen dieser logischen Operatoren haben.

Betrachtet man die Notation im Hinblick auf die festgelegten Bewertungskriterien ergibt sich folgendes Bild:

Sprachumfang Die Notation ist in ihrem Umfang sehr gut auf die Problemstellung „Beschreiben einer Bedingung zur dynamischen Prüfung eines Spreadsheet-Elements“ zugeschnitten. Sie kann allerdings nur als Grundlage für das eigene Austauschformat dienen, da bestimmte Elemente fehlen um den geplanten Umfang ausdrücken zu können. Dazu gehören *or-assertion* Elemente für die Zählung von Spreadsheet-Elementen eines Typs sowie Möglichkeiten um Testeingaben zu definieren und festzulegen ob eine Bedingung eine Invariante oder eine Nachbedingung ist.

Dokumentation : Eine Dokumentation existiert nur in Form des Artikel [BCP+03] und der Master-Thesis [Cre05]. Allerdings ist das kein großes Manko da die Sprache weder komplex noch umfangreich ist und nur als Basis für eine eigene Sprache dienen könnte.

Implementierungsaufwand : Es existieren keine öffentlich zugänglichen Parser-Implementierungen, sodass das Parsen selbst implementiert werden müsste.

Erweiterbarkeit : Die Erweiterbarkeit der Notation hängt der Art der Parser-Implementierung ab. Falls es möglich ist einen Schema- oder Grammatikbasierten Parser oder Parser-Generator zu erstellen muss im Fall einer Erweiterung nur das Schema bzw. die Grammatik angepasst werden. Falls das nicht der Fall ist müsste beispielsweise im Falle eines Expertensystems, bei dem jedes Sprach-Element an ein eigenes Parse-Modul weitergereicht wird das Parse-Modul des Oberelements angepasst und ein Modul für das neue Element hinzugefügt werden.

Die Notation kann nur als Grundlage für eine eigenes Austauschformat dienen, da beispielsweise das Zählen von Elementen eines Typs nicht spezifiziert ist. Außerdem

UML OCL

OCL (Object Contraint Language) ist ein Bestandteil der weit verbreiteten UML (Unified Modeling Language). Uml wird zum Modellieren von Softwaresystemen eingesetzt. Die OCL

erweitert die UML um eine prädikatenlogische und somit formal überprüfbare Sprache, mit der Constraints (in der englischen Mathematik für „Zwangsbedingung“) formuliert werden können. Das Ziel ist es so das UML Modell zu vervollständigen. Die Anwender der OCL sind also hauptsächlich Software-Architekten und -Entwickler. Grundsätzlich ist es möglich aus OCL Ausdrücken automatisch Java-Code zu generieren. Allerdings schrieb Breu 2005 „Das hierin vorhandene Potential wird derzeit noch von keinem Tool vollständig genutzt.“ und daran scheint sich bis heute nicht viel geändert zu haben. Ein vielversprechendes Werkzeug hierfür ist das an der TU Dresden entwickelte DresdenOCL. Doch auch dieses Tool produziert aktuell Code, der noch stark angepasst werden muss.

Sprachumfang: Die OCL bietet weit mehr als den benötigten Sprachumfang. Allerdings konnte keine Möglichkeit gefunden werden eine Untermenge von OCL zu erstellen, wie das z.B. in der UML mit Hilfe von UML Profilen möglich ist.

Dokumentation : Zur OCL existieren ausführliche Dokumentationen. Allerdings sind diese vermutlich zu technisch für Endanwender.

Endbenutzertauglichkeit : Da die OCL eine prädikatenlogische Sprache und ähnlich komplex ist wie eine Programmiersprache halte ich sie nicht für geeignet von Endbenutzern verwendet zu werden.

Implementierungsaufwand : Der Einsatz als Beschreibungssprache für Spreadsheet-Bedingungen unterscheidet sich von dem eigentlichen Einsatzzweck als Beschreibungssprache für Constraints in UML Modellen. Dadurch kann nur in geringem Maß auf vorhandene Bibliotheken zurückgegriffen werden. Deswegen schätze ich den Implementierungsaufwand als hoch ein.

3.2.3. Eigene Sprache

Da innerhalb des festgelegten Zeitfensters keine Sprache gefunden wurde, von der erwartet werden konnte, dass sie allen Bewertungskriterien in ausreichendem Maße genügt, musste damit begonnen werden eine eigene Beschreibungssprache zu erstellen. Zum Erstellen eigener Sprachen gibt es eine Vielzahl möglicher Technologien und Werkzeuge. Ich habe mich mit den folgenden drei Möglichkeiten auseinandergesetzt:

JavaCC steht für „Java Compiler Compiler“und kann Parser für eine große Zahl von Grammatiken[1] erstellen. In Kombination mit anderen Werkzeugen, wie JJTree kann man so aus einer Eingabe, die der Grammatik folgt, einen Baum der Sprachelemente erstellen. Im Fall einer Notation für Vorschriften, würde man diesen Baum durchlaufen und währenddessen das eigene Datenmodell aufbauen.

XML steht für extensible Markup Language (erweiterbare Auszeichnungssprache) und ist eine hierarchische Auszeichnungssprache, die in den letzten Jahren zu großer Verbreitung gefunden hat. Dadurch existiert eine Vielzahl an Werkzeugen und in vielen

[1] LL(k) Grammatiken. Für weitere Informationen zum Thema Compiler und Grammatiken siehe [ALSU07]

höheren Programmiersprachen sind Klassenbibliotheken enthalten, die den Umgang mit XML Dateien erleichtern. Um in Java XML zu serialisieren gibt es prinzipiell drei Möglichkeiten:

- Die Verwendung von XML Bindung wie z.B. dem Standardprodukt JAXB (Java Architecture for XML Binding). Solche Lösungen können mit zusätzlichen Informationen, die z.B. über Annotationen im Quelltext hinzugefügt werden Objektbäume des Klassenmodells der Anwendung automatisch serialisieren und deserialisieren.
- Die automatische Erstellung eines Objektbaums z.B. mit DOM (Data Object Model). Der Objektbaum besteht hier im Unterschied zur XML Bindung allerdings aus Objekten der eingesetzten API. Die Anwendung, die die API einsetzt muss den Objektbaum also durchlaufen und transformieren. Der Vorteil gegenüber der XML Bindung ist, dass mehr Anwendungswissen eingesetzt werden kann. Dadurch können z.B. Schwierigkeiten mit Interfaces umgangen werden.
- Schrittweises Lesen des Eingabestroms z.B. mit SAX (Simple API for XML). Diese Methode benötigt bei guter Implementierung weniger Arbeitsspeicher als das Erstellen eines Objektbaums und ist dabei genauso universell einsetzbar. Allerdings besteht eine größere Gefahr, dass bei der Implementierung Fehler gemacht werden.

3.2.4. Entscheidung

Da XML durch XML Bindung zeitgleich mit dem Richtlinien-Modell von SIF erweitert werden kann und es die umfangreichste Werkzeug-Unterstützung bietet fällt die Wahl hierauf. Im Punkt Endbenutzertauglichkeit ist eine kompaktere Sprache vermutlich besser. Allerdings ist XML weit verbreitet und es existiert eine große Menge von Tools, die Endanwender bei der Erstellung unterstützen. Das Format soll es erlauben Vorschriften zu spezifizieren die Invarianten, Testeingaben und Nachbedingungen beinhalten. Als Invarianten und Nachbedingungen sollten ein Wertevergleich, eine Überprüfung ob sich ein Wert in einem Intervall befindet und die Zählung von Elementen eines Typs angegeben werden können. Außerdem sollten Ein- und Ausgabezellen gespeichert werden können.

3.3. Integration in SIF

Die Möglichkeit das Framework um dynamische Prüfungen zu erweitern wurde in der ersten Ausbaustufe bereits berücksichtigt. Der dazu vorgesehene Ablauf wird in Abbildung 3.1 dargestellt. Dieser geht von der Annahme aus, dass nur diejenige Elemente, die bei der statischen Prüfung auffallen anschließend einer dynamischen Prüfung unterzogen werden sollen. Das ist zwar ein interessanter Ansatz um Leistung zu sparen, schmälert aber gleichzeitig die Effektivität der Prüfung, da nicht alle Elemente, die möglicherweise einen Verstoß verursachen untersucht werden.

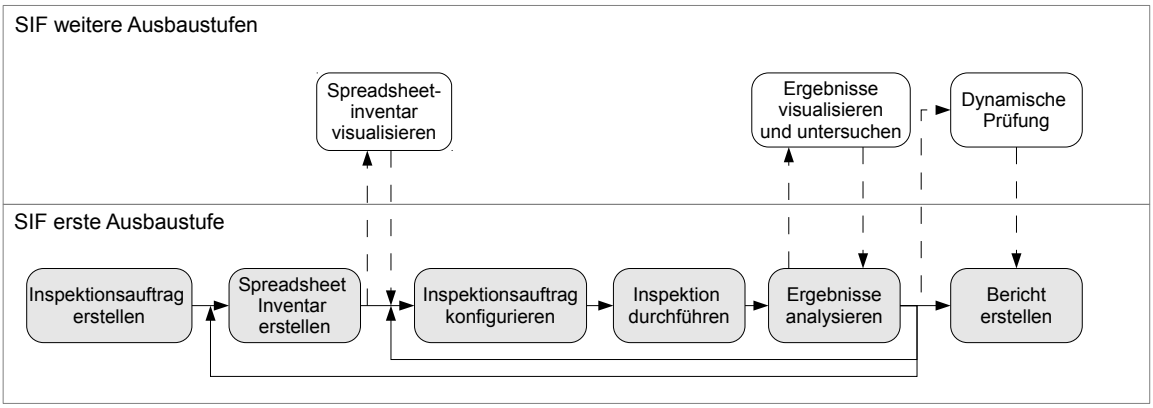

Abbildung 3.1.: Ablauf einer Inspektion in der ersten Ausbaustufe (adaptiert aus [Zit12])

Stattdessen soll die dynamische Prüfung im Rahmen der normalen Prüfung für alle Spreadsheet-Elemente durchgeführt werden, sodass auch Spreadsheet-Elemente, die keine Verstöße bei der statischen Prüfung gezeigt haben geprüft werden.

Vorschriften, die eine dynamische Prüfung, also eine Ausführung des Spreadsheets voraussetzen haben einige gemeinsame Eigenschaften. So wird nahezu jede Vorschrift einen Bezug zu einer Zelle oder einem Bereich im Spreadsheet, also ein Ziel haben. Da außerdem nicht nur Werte sondern auch Eigenschaften überprüfbar sein sollen müssen solche Vorschriften eine Angabe enthalten, welche Eigenschaft das sein soll. Außerdem muss jede Vorschrift einen SOLL-Wert enthalten, der bei der Prüfung verglichen wird. Die Tatsache, dass eine Vorschrift ein Ziel und die gleichen Testeingaben für eine Vielzahl von Invarianten und Nachbedingungen angibt führt dazu, dass es sinnvoller ist die konkreten Bedingungen als Unterelemente der eigentlichen Vorschrift anzusehen, statt mehrere Vorschriften mit dem gleichen Ziel und den gleichen Testeingaben anzugeben. Eine Alternative wäre ein Behälter für ein Ziel und mehrere Testeingaben doch auch dieser müsste von jeder Invariante und Bedingung referenziert werden, was sowohl die Beschreibung der Vorschrift als auch die Durchführung unnötig erschweren würde. Aus diesem Grund wird ein neues Element in die Metapher eingeführt: die Bedingung.

Bedingung Eine Bedingung spezifiziert einen Zustand in dem sich ein oder mehrere Spreadsheet-Elemente befinden müssen. Es ist ein Teil einer Vorschrift zur dynamischen Prüfung. Als zu überprüfendes Ziel können vom Benutzer Zellen, Bereiche oder eine Klasse von Spreadsheet-Elementen angegeben werden. Die Bedingung kann entweder als Invariante, die immer gelten muss, oder als Nachbedingung die nach der Eintragen von Testeingaben in das Spreadsheet gelten muss spezifiziert werden.

Da es viele verschiedene Arten von Vorschriften geben kann, die eine Ausführung des Spreadsheets zu ihrer Überprüfung benötigen werden austauschbare Unterelemente für den neuen Prüfstand benötigt. Prinzipiell böte sich hier also ein aus Prüfwerkzeugen zusammengesetzter Prüfstand (siehe Kapitel 2.3.1) an. Da jedoch zusammengesetzte Prüfstände und

Prüfwerkzeuge jeweils auf eine Klasse von Spreadsheet-Elementen beschränkt sind, wird stattdessen ein **monolithischer Prüfstand** verwendet.

Auswertung von Formeln

Um die Auswertung der Formeln nicht selbst implementieren zu müssen soll die bereits eingesetzte externe Klassenbibliothek dazu eingesetzt werden. Die einzige aktuell verwendete Klassenbibliothek Apache POI unterstützt die Auswertung von Formeln bereits. Falls ein neues Format unterstützt werden soll und die entsprechende Klassenbibliothek keine Formelauswertung anbietet, führt das allerdings dazu, dass die dynamische Prüfung mit diesem Format nicht funktioniert. Das heißt in diesem Fall muss dennoch eine eigene Formelauswertung implementiert werden.

Ablauf

Der Ablauf von SIF wird zur Realisierung der dynamischen Prüfung verändert und erweitert. Abbildung 3.2 stellt dies dar. Neu hinzu kommt die Deserialisierung der XML Spezifikation, wodurch es unnötig wird Richtlinien programmatisch zu erstellen. Verändert wird die Erstellung des internen Datenmodells. Durch das Hinzufügen einer neues Prüfstands wird auch das Durchführen der Inspektion verändert.

Richtlinie erstellen Im ersten Schritt wird die Richtlinie erstellt. Entweder programmatisch oder indem eine entsprechende XML Datei deserialisiert wird. Die Anwendung übergibt die so erhaltene Richtlinie zusammen mit dem Pfad des Spreadsheets an SIF. Diesen Pfad kann sie entweder der Richtlinie entnommen oder selbst bestimmt haben z.B. durch einen Auswahldialog.

Inspektionsauftrag erstellen Die Erstellung des internen Objektmodells wird in zwei Teilschritte zerlegt: Die Erstellung des Modells der externen Klassenbibliothek und die Transformation von diesem in das interne SIF Objektmodell. Beide separat erstellten Objektbäume werden im Inspektionsauftrag abgelegt. Anschließend wird das Spreadsheet gescannt und so das Spreadsheet-Inventar erstellt.

Inspektion durchführen Die Inspektion wird von einem neuen Prüfstand durchgeführt. Da Invarianten immer gelten sollen, werden diese zu Beginn und zu den Zeitpunkten, an denen sich das Spreadsheet ändert überprüft (siehe Abbildung 3.3). Nachdem die Invarianten ganz zu Beginn überprüft wurden wird das Spreadsheet-Modell (der externen Klassenbibliothek) kopiert, sodass das Original erhalten bleibt. Anschließend werden die Testeingaben eingefügt. Nachdem die Invarianten erneut geprüft wurden, werden die Formeln mit Hilfe der externen Klassenbibliothek ausgewertet. Anschließend werden zuerst die Invarianten und danach die

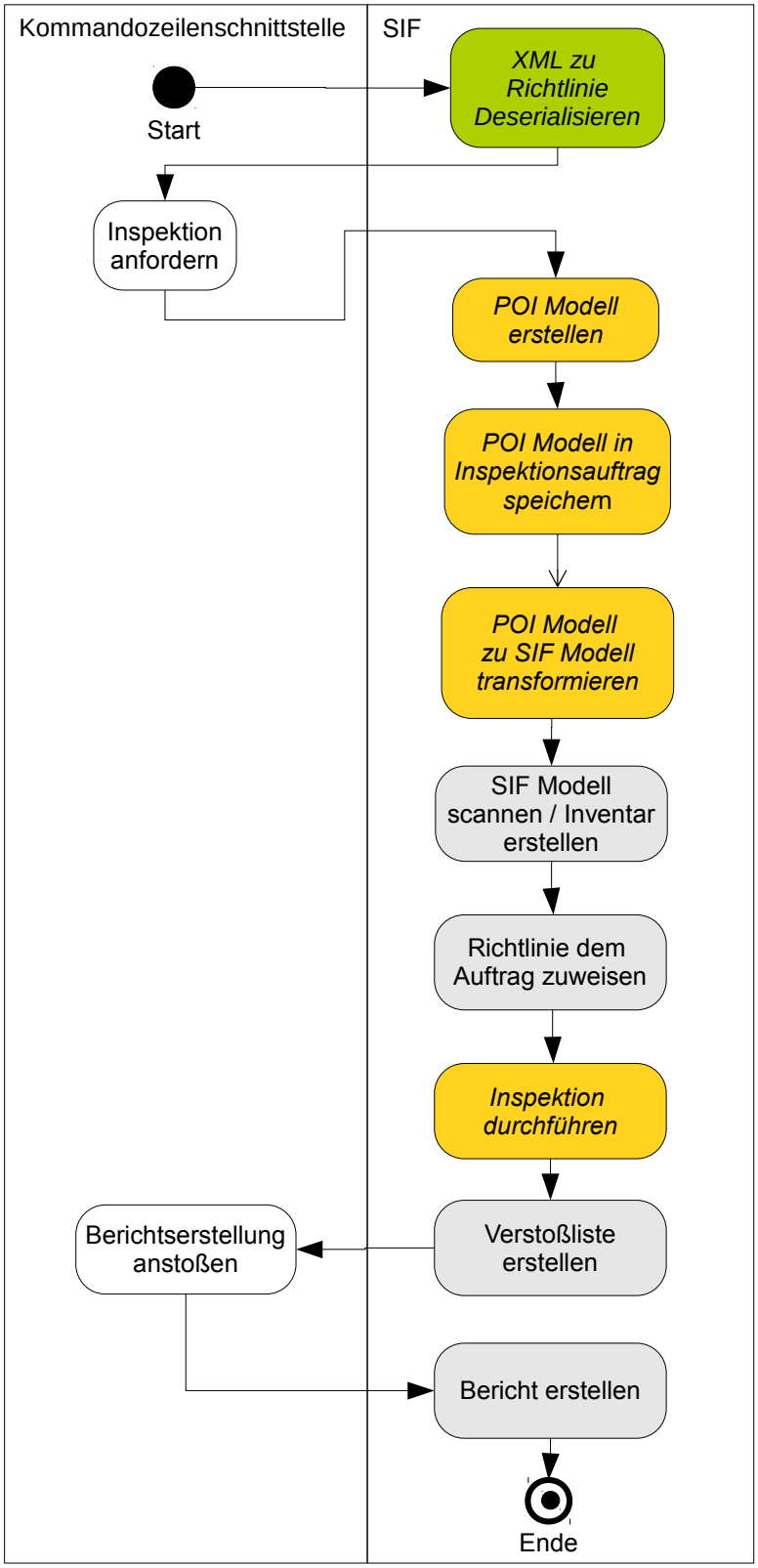

Abbildung 3.2.: Ablauf in der zweiten Ausbaustufe

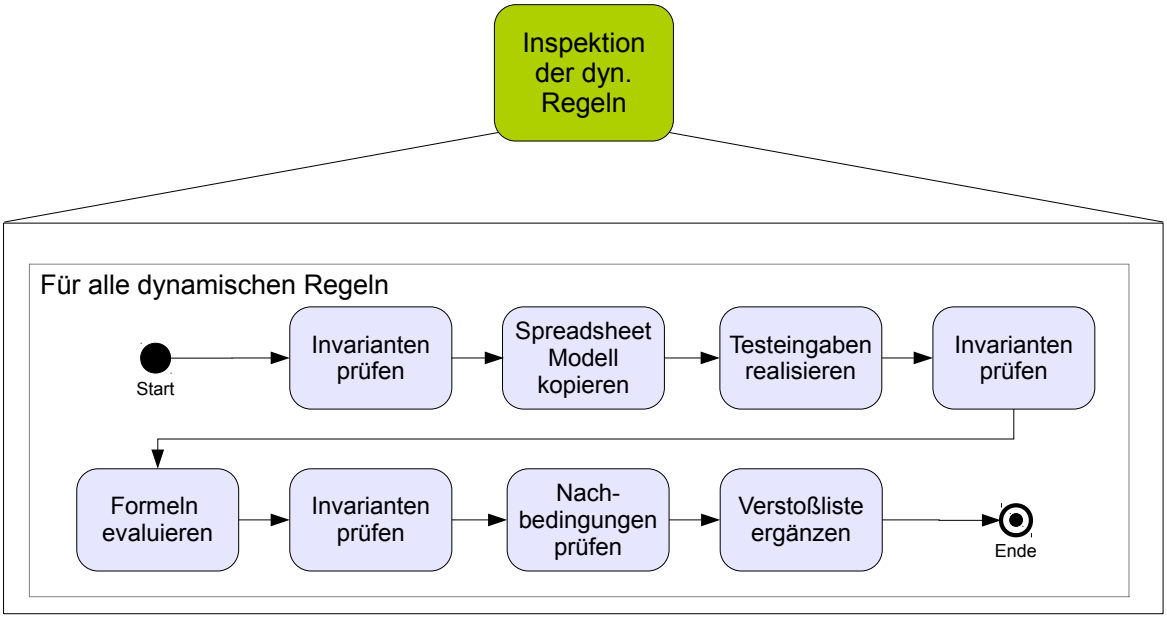

Abbildung 3.3.: Inspektion dynamischer Vorschriften

Nachbedingungen geprüft. Die Reihenfolge dieser beiden Schritte ist willkürlich gewählt. Anschließend werden alle aufgetretenen Verstöße zur Verstoßliste hinzugefügt.

3.3.1. Überprüfen der Metapher

Um zu beurteilen wie gut die dynamische Prüfung in die Metapher passt untersuche ich, inwieweit ein Fahrzeug in einem Kfz-Prüfzentrum ausgeführt wird. In einer solchen Einrichtung werden nicht nur Sichttests durchgeführt, sondern auch allerlei Tests, bei denen jeweils mindestens das untersuchte Teilsystem ausgeführt wird. So wird beispielsweise bei einem Bremsanlagentest eine Wirkungsprüfung auf einem Bremsprüfstand durchgeführt [Klu90]. Auf einem solchen Bremsprüfstand werden die Räder durch eine externe Rolle oder durch ein Rollband angetrieben [Rot95]. Um die die Bremswirkung zu messen wird die Bremsanlage betätigt. Das gesamte Fahrzeug wird für Bremstests nur in Bewegung gesetzt, falls es zu groß für den Bremsstand ist. In diesem Fall wird die Bremsleistung mit einem schreibenden Bremsmeßgerät aufgezeichnet [Klu90]. Laut § 29 Anlage IIIa der StVZO gehört zur Hauptuntersuchung von PKW, die in Deutschland regelmäßig durchgeführt werden muss, eine Messung des Fahrgeräusches. Außerdem beginnt die Hauptuntersuchung seit dem 1. Juli 2012 mit einer Probefahrt von mindestens 8 km/h um elektronische Systeme zu aktivieren. Man also davon sprechen, dass das Fahrzeug ausgeführt wird und nicht nur statische sondern auch dynamische Prüfungen durchgeführt werden. Das Fahrzeug wird außerdem in einer Umgebung ausgeführt, die nicht dem täglichen Gebrauch entspricht, was mit der Ausführung des Spreadsheets durch eine Klassenbibliothek statt der eigentlichen Spreadsheet-Umgebung verglichen werden kann [Tue12].

4. Die Implementierung

In diesem Kapitel wird zunächst der IST Zustand der Implementierung von SIF vor der Erweiterung beschrieben und anschließend, wie SIF erweitert wurde. Beides wird bewusst etwas technischer beschrieben als in [Zit12], da das gut gemeinte Auslassen von technischen Details dort meines Erachtens das Verständnis eher behindert als gefördert hat. Ich werde dennoch nicht jede Klasse und Methode aufzählen, sondern lediglich die Wichtigsten nennen und ihre Beziehungen untereinander beschreiben. Die Implementierung von SIF verwendet dabei englische Beriffe als Klassennamen. Es werden also nicht die Begriffe der Metapher verwendet sondern englische Pendants mit den Präfixen „Abstract"für abstrakte Klassen oder „I"für Interfaces also Schnittstellen. In Tabelle 4.1 werden einige aus der Metapher bekannte Begriffe auf die englischen Klassennamen abgebildet.

Begriff in der Metapher	`Klassenname`	**Bemerkung**
(Empfang)	`FrontDesk`	Der Empfang ist in der Metapher nicht aufgeführt. Allerdings ist er für die Implementierung eine wichtige Komponente, da er die Schnittstelle zur aufrufenden Anwendung darstellt.
Inspektionsauftrag	`InspectionRequest`	
Prüffeld	`TestBay`	
Prüfstand	`AbstractTestFacility`	Je nach Implementierung als Unterklasse von `MonolithicTestFacility` oder `CompositeTestFacility`.
Richtlinie	`Policy`	
Verstoß	`IViolation`	
Vorschrift	`AbstractPolicyRule`	Je nach Implementierung der zugehörigen Prüfwerkstatt als Unterklasse von `MonolithicPolicyRule` oder `CompositePolicyRule`.
(Vorschriftsmanager)	`PolicyManager`	Dieser ist, wie der Empfang, nicht in der Metapher aufgeführt aber eine zentrale Komponente. Er bestimmt, welche Vorschrift mit welcher Prüfwerkstatt geprüft wird.

Tabelle 4.1.: Abbildung der Metapher auf Klassen

4.1. IST-Zustand

4.1.1. Aufbau

Das SIF Design besteht aus vier Paketen, die im Folgenden beschrieben sind. Abbildung 4.1 stellt die Pakete und die wichtigsten in ihnen enthaltenen Klassen dar. Damit die Abbildung nicht zu unübersichtlich wird werden lediglich die Beziehungen der Klassen über Paketgrenzen hinweg dargestellt.

FrontOffice

Das Paket `FrontOffice` enhält die Klasse `FrontDesk`, die die Schnittstelle nach außen darstellt und öffentliche Methoden anbietet um z.B. Inspektionsaufträge anzufragen oder einen Bericht zu erstellen. In `FrontOffice` enthalten sind außerdem die Klassen `PolicyManager` und `InspectionManager`, an die Anfragen von `FrontDesk` delegiert werden und die die Pakete `IO` und `TechicalDepartment` verwenden um diese Anfragen zu bearbeiten.

TechnicalDepartment

Im Paket `TechnicalDepartment` sind die Prüflogik und die Element-Scanner Implementierungen untergebracht. Die Klasse TechnicalManager deligiert die meisten Aufrufe an TestBayManager oder ScanningManager

IO

Mit den in diesem Paket enthaltenen Klassen können Spreadsheets ausgelesen und in das eigene Datenmodell transformiert werden. Schnittstelle nach zu anderen Paketen ist die Klasse `DataFacade`.

4.1.2. Allgemeines zum Hinzufügen neuer Vorschriften

Es ist schon in der ersten Ausbaustufe vorgesehen SIF um zusätzliche Vorschriften zu erweitern, indem Unterklassen von `MonolithicPolicyRule` bzw. `CompositeTestFacility` und `MonolithicTestFacility` bzw. `CompositePolicyRule` gebildet werden. Diese Unterklassen müssen beim Start bei der von `FrontOffice` verwendeten Instanz von `PolicyManager` registriert werden. Außerdem müssen die Klassenfelder, die durch die Anwendung konfigurierbar sein sollen im Code mit der Annotation `@ConfigurableParameter` markiert werden.

Das kann allerdings nur innerhalb von SIF selbst stattfinden, denn es existieren keine öffentlichen Methoden um Klassen beim `PolicyManager` zu registrieren. Außerdem existiert keine Methode, um die aktuell verwendete Instanz von `PolicyManager` zu ermitteln. Es ist

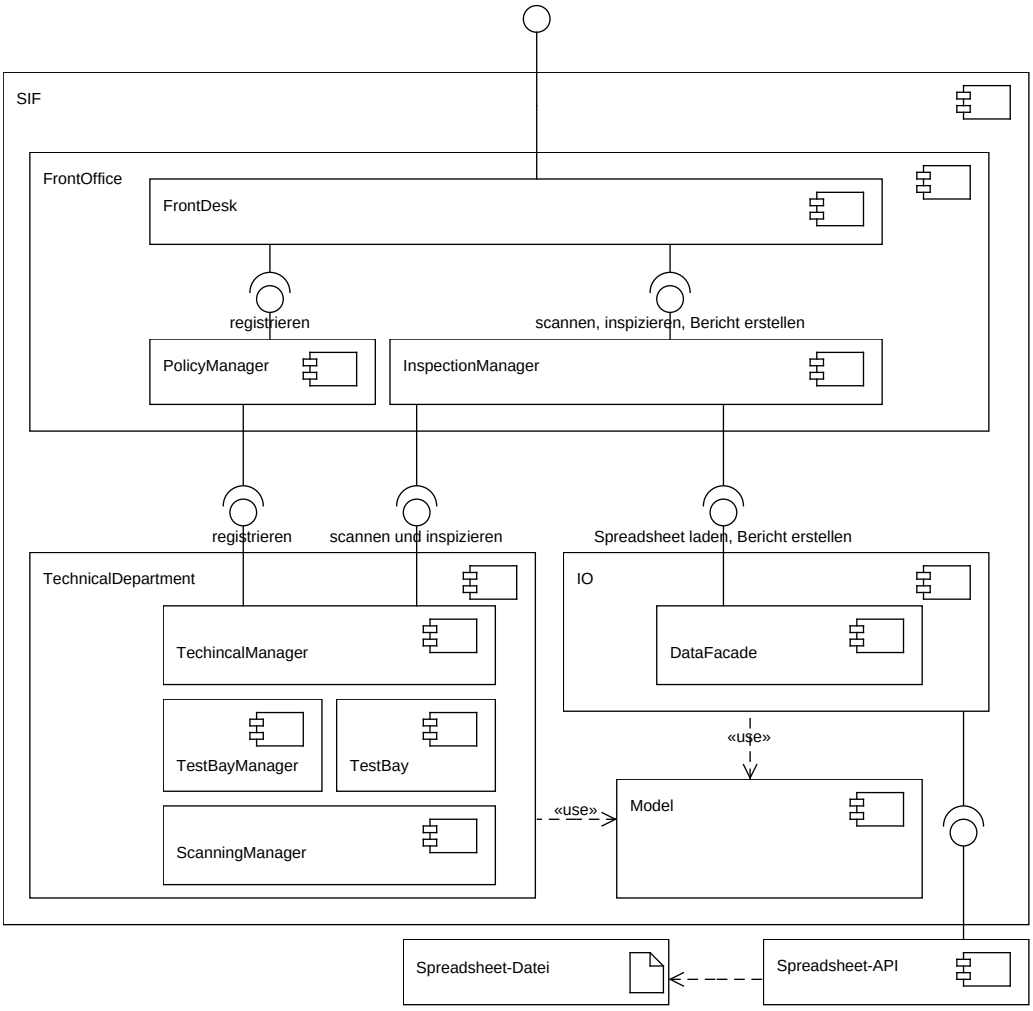

Abbildung 4.1.: Aufbau von SIF. Angelehnt an [Zit12, Abbildung 8.1], jedoch detailierter.

allerdings ohne größeren Aufwand möglich `FrontOffice` um solche Methoden zu erweitern, sodass auch eine auf Basis des Rahmenwerks entwickelte Anwendung eigene Vorschriften hinzufügen könnte.

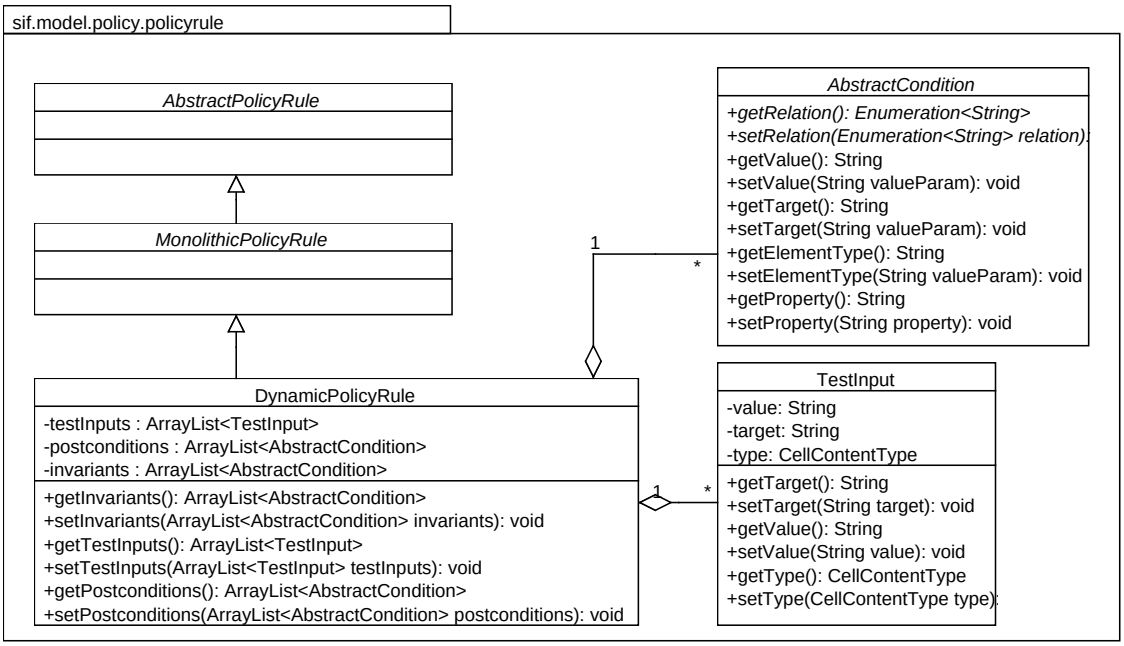

Abbildung 4.2.: Aufbau einer dynamischen Vorschrift. Details der abstrakten Elternklassen werden ausgelassen.

4.2. Erweiterung

4.2.1. Klassenmodell

In Kapitel 3.3 wurde spezifiziert, dass eine dynamische Vorschrift Testeingaben, Nachbedingungen und Invarianten haben soll. Nachbedingungen und Invarianten haben die gleichen Eigenschaften, lediglich der Prüfzeitpunkt unterscheidet sich. In Kapitel 3.3 wurde festgestellt, dass diese gleichen Eigenschaften ein Ziel, ein SOLL-Wert und eine Eigenschaftsangabe sind. Daher werden sie durch die selbe abstrakte Klasse implementiert: `AbstractCondition`. Je nachdem, ob es sich um einen Vergleich, ein Intervall oder ein Zählen von Elementen handelt handelt wird eine andere Spezialisierung verwendet (siehe 4.2). Das Ziel einer Bedingung wird durch zwei Attribute implementiert, nämlich `target` und `elementType`. `Target` wird verwendet, wenn ein Ziel über eine Addresse oder einen Namen direkt angesprochen werden kann. `ElementType` hingegen wird eingesetzt, wenn alle Spreadsheet-Elemente eines Typs, also auch Spezial-Elemente, die mit einem Element-Scanner erstellt wurden (siehe Kapitel 2.3.1) überprüft werden sollen.

Außerdem werden an verschiedenen Stellen Unterklassen der bestehenden Klassen gebildet.

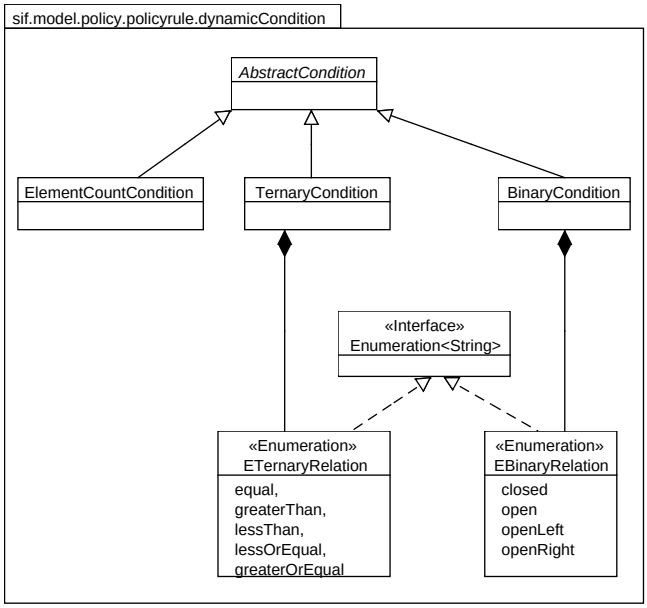

Abbildung 4.3.: Vererbungshierarchie der Bedingungen.

`DynamicPolicy` erweitert die bisherige Richtlinienklasse `Policy` um Listen für Ein- und Ausgabezellen sowie einen optionalen Spreadsheet-Pfad.

`DynamicInspectionRequest` erweitert die Implementierung des Inspektionsauftrags um eine Feld zum Halten des Spreadsheet-Modells der externen Bibliothek.

`DynamicTestFacility` erweitert `MonolithicTestFacility` und implementiert so einen neuen monolithischen Prüfstand. Sie besitzt außerdem eine Implementierung von IDynamicSpreadsheetRunner. Die Implementierung dieser Schnittstelle ist abhängig von der zum Lesen und Ausführen des Spreadsheets eingesetzten Klassenbibliothek.

Für die Überprüfung der Bedingungen wurden die abstrakte Klasse `AbstractConditionChecker` sowie die in Abbildung 4.3 dargestellten Implementierungen geschaffen. Diese Klassen implementieren ein modifiziertes *Schablonenmethode* Entwurfsmuster (Template Method Pattern), D.h. der konkrete Ablauf wird von `AbstractConditionChecker` vorgegeben und lediglich die Implementierung von Methoden, die Teilschritte wie die Überprüfung der Bedingung durchführen wird in den konkreten Klassen implementiert. Die Modifizierung des Entwurfsmusters besteht darin, dass die „Hauptmethode“ in der abstrakten Klasse nicht final ist um die Erweiterbarkeit nicht unnötig einzuschränken.

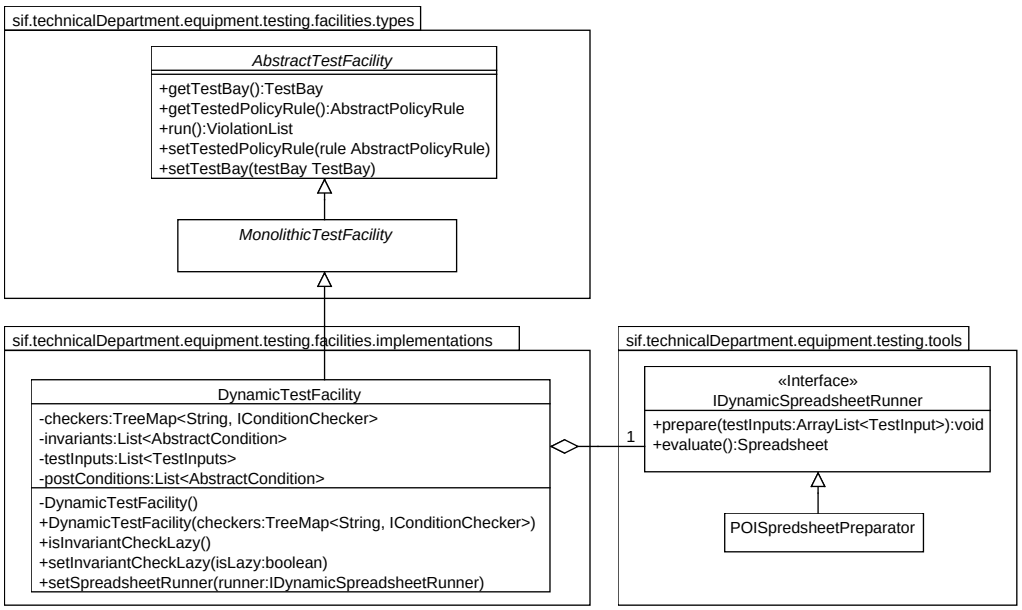

Abbildung 4.4.: Vererbungshierarchie der Prüfstände

4.2.2. Erweiterbarkeit

Austauschbare Komponenten Um die Austauschbarkeit von Komponenten wie den Klassen zum Einfügen der Testeingaben oder Kopieren zu gewährleisten, wurde vor allem in Paket `technicalDepartment` das *Strategie* Entwurfsmusters eingesetzt und insgesamt soweit möglich gegen Schnittstellen programmiert und nicht gegen konkrete Implementierungen, wie es im Sinne guter objektorientierter Entwicklung ohnehin getan werden sollte.

Hinzufügen neuer Bedingungen Um es zu ermöglichen, neue Bedingungen hinzuzufügen wurde in `FrontDesk` die Methode register(AbstractConditionl.class, AbstractConditionChecker.class) implementiert. So kann eine Anwendung neue Bedingungen hinzufügen ohne den Code von SIF verändern zu müssen.

4.2.3. Austauschformat

Die Serialisierung wird in das Paket IO integriert. Um das Auslesen des externen Datenmodells und die Transformation in das eigene Datenmodell zu trennen, wird die Klasse DataFacade so angepasst, dass sie diese Tätigkeiten als getrennte Methoden anbietet. Zum Serialisieren der `DynamicPolicy` Objekte sind die Getter der Felder, die im Austauschformat vorhanden sein sollen mit Annotationen versehen, die JAXB mitteilen, ob das Feld als Unterelement oder als Attribut seiner Klasse zu serialisieren ist. In diesen Annotationen ist

außerdem angegeben, welchen Namen das Element oder Attribut tragen soll. Für Klassen, die nicht von Haus aus mit JAXB serialisierbar sind, werden Implementierungen der JAXB Klasse XMLJavaAdapter erstellt, um JAXB so mitzuteilen wie sie zu serialisieren sind.

Umsetzung

Um eine Richtlinie abzubilden, sollen folgende XML Elemente eingeführt werden:

`dynamicPolicy` ist das Wurzelelement. Es stellt eine Richtlinie dar. Ihm können optional als Attribute ein Name, ein Autor und eine Beschreibung hinzugefügt werden. Diese Attribute tragen die Namen name, author und description. Es besitzt folgende Unterelemente:

- `rules`: In dieses Unterelement können beliebig viele Vorschriften (`rule` Elemente) eingefügt werden.
- `inputCells` / outputCells: In diesen Elementen sind die erkannten Ein- und Ausgabezellen gespeichert. Es kann beliebig viele `inputCell` bzw. `outputCell` Elemente beinhalten.
- `spreadsheetFilePath`: Der Dateipfad zu dem Spreadsheet, auf das sich die Richtlinie bezieht.

`rule` stellt eine Vorschrift dar. Optional können ein Autor (`author`), ein Name (`name` und eine Gewichtung (`severityWeight`) in Form von Attributen angegeben werden. Invarianten, Testeingaben und Nachbedingungen können in den Unterelemente `invariants` `testInputs` und `postconditions` eingefügt werden.

`testInput` stellt eine Testeingabe dar. Für eine Testeingabe müssen das Ziel, Zelltyp und der Wert angegeben werden. Hierfür existieren die Unterelemente `target`, `type` und `value`.

`compare` ist eine Bedingung und stellt einen Wertevergleich dar. Das Element kann also in `invariants` oder `postconditions` eingefügt werden. Das Element besitzt folgende Unterelemente:

- `target`: Hiermit kann das Ziel entweder in A1 Notation oder per Name angegeben werden.
- `elementType`: Kann alternativ zu `target` eingesetzt werden. So lässt sich eine Menge von Spreadsheet-Elementen eines Typs als Ziel angeben. Die Bedingung wird für alle im Spreadsheet enthaltenen Elemente dieses Typs überprüft.
- `value`: Der SOLL-Wert.
- `relation`: Die Art des Vergleichs. Mögliche Werte dieses Unterelements sind *equal*, *greaterThan*, *lessThan*, *lessOrEqual* und *greaterEqual*.

`interval` ist ebenfalls eine Bedingung und stellt die Überprüfung dar, ob sich der Zellwert im angegebenen Intervall befindet. Das Element besitzt die gleichen Unterelmente wie `compare` für Ziel, (ersten) Wert und die genaue Ausgestaltung. Der Wert wird als untere grenze des Intervalls betrachtet. Zusätzlich muss noch das Element `value2` angegeben werden um die obere Intervallgrenze zu bestimmen. Die möglichen Werte bei `relation` unterscheiden sich. Hier sind *open*, *closed*, *openLeft* und *openRight* möglich.

`count` ist die dritte Bedingung und stellt die Zählung der Elemente einer Klasse dar. Bei ihr kann das Ziel nur mit `elementType` angegeben werden. Der durch `value` angegebene Wert entspricht der erwarteten Anzahl an Elementen, der angegebenen Klasse.

5. Evaluation

Um zu untersuchen wie gut die erstellte zweite Ausbaustufe von SIF geeignet ist um dynamische Prüfungen zu spezifizieren und durchzuführen wurde eine Benutzerevaluation durchgeführt. Dazu wurde zusätzlich zu der im vorigen Kapitel beschriebenen Implementierung eine Kommandozeilen-Schnittstelle erstellt, die es erlaubt SIF wie in 3.2 dargestellt auszuführen. Im Kontext der Umfrage wurde statt dem Begriff „Vorschrift“ der Begriff „Regel“ gebraucht, da der Sinn des Elements `rule` so deutlicher wurde und Endanwender keinen Bezug zur hauptsächlich für die Entwicklung relevanten Metapher haben.

5.1. Rahmenbedingungen

5.1.1. Arbeitsumgebung der Probanden

Als Werkzeug um die Spezifikation im XML Format zu erstellen wurde der XML Editor der Entwicklungsumgebung Eclipse eingesetzt. Dieser Editor bietet Hilfsmittel wie Autovervollständigung und kann sowohl im Quelltext-Modus als auch im graphischen Modus verwendet werden. Als Spreadsheet-Umgebung wurde LibreOffice 3.6.3.2 unter Kubuntu 12.04 eingesetzt.

5.1.2. Prüfling

Als zu prüfendes Spreadsheet wurde das in Abbildung 5.2 dargestellte Spreadsheet verwendet. Dieses Spreadsheet implementiert einen BaFöG Rechner, mit dessen Hilfe Erlasse auf den Rückzahlungsbetrag des BaFöG Darlehens berechnen werde können. Zellen, in denen Anwender Eingaben tätigen können sind gelb. Zellen, in denen Berechnungen dargestellt werden sind stattdessen grau. Eine Ausnahme bildet die Zelle, in der der Rückzahlungsbetrag bei vorzeitiger Rückzahlung dargestellt wird. Diese ist die einzige reine Ausgabezelle, da sie nicht referenziert wird und wird grün dargestellt. Die restlichen Zellen, in denen Formeln enthalten sind, sind also Zwischenzellen. Dieses Spreadsheet hat allerdings auch schon einen Fehler: Wenn die Summe aus den Feldern B17 und B19 kleiner ist als die Summe aus B8 und B10 werden negative Werte berechnet. Der BaFöG Empfänger würde also nochmals Geld vom BaFöG Amt erhalten.

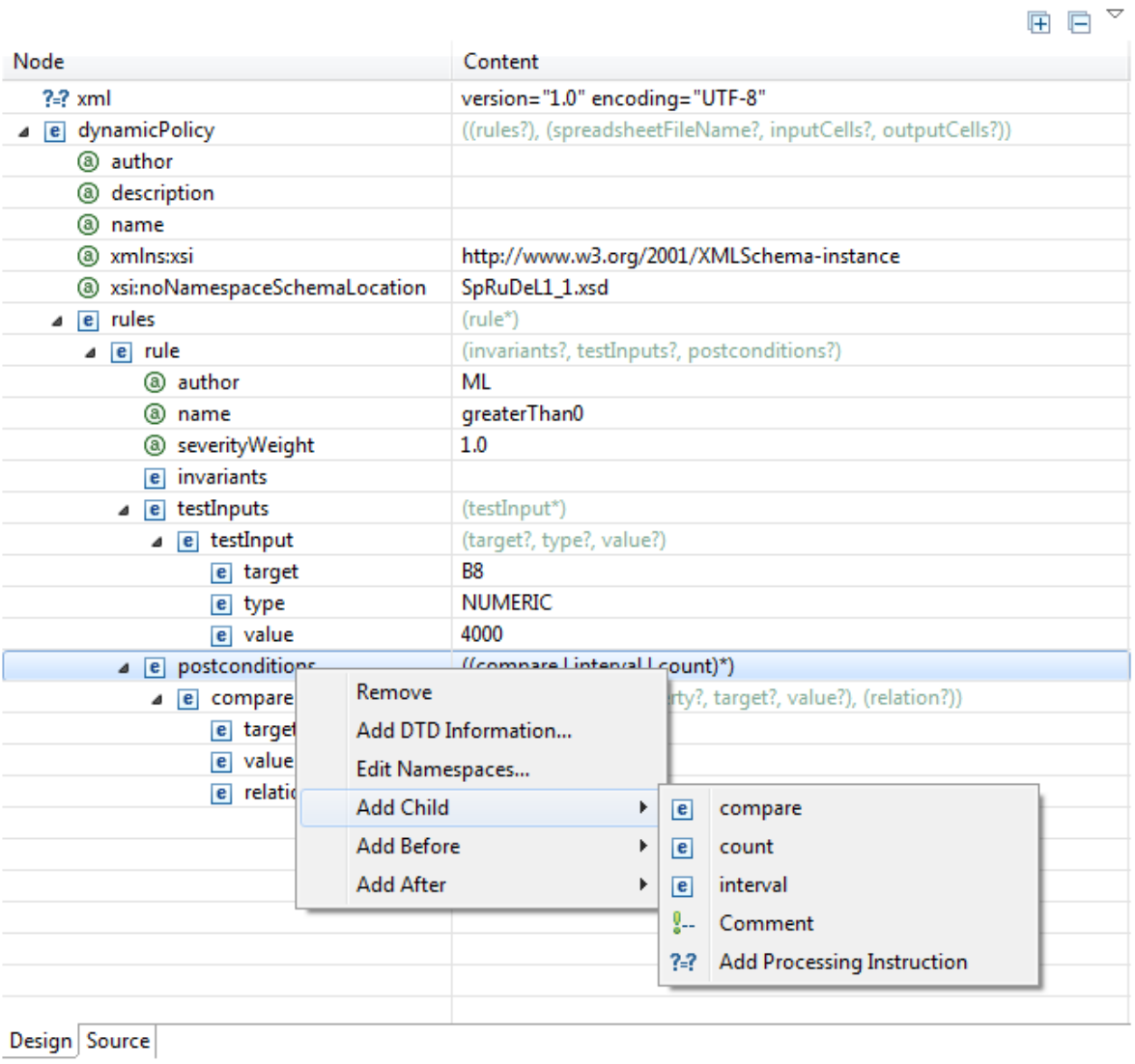

Abbildung 5.1.: Der Eclipse XML Editor im graphischen Modus

5.1.3. Aufgabenstellung

Um zu untersuchen, wie gut die Probanden in der Lage sind das XML Austauschformat zu verwenden um Vorschriften zu spezifizieren, wurden ihnen eine Beschreibung der Sprachelemente auf zwei Seiten ausgehändigt und es wurden zwei Aufgaben gestellt. Die erste Aufgabe bestand darin, eine Regel zu spezifizieren, die vorschreibt, dass „Restschuld Gesamt“ nie den Wert 10.000 übersteigt. Hiermit sollte untersucht werden, wie gut die Probanden den Unterschied zwischen Invarianten und Nachbedingungen mit Testeingaben erfassen können und ob sie in der Lage sind, einfache Regeln zu spezifizieren. Für die zweite Aufgabe wurde die Berechnung des Rückzahlungsbetrags bei vorzeitiger Rückzahlung so

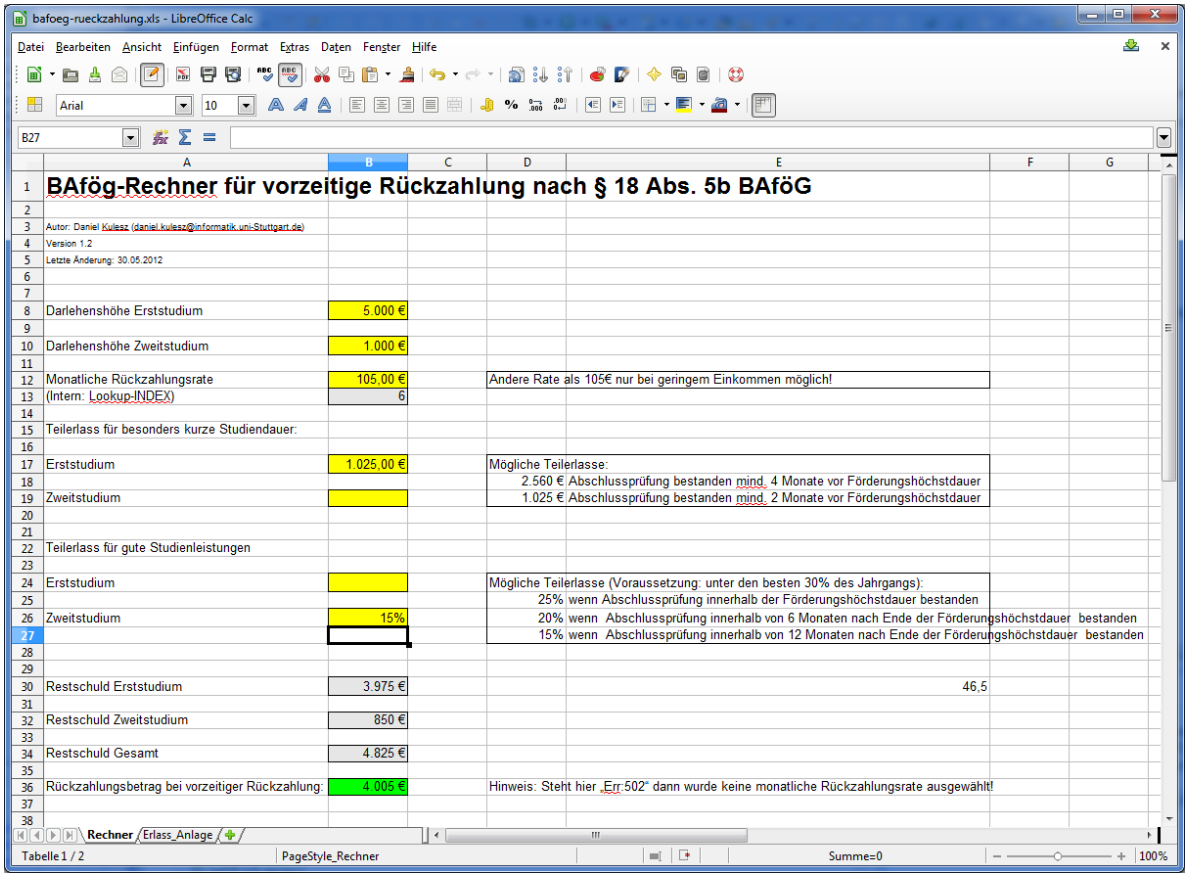

Abbildung 5.2.: Zur Evaluation verwendetes Spreadsheet

modifiziert, dass bei einer Restschuld von weniger als 10.000 ein falsches Ergebnis berechnet wird. In der zweiten Aufgabe wurde erklärt, dass das zweite Spreadsheet verändert wurde und das erste Spreadsheet als Test-Orakel verwendet werden kann. Anschließend wurde die Aufgabe gestellt, einen Testfall zu spezifizieren, der einen Verstoß erzeugt.

5.1.4. Die Probanden

Vor der eigentlich Evaluation wurde eine Pilot-Evaluation mit drei Probanden durchgeführt. Dabei viel auf, dass das modifizierte Spreadsheet auf den Rechnern der Probanden in der falschen Version kopiert wurde, wodurch es etwas seltener Fehler produzierte und so die Bearbeitung der zweiten Aufgabe erschwerte.

Leider fanden sich für die eigentliche Evaluation weniger Probanden als erwartet. Dadurch konnte die Evaluation nur mit fünf weiteren Probanden durchgeführt werden. Außerdem bestanden die Probanden ausschließlich aus Informatik und Softwaretechnik Studenten sowie Mitarbeitern des Instituts für Softwaretechnik. Die Evaluation lässt also keine Schlüsse

Frage	Ergebnis im Median
Verständlichkeit der Aufgabenstellung	Leicht
Verständlichkeit der Sprach-Beschreibung	Moderat
Schwierigkeit: Erstellen von Regeln	Leicht bis moderat

Tabelle 5.1.: Median der Fragen zur Schwierigkeit

darauf zu, wie erfolgreich ein Endanwender ohne Erfahrung auf diesem Gebiet Richtlinien spezifizieren kann.

5.2. Resultate

5.2.1. Spezifizieren von Regeln

Aufgabe 1 Alle Probanden konnten Aufgabe 1 lösen und eine Invariante erstellen, die vorschreibt, dass „Restschuld Gesamt“, also der Wert aus Zelle B34 nicht größer als 10.000 werden darf.

Aufgabe 2 Nur einem Probanden gelang es, Aufgabe 2 zu lösen, also eine Vorschrift zu erstellen, die einen richtig positiven Verstoß meldete. Ein anderer Proband erstellte eine Regel, die einen Verstoß meldete, der durch den schon im Original vorhandenen Fehler ausgelöst wurde. Zwei weitere Probanden erstellten zwar Regeln, die eigentlich zu einem Verstoß geführt hätten aber durch Syntaxfehler, die z.T. aus fehlenden Informationen in der Sprachbeschreibung herrührten, nicht durchgeführt werden konnten.

5.2.2. Umfrage

Nach Abschluss der Aufgaben erhielten die Probanden außerdem eine Umfrage. Dabei wurde zunächst nach der Verständlichkeit der Aufgabenstellung und der Sprachbeschreibung gefragt. Anschließend wurde gefragt, wie schwer es dem Probanden gefallen ist Vorschriften zu erstellen. Die Antwortmöglichkeiten dieser drei Fragen waren jeweils die gleichen und reichten von „sehr leicht“ bis „sehr schwer“. Es folgten drei Fragen, die den Sinn hatten, die Erfahrung des Probanden mit XML zu erfassen. Das Median der Ergebnisse der ersten drei Fragen dieser Umfrage ist in Tabelle 5.1 dargestellt.

Zusammenfassung Zusammenfassend lässt sich sagen, dass die Evaluation gezeigt hat, dass das die Kombination aus XML Austauschformat und SIF nicht geeignet war um innerhalb kurzer Zeit komplexe Regeln zu spezifizieren. Da es sich bei den Probanden um Personen mit technischem Hintergrund handelte ist anzunehmen, dass Personen mit weniger technischem Hintergrund noch größere Schwierigkeiten haben werden.

5.2.3. Fehler in SIF

Durch die Evaluation fielen mehrere Probleme in SIF auf. Einer davon war schon in der ersten Ausbaustufe vorhanden. Er bewirkt, dass Werte mit mehreren Nachkommastellen falsch ausgelesen werden und wird eventuell durch die Klassenbibliothek Apache POI verursacht. Ein anderer Fehler kam in der zweiten Ausbaustufe hinzu. Es handelt sich um die Tatsache, dass sich als TestInput alle Werte angeben lassen, also auch Werte, die durch eine Gültigkeitsprüfung im Spreadsheet nicht zugelassen sind. Es wurde zwar nicht spezifiziert, dass das nicht möglich sein soll und kann auch in einer Spreadsheet-Umgebung durch den Endanwender mit „Copy and Paste“ erfolgen, stellt aber dennoch keine normale Nutzung des Spreadsheets dar und sollte somit nicht möglich sein. Außerdem viel auf, dass die meisten Benutzer nicht erwarteten, dass Prozentwerte als Wert zwischen 0 und 1 angegeben werden sollten und eine Eingabe von einem Wert zwischen 0 und 100 gefolgt von einem Prozentzeichen zu Fehlern in der Formelberechnung von Apache POI führt. Im Fall von Zellen, die als Prozentwerte formatiert sind sollte also eine Konversion in SIF implementiert werden.

6. Rückblick

Rückblickend lässt sich sagen, dass das Austauschformat zu viel Zeit in Anspruch genommen hat. Zwar war die Zeit, die für die Suche nach einer geeigneten Sprache aufgewendet wurde begrenzt, aber die anschließende Konzipierung und Implementierung des Austauschformats haben mehr Aufwand verursacht, als gedacht. Diese Zeit wäre vermutlich besser investiert gewesen, wenn stattdessen ein genaueres und weitere Bestandteile umfassendes Konzept erstellt worden wäre. Die Evaluation hätte entweder in einem größeren Rahmen stattfinden oder ganz aufgegeben werden müssen, da die daraus gewonnenen Erkenntnisse nicht besonders umfangreich und vor allem nicht repräsentativ sind.

6.1. Zukünftige Arbeiten

Die möglichen zukünftigen Arbeiten auf Basis von SIF sind umfangreich. Der meiner Meinung nach nächste Schritt sollte die Erstellung eines graphischen Editors für Richtlinien sein, da dieser den Benutzer bei der Erstellung entlastet und die Schnittstelle

A. Anhang

A.1. Aufbau der CD-ROM

Die beiliegende CD-ROM ist folgendermaßen aufgebaut:

Ausarbeitung Der Ordner ausarbeitung enthält dieses Dokument im PDF-Format.

Evaluation Der Ordner evaluation enthält die Evaluations Ordner in dem Zustand wie sie nach der Bearbeitung durch die Probanden jeweils waren.

Implementierung Der Ordner Implementierung enthält die Eclipse Projekte und den Quell-Code des Spreadsheet Inspection Framework und der Kommandozeilen-Schnittstelle .

UML Der Ordner UML enthält die UML Diagramme aus dem Bericht im UMLet Format.

A.2. Evaluation

Evaluation: Dynamische Spreadsheetprüfung mit SIF

Spreadsheets werden in Unternehmen und Verwaltung in hohem Maß eingesetzt, sind jedoch sehr häufig fehlerhaft. Mit der XML-basierten Sprache SpRuDeL lassen sich Regeln für Spreadsheets formulieren, die sich anschließend mit SIF (Spreadsheet Inspection Framework) überprüfen lassen.

In dieser Evaluation soll die Qualität eines Spreadsheets gesichert werden, mit dessen Hilfe sich Erlasse (Vergünstigungen) auf das BAFöG Darlehen berechnen lassen. Diese Erlasse werden gewährt, wenn ein Studium besonders schnell oder mit besonders guten Leistungen abgeschlossen wird bzw. wenn das Darlehen vorzeitig zurückgezahlt wird.

Aufgabenstellung

Auf Ihren Rechnern finden Sie im Home Ordner das Spreadsheet „bafoeg-rueckzahlung.xls“. Mit Hilfe dieses Spreadsheets kann der Rückzahlungsbetrag des BAFöG unter Berücksichtigung der Erlasse berechnet werden. Außerdem befindet sich dort eine Vorlage für Regeln mit dem Namen „rules.xml“.
Öffnen Sie diese Vorlage mit Eclipse, um Regeln zu erstellen. Eine Beschreibung der Sprache finden Sie auf den folgenden Seiten! Mit den Skripten „check_orig.sh“ und „check_mod.sh“ können Sie Ihre Regeln auf Spreadsheet 1 bzw. 2 anwenden. Dabei wird ein Report in /reports erstellt.

1. Erstellen Sie eine Regel, die sicherstellt, dass Restschuld Gesamt immer kleiner oder gleich 10.000€ ist.
2. Das Spreadsheet bafoeg-rueckzahlung.xls arbeitete bisher korrekt. Ein Kommilitone hat das Spreadsheet verändert und als bafoeg-rueckzahlung_mod.xls abgespeichert.
   Erstellen Sie eine Regel mit Testeingaben, die überprüft, ob das Spreadsheet noch korrekt rechnet (also die gleichen Ergebnisse liefert wie bafoeg-rueckzahlung.xls).

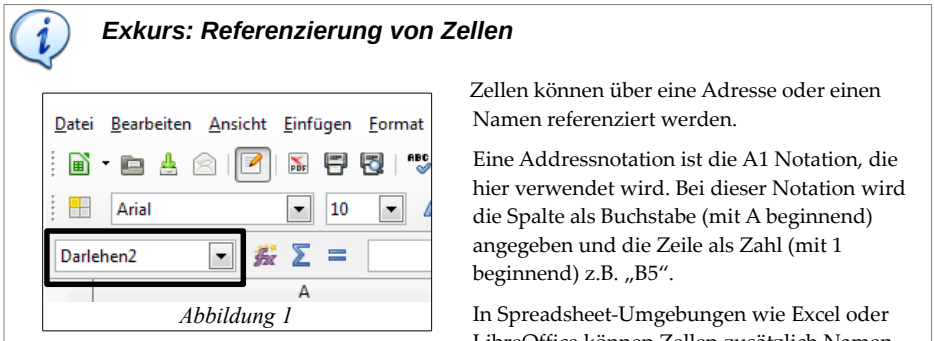

Abbildung 1

Zellen können über eine Adresse oder einen Namen referenziert werden.

Eine Addressnotation ist die A1 Notation, die hier verwendet wird. Bei dieser Notation wird die Spalte als Buchstabe (mit A beginnend) angegeben und die Zeile als Zahl (mit 1 beginnend) z.B. „B5“.

In Spreadsheet-Umgebungen wie Excel oder LibreOffice können Zellen zusätzlich Namen zugewiesen werden. Diese können über das in Abbildung 1 dargestellte Textfeld betrachtet und verändert werden.

1

Die Beschreibungssprache SpRuDeL (Spreadsheet Rule Definition Language)

Der Aufbau eines SpRuDeL Dokuments ist folgendermaßen (* = 0...∞, ? = 0...1):

```
<dynamicPolicy>
      <rules>
            <rule name="[string]"> *
                  <invariants>
                        (<compare /> | <compare /> |<compare /> )
                  </invariants> ?
                  <testInputs>

                  </testInputs> ?
                  <postconditions>

                  </postconditions> ?
            </rule>
      </rules>
      <inputCells></inputCells> ?
      <outputCells></outputCells> ?
</dynamicPolicy>
```

? = 0 ... 1
* = 0 ... ∞

Die Testeingaben einer Regel lassen sich durch beliebig viele `testInput` Elemente innerhalb von **`testInputs`** spezifizieren. `TestInputs` hat folgende Unterelemente:

- **`target`**: Gibt die Zelle als A1-Addresse (z.B. „B5“) bzw. Zellname an, in die geschrieben werden soll. Der Name einer Zelle wird im in Abbildung 1 dargestellten Textfeld angezeigt.
- **`type`**: Hier muss der Typ des Zelleninhalts der geschrieben werden soll ausgewählt werden
- **`value`**: Beinhaltet den Wert, der geschrieben werden soll. Hier kann eine Zahl mit einem Punkt als Trennzeichen, ein String oder true bzw. false eingetragen werden.

```
<testInput>
      <target>[A1Adress] or [Name]</target>
      <type>NUMERIC, TEXT, BOOLEAN, BLANK, ERROR</type>
      <value>[double], [string], true/false</value>
</testInput>
```

In die Elemente **`invariants`** und **`postconditions`** können Bedingungen eingefügt werden. Bedingungen in `invariants` müssen immer gelten. Bedingungen in `postconditions` hingegen nur nach dem Einfügen der Testeingaben aus `testInputs`
Solche Bedingungen besitzen jeweils mehrere der folgenden Unterelemente:

- **`target bzw. elementType:`**
  Das Ziel einer Bedingung kann entweder wie bei einem TestInput durch `target` oder durch `elementType` angegeben werden. Mit `elementType` lassen sich alle Spreadsheet-Elemente eines bestimmten Typs überprüfen. Dazu gehören unter anderem:
  - **`InputCell:`** Eingabezellen, also Zellen auf die verwiesen wird aber die nicht selbst auf andere Zellen verweisen
  - **`OutputCell:`** Ausgabezellen, also Zellen die auf andere Zellen verweisen aber auf die nicht von anderen Zellen aus verwiesen wird.
  - **`IntermediateCell:`** Zellen mit ein- und ausgehenden Verweisen.

- **value:**
  Spezifiziert den erwarteten Wert der Invariante oder Nachbedingung. Intervalle besitzen zusätzlich noch eine Obergrenze, die mit dem Element **value2** angegeben wird.
- **relation:**
  Legt die Art des Vergleichs oder des Intervalls fest.

Es gibt drei Arten von Bedingungen: compare, interval und count:

- **compare**: Beschreibt einen Wertevergleich also z.B. „Zelle B5 muss größer 0 sein“.

```
<compare> *
      (<target>[A1Adress] or [Name]</target>
          | <elementType>[sif class name]</elementType>)
      <property>[sif class property]<property> ?
      <value>[double]</value>
      <relation>[equal, greaterThan, greaterOrEqual, lessThan, lessOrEqual]
      </relation>
</compare>
```

? = 0 ... 1
* = 0 ... ∞
| = oder

- **interval**: Hiermit lässt sich überprüfen, ob sich ein Wert sich innerhalb des angegebenen Intervalls befindet. `Open` bedeutet hierbei, dass das die Intervallgrenzen zum Intervall gehören während `closed` bedeutet, dass die Intervallgrenzen nicht zum Intervall gehören. Mit `openLeft` bzw. `openRight` kann die linke (untere) bzw. rechte (obere) Intervallgrenze als offen definiert werden.

```
<interval> *
      (<target>[A1Adress] or [Name]</target>
          | <elementType>[sif class name]</elementType>)
      <property>[sif class property]<property> ?
      <value>[double]</value>
      <value2>[double]</value2>
      <relation>[open, closed, openLeft, openRight]</relation>
</interval>
```

- **count**: Überprüft die Anzahl der Vorkommnisse eines bestimmten Spreadsheet Elements(Eingabezellen, Referenzen, ...). Mögliche Werte für `elementType` siehe oben.

```
<count> *
      <elementType>[sif class name]</elementType>
      <value>[int]</value>
</count>
```

Umfrage: Anhang zur dynamischen Spreadsheetprüfung *Autor: Manuel Lemcke, Universität Stuttgart, Institut für Softwaretechnologie*

Dynamische Spreadsheetprüfung

28. Wie verständlich fanden Sie die Aufgabenstellung?
- ☐ Sehr leicht
- ☐ Leicht
- ☐ Moderat
- ☐ Schwer
- ☐ Sehr schwer

29. Wie verständlich fanden Sie die Regel-Beschreibung?
- ☐ Sehr leicht
- ☐ Leicht
- ☐ Moderat
- ☐ Schwer
- ☐ Sehr schwer

30. Wie schwer ist es Ihnen gefallen, die Regeln zu erstellen?
- ☐ Sehr leicht
- ☐ Leicht
- ☐ Moderat
- ☐ Schwer
- ☐ Sehr schwer

31. Haben Sie einen graphischen XML Editor verwendet um die Aufgabe zu lösen?
- ☐ Ja, ich habe einen graphischen XML Editor verwendet
- ☐ Nein, ich habe den Quelltext bearbeitet

32. Erfahrung mit XML: Bearbeiten Sie gelegentlich XML Dokumente mit einem Editor?
- ☐ Ja
- ☐ Nein

33. Wann haben Sie das erste mal ein XML Dokument bearbeitet?
- ☐ Vor weniger als einem Jahr
- ☐ Vor weniger als drei Jahren
- ☐ Vor mehr als drei Jahren
- ☐ Noch nie

34. Sonstige Anmerkungen

1

Literaturverzeichnis

[ALSU07] A. Aho, M. Lam, R. Sethi, J. Ullman. **Compilers, principles, techniques, and tools**. Addison-Wesley series in computer science. Addison-Wesley Pub. Co., 2. Auflage, 2007. ISBN 978-0-321-54798-9. (Zitiert auf Seite 22)

[BCP+03] M. Burnett, C. Cook, O. Pendse, G. Rothermel, J. Summet, C. Wallace. End-user software engineering with assertions in the spreadsheet paradigm. In **Proceedings of the 25th International Conference on Software Engineering**, ICSE '03, S. 93–103. IEEE Computer Society, Washington, DC, USA, 2003. URL http://dl.acm.org/citation.cfm?id=776816.776828. (Zitiert auf den Seiten 7, 20 und 21)

[Cre05] E. Creswick. **Dynamic, Incremental Assertion Propagation in End-user Programming**. Oregon State University, 2005. URL http://hdl.handle.net/1957/22887. (Zitiert auf den Seiten 20 und 21)

[FLS04] K. Frühauf, J. Ludewig, H. Sandmayr. **Software-Prüfung : eine Anleitung zum Test und zur Inspektion**. vdf-Lehrbuch Informatik. vdf, Zürich, 5., überarb. Auflage, 2004. 168 S. : Ill., graph. Darst.; ISBN 3-7281-2906-2, deutsch. (Zitiert auf den Seiten 11 und 12)

[KAB+11] A. J. Ko, R. Abraham, L. Beckwith, A. Blackwell, M. Burnett, M. Erwig, C. Scaffidi, J. Lawrance, H. Lieberman, B. Myers, M. B. Rosson, G. Rothermel, M. Shaw, S. Wiedenbeck. The State of the Art in End-User Software Engineering. **ACM Computing Surveys**, 43(3):21ff, 2011. doi:10.1145/1922649.1922658. URL http://doi.acm.org/10.1145/1922649.1922658. (Zitiert auf Seite 12)

[Klu90] H.-P. Klug. **Nutzfahrzeug-Bremsanlagen: Aufbau und Funktion; Prüf- und Wartungsarbeiten**. Vogel, Würzburg, 2., überarb. u. erw. Auflage, 1990. (Zitiert auf Seite 27)

[LL07] J. Ludewig, H. Lichter. **Software Engineering - Grundlagen, Menschen, Prozesse, Techniken**. dpunkt.verlag, 2007. (Zitiert auf den Seiten 11 und 13)

[Pan08] R. R. Panko. What We Know About Spreadsheet Errors. **Journal of End User Computings**, Vol. 10, No 2:15–21, 1998, überarbeitet 2008. URL http://panko.shidler.hawaii.edu/ssr/Mypapers/whatknow.htm. (Zitiert auf den Seiten 9 und 12)

[PRM01] J. F. Pane, C. A. Ratanamahatana, B. A. Myers. Studying the language and structure in non-programmers' solutions to programming problems. **International Journal of Human-Computer Studies**, 54(2):237 – 264, 2001. doi:10.1006/ijhc.2000.0410. URL http://www.sciencedirect.com/science/article/pii/S1071581900904105. (Zitiert auf Seite 21)

[Rot95] W. Rothmann. **Bremsprüfstand für Kraftfahrzeuge, insbesondere Pkw, mit ABS-Bremsanlagen**. EP0280785. Satoriusstrasse 11, Essen 1, 1995. URL http://www.freepatentsonline.com/EP0280785B2.html. (Zitiert auf Seite 27)

[SSM05] C. Scaffidi, M. Shaw, B. Myers. Estimating the numbers of end users and end user programmers. In **Visual Languages and Human-Centric Computing, 2005 IEEE Symposium on**, S. 207 – 214. 2005. doi:10.1109/VLHCC.2005.34. (Zitiert auf Seite 9)

[Tue12] Infoblatt des TÜV SÜD zur 47. Änderungsverordnung der Hauptuntersuchung, 2012. URL http://www.tuev-sued.de/hu2012. TÜV SÜD AutoService GmbH. (Zitiert auf Seite 27)

[Zit12] S. Zitzelsberger. **Fehlererkennung in Spreadsheets**. Diplomarbeit, Universität Stuttgart, Holzgartenstr. 16, 70174 Stuttgart, 2012. URL http://elib.uni-stuttgart.de/opus/volltexte/2012/7056. (Zitiert auf den Seiten 7, 13, 14, 15, 24, 29 und 31)

Alle URLs wurden zuletzt am 25. 02. 2013 geprüft.

Erklärung

Ich versichere, diese Arbeit selbstständig verfasst zu haben. Ich habe keine anderen als die angegebenen Quellen benutzt und alle wörtlich oder sinngemäß aus anderen Werken übernommene Aussagen als solche gekennzeichnet. Weder diese Arbeit noch wesentliche Teile daraus waren bisher Gegenstand eines anderen Prüfungsverfahrens. Ich habe diese Arbeit bisher weder teilweise noch vollständig veröffentlicht. Das elektronische Exemplar stimmt mit allen eingereichten Exemplaren überein.

Ort, Datum, Unterschrift