PRAXISBERICHT

THEMA

Analyse von Arbeitserleichterungen durch die Nutzung von xText im Workflow zur Konfiguration von Pflichtprüfungen für profil c/s

Eingereicht von: Niels Gundermann

Hiermit versichere ich, dass ich die vorliegende Arbeit selbstständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel benutzt sowie Zitate kenntlich gemacht habe.

Neubrandenburg, Januar 2015

Niels Gundermann

Inhaltsverzeichnis

Abbildungsverzeichnis

Listings

1 Einleitung

Geschäftsprozessplanung und -optimierung in einem Unternehmen haben zum Ziel, die Kosten eines Produktes zu senken, die Produktionszeit herabzusetzen oder die Qualität zu verbessern. [SGS08, S. 56] Eine Optimierung kann mit unterschiedlichen Mitteln erreicht werden. Beispielsweise können neue und effizientere Arbeitsverfahren eingesetzt werden. Auch die Verwendung neuerer Tools und Frameworks, können dem Bearbeiter Routinearbeiten abnehmen.
In diesem Bericht soll analysiert werden, inwiefern *Xtext* für die data experts gmbH (deg) in Bezug auf Kosten- und Zeiteinsparung von Nutzen ist.

1.1 Vorgehen

Nach der Definition einiger Begriffe wird ein entsprechender Geschäftsprozess aus der deg erklärt. Die Arbeitsabläufe werden dahingehend untersucht, ob *Xtext* theoretisch zur Anwendung kommen kann. Anschließend wird analysiert wie *Xtext* konkret in den entsprechenden Arbeitsabläufen zu einer Kosten- oder Zeiteinsparung führen könnte.

2 Grundlegende Begriffe

In Kapitel 3 wird Bezug auf die Begriffe *Geschäftsprozess (Prozess)* und *Workflow* genommen. Aus diesem Grund wird hier eine Abgrenzung zwischen den beiden Begriffen vorgenommen. Weiterhin werden in Kapitel 4 die Begriffe *Syntax* und *Semantik* verwendet. Wie diese Begriffe in diesem Bericht zu verstehen sind, ist ebenfalls in diesem Kapitel definiert.

Geschäftsprozess

Unter einem Geschäftsprozess (Prozess) versteht man eine Zusammenfassung der erforderlichen betrieblichen Abläufe, die zur Erstellung von Produkten und Dienstleistungen notwendig sind. (vgl. [PA05, S.4]) Bei der Darstellung mittels der EPK[1] (welche in diesem Bericht verwendet wird) werden Prozesse durch Zustände und Funktionen visualisiert.

Workflow

Ein Workflow beschreibt die beteiligten Personen und die von ihnen auszuführenden Arbeitsschritte innerhalb des Prozesses. [fESE14] Es handelt sich um eine genaue Beschreibung der Arbeitsschritte, die abgearbeitet werden müssen, um den Prozess von Anfang bis Ende durchzuführen.

Syntax

Die Syntax ist ein Bestandteil einer Sprache. Durch die Syntax wird die Notation der Sprache definiert. Sie beschreibt demnach die Sprachkonstrukte die für eine Sprache zulässig sind und mit denen man sich in dieser Sprache ausdrücken kann. In Bezug auf Programmiersprachen beschreibt die Syntax somit die Sprachkonstrukte, mit denen der Nutzer ein Programm beschreiben kann. (vgl. [MSL+13, S.26]) Zur Beschreibung einer Syntax wird eine *Grammatik* verwendet[2]. (vgl. [Hed12, S.26])

[1] Ereignisgesteuerte Prozesskette

[2] Beschreibung einer Grammatik ist im Praxisbericht *Entwicklung einer Grammatik für eine DSL mit xText am Beispiel einer Sprache zur Definition von Pflichtprüfungen in profil c/s* [Gun14] zu finden.

Semantik

Die Semantik ist ebenfalls ein Bestandteil einer Sprache. Sie enthält die Regeln für die strukturelle Zusammensetzung der Sprachkonstrukte, die von der Syntax definiert werden. Bezogen auf Programmiersprachen beschreibt die Semantik die Bedeutung aufeinander folgender Sprachkonstrukte. (vgl. [MSL+13, S.26])

Xtext

Xtext ist ein Framework mit dem es einerseits möglich ist, in sehr kurzer Zeit neue Sprachen zu entwickeln. Andererseits bietet es auch für bestehende Sprachen in Verbindung mit Eclipse ein entsprechendes Tooling an, mit dem sich die Arbeit mit dieser Sprache effizienter gestalten lässt. Das äußert sich darin, dass mit *Xtext* Editoren generiert werden können die Fehler in der Sprache erkennen und den Nutzer darauf hinweisen. Diese Fehler werden dabei wie gewohnt in der Eclipse IDE angezeigt. (vgl. [114, S.113])

3 Beschreibung des Prozesses

Für die Analyse in dieser Arbeit wurde der Prozess zur Umsetzung einer Pflichtprüfung innerhalb von profil c/s herangezogen.[3] Der Workflow dieses Prozesses umfasst alle Arbeitsschritte von der Anforderung des Kunden, über die Analyse und Implementierung der Pflichtprüfung, bis hin zur Abnahme der Umsetzung durch den Kunden. Um die Übersicht zu behalten, ist es sinnvoll den Gesamtprozess in drei Teilprozesse einzuteilen, die in den nachfolgenden Kapiteln beschrieben werden.

3.1 Anforderung und Analyse

Am Anfang des Prozesses steht immer eine Anforderung vom Kunden, die in der deg analysiert werden muss. Die deg erstellt für diese Anforderung ein entsprechendes Angebot. In diesem Beispiel wird davon ausgegangen, dass zur Umsetzung der Anforderung eine neue Pflichtprüfung implementiert werden muss. Nach der Annahme des Angebotes durch den Kunden, ist dieser Teilprozess (Abbildung 1) beendet.

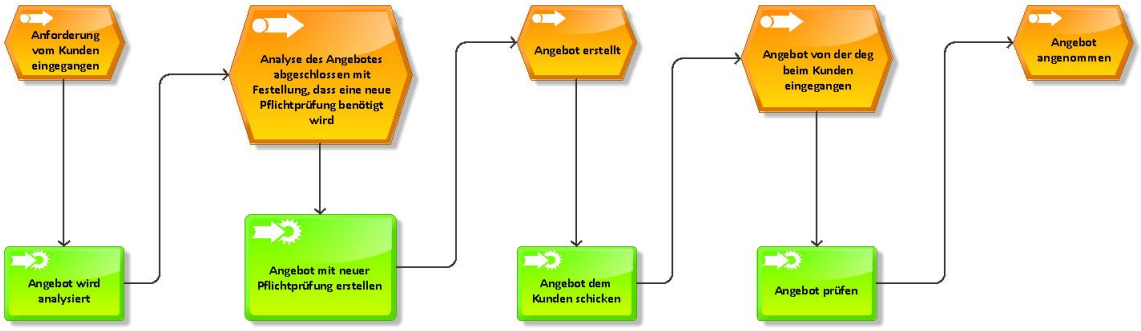

Abbildung 1: Teilprozess: Anforderung und Analyse

3.2 Umsetzung

Nach der Annahme des Angebotes wird der Algorithmus für die neue Pflichtprüfung von den Mitarbeitern der deg implementiert und die Prüfungskonfiguration definiert. Im Anschluss daran wird ein Codereview durch-

[3] Informationen über *Pflichtprüfungen* und *profil c/s* sind im Praxisbericht: *Entwicklung einer Grammatik für eine DSL mit xText am Beispiel einer Sprache zur Definition von Pflichtprüfungen in profil c/s* (vgl. [Gun14]) zu finden.

geführt, um grobe Fehler bei der Implementierung auszuschließen. Nachfolgend findet die Installation der Pflichtprüfung statt, in deren Anschluss entsprechende Tests durchgeführt werden. Nach erfolgreichem Testen ist dieser Teilprozess (Abbildung 2) beendet.

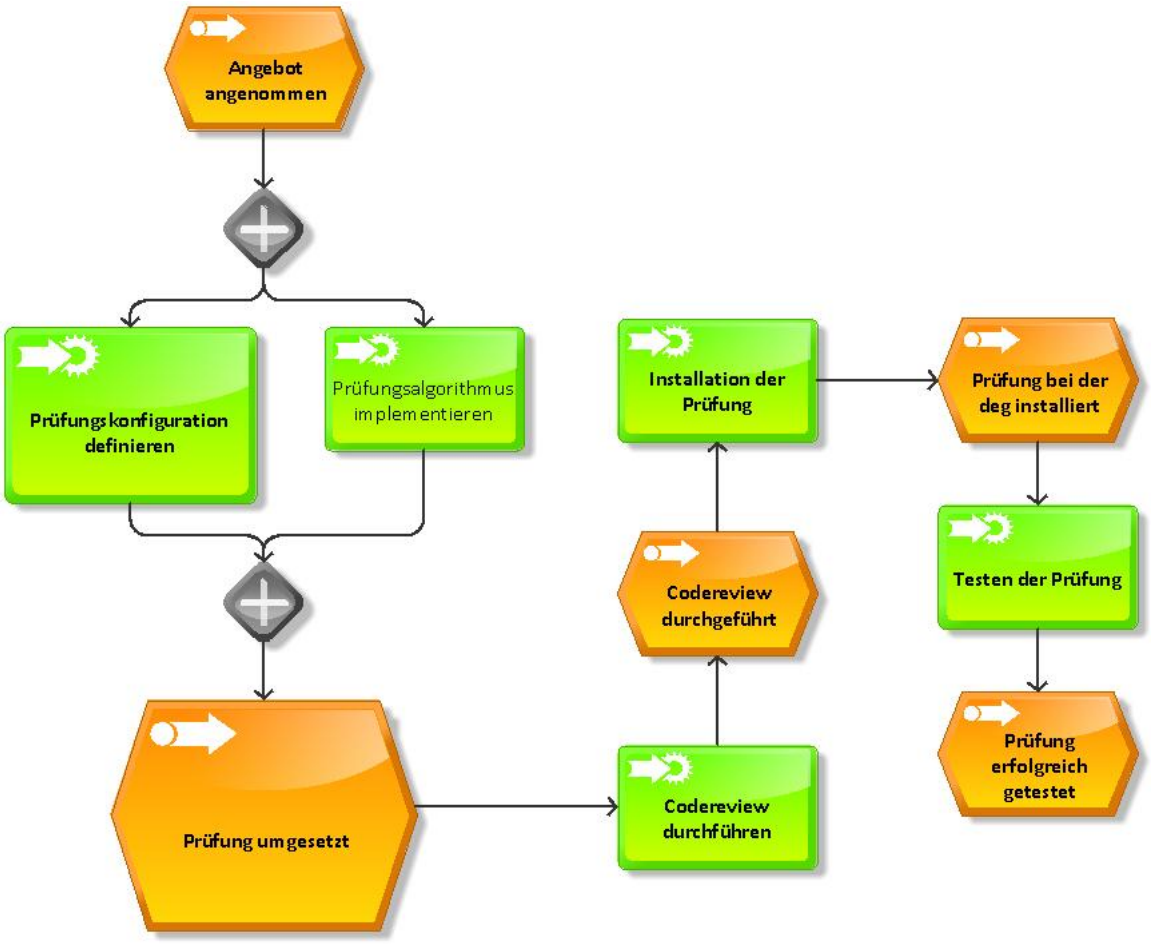

Abbildung 2: Teilprozess: Umsetzung

3.3 Auslieferung

Der letzte Teilprozess wird hauptsächlich vom Kunden durchgeführt. Nach der Auslieferung zum Kunden muss die Auslieferung dort installiert werden, bevor mit dem Testen begonnen werden kann. Wird die Prüfung positiv getestet, ist der Prozess für die Betrachtung in diesem Praxisbericht beendet. Falls die Prüfung negativ getestet wurde, beginnt die Fehlersuche beim Kunden. Der Kunde kann dabei einen Fehler bei der Installation der Prüfung gemacht haben. Ist dies der Fall, muss die Prüfung neu installiert werden und der Test beginnt erneut. Falls der Fehler vom Kunden nicht

nachvollziehbar ist, wird dieser Fehler an die deg gemeldet. Dort beginnt wiederum eine neue Fehlersuche. Ausgehend davon, dass wirklich ein Fehler vorliegt, gibt es zwei Fehlerursachen. Die erste mögliche Ursache bezieht sich auf die Implementierung des Prüfungsalgorithmus. Die zweite mögliche Ursache könnte eine falsche Definition der Prüfungskonfiguration sein. Je nachdem muss der konkrete Fehler gefunden und behoben und die Änderung neu an den Kunden ausgeliefert werden (Siehe Kapitel 3.1 und Kapitel 3.2). Dieser Teilprozess wird in Abbildung 3 visualisiert.

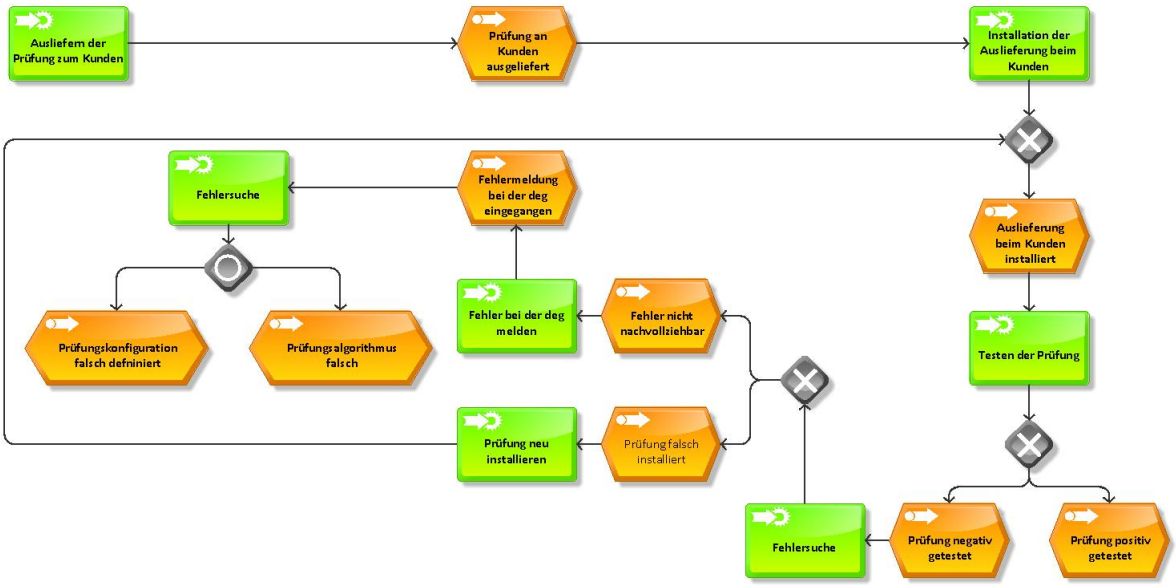

Abbildung 3: Teilprozess: Auslieferung

3.4 Effizientere Workflows mit *Xtext*

Nach der Beschreibung des Gesamtprozesses wird nun betrachtet, welche Arbeitsschritte in dem Workflow des Prozesses durch *Xtext* effizienter gestaltet werden könnten. Im Workflow des ersten Teilprozesses (siehe Kapitel 3.1) ist eine direkte Effizienzsteigerung[4] durch *Xtext* nicht möglich, da hier noch keine Implementierung vorgenommen wird. Allerdings könnte eine gut lesbare Sprache, welche für die Beschreibung der Pflichtprüfungen eingesetzt wird auch für die Beschreibung der Prüfung im Angebot

[4]Direkt meint, dass sich eine Änderungen des Arbeitsschritts nicht direkt auf die Effizienz dieses Arbeitsschitts auswirken würde, sondern indirekt die Effizienz eines anderen Arbeitsschritts steigert.

zur Anwendung kommen. Somit könnte die Definition der Prüfungskonfiguration vom Angebot einfach übernommen werden und müsste nicht in eine andere Syntax überführt werden. Insofern würde die Veränderung des Arbeitschritts *Angebot mit neuer Pflichtprüfung erstellen* zu einer Zeit- und Kosteneinsparung beim Arbeitsschritt *Prüfungskonfiguration definieren* im Workflow des Teilprozesses aus Kapitel 3.2 führen.
Im Workflow des zweiten Teilprozesses (siehe Kapitel 3.2) kann *Xtext* dabei helfen, die Prüfungskonfiguration zu definieren. Das hätte auch Auswirkungen auf das Codereview. Beim Testen der Prüfung sehe ich keine Möglichkeit für die Nutzung von *Xtext*.
Im Workflow des dritten Teilprozesses (siehe Kapitel 3.3) wäre es durch den Einsatz von *Xtext* möglich, die Fehlersuche zu vereinfachen. Nicht nur bei der deg könnten Fehler in der Prüfungskonfiguration schneller erkannt werden. Auch der Kunde kann durch die Nutzung von *Xtext* seine Fehlersuche ausweiten.

4 Analyse der Arbeitserleichterung

Im vorherigen Kapitel wurden folgende 4 Arbeitsschritte ausfindig gemacht, die mittels *Xtext* direkt effizienter gestaltet werden können.

1. Prüfungskonfiguration definieren (siehe Kapitel 3.2)
2. Codereview (siehe Kapitel 3.2)
3. Fehlersuche (des Kunden, siehe Kapitel 3.3)
4. Fehlersuche (der deg, siehe Kapitel 3.3)

4.1 Effizienzsteigerung mit *Xtext* durch eine Grammatik

Für die folgende Analyse wird in Betracht gezogen, dass für die Sprache zur Definition von Prüfungskonfigurationen eine Grammatik existiert, die von *Xtext* verwendet werden kann. Diese Grammatik wurde im Praxisbericht *Entwicklung einer Grammatik für eine DSL mit xText am Beispiel einer Sprache zur Definition von Pflichtprüfungen in profil c/s* (vgl. [Gun14]) entwickelt. Um alle Konfigurationsdateien bzgl. der Syntax zu validieren, waren noch einige Erweiterungen der Grammatik von Nöten (siehe Anhang A). Beispielsweise mussten Kommentare in die Menge der Terminale aufgenommen werden.

Listing 1: Kommentare mit '#' als Terminale

```
1 terminal SL_COMMENT:
2         '#' !('\n' | '\r')* ('\r'? '\n')?;
```

Darüber hinaus wurden die Terminale zur Definition von *Aktionen*, *Wirkungen* und *Klassennamen* auf die in profil c/s verwendeten Bezeichnungen eingeschränkt. Listing 2 zeigt dies für die *Wirkungen*. Die Umsetzung für die *Aktionen* und *Klassennamen* ist analog dazu im Anhang A zu finden.

Listing 2: Terminale für Wirkungen

```
1 WIRKUNG:
2         'VERHINDERT_AKTION' | 'OHNE' | 'WARNUNG';
```

Erzeugt man mit *Xtext* einen Editor, der die verwendete Syntax in den Konfigurationsdateien hinsichtlich dieser Grammatik überprüft, können dadurch folgende Fehlerquellen ausgeschlossen werden.

- Verwendung falscher Sprachkonstrukte
- Zuweisung von Aktionen, die in profil c/s nicht existieren
- Zuweisung von Wirkungen, die in profil c/s nicht existieren
- Zuweisung von Klassennamen (Prüfungsalgorithmen), die in profil c/s nicht existieren

Voraussetzung dafür ist jedoch, dass *Aktionen*, *Wirkungen* und *Klassennamen* auch in der Grammatik gepflegt werden.
Die Arbeit wäre durch die wegfallenden Fehlerquellen effizienter. Darüber hinaus muss der Entwickler auch nicht mehr nach dem qualifizierten Klassennamen suchen, weil ihn dieser vom Editor vorgeschlagen wird. Dies wäre auch bei der Suche nach den richtigen Bezeichnungen für die *Aktionen* oder *Wirkungen* von Vorteil.
Beim Codereview muss bedacht werden, dass keine Semantik geprüft wird. Besonders ist darauf zu achten, dass sämtliche IDs korrekt deklariert und zugewiesen sind.
Bei der Fehlersuche kann somit ein syntaktischer Fehler ausgeschlossen werden.[5] Die Mitarbeiter der deg können sich bei der Fehlersuche auf die fachliche Korrektheit der Prüfungskonfiguration und auf die semantischen Zusammenhänge konzentrieren. Der Kunde geniest bei der Fehlersuche dadurch derzeit keinen Vorteil. Wäre der Kunde jedoch in der Lage die Konfigurationsdateien zu lesen und zu verstehen, könnte auch dieser die fachlichen Konzepte der Prüfungskonfiguration prüfen und sie mit dem Angebot abgleichen.[6]

4.2 Effizienzsteigerung mit *Xtext* durch Validierung

Mittels Validierungen können bei *Xtext* semantische Zusammenhänge geprüft werden. In dem Beispiel der Konfigurationsdateien sind im Folgen-

[5] Sollte es dennoch zu einem Syntaxfehler kommen, muss die Grammatik angepasst werden.

[6] Neben einem Fehler in der Konfiguration kann auch ein vom Kunden nicht ausreichend geprüftes, aber angenommenes Angebot zu einem nicht erwarteten Ergebnis des Tests führen.

den 3 Askpekte beschrieben, die mit Hilfe der Grammatik nicht abgesichert werden können.

1. Deklaration der IDs
   In den Konfigurationsdateien werden die Prüfungen zu den Anträgen über die Prüfungs-ID und die ID des Antrags bzw. das Schlüsselwort der Antragsart zugewiesen. Die Prüfungs-ID muss dazu im Vorfeld deklariert werden.
   Andersherum sind auch Deklarationen von IDs unnötig, die keinem Antrag zugewiesen werden.

2. Durchgängig laufende Nummer
   Die laufende Nummer, die bei der Zuweisung mehrerer *Wirkungen* und *Aktionen* zur Prüfungskonfiguration benutzt wird, muss bei eins beginnen, fortlaufend sein und darf keine Zahl überspringen.[7] Wird eine Nummer ausgelassen, werden die Konfigurationen mit den nachfolgenden Nummern nicht berücksichtigt.

3. Einmalige Definition bestimmter Eigenschaften
   Die Eigenschaften *Kurzbezeichnung*, *Langtext* und *Klassenname* einer Konfiguration dürfen nur einmal definiert werden. Werden sie mehrfach definiert, wird die jeweilige Konfiguration verwendet, die in der Datei am weitesten vorne steht.

Um diese Aspekte abzusichern, sind Validierungen nötig, die im Zuge dieses Berichtes implementiert wurden. In den folgenden Kapiteln wird auf diese Implementierung (siehe Anhang A) eingegangen.
Für sämtliche Validierungen bietet *Xtext* eine Validatorklasse. Die Methoden dieser Klasse, die mit *Check* annotiert sind, werden bei der Validierung ausgeführt. Entsprechende Warnungen oder Fehler, werden innerhalb dieser Methoden mit *warning(String meldung, EStructuralFeature feature, int index)* oder *error(String meldung, EStructuralFeature feature, int index)* implementiert. Die *meldung* beinhaltet die Fehlermeldung oder Warnung, die dem Nutzer angezeigt wird. Die anderen Parameter *feature* und *index* bestimmen

[7] Genaueres dazu ist im Praxisbericht *Entwicklung einer Grammatik für eine DSL mit xText am Beispiel einer Sprache zur Definition von Pflichtprüfungen in profil c/s* [Gun14] zu finden.

die Position der Warnung oder des Fehlers im Eclipse-Editor. Das *feature* beschreibt dabei die in der Konfigurationsdatei definierte Eigenschaft und der *index* bestimmt um welche Prüfung es sich handelt. Mit beiden Werten (Eigenschaft und Prüfung) kann der Editor die entsprechende Stelle in der Konfigurationsdatei genau lokalisieren.

4.2.1 Absicherung der ID-Deklaration

Bei der Absicherung der notwendigen Deklarationen von Prüfungs-ID wird mit folgenden Code-Ausschnitt geprüft, ob die Prüfungs-IDs deklariert wurden, die bei der Zuweisung der Prüfungen zu Anträgen verwendet werden.

Listing 3: Validierung der Deklaration von Prüfungs-IDs für die Zuweisung zu Anträgen

```
1  @Check
2  def checkIdsFuerAntragszuweisungDeklariert(Konfiguration konfiguration) {
3      checkIdsVonZuweisungDeklariert(konfiguration.spezantragszuweisung,
           konfiguration.usedids.get(0),
4          DslPackage.Literals.KONFIGURATION__SPEZANTRAGSZUWEISUNG)
5  }
6
7  @Check
8  def checkIdsFuerAntragsArtzuweisungDeklariert(Konfiguration konfiguration)
       {
9      checkIdsVonZuweisungDeklariert(konfiguration.antragszuweisung,
           konfiguration.usedids.get(0),
10         DslPackage.Literals.KONFIGURATION__ANTRAGSZUWEISUNG)
11 }
12
13 def checkIdsVonZuweisungDeklariert(EList<String> konfigurationen, String
       konfigFuerDeklarierteIDs,
14     EStructuralFeature feature) {
15     var index = 0
16     for (konfiguration : konfigurationen) {
17         var zugewiesenePruefungen =
               extrahiereDeklarierte Pruefungen(konfiguration)
18         for (pruefung : zugewiesenePruefungen) {
19             var deklarierte Pruefungen =
                   extrahiereDeklarierte Pruefungen(
                   konfigFuerDeklarierteIDs)
20             if (!deklarierte Pruefungen.contains(pruefung.trim)
                   ) {
21                 warning(String.format(
                       WARNUNG_KEINE_ID_DEKLARIERT, pruefung)
                       , feature, index)
22             }
```

```
23                      }
24                      index++
25              }
26      }
```

Weiterhin muss geprüft werden, ob alle deklarierten IDs mindestens einem Antrag zugeordnet sind. Das wird wie folgt umgesetzt.

Listing 4: Validierung der Verwendung von deklarierten Prüfungs-IDs

```
1       @Check
2       def checkIdVerwendung(Konfiguration konfig) {
3               var deklariertePruefungen = extrahiereDeklariertePruefungen(konfig
                    .usedids.get(0))
4               for (String pruefung : deklariertePruefungen) {
5                       checkVerwendungInAntrag(konfig.antragszuweisung, konfig.
                            spezantragszuweisung, pruefung)
6               }
7       }
8
9       def checkVerwendungInAntrag(EList<String> zuweisungenZuAntragsart, EList<
            String> zuweisungenZuAntrag,
10              String pruefung) {
11              for (antragsartZuweisung : zuweisungenZuAntragsart) {
12                      var zugewiesenePreufungen =
                            extrahiereDeklariertePruefungen(antragsartZuweisung)
13                      if (zugewiesenePreufungen.contains(pruefung)) {
14                              return
15                      }
16              }
17              for (antragsZuweisung : zuweisungenZuAntrag) {
18                      var zugewiesenePreufungen =
                            extrahiereDeklariertePruefungen(antragsZuweisung)
19                      if (zugewiesenePreufungen.contains(pruefung)) {
20                              return
21                      }
22              }
23              error(String.format(WARNUNG_PRUEFUNG_KEINEM_ANTRAG_ZUGEWIESEN,
                    pruefung),
24                      DslPackage.Literals.KONFIGURATION__USEDIDS)
25
26      }
```

Der entsprechende Fehler wird genau an der Stelle angezeigt, wo die nicht deklarierte Prüfungs-ID verwendet wird bzw. wo die nicht verwendete Prüfungs-ID deklariert wird.
Durch diese Validierung wird dem Entwickler das Prüfen der Vollständigkeit der deklarierten Prüfung abgenommen. Außerdem wird zusätzlich überprüft, ob unnötige Deklarationen stattgefunden haben.

4.2.2 Durchgängig laufende Nummer

Die laufende Nummer ist nur für zwei Eigenschaften einer Prüfungskonfiguration wichtig[8]. Die Validierung wird wie folgt umgestzt.

Listing 5: Validierung der durchgängig laufenden Nummer

```
1  @Check
2  def checkLaufendeNummerWirkung(Konfiguration konfiguration) {
3          checkFehltZahlInLaufenderNummer(
4                  extrahierePruefungZuLaufendeNummerMapFuerEigenschaft(
                       konfiguration.pruefungswirkung), "Wirkung",
5                  DslPackage.Literals.KONFIGURATION__PRUEFUNGSWIRKUNG);
6  }
7
8  @Check
9  def checkLaufendeNummerAktion(Konfiguration konfiguration) {
10         checkFehltZahlInLaufenderNummer(
11                 extrahierePruefungZuLaufendeNummerMapFuerEigenschaft(
                       konfiguration.pruefungsaktion), "Aktion",
12                 DslPackage.Literals.KONFIGURATION__PRUEFUNGSAKTION);
13 }
14
15 def checkFehltZahlInLaufenderNummer(HashMap<String, ArrayList<Integer>>
       map, String eigenschaft,
16         EStructuralFeature feature) {
17         var index = 0
18         for (pruefung : map.keySet) {
19                 var nummern = map.get(pruefung)
20                 for (var ln = 1; ln < nummern.size; ln++) {
21                         if (!nummern.contains(ln)) {
22                                 error(String.format(
                                       WARNUNG_KEINE_LAUFENDE_NUMMER,
                                       eigenschaft, pruefung,
23                                 ln), feature, index) }
24                         index++
25                 }
26         }
27 }
```

Dadurch wird eine Fehleranalyse weitaus einfacher. Das Suchen von nicht vorhandenen laufenden Nummern wird umso komplizierter, je unübersichtlicher und größer die Datei wird.
Abgesehen von der Vereinfachung der Fehleranalyse erspart diese Validierung auch beim definieren von Prüfungskonfigurationen Arbeit, da der Editor überprüft, welche die nächste laufende Nummer ist.

[8] *Wirkung* und *Aktion*

4.2.3 Einmalige Definition bestimmter Eigenschaften

Diese Prüfung verhindert Nebeneffekte, die entstehen, wenn eine Eigenschaft in der Datei versehentlich doppelt definiert wird. Bei den Eigenschaften, die eine laufende Nummer benötigen, ist dieses Problem durch die zweite beschriebene Validierung (siehe Kapitel4.2.2) abgesichert. Bei den anderen Eigenschaften wird dies wie folgt sicher gestellt.

Listing 6: Validierung der einmaligen Definition

```
1   @Check
2   def checkMehrfacheDefinition(Konfiguration konfig) {
3           var deklariertePruefungen = extrahiereDeklarierte
                Pruefungen(konfig.usedids.get(0))
4           for (String pruefung : deklarierte Pruefungen) {
5                   checkMehrfacheVerwendungDerPruefung(konfig.
                        pruefungskurzbezeichnung, pruefung,
6                           DslPackage.Literals.
                                KONFIGURATION__PRUEFUNGSKURZBEZEICHNUNG)
7                   checkMehrfacheVerwendungDerPruefung(konfig.
                        pruefungslangtext, pruefung,
8                           DslPackage.Literals.
                                KONFIGURATION__PRUEFUNGSLANGTEXT)
9                   checkMehrfacheVerwendungDerPruefung(konfig.
                        pruefungsichtbarkeit, pruefung,
10                          DslPackage.Literals.
                                KONFIGURATION__PRUEFUNGSICHTBARKEIT)
11          }
12  }
13
14  def checkMehrfacheVerwendungDerPruefung(EList<String> list, String
        pruefung,
15  EAttribute attribute) { var index = 0
16          var gefundeneKonfigurationen = 0
17          for (konfiguration : list) {
18                  var pruefungBenutzt = extrahierePruefung(konfiguration)
19                  if (pruefungBenutzt.equals(pruefung)) {
20                          if (gefundeneKonfigurationen == 0)
21                                  gefundeneKonfigurationen = 1
22                          else
23                                  error(String.format(
                                        WARUNUNG_DOPPELT_GENUTZTE_PRUEFUNG,
                                        attribute.name,
24                                  pruefung), attribute, index)
25                  }
26                  index++
27          }
28  }
```

Durch die Validierungsregeln können weitere Fehler ausgeschlossen werden, die sich auf die Semantik der Sprache zur Definition von Prüfungskonfigurationen beziehen. Beim Definieren der Prüfungskonfigurationen hat das einen klaren Vorteil. Die oben beschriebenen Validierungen werden vom Editor ausgeführt und müssen nicht vom Entwickler überprüft werden.
Beim Codereview kann der Fokus hauptsächlich auf die fachlich korrekte Definition der Eigenschaften gelegt werden[9].
Für die Fehleranalyse gilt dasselbe. Auch hier kann man sich weitgehend auf die fachlichen Aspekte einer Prüfungskonfiguration konzentrieren.
Das unterstützt auch den Kunden. Dieser kann selbst die Überprüfung der Konfigurationsdatei vornehmen, weil syntaktische Fehler und semantische Fehler, die nichts mit den fachlichen Festlegungen zu einer Prüfungskonfiguration (Festlegungen aus dem Angebot) zu tun haben, auszuschließen sind.

[9] Bspw. ob die Kurzbezeichnung korrekt ist, oder alle nötigen Aktionen definiert wurden

5 Fazit

In diesem Bericht wurde gezeigt, dass durch den Einsatz von *Xtext* einige Fehlerquellen bei der Definition von Prüfungskonfigurationen ausgeschlossen werden können. Voraussetzung dafür ist jedoch die Definition einer Grammatik. Weiterhin bedarf es für die Absicherung semantischer Zusammenhänge entsprechende Validierungsregeln. Mit *Xtext* lässt sich beides umsetzen und sehr einfach in Eclipse integrieren.
Durch den Ausschluss bestimmter Fehlerquellen lässt sich bei den Arbeitsschritten, zu deren Unterstützung *Xtext* eingesetzt werden kann (genannt zu Beginn von Kaptiel 4), eine Effizienzsteigerung erzielen. Das hat auch eine Auswirkung auf den Gesamtprozess (siehe Kapitel 3). Auch das Durchführen von Prozessschleifen, die Aufgrund von fehlgeschlagenen Tests auftreten, wird im Gesamtprozess reduziert. Demnach wird der Prozess beim Einsatz von *Xtext* schneller einen Endzustand erreichen.
Als zusätzlichen Arbeitsaufwand könnte man die Pflege der Grammatik betrachten. Diese muss immer aktualisiert werden, wenn *Akionen* oder *Klassennamen* neu hinzukommen, oder wenn an den Bezeichnungen der bestehenden *Akionen* und *Klassennamen* etwas geändert wird. Das Editieren der Grammatik ist jedoch auch in Bezug auf diesen Aspekt ein Vorteil. Wenn die Grammatik geändert wird, werden in den Konfigurationsdateien an den entsprechenden Stellen Fehler angezeigt. Dadurch entfällt das Suchen entsprechender Stellen in den Konfigurationsdateien, was bei vielen Konfigurationen ein erheblich größerer Arbeitsaufwand wäre, als das Ändern einer Zeile in der Grammatik.

A Implementierungen

Grammatik für die Syntax der Konfigurationsdateien

Datenträger: Implementierung/xtext/de.deg.eler.ft.vp/src/de/deg/eler/ft/vp/Dsl.xtext

Validator-Klasse für die Konfigurationsdateien

Datenträger: Implementierung/xtext/de.deg.eler.ft.vp/src/de/deg/eler/ft/vp/validation/DslValidator.xtend

Literatur

[114] *Xtext Documentation*. URL: http://www.eclipse.org/Xtext/cumentation/2.6.0/Xtext Documentation.pdf, September 2014. Zuletzt eingesehen am 24.11.2014.

[fESE14] EXPERIMENTELLES SOFTWARE ENGINEERING, FRAUNHOFER-INSTITUT FÜR: *Workflow*. URL: http://www.software-kompetenz.de/servlet/is/2368/, 2001-2014. Zuletzt eingesehen am 26.11.2014.

[Gun14] GUNDERMANN, NIELS: *Entwicklung einer Grammatik für eine DSL mit xText am Beispiel einer Sprache zur Definition von Pflichtprüfungen in profil c/s*. Technischer Bericht, data experts gmbH, 2014.

[Hed12] HEDTSTÜCK, ULRICH: *Einführung in die Theoretische Informatik*. Oldenbourg Wissenschaftsverlag, 2012.

[MSL+13] MARKUS VOELTER, SEBASTIAN BENZ, LENNART KATS, MATS HELANDER, EELCO VISSER und GUIDO WACHSMUTH: *DSL Engineering*. CreateSpace Independent Publishing Platform, 2013.

[PA05] PROF. DR. ALLWEYER, THOMAS: *Geschäftsprozessmanagement*. W3L GmbH, 2005.

[SGS08] SCHNEIDER,GABRIEL , GEIGER, INGRID KATHARINA und SCHEURING, JOHANNES : *Prozess- und Qualitätsmanagemen*. Compendio Bildungsmedien AG, 2008.