Analyse des Quellcodes

BUBE Online

Version 1.0 – 22.07.2013

Ansprechpartner
Christian Schneider
christian.schneider@quinscape.de
Telefon 0231 / 533 831 433
Telefax 0231 / 533 831 111

Inhalt

1 Einleitung

Dieses Dokument betrachtet das vorliegende System BUBE Online.

Der aktuelle Stand der Software wurde durch die Betrachtung einzelner logischer Module:

- Fachmodule
- Navigation
- Berichtswesen

dokumentiert. Die logische Differenzierung dient der Übersicht. Aus technischer Sicht ist die Software monolithisch aufgebaut, eine Differenzierung in fachliche Module erfolgt nur über ein Benennungsschema (S für Stammdaten, E für 11.BImSchV, P für PRTR und G für 13.BImSchV).

Auf Quellcodeebene findet keine fachliche Differenzierung statt. Hier wurde eine Trennung in technische Module durchgeführt, die sich primär am eingesetzten Framework Struts orientieren.

Das folgende Kapitel zeigt die Anforderungen und Faktoren auf, die Wartbarkeit eines Softwaresystems schon während des Entwurfs zu berücksichtigen, um so die Software auch über Jahre hinweg kontinuierlich weiterentwickeln zu können. Gemessen hieran kann eine Abschätzung der Wirtschaftlichkeit einer Pflege der vorliegenden Software getroffen werden.

Die darauf folgenden Kapitel zeigen eine technische Betrachtung, sowie eine Betrachtung der Module und modulübergreifende Aspekte. Die Betrachtungen beinhalten eine Einschätzung über die Zukunftsfähigkeit der Komponenten.

2 Wartbarkeit als Entwurfsziel

Das System ist in sich selbst funktionsfähig und stabil. Es erfüllt – soweit bekannt – seine Aufgabe zufriedenstellend und kann produktiv eingesetzt werden. Es befindet sich im Zustand der Wartung, die zwei Ziele verfolgt:

1. Anpassung der Software an veränderte technische und fachliche Anforderungen.
2. Beseitigung von Fehlern

Daher wird in diesem Dokumente die Perspektive in Hinblick auf die Wartbarkeit und auf die Möglichkeiten von zukünftigen Änderungen an dem System betrachtet. Eine langlaufende Planung ist in großen Teilen identisch mit den Anforderungen an das Produktmanagement. An diese sind die folgenden Betrachtungen angelehnt.

2.1 Technische Grundlage

Ein Softwaresystem, welches auf eine längere Wartungsphase ausgelegt ist, muss über eine entsprechend stabile Architektur verfügen. Die Anforderungen an eine solche Architektur sind sehr anspruchsvoll.

Eine grundlegende Herausforderung beim Entwurf eines Systems ist die Kombination von fachlichen Anforderungen und der technischen Realisierung. In diesem Spannungsfeld muss eine eingehende Analyse erfolgen.

2.2 Gleichförmigkeit

Bei der Planung einer Wartung eines Softwaresystems über mehrere Jahre hin ist es sehr wahrscheinlich, dass das mit der Wartung betraute Personal während dieser Zeit wechselt.

Ist die Software gleichförmig aufgebaut, so erleichtert dies die Wartung, da ein Entwickler die generellen Konzepte erlernen muss und sich so innerhalb des Quellcodes schneller orientieren kann.

Um eine effektive Wartung zu realisieren, ist es sinnvoll ein einheitliches Entwicklungsmodell zu definieren.

Handlungsbedarf BUBE Online

Keiner. Das System ist gleichförmig aufgebaut, so dass sich Konzepte gut auf andere Module übertragen lassen. Die Konzepte müssen bei der Wartung in Hinblick auf den Aspekt der Gleichförmigkeit weitergeführt werden, d.h. es werden auch die hier als weiterer Handlungsbedarf identifizierten Punkte fortgeführt, um die Gleichförmigkeit zu erhalten.

2.3 Kontinuierlicher technischer Migrationspfad

Eine über mehrere Jahre ausgelegte Wartung bedarf neben der fachlichen Weiterentwicklung einer ständigen technischen Weiterentwicklung. Diese Migrationspfade müs-

sen mit der Entwicklung geplant werden, da die Aktualisierung von Frameworks in der Regel einen umfassenden Test nach sich zieht. Diese Aufwände müssen in Zyklen von 12 bis 18 Monaten eingeplant werden.

Viele zugrundeliegende Technologien vollziehen diesen Wandel ebenfalls. Für die meisten Unternehmen und Open Source Anbieter ist es jedoch nur wirtschaftlich die neusten Versionen weiter zu entwickeln. Ältere Version gehen irgendwann in die Phase EOL („End of Life“) über und werden nicht weitergepflegt.

Ein technischer Migrationspfad bedarf umfassender Beratung und Marktbeobachtung durch die Entwicklung. Wird der Aufwand in eine technische Aktualisierung nicht investiert, so wird das System spätestens mit dem EOL seiner Teilkomponenten selbst in den EOL überführt.

Handlungsbedarf BUBE Online

Einige Komponenten des Systems sind veraltet, vgl. hierzu Kapitel 3 Technische Betrachtung. Ein signifikanter Anteil der Aufwände muss in die Aktualisierung der technischen Grundlagen investiert werden.

2.4 Einzelschrittausführung

Die Einzelschrittausführung (Debugging) ist eines der wichtigsten Werkzeuge bei der Suche nach Fehlern und der Erprobung von Funktionalitäten. Ein Softwaresystem besteht aus vielen tausend Zeilen Quellcode, der Zeile für Zeile in sehr schneller Geschwindigkeit abgearbeitet werden.

Mit der Einzelschrittausführung ist es für einen Entwickler möglich den Programmfluss an einer beliebigen Stelle anzuhalten und die nun folgenden Programmzeilen, Schritt für Schritt durchzugehen. Insbesondere ist dies notwendig, wenn durch einen Fehler kein Programmabbruch erfolgt, sondern z.B. ein falsches Berechnungsergebnis generiert wird. So kann er sich die Auswirkungen jeder Programmzeile ansehen und bewerten.

Die Einzelschrittausführung kann den Quellcode immer operationsweise ausführen, danach ist eine Betrachtung der Werte möglich, sofern diese im Quellcode definiert vorliegen. Im Folgenden soll dies an einem Beispiel aus dem Quellcode dargestellt werden.

Die folgende *einzeilige* Anweisung aus BUBEOnline[1]

```
1. dForm.set("s_jahr", BasicFunktionBean.changeStringEmpty(String.va
   lueOf(BasicFunktionBean.convertInt(BasicFunktionBean.datetoString
   format(new Date(), Constants.FORMAT_YEAR)) - 1)));
```

belegt das Feld „s_jahr“ mit dem aktuellen Jahr - 1. Dazu werden folgende Schritte ausgeführt:

1. Erzeuge ein aktuelles Datum
2. Extrahiere das Jahr des aktuellen Datums in eine Zeichenkette (String)

[1] de.uba.bube.actions.AuswertungBearbeitungszustandAction

3. Wandle diese Zeichenkette in einen Zahlenwert (Integer)
4. Ziehe 1 von dieser Zahl ab
5. Wandle diesen Zahlenwert in eine Zeichenkette (String)
6. Falls diese Zeichenkette ungefüllt ist, liefere eine leere Zeichenkette, sonst die Jahreszahl.

Im Einzelschrittmodus hat man – ergänzend zu der sehr schlechten Lesbarkeit des Quellcodes – keinen einfachen Zugriff auf die Zwischenergebnisse von Schritt 1 bis 6, da die Methode keine Zwischenergebnisse definiert. Liegt die Methode dagegen wie folgt vor:

```
1. String previousYear = getPreviousYear();
2. dForm.set("s_jahr", previousYear);
   […]
3. private String getPreviousYear() {
4.         Calendar calendar = GregorianCalendar.getInstance(LOCALE);
5.         calendar.add(Calendar.YEAR,-1);
6.         int year = calender.get(Calendar.YEAR);
7.         return String.valueOf(year);
8. }
```

ist mit der Einzelschrittausführung an jeder Stelle der Zugriff auf die Zwischenergebnisse möglich. Zudem ist die Funktion klarer strukturiert und der Zweck unmittelbar erkennbar. Darüber hinaus kann die Funktion previousYear auch an anderen Stellen des Systems genutzt werden.

Bitte beachten Sie, dass beide Implementierungsvarianten im System das gleiche korrekte Ergebnis liefern und *aus fachlicher Sicht vollkommen korrekt* sind. In dieser Betrachtung wird der Fokus auf die *Wartbarkeit und Weiterentwicklungspotential des Systems* durch einen Entwickler gelegt.

An diesem Beispiel ist gut zu erkennen, dass die Art der Umsetzung signifikanten Einfluss auf die Wartung hat. Das Nachvollziehen des Kontrollflusses in der zweiten Variante ist um Faktoren effizienter bei der Wartung.

Handlungsbedarf BUBE Online

Im Quellcode von BUBEOnline finden sehr häufig die gezeigten Konstrukte. Im Laufe der Wartung hat sich gezeigt, dass die Bearbeitung diese bei der Analyse von Fehlverhalten der Software hinderlich sind.

Dennoch ist davon abzuraten, diese während der Wartung hier und da anzupassen, da man ansonsten eine Codebasis aus gemischten Ansätzen aufbaut (vgl. Kap. 2.2 Gleichförmigkeit). Diese wäre noch aufwändiger zu warten, da sie zwei Herangehensweisen konkurrierend nebeneinander stellt.

2.5 Regressionstest

Ein Regressionstest[2] ist die kontinuierliche Wiederholung von Testfällen, um die Korrektheit der bisher definierten Testfälle abzudecken und Nebeneffekte von Änderungen aufzudecken.

Beispiel: In einem Textfeld in Modul S wurde bisher immer eine Eingabe größer als 0 erwartet. In Modul P wurde eine Funktion so implementiert, dass sie unter Annahme, dass das Feld immer größer als 0 ist, funktioniert. Nun wird Aufgrund einer Änderung der Anforderung in S eine Eingabe von 0 möglich, die aber in P zu einem Fehler führt. Dieser Fehler wird vermutlich erst im Produktivsystem oder durch Zufall bemerkt.

Dieses Problem ist ein klassisches Qualitätssicherungsproblem. Theoretisch muss ein System nach jeder Änderung, selbst bei Maskenanpassungen, vollständig getestet werden. Der dafür anfallende Aufwand steht jedoch in keinem Verhältnis zum Umfang vieler Änderungen.

Daher ist zur Sicherstellung der Qualität einer Software die Nutzung von *automatisierten* Tests maßgeblich. Automatisierte Tests können in voller Geschwindigkeit der zugrundliegenden Hardware durchlaufen werden.

Bitte beachten Sie, dass eine Testmenge *niemals* vollständig sein kann, d.h. ein erfolgreich durchlaufender Test bedeutet *nicht*, dass die Software fehlerfrei ist, er bedeutet, dass in Bezug auf eine definierte Testfallmenge, genau diese Testfälle nicht zu Fehlern geführt haben.

In der Praxis begegnet man diesem Problem damit, dass man von einer Grundmenge an Tests ausgeht, die während der Entwicklung eines Systems aufgebaut werden. Im Zuge der Produktivphase und Wartungen entstehen nun Fehler oder Änderungen, die nicht durch diese Testfallmenge abgedeckt sind. Ist dies der Fall, so wird nach Beseitigung des Fehlers oder Umsetzung der Änderung genau für diesen Fall ein oder mehrere neue Testfälle geschrieben. Zudem deckt man so Pfade ab, die

Wie auch bei der Einzelschrittausführung muss die Software auf Testbarkeit ausgelegt sein. Die zugrundeliegende Technologie muss geeignet sein, um automatisierte Testfälle definieren und zu jedem Zeitpunkt ausführen zu können.

Der große Vorteil von automatisierten Tests, die immer gleiche Durchführung von immer gleichen Prozessschritten, ist in bestimmten Szenarios auch der größte Nachteil. Sie liefern stetig vergleichbare Ergebnisse mit vorherigen Testläufen. Die Varianz, die ein menschlicher Tester in einen Test einbringt, indem er bei jedem Testlauf nicht die exakt gleichen Arbeitsschritte macht, kann er nicht bieten.

Daher ist die Kombination von automatischen Regressionstests und einem fachlichen Test eine notwendige Kombination bei der Wartung der Software.

Handlungsbedarf BUBE Online

Für die Erstellung von automatisierten Oberflächentests ist BUBE Online nicht ausgelegt. Eine nachträgliche Erstellung würde sehr hohe Aufwände erzeugen. Zum aktuel-

[2] http://de.wikipedia.org/wiki/Regressionstest

len Zeitpunkt wird die Qualität alleinig durch interne Funktionstests und fachlich hochgradig fundierte Tests des Auftraggebers gesichert.

2.6 Dokumentation

Die Dokumentation der Funktionsweise von Klassen und deren Methoden wird in der zugrundeliegenden Programmiersprache Java mit zwei Konstrukte unterstützt:

- JavaDoc-Dokumentation, mit der Klassen oder deren Methoden dokumentiert werden. Zudem werden diese JavaDoc-Kommentare in der Entwicklungsumgebung bei der Nutzung von Methoden dem Entwickler angezeigt. Sie dienen hier quasi als Handbuch während der Entwicklung und alleine aus dieser Tatsache ergibt sich schon ihre Bedeutung
- Inline-Dokumentation, mit der innerhalb von Methoden komplexe Sachverhalte näher beschrieben werden können.

Die Dokumentation steigert das Verständnis des Quellcodes, insbesondere, wenn es in einer langlaufenden Wartungsphase zu personellen Veränderungen kommt. Sie ermöglicht einem Entwickler bei der Analyse der Methode diese von einem Ausgangspunkt zu betrachten und beschleunigt so die Einarbeitung und das Einfinden in den Quellcode.

Entscheidend ist die Qualität der Dokumentation, wie folgendes Beispiel[3] zeigt:

```
/**
 * Berechnung der 01
 * @param efaktor als Efaktor
 * @param menge als Menge
 * @param ufaktor als Ufaktor
 * @exception Exception
 * @return Freisetzung
 */
public static double calc01(…
```

In dieser Methode erfolgt die Berechnung einer Freisetzung auf Basis der Menge und der beiden angegebenen Faktoren. Die Dokumentation selbst ist in diesem Fall nicht brauchbar. Die Funktionsweise lässt sich erst ermitteln, wenn die Methode analysiert wird. Alternativ hier ein Beispiel aus der jüngeren Wartungsphase[4]:

```
/**
 * <p>Rundet <code>decimal</code> nach folgenden Regeln, wenn <code>isUBA=false</code>:<br/>
 * <ul>
 * <li>Wenn decimal > 100, dann keine Nachkommastellen</li>
 * <li>Wenn decimal <=100 und >1, dann drei Nachkommastellen</li>
 * <li>Wenn decimal <=1, dann zeige drei signifikante Stellen</li>
 * </ul>
```

[3] de.uba.bube.util.CalcBean.calc01(double, double, double)

[4] de.uba.bube.util.CommonUtils.round(BigDecimal, int, boolean)

```
 * </p><p>
 * Wenn <code>isUBA=true</code>, dann zeige <strong>immer</strong> drei signifikante Stellen.
 * <p/><p>
 * Wenn <code>decimal=null</code>, dann gebe einen Leerstring zurück.
 * <p/>
 *
 * @param decimal
 * @param isUBA
 * @return {@link String} gerundeter Wert
 */
  public static String round(BigDecimal decimal, boolean isUBA) {
```

Der im ersten Moment etwas überladen wirkende Kommentar stellt sich für den Entwickler in der Entwicklungsumgebung wie folgt dar:

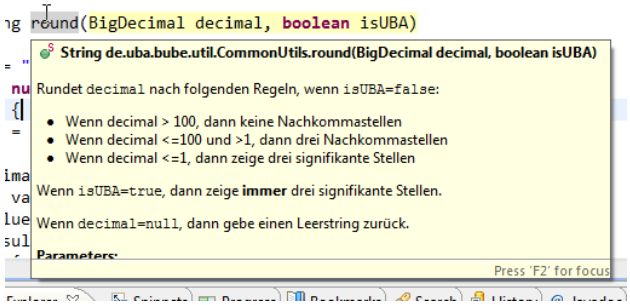

Abbildung 1: Anzeige der Javadokumentation während der Entwicklung

Hier wird mit einem Blick deutlich, was diese Methode leistet und was passiert, wenn der Schalter „isUBA“ gesetzt ist oder nicht. Für den Entwickler bedeutet dies, dass er mehrere Sekunden bis Minuten spart, um zum Beispiel Herauszufinden, was „isUBA“ bewirkt.

Die Erstellung der Dokumentation bei der Entwicklung erzeugt zusätzlichen Aufwand, dieser sollte jedoch bei der Planung einer langen Wartungsphase einkalkuliert werden. Er amortisiert sich mit steigendem Alter der Applikation oder (Teil)Neuimplementierungen sehr schnell.

Handlungsbedarf BUBE Online

Die Dokumentation von BUBE Online ist sehr heterogen. Subjektiv geschätzt sind ca. 20-30% des Quellcodes dokumentiert[5]. Viele Teile des Systems, wie z.B. die Action-Klassen, können nur bedingt dokumentiert werden, da sie in einer Methode sehr viele Aktionen zusammenfassen.

Hier zeigt sich Handlungsbedarf in der Umstrukturierung solcher überlangen Methoden (vgl. Kapitel 3.2 Redundanzen und 0 Methodenlängen), wie z.B. der Action-

[5] Ausgehend von der Qualität der Dokumentation, die für die Entwicklung nutzbar ist.

Klassen, die mehrere Benutzerinteraktionen in einer Methode durch Fallunterscheidungen implementieren.

Eine Nachdokumentation eines Systems ist i.d.R. zu aufwändig, so dass dies nur punktuell bei neuen Funktionalitäten sinnvoll ist.

3 Technische Betrachtung

Bei der Betrachtung des technischen Gerüsts der Software ist festzustellen, dass diese über ihre bisherige Lebenszeit nicht den Entwicklungen der allgemein anerkannten Regeln der Technik gefolgt ist. Daraus folgen eine zunehmende Alterung der Komponenten und damit eine zunehmende Entkopplung aktueller Technologien.

Ähnlich wie bei sich immerfort ändernden fachlichen Anforderungen, ist es auch für eine langfristige Nutzbarkeit einer Software notwendig über den Support eine kontinuierliche Anpassung vorzunehmen. Die notwendigen technischen Anpassungen müssen analog zu fachlichen Entwicklungen entlang eines Migrationspfades geplant werden. Im Folgenden möchten wir dies an konkreten Beispielen darlegen.

3.1 Verwendete Software Bibliotheken

Die Software nutzt als zentrales Framework zur Erzeugung der Webseiten und Kontrollflusssteuerung Struts[6] in der Version 1.2.8. Das Framework Struts wurde mit Wirkung zum 05.04.2013 hat seinen EOL (End Of Life) gesetzt und wird nicht mehr weiterentwickelt. Die letzte Version ist 1.3.10[7]. Insbesondere werden für diese Version *keine Sicherheitspatches* mehr bereitgestellt. Auch wenn die Software grundsätzlich weiter lauffähig ist, besteht aus Sicht der Wartbarkeit und Sicherheit hier dringender Handlungsbedarf, das Framework zu ersetzen.

Als Persistenzframework wird Hibernate[8] 3.2.5.ga eingesetzt. Die momentan aktuelle Version ist 4.2.3, die letzte Version der 3-Familie ist 3.6, die mit dieser Version beendet ist[9]. Die Versionen der 3.2-Familile befinden sich seit Oktober 2011 im EOL[10]. Da Hibernate für den Zugriff auf die Datenbank ausgelegt ist, ist es vertretbar in einem bestehenden System diese Version weiter einzusetzen. Die Version entsprechend alt und dadurch ausgiebig getestet. Zudem unterliegt das Datenmodell keinen signifikanten Änderungen. Es muss jedoch in Betracht gezogen werden, dass bei potentiell auftauchenden Fehlern im Framework, diese nicht mehr durch den Hersteller oder die Community behoben werden. Diese müssten dann in Dienstleistung u.U. mit signifikantem Aufwand (Analyse des Quellcodes, Behebung des Fehlers, Patchen) beseitigt werden. Wir möchten jedoch darauf hinweisen, dass die Wahrscheinlichkeit für ein solches Ereignis eher gering ist.

---

[6] http://struts.apache.org/

[7] http://struts.apache.org/struts1eol-press.html

[8] http://hibernate.org/

[9] https://community.jboss.org/wiki/HibernateRoadmap

[10] https://access.redhat.com/support/policy/updates/jboss_notes/

3.2 Redundanzen

Der Quellcode ist stark auf Funktionsblöcke ausgelegt. Diese Auslegung macht es notwendig, dass in den Funktionsblöcken Fallunterscheidungen durchgeführt werden, die wiederum zu Redundanzen führen. Dies soll an zwei Beispielen dargelegt werden.

Java-Quellcode

Der technische Aufbau der Software hat den Anschein, als sei er auf Basis einer definierten Grundlage generiert worden. Diese Theorie wird gestärkt durch das Auftreten von redundantem Code an signifikant vielen Stellen im System.

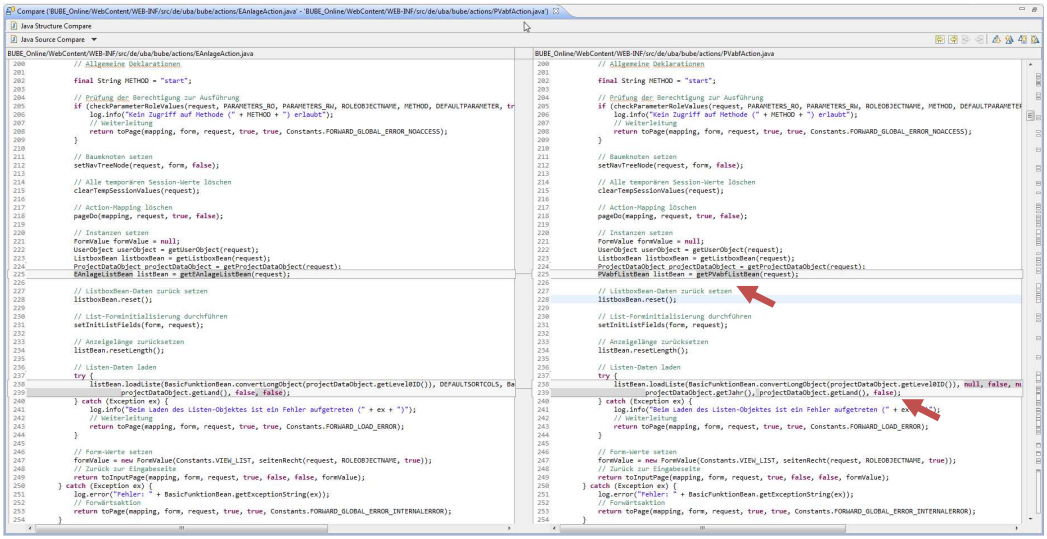

Abbildung 2: Vergleich start-Methode EAnlageAction und PVabfAction

In Abbildung 2 ist ein Vergleich der start-Methode zweier Action-Klassen[11] zu sehen. Die Auswahl der Klassen erfolgte unter dem Gesichtspunkt, dass sie aus unterschiedlichen Modulen stammen und thematisch unterschiedlich sind, um so auszuschließen, dass fachlich ähnliche Objekte auch über ähnlichen Quellcode verfügen. Gewählt wurden hierbei EAnlageAction (Anlage, Modul 11BV) und PVabfAction (Verbringung Abfälle, Modul PRTR).

Die dargestellten 54 Zeilen der Methode „start“ unterscheiden sich in genau 2 Zeilen, dies entspricht einer Codegleichheit von ca. 96%. Die Differenzen bestehen darin, dass unterschiedliche Objektlisten entsprechend des Fachobjekts erstellt werden und der Aufruf zur Erzeugung der Liste leicht variiert. Für weitere Methoden, wie „execute“, „addEntry“, „select“, etc. lassen sich ähnliche Ergebnisse nachvollziehen.

Im System sind zurzeit 61 Action-Klassen enthalten, die variierend über ca. 2-5 solcher stark redundanten Methoden verfügen.

[11] In Action-Klassen sind die Funktionalitäten zur Verwaltung eines Objekts, wie z.B. einer Anlage, gekapselt.

Grundlegend kann gesagt werden, dass die Menge des redundanten Quellcodes im Vergleich zum objektspezifischen Quellcode sehr hoch ist. Dies deuten wir, neben den identischen Inline-Kommentaren, als Indikator, dass diese Klassen initial durch ein Script aus der Datenbankdefinition oder ähnlichen Strukturen generiert wurden und dann im Laufe des Projektes einige Stellen durch Änderungswünsche modifiziert wurden.

Treten nun im Rahmen der Änderungen am Ablauf dieser redundant ausgelegten Programmteile auf, so sind – neben der Änderung selbst – zusätzliche Aufwandstreiber vorhanden:

- Prüfung jeder Stelle, ob sie dem ursprünglichen Aufbau entspricht oder im Rahmen einer Weiterentwicklung bereits verändert wurden. Ist dies der Fall, muss überprüft werden, ob diese Stellen mit der gewünschten Veränderung verträglich sind.
- Änderung an exakt einer Stelle. Dies ist mitunter die wahrscheinlichste Fehlerquelle, da es eine monotone Ersetzungsarbeit ist. Wird diese zudem von unterschiedlichen Personen durchgeführt, so steigert dies nochmal die Fehlerwahrscheinlichkeit.
- Jede dieser Stellen muss aufgrund der vorgenannten Situationen getestet werden.

Moderne Architekturkonzepte arbeiten heute mit Strategie-Implementierungen und Inversion-Of-Control[12] (IOC). Das IOC-Konzept injiziert einem Objekt eine bestimmte Strategie (z.B. zur Prüfung der Rechte beim Zugriff auf Objekte). Muss für ein Objekt eine abweichende Strategie umgesetzt werden, kann diese statt der Standardstrategie eingebunden werden. Dies senkt die Menge von redundantem Code um ein Vielfaches und erhöht gleichzeitig die Wartbarkeit.

Oberflächenmasken (JSP)

Die Oberflächenmasken eines Objekts enthalten alle objektbezogenen Maskenteile[13]:

```
<jsp:include page="status_info.jsp" flush="true"/>
<jsp:include page="ebetrst_search.jsp" flush="true"/>
<jsp:include page="ebetrst_list.jsp" flush="true"/>
<jsp:include page="ebetrst_detail.jsp" flush="true"/>
<jsp:include page="info.jsp" flush="true"/>
```

Es werden die Informationsleiste, die Suchseite, die Listenseite, die Detailseite und der Informationsbereich am Ender der Zeile eingebunden. Der Detailbereich einer Seite unterteilt sich nochmals in

```
<logic:match name="ebetrstForm" property="view" value="list_form_neu">
<logic:match name="ebetrstForm" property="view" value="list_form_bearbeiten">
<logic:match name="ebetrstForm" property="view" value="list_form_anzeigen">
```

[12] http://de.wikipedia.org/wiki/Inversion_of_Control

[13] Hier am Beispiel WebContent\WEB-INF\jsp\ebetrst.jsp

Damit werden die drei Darstellungsmodi „Neu“, „Bearbeiten“ und „Anzeigen“ differenziert. Welche Teile der Maske angezeigt werden, ergibt sich aus dem aktuellen Anzeigestatus.

Die Eingabefelder selbst sind nochmals in eine weitere Datei ausgelagert, um die Redundanz beim Eingabe- und Neumodus zu senken.

Dieses Konzept ist konsequent für alle verfügbaren Masken angewandt worden. Dies bewirkt, dass sowohl die Definition der Oberflächenmasken, als auch die zugehörige Logik in Java-Klassen, sehr großflächig (vgl. Kap. 0 Methodenlängen) ausgelegt sind. In Abbildung 3 ist diese Abhängigkeit nochmals grafisch dargestellt.

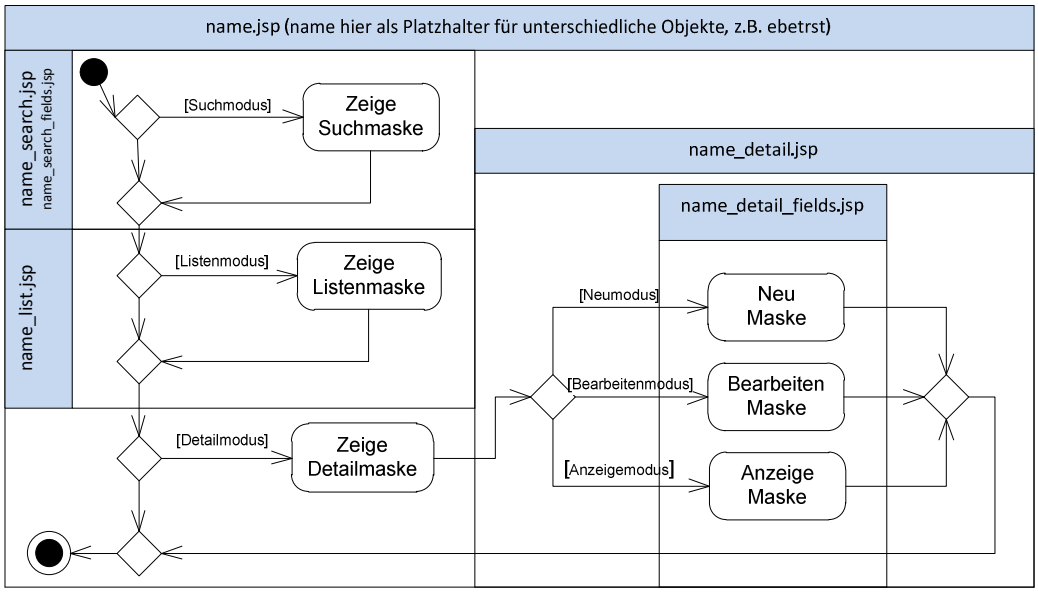

Abbildung 3: Strukturabhängigkeit Bildschirmmasken eines Objekts

Die Großflächigkeit hilft zwar während der Entwicklungsphase logische Bestandteile zusammenzuhalten und durch Kopieren einfach zu vervielfachen, erzeugt durch seine durch die Fallunterscheidungen eingebrachte Komplexität entsprechende Aufwände bei der Wartung.

In den detail-field-Masken muss dann entsprechende Logik umgesetzt werden, wie beispielhaft in der ebtrst_detail_fields.jsp:

```
<logic:match name="ebetrstForm" property="view" value="list_form_neu">
  <html:text property="i_astnr" maxlength="20" styleClass="normalbox farbe-grau"
          tabindex="10" disabled='<%=disableField.equals("true")?true:false%>'/>
</logic:match>
<logic:notMatch name="ebetrstForm" property="view" value="list_form_neu">
  <html:text property="i_astnr" maxlength="20" styleClass="normalbox farbe-grau"
          tabindex="10" disabled="true"/>
</logic:notMatch>
```

In diesem Beispiel wird eine Unterscheidung zwischen Neu-Modus und den anderen Modi gemacht. Der Code zur Anzeige der Arbeitsstättennummer ist zum Großteil re-

dundant. Der Grund für die redundante Auslegung ist hier allerdings keine schlechte Programmierung, sondern die Notwendigkeit durch die technologische Grundlage diesen Code so auszuführen.

Dieses Beispiel zeigt auch sehr schön, dass die Vermischung unterschiedlicher Systemzustände dazu führt, dass Quellcode unter dieser Voraussetzung die Tendenz hat unnötige Operationen zu enthalten.

3.3 Codemetriken

Die Erstellung von Codemetriken kann Hinweise auf Schwachstellen innerhalb der Software bieten. Dabei wurde die Erfahrung bei der Wartung des Quellcodes in Hinblick auf die Auswahl der Metriken genutzt.

Methodenlängen

In der vorliegenden Software sind Geschäftslogiken meist sehr umfangreich programmiert. Viele Methoden enthalten sehr viele Fallunterscheidungen, die unterschiedliche Fälle umsetzen.

In der Praxis zeigt sich, dass Methoden, die sich über mehr als eine Bildschirmseite erstrecken, bereits Einfluss auf die Lesbarkeit haben, da sie nicht mehr vollständig überblickt werden können. Die Bearbeitungsgeschwindigkeit bei der Wartung solcher Methoden sinkt hierdurch, da bei einer Änderung durch den Quellcode gescrollt werden muss. Gleichzeitig steigt die Wahrscheinlichkeit für Fehler, insbesondere bei stark verschachtelten Kontrollflüssen.

Ausnahmen von dieser Regel bilden lediglich durch unterstützende Tools wie in diesem Fall Castor generierte Klassen, die nicht mehr durch einen Entwickler modifiziert werden müssen.

Die Analyse hat ergeben, dass die vorliegende Version insgesamt 322 Methoden mit einer Anzahl über 150 Zeilen enthält, dabei ist die längste Methode[14] 2.876 Zeilen lang. Die Ursache der langen Methoden liegt zumeist in der redundanten Ausführung der Programmierung.

Insgesamt gibt es 58 Klassen, die einen Richtwert von 2.000 Zeilen überschreiten, die längste Klasse ist XML0Bean mit insgesamt 8.362 Zeilen.

Eine nachträgliche Analyse dieser Methoden ist sehr umfangreich, der der Kontrollfluss in einem Editor nur sehr aufwändig verfolgt werden kann. Als Beispiel soll folgender Ausschnitt in Abbildung 4 dienen.

[14] XML1Bean.importEintraege

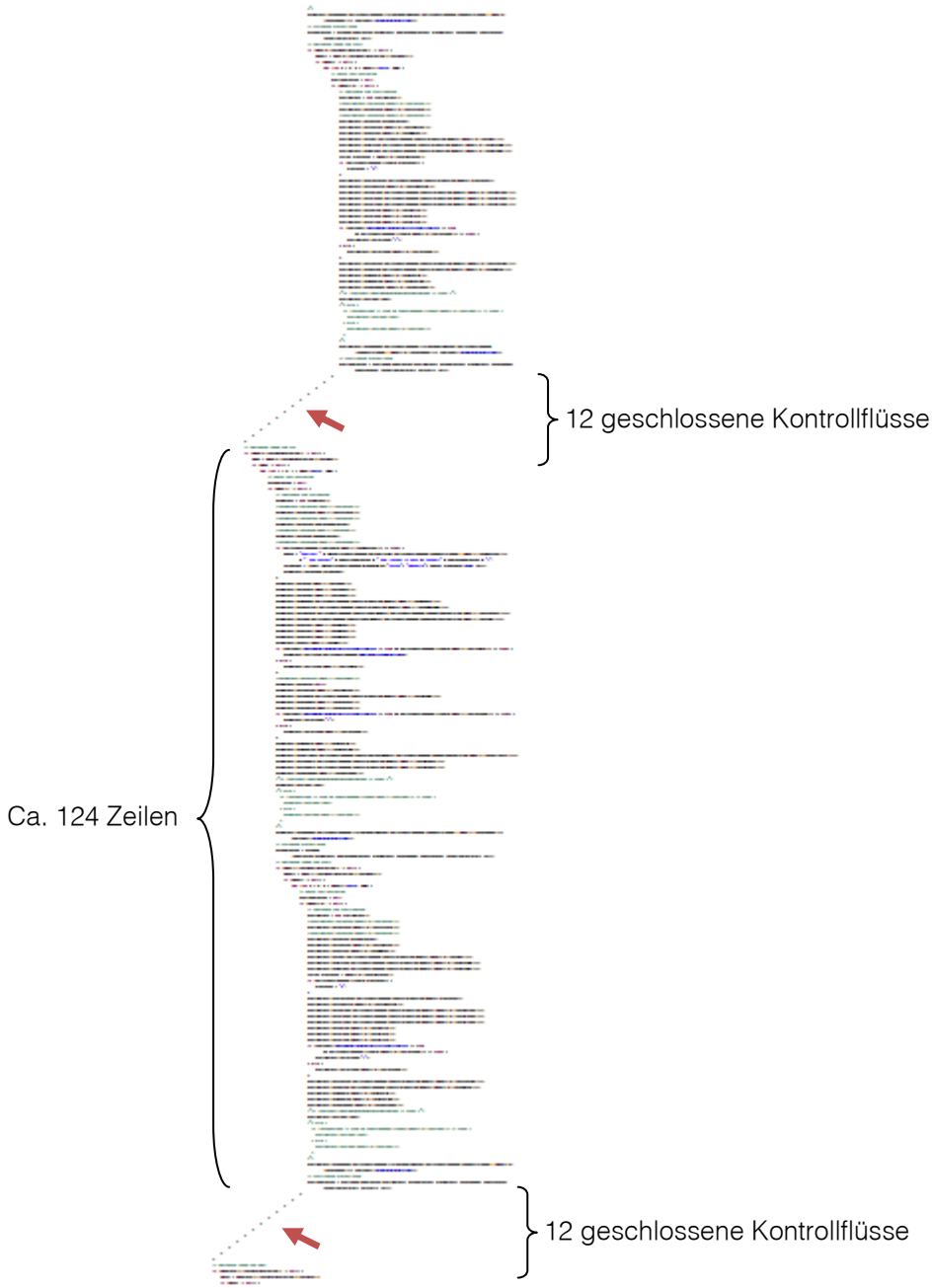

Abbildung 4: Strukturansicht einer langen Methode

Das Nachvollziehen der Kontrollflüsse innerhalb dieser Methode hat während der Wartung zwei gravierende Nachteile:

1. Es ist nicht klar erkennbar, welcher Code innerhalb eines Kontrollflusses ausgeführt wird. Dementsprechend müssen bei Codeänderungen zusätzliche

Analysen des umgebenden Codes vorgenommen werden, was entsprechenden Aufwand verursacht.

2. Die Wahrscheinlichkeit die Änderung fehlerhaft umzusetzen stark mit der Codekomplexität steigt.

Soll das System weiterhin mit moderatem Aufwand zu warten bleiben, so muss hier, wie auch an weiteren Codestellen, die Logik grundlegend überarbeitet werden.

Die Erfahrung zeigt, dass die Aufwände für die Wartung solcher Codefragmente kontinuierlich steigen, da sich aufgrund der Komplexität keine grundlegenden Strukturen geschaffen werden können.

Signaturlängen

Die Längen von Methodensignaturen[15] sind i.d.R. nur bis zu sieben Parametern gut beherrschbar. Je länger die Signaturen sind optisch schwer zu erfassen, so dass ein Entwickler hier signifikant mehr Zeit zur Analyse benötigt.

Als Beispiel soll hier eine Methode aus dem XML-Import dienen:

```
public XMLStatusObject importEintraege(Document doc, String xsdFile,
String modul, boolean importStammdaten, boolean importPRTR, boolean im-
portGFA, boolean importE11, UserObject userObject, RechteBean
rechteBean, boolean checkRechte, boolean saveDisableFields, boolean
setDefaultBearb, boolean setSpecialValues, boolean updateOnly, Long up-
dateIntkey, boolean setEmptyJahr, boolean calcGhs, boolean calcEmis)
throws ParameterException, HibernateException, MarshalException, Vali-
dationException, Exception {
```

Die hier abgebildete Methode ist genau eine Signatur. Mit dem folgenden Aufruf

xmlStatusObject = importEintraege(doc, xsdFile, modul, true, false, false, true, userObject, rechteBean, false, true, true, false, true, true, false, true, true, sess);

und der Fragestellung „Ist bei diesem Aufruf updateOnly ‚true' oder ‚false'?" wird deutlich, dass die Beantwortung der Frage zum einen viel Zeit in Anspruch nimmt (bei der Veränderung und Testläufen von Quellcode muss diese Stelle ggf. mehrfach betrachtet werden) und aufgrund der Möglichkeit des Verzählens eine zusätzliche Fehlerquelle darstellt.

Insgesamt wurden 286 Methoden mit Parameterlisten mit mehr als sieben Parametern identifiziert. Es ist empfehlenswert diese Stellen entsprechend neu zu strukturieren.

[15] Als Signatur wird die Menge der Parameter benannt, die einer Methode übergeben wird.

4 Module

In diesem Kapitel werden die einzelnen Module von BUBE Online betrachtet. Die Module werden in Hinblick auf ihren Wartungsstand und in Hinblick auf ihre Wiederverwertbarkeit betrachtet.

4.1 Grundlegende Hinweise zur Wiederverwertbarkeit

In Hinblick auf Bestrebungen wie dem Vorhaben MIFLEX wird auf eine Betrachtung der Wiederverwendbarkeit der Masken nicht eingegangen. Es ist davon auszugehen, dass die Wiederverwendung der technischen Grundlagen *nicht* möglich ist. Die Gründe hierfür sind:

- Wunsch nach einem anderen Oberflächenkonzept wie MIFLEX, da hier der Übergang von einer objektbasierten zu einer prozessorientierten Oberfläche vollzogen werden soll.
- Die Masken basieren auf einer mittlerweile veralteten Technologie

Der Schwerpunkt der Betrachtung liegt auf dem Datenmodell, der Berechtigungsstruktur und der Geschäftslogik.

4.2 Fachmodule

Die Wiederverwertbarkeit der Fachmodule ist stark gekoppelt an ihrer graphischen Repräsentation in Form der Eingabemasken und an das Datenmodell. Ändert sich eines dieser beiden Elemente ist davon auszugehen, dass das Fachmodul neu implementiert werden muss.

Benutzerverwaltung

Die Benutzerverwaltung besteht aus einer Login-Verwaltung und basiert auf fest definierten Rollen bzw. Benutzergruppen (14 Rollendefinitionen gemäß Tabelle r_rolle) und Status (8 Status gemäß Tabelle R0051).

Ein Benutzer kann immer nur genau einer Benutzergruppe zugeordnet sein. Einem Benutzer können zudem jeweils genau ein Bundesland, eine Behördenkennung und/oder eine Aufgabenbereichskennziffer zugeordnet sein. Aus dieser Kombination werden innerhalb der Funktionen die Zugriffsrechte berechnet.

Zudem können einem Benutzer noch Arbeitsstätten (z_astnr) und Gemeindekennziffern (z_gkznr) zugeordnet werden, die ebenfalls Einfluss auf die Zugriffsrechte haben.

Die Benutzerverwaltung selbst verfügt über die klassischen Pflegefunktionalitäten (Neu, Ändern, Löschen). Eine Pflege ist nur für Administratoren vorgesehen.

Wiederverwertbarkeit

Bei der Betrachtung der Wiederverwertbarkeit muss das Benutzer-Konzept von BUBE Online geprüft werden. Die Erfahrung zeigt, das Benutzer-Rechte-Rollen-Konzepte,

die eine beliebige Granulierung von Objekt und Datenrechten[16] dazu tendieren nur noch mit hohen Aufwänden administrierbar zu sein. Ist dann eine Berechtigungsstruktur eingestellt, so bleibt diese meist unverändert und die mit viel Aufwand hergestellte Rollen-Rechte-Verwaltung wird nicht genutzt. Oft ist es zielführender die meist in Frameworks vorhandenen Benutzerverwaltungen zu begrenzen.

Die stark auf die Benutzergruppe (Rolle) fokussierte Betrachtung in BUBE Online und die damit verbundene Einfachheit sehen wir daher als Vorteil des bestehenden Systems an.

In Hinblick auf die Analyse der Geschäftslogik sind Aufwände bei der Lokalisierung von Rechteeinschränkungen zu erwarten, da an vielen Stellen im Quellcode die Primärschlüssel (die hier die Rolle eines fachlichen Schlüssels übernehmen) direkt genutzt wurden, wie in diesem Beispiel[17]:

```
if ((userObject.getBenutzgr() == 9 ||
     userObject.getBenutzgr() == 10 /*|| userObject.getBenutzgr() == 14*/)
  && BasicFunktionBean.convertInt(item.getBearb()) > 1) {
       oPruef.setErrors(-1);
       return oPruef;
}
```

Da das Rollenkonzept ein zentraler, modulübergreifender Baustein der Geschäftslogik ist, können viele Regeln aus dem bestehenden Quellcode wiederverwendet werden, auch wenn der Quellcode teilweise neu geschrieben werden muss.

Stammdaten

Die Stammdatenverwaltung ist das zentrale Register für die Fachmodule. Prinzipiell handelt es sich hier um eine hierarchische Verwaltung von Objekten.

Auf diesem Modul basieren folgende 16 Datenfunktionen:

Funktionen auf den Stammdaten:

- "Übersicht anzeigen"
- "Prüfen der Daten"
- "Ergebnis der letzten Prüfung anzeigen"
- "Stammdatenabgleich"
- "Ergebnis des letzten Stammdatenabgleichs anzeigen"
- "Behörden-Nr./Arbeitsstätten-Nr./PRTR-Kennung/GFA-ID-Nr. ändern"

---

[16] Zugriffsrechte werden in zwei Hauptgruppen eingeteilt. Es gibt zum einen Objektrechte, die definieren, ob ein Benutzer ein bestimmtes Objekt manipulieren darf (z.B. Anlegen einer Arbeitsstätte). Zum anderen gibt es Datenrechte, die die Menge der Daten einschränkt, die ein Benutzer sehen darf (z.B. nur Arbeitsstätten aus einem Bundesland). Während das Objektrecht primär Einfluss auf den Kontrollfluss hat, wirkt das Datenrecht entlang des Kontrollflusses als Filter.

[17] de.uba.bube.business.GGfaBean.pruefeEintrag(Long, PruefObject, UserObject, RechteBean, Session)

- "Stammdatenübergabe"

Importmöglichkeiten:

- "der bundeseinheitlichen Referenztabellen (XML0)"
- "der bundeslandspezifischen Referenztabellen (XML0)"
- "von Stammdaten (XML1)"
- "von Stammdaten und Daten PRTR/11.BImSchV/13.BImSchV (XML1)"

Datensatzbezogener Export:

- "von Stammdaten und Daten PRTR/11.BImSchV/13.BImSchV der gewählten Arbeitsstätte (XML1)"
- "von Stammdaten der gewählten Arbeitsstätte (XML1)"

Löschen von Daten:

- "Alle Daten des gewählten Berichtsjahres löschen"

Referenzdatenexport:

- "der bundeseinheitlichen Referenztabellen (XML0)"
- "der bundeslandspezifischen Referenztabellen (XML0)"

11BV

Das Modul 11BV dient zur Erfassung der Daten zur Erstellung des Emissionsberichts.

Auf diesem Modul basieren folgende 19 Datenfunktionen:

Funktionen auf den Daten 11BV:

- "Übersicht anzeigen"
- "Prüfen der Daten"
- "Ergebnis der letzten Prüfung anzeigen"
- "Erzeugung der Freisetzungen Luft als PRTR-Bericht (PDF)"
- "Betriebs- und Geschäftsgeheimnisse vollständig löschen"
- "Ausgabe der Emissionserklärung (PDF)"
- "Abgabebericht herunterladen (Textdatei)"
- "Erzeugung der Freisetzungen Luft für PRTR"
- "Kennzeichnung aller Anlagen/AN mit Ausnahme gem. § 6 der 11.BImSchV"
- "Anlagenweise Ausgabe der Emissionserklärung (PDF)"
- "Kennzeichnung aller Anlagen/AN mit außer Betrieb im Erklärungszeitraum"
- "Email-Versand mit variablem Inhalt"

Datenübernahme

- "Datenübernahme aus dem letzten Berichtsjahr (Datensatzbezogen)"

Importfunktionen

- "der bundeseinheitlichen Referenztabellen (XML0)"
- "der bundeslandspezifischen Referenztabellen (XML0)"
- "von Betreibern (XML1)"

Datensatzbezogene Exportfunktionen

- "des gewählten Betreibers (XML1)"

Löschung

- "Alle Daten des gewählten Berichtsjahres löschen"

Datensatzbezogene Importfunktionen

- "des gewählten Betreibers (XML1)"

PRTR

Das Modul PRTR dient zur Erfassung und Verwaltung der Daten zur PRTR-Berichtspflicht.

Auf diesem Modul basieren folgende 18 Datenfunktionen:

Funktionen auf den Daten PRTR:

- "Übersicht anzeigen"
- "Prüfen der Daten"
- "Ergebnis der letzten Prüfung anzeigen"
- "PRTR-Bericht (PDF)"
- "Vertraulichkeit auf 'Kein Schutzgrund' setzen"
- "Kennzeichnung der Freisetzungen/Verbringungen 'Keine Schwellenwertüberschreitung'"
- "PRTR-Bericht (PDF) mit Schutzgründen (UBA)"
- "Abgabebericht herunterladen (Textdatei)"
- "Email-Versand mit variablem Inhalt"

Datenübernahme:

- "Datenübernahme aus dem letzten Berichtsjahr"

Import:

- "der bundeseinheitlichen Referenztabellen (XML0)"
- "der bundeslandspezifischen Referenztabellen (XML0)"
- "von Betriebseinrichtungen (XML1)"

- "des Prüftstatus Abfall (XML4)"
- "des Prüfstatus Abwasser (XML4)"

Datenbezogener Export:

- "der gewählten Betriebseinrichtung (XML1)"

Löschen:

- "Alle Daten des gewählten Berichtsjahres löschen"

Datenbezogener Import:

- "der gewählten Betriebseinrichtung (XML1)"

13BV

Das Modul 13BV dient zur Erfassung und Verwaltung der Daten zur 13.BImSchV-Berichtspflicht.

Auf diesem Modul basieren folgende 12 Datenfunktionen:

Funktionen auf den Daten 13BV:

- "Prüfen der Daten"
- "Ergebnis der letzten Prüfung anzeigen"
- "Abgabebericht herunterladen (Textdatei)"
- "Ausgabe der GFA-Meldung (PDF)"
- "Email-Versand mit variablem Inhalt"

Datenübernahme:

- "Datenübernahme aus dem letzten Berichtsjahr"

Import:

- "der bundeseinheitlichen Referenztabellen (XML0)"
- "der bundeslandspezifischen Referenztabellen (XML0)"
- "von Großfeuerungsanlagen (XML1)"

Datenbezogener Export:

- "der gewählten Großfeuerungsanlage (XML1)"

Löschen:

- "Alle Daten des gewählten Berichtsjahres löschen"

Datenbezogener Import:

- "der gewählten Großfeuerungsanlage (XML1)"

4.3 Navigation

Die Navigation basiert zum einen auf einer statischen Modulnavigation, sowie einem objektorientieren Navigationsbaum, in dem modulabhängig die Objekthierarchie angezeigt wird.

Die Generierung der Navigation erfolgt über die Klassen des Packages

de.uba.bube.commons.struts.navtree

Bei der Erstellung der Navigation wird das komplette HTML-Fragment, abweichend von der sonstigen Umsetzung generiert. Die Generierungsmethode ist 2.264 Zeilen lang und besteht aus mehreren Kaskaden von Fallunterscheidungen.

Es wird empfohlen an der Navigation keine weiteren strukturellen Veränderungen vorzunehmen, da dies nur mit unverhältnismäßigem Aufwand durchführbar ist.

4.4 Berichtsgenerierung

Die Generierung eines Berichts in BUBE Online durchläuft mehrere Stufen. Dabei sind wie in Abbildung 5 dargestellt eine BUBE Online-spezifische Komponente und eine unabhängige Standardkomponente zu untscheiden.

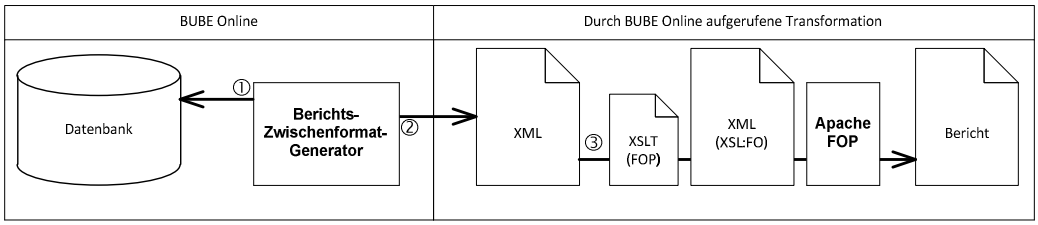

Abbildung 5: Berichtsgenerierung in BUBE Online

Wird die Generierung eines Berichts ausgelöst, so werden über eine BUBE Online-spezifische Funktion die notwendigen Daten aus der Datenbank ausgelesen (1) und als XML-Datei abgelegt (2).

Die hierfür verwendeten Objekte sind stark an dem Datenmodell orientiert und ähneln mehr oder weniger den Sturkturen in der XML1-Struktur.

Diese XML-Datei ist nun die Grundlage für die Berichtserzeugung. Die Erzeugung des Berichts selbst erfolgt mit dem Produkt Apache FOP[18] auf Basis einer berichtsspezifischen Transformationsvorschrift.

Aus der in Schritt (2) entstandenen XML-Datei wird nun durch eine XSL-Transformation[19] eine FOP-kompatible Datei erzeugt (3) und diese anschließend mit dem FOP-Generator in den endgültigen Bericht, z.B. im Format PDF, umgewandelt.

[18] http://xmlgraphics.apache.org/fop/

[19] http://de.wikipedia.org/wiki/XSL_Transformation

Die XSL-Transformation und das Produkt Apache FOP sind unabhängig von BUBE Online. Wird das Ausgangsformat der XML-Datei (Schritt (2)) korrekt erzeugt, wird auch immer der Bericht korrekt erzeugt.

Wiederverwertbarkeit

Aufgrund der Tatsache, dass die Berichtsgenerierung in eine Exportphase und eine Transformationsphase unterteilt ist, kann – *unter der Voraussetzung, dass die XML-Datei in der Exportphase (2) vollständig generiert werden kann* – die Berichtsgenerierung auf Basis dieses XML-Formats wiederverwendet werden.

5 Zusammenfassung

Der Kern des Verfahrens BUBE Online ist ein plausibler und vollständiger Datenbestand, der in einem geeigneten Datenformat vorliegt, um die aus den Verfahrensrahmenbedingungen entstehenden Berichts- und Prüfpflichten abzudecken.

Für die Herstellung dieses Datenbestandes liefert das Softwaresystem BUBE Online die Grundlage, indem es unterschiedlichen Akteuren innerhalb des Verfahrens verschiedene Interaktionen mit dem Datenbestand ermöglicht.

Auf abstrakter Ebene unterteilt sich das System in die in Abbildung 6 dargestellten vier Komponenten und weist damit einen klassischen Aufbau auf.

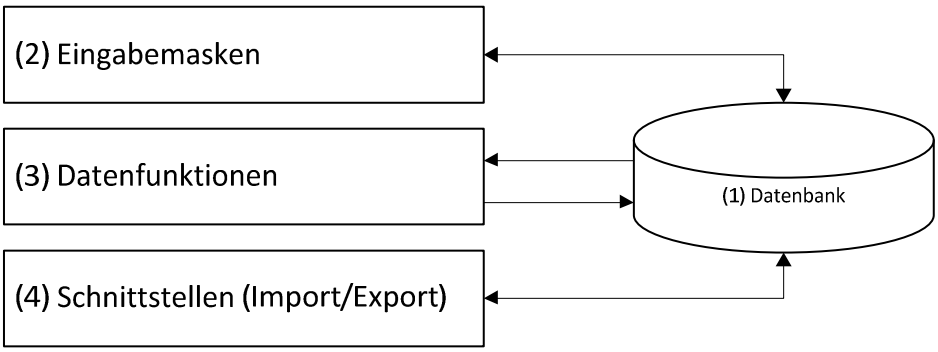

Abbildung 6: Funktionsgruppen von BUBE Online

Die zentrale Komponente der Datenbank (1) ist grundsätzlich der Ausgangspunkt für alle weiteren Komponenten. Entweder liefert sie Daten oder nimmt Daten entgegen. Liegen die Daten konsistent vor, so können sie weiterverarbeitet[20] werden.

Die Eingabemasken (2) dienen zur konsistenten Eingabe und Veränderung von Daten in der Datenbank (1). Auf dem bestehenden Datenbestand gibt es für unterschiedliche Akteure Funktionen (3), die auf den Daten implementiert sind, um bestimmte Arbeitsabläufe zu vereinfachen, die Konsistenz sichern (z.B. Freisetzungsberechnungen) oder Auswertungen über die Daten erstellen. Analog zu den Eingabemasken gibt es für den Datenaustausch entsprechende Import/Export-Schnittstellen (4).

Diese Komponenten sind in sich abgeschlossen und bieten somit eine gute Strukturierung, um sie einzeln in Bezug auf ihre Wiederverwendbarkeit zu prüfen.

5.1 Datenbank / Persistenzframework

Das vorliegende Datenmodell ist ein Datenspeicher, der die aktuelle Datenlage repräsentiert. Auf diesem Modell basiert das vollständige Softwaresystem BUBE Online, die Rechteverwaltung, die Geschäftslogik, die Masken und die Berichtsgenerierung.

[20] Ein gutes Beispiel hierfür ist BUBE UDA, mit dem sehr einfach Auswertungen auf dem Datenbestand durchgeführt werden können. Dies ist ein ideales Beispiel einer Weiternutzung der Daten, unter der Annahme, dass die Daten plausibel sind.

Da eine Software grundsätzlich stark an ihren Speicher und die Speicherstruktur gekoppelt ist, würde ein vollständig neues Datenmodell die Nutzung bestehender Werte aus BUBE Online weitestgehend verhindern.

Die folgende Tabelle gibt einen Überblick über die Auswirkungen einer Änderung des Datenmodells. In der letzten Spalte ist die Art der Auswirkung angegeben (K=Konzeption notwendig, N=Neuimplementierung, M=Migration, G=geringe Änderungen, -=nicht vorhanden).

Auswirkung	Erläuterung	Art
Datenmodell	Wie oben beschrieben	N
Eingabemasken	Da die Eingabelogik sich stark auf das Geflecht von Katalogtabellen (R-Tabellen) und den Objekttabellen bezieht, müssen die Datenbindungen grundlegend überarbeitet werden. Da mit MIFLEX der Übergang von einer objektorientierten zu einer prozessorientierten Eingabe vollzogen werden soll, muss die Eingabelogik in jedem Fall ausgetauscht werden.	N
Geschäftslogik	Da die Geschäftslogik unmittelbar auf dem Datenmodell arbeitet, muss diese entsprechend umfassend überarbeitet werden.	N
Rechtesystem	Die Filterung der Datenrechte beruht auf einem dem Datenmodell inhärentem Konzept der Abbildung auf Bundesland, Behördenkennung, AKZ und ggf. Arbeitsstätten und GKZ. Die im System verwendeten Filterdefinitionen müssen analysiert werden und an ein verändertes Konzept angepasst werden.	K,N
Berichtsgenerierung	Die Berichtsgenerierung basiert auf einer Abbildung des Datenmodells in ein XML-Zwischenformat, welches dann nach PDF überführt wird. Zwar kann das Zwischenformat beibehalten werden, die Hauptaufwände würden aber auf das Erstellen des Zwischenformats entfallen.	N
Schnittstellen	Die Erzeugung von Exportdaten und der Import von externen Daten muss komplett neu erstellt werden, damit die Daten den korrekten Datenfeldern zugeordnet werden können	N
Vorgelagerte Systeme	Indirekte Auswirkungen, sofern nicht direkt auf die Datenbank von BUBE Offline zugegriffen wird, siehe andere Module	-
Nachgelagerte Systeme	Für Auswertungen, die in BUBE UDA realisiert wurden, fällt entsprechender Migrationsaufwand auf das neue Datenmodell an. Die Software BUBE UDA selbst ist aufgrund ihres Konzepts nicht betroffen und kann unverändert bleiben.	M

Schulung	DB-Administratoren, Vielfach ist Know-How bzgl. des Datenmodells aufgebaut worden, das wieder neu aufgebaut werden muss.	ja

5.2 Eingabemasken

Die Änderung der Eingabemasken und ihrer Logik kann zu einem Großteil ohne Änderung des Datenmodells und damit verbundener Komponenten erfolgen. Kleinere Änderungen oder Ergänzungen werden vermutlich notwendig sein, ein Großteil wird jedoch überführbar sein.

Die folgende Tabelle gibt einen Überblick über die Auswirkungen einer Änderung der Eingabemasken. In der letzten Spalte ist die Art der Auswirkung angegeben (K=Konzeption notwendig, N=Neuimplementierung, M=Migration, G=geringe Änderungen, -=nicht vorhanden).

Auswirkung	Erläuterung	Art
Datenmodell	Da das Datenmodell von BUBE Online stark auf die Fachobjekte bezogen ist und auch mit anderen Masken wieder Fachobjekte verwaltet werden, wären die Einflüsse auf das Datenmodell gering.	G
Eingabemasken	Wie oben beschrieben	N
Geschäftslogik	Werden aufgrund der Oberflächenänderungen neue Datenobjekte notwendig, weil z.B. die Darstellungsobjekte anders geschnitten sind, muss auch die Geschäftslogik angepasst werden. Es ist davon auszugehen, dass eine Anpassung einer Neuimplementierung gleich kommt.	N
Rechtesystem	Da das Rechtesystem stark mit dem Datenmodell verknüpft ist und recht einfach gestaltet ist, wären hier vermutlich keine bis geringe Änderungen notwendig.	G
Berichtsgenerierung	Da die Berichtsgenerierung unmittelbar auf dem Datenmodell arbeitet, würden sich hier nur Änderungen bei Änderung des Datenmodells ergeben.	-
Schnittstellen	Da die Schnittstellen unmittelbar auf dem Datenmodell arbeiten, würden sich hier nur Änderungen bei Änderung des Datenmodells ergeben.	-
Vorgelagerte Systeme	Da vorgelagerte Systeme nur auf den Schnittstellen basieren, würden sich hier nur Änderungen bei Änderung der Schnittstellen ergeben.	-
Nachgelagerte Systeme	Da nachgelagerte Systeme in der Regel auf den Datenbestand oder Exporten arbeiten, ergeben sich hier keine Änderungen.	-

Schulung	Benutzerschulungen	ja

5.3 Datenfunktionen

Die Datenfunktionen berechnen aus bestehenden Daten neue Daten, indem sie diese Daten mit Algorithmen verarbeiten. Die Algorithmen selbst laufen i.d.R. auf dem Datenmodell. Die Benutzerinteraktion ist meistens auf die Abfrage von Parametern beschränkt.

Die folgende Tabelle gibt einen Überblick über die Auswirkungen einer Änderung der Datenfunktionen. In der letzten Spalte ist die Art der Auswirkung angegeben (K=Konzeption notwendig, N=Neuimplementierung, M=Migration, G=geringe Änderungen, -=nicht vorhanden).

Auswirkung	Erläuterung	Art
Datenmodell	Da die Datenfunktionen auf dem Datenmodell arbeiten, ergeben sich keine oder nur minimale Änderungen. Umgekehrt hätte eine Änderung am Datenmodell eine Neuimplementierung der Datenfunktionen zur Folge.	G
Eingabemasken	Bei einer Änderung der Datenfunktionen muss auf eine Anpassung der Eingabemasken erfolgen.	M
Geschäftslogik	wie beschrieben	N
Rechtesystem	Da das Rechtesystem stark mit dem Datenmodell verknüpft ist und recht einfach gestaltet ist, wären hier vermutlich keine bis geringe Änderungen notwendig.	G
Berichtsgenerierung	Da die Berichtsgenerierung unmittelbar auf dem Datenmodell arbeitet, würden sich hier nur Änderungen bei Änderung des Datenmodells ergeben.	-
Schnittstellen	Da die Schnittstellen unmittelbar auf dem Datenmodell arbeiten, würden sich hier nur Änderungen bei Änderung des Datenmodells ergeben.	-
Vorgelagerte Systeme	Da vorgelagerte Systeme nur auf den Schnittstellen basieren, würden sich hier nur Änderungen bei Änderung der Schnittstellen ergeben.	-
Nachgelagerte Systeme	Da nachgelagerte Systeme in der Regel auf den Datenbestand oder Exporten arbeiten, ergeben sich hier keine Änderungen.	-
Schulung	Benutzer	Ja

5.4 Schnittstellen Import/Export

Die aktuellen Schnittstellen definieren ein Import- und Exportformat, mit dem es möglich ist, die Daten in vor- oder nachgelagerte Systeme bei allen Akteuren zu nutzen. Eine Umstellung auf ein anderes Format, z.B. auf XU:Betrieb, hat Auswirkungen auf unterschiedliche Systemteile.

Die folgende Tabelle gibt einen Überblick über die Auswirkungen einer Änderung der Schnittstellen. In der letzten Spalte ist die Art der Auswirkung angegeben (K=Konzeption notwendig, N=Neuimplementierung, M=Migration, G=geringe Änderungen, -=nicht vorhanden).

Auswirkung	Erläuterung	Art
Datenmodell	Sofern die Daten inhaltlich weiter den Objekten im Datenmodell entsprechen, sollten sich hier nur minimale Anforderungen ergeben, analog zu den üblichen Wartungssituationen.	-
Eingabemasken	Da der Import/Export nur auf Datenebene arbeitet, sind hier keine Änderungen zu erwarten.	-
Geschäftslogik	Da der Import/Export nur auf Datenebene arbeitet, sind hier keine Änderungen zu erwarten.	-
Rechtesystem	Da das Rechtesystem stark mit dem Datenmodell verknüpft ist und der Import/Export auf Datenebene als Transformation zwischen Datentransportformat und Datenmodell arbeitet, wären hier vermutlich keine bis geringe Änderungen notwendig.	-
Berichtsgenerierung	Da die Berichtsgenerierung unmittelbar auf dem Datenmodell arbeitet, würden sich hier nur Änderungen bei Änderung des Datenmodells ergeben.	-
Schnittstellen	Die Umsetzung der Schnittstellen muss neu implementiert werden.	N
Vorgelagerte Systeme	Bei vorgelagerten Systemen, die z.B. eine Importdatei für BUBE Online erzeugen, müssen Migrationsaufwände von den Betreibern der Systeme getragen werden.	M
Nachgelagerte Systeme	Bei nachgelagerten Systemen, die auf einem Export aus BUBE Online basieren, muss mit Migrationsaufwand bei den Betreibern der Systeme gerechnet werden.	M
Schulung	Benutzer	ja

6 Fazit

Die folgende Tabelle fasst nochmal alle Auswirkungen in Bezug auf die vier abstrakten Komponenten zusammen. Bitte beachten Sie, dass teilweise starke Abhängigkeiten zwischen den Auswirkungen und den unterschiedlichen Komponenten bestehen. Diese sind in den vorstehenden Tabellen beschrieben.

Auswirkung	Datenbank	Eingabemasken	Datenfunktionen	Schnittstellen
Datenmodell	N	G	G	-
Eingabemasken	N	N	M	-
Eingabelogik	N	N	N	-
Rechtesystem	K,N	G	G	-
Berichtsgenerierung	N	-	-	-
Schnittstellen	N	-	-	N
Vorgelagerte Systeme	-	-	-	M
Nachgelagerte Systeme	M	-	-	M
Schulung	X	X	X	X

(K=Konzeption notwendig, N=Neuimplementierung, M=Migration, X=notwendig, -=nicht vorhanden)

Aufgrund der in BUBE Online genutzten Programmierparadigmen ist die Geschäftslogik über alle Komponenten verteilt. Für eine Neuimplementierung kann daher das Altsystem *nicht* als Spezifikation im Sinne „Design by example“ genutzt werden, da die Wahrscheinlichkeit sehr hoch ist, dass das neu erstellte System nicht vollständig sein wird.

Der Hintergrund ist hier die in Kapitel 3.2 und 3.3 dargestellte Quellcodequalität, die die Analyse der Software entsprechend aufwändig gestaltet. Hinzu kommen viele Stelle mit inhärenter Logik, wie in diesem Beispiel[21] aus dem Befüllen der Textfelder in der Betriebsstätte im Modul 11BV:

```
if (["s_gemde" ist leer])) {

  if ([ort ist nicht leer) {

    [Lese aus R1003 den Datensatz zum Ort]

    [Wenn R1003-Satz gefunden, trage s_gemde ein]

    [Sonst setze Inhalt]

}}
```

[21] de.uba.bube.actions.EBetrstAction.getFormFields(ActionForm, HttpServletRequest), hier als Pseudocode, da der originale Quellcode 31 Zeilen lang ist.

Bei einer Neuimplementierung kann nicht davon ausgegangen werden, dass 100% dieser Stellen gefunden oder mindestens korrekt analysiert werden, sofern nicht in eine sehr umfassende Analysephase investiert wird oder umfangreiche Kenntnisse bezüglich des Quellcodes vorliegen.

Auf Basis des vorliegenden Quellcodes wird empfohlen einen Migrationspfad für die Modernisierung von BUBE Online zu planen, der es erlaubt einen kontinuierlichen Übergang der Module möglich zu machen.

Aus gesamtwirtschaftlicher Sicht sollten bei Änderungen von Schnittstellenformaten zudem die Auswirkungen auf vor- und nachgelagerte System bewertet werden, die sowohl beim Auftraggeber, als auch bei Nutzern von Daten der Software auftreten können.

In Bezug auf die weitere Wartung machen die in diesem Dokument aufgezeigten Schwächen der Software zwar eine Erhaltung des Status Quo durchaus möglich, jedoch ist mit steigenden Wartungskosten, insbesondere bei der Umsetzung neuer Module oder Funktionalität zu rechnen.

Der Umfang der Migration der Komponenten, die ihren EOL erreicht haben, kommt hier der Neuimplementierung der betroffenen Systemteile gleich.