HAW HAMBURG

BACHELORTHESIS
Steven Beyermann

Entwicklung eines automatisierten Testdaten-Generators für XML-basierte Produktdaten im E-Commerce

FAKULTÄT TECHNIK UND INFORMATIK
Department Informatik

Faculty of Computer Science and Engineering
Department Computer Science

HOCHSCHULE FÜR ANGEWANDTE WISSENSCHAFTEN HAMBURG
Hamburg University of Applied Sciences

Steven Beyermann

Entwicklung eines automatisierten Testdaten-Generators für XML-basierte Produktdaten im E-Commerce

Bachelorarbeit eingereicht im Rahmen der Bachelorprüfung
im Studiengang *Bachelor of Science Technische Informatik*
am Department Informatik
der Fakultät Technik und Informatik
der Hochschule für Angewandte Wissenschaften Hamburg

Betreuender Prüfer: Prof. Dr. Olaf Zukunft
Zweitgutachter: Prof. Dr. Marina Tropmann-Frick

Eingereicht am: 22. Januar 2021

Steven Beyermann

Thema der Arbeit

Entwicklung eines automatisierten Testdaten-Generators für XML-basierte Produktdaten im E-Commerce

Stichworte

automatisiert, Testdaten, Generator, generieren, XML, WebShop

Kurzzusammenfassung

Als Ziel dieser Arbeit wird ein Testdatengenerator für WebShops implementiert, welcher automatisiert, gemäß Angaben und Spezifikationen des Anwenders, Produktdaten generiert oder in ihrer Menge reduzieren kann.

Steven Beyermann

Title of Thesis

Development of an automated testdata generator for XML-based productdata in E-Commerce

Keywords

automated, testdata, generator, generating, XML, webshop

Abstract

The goal of this thesis is the development of an testdata generator for webshops. This generator generates productdata or reduces a given dataset, by following specific commands and the input of the user.

Inhaltsverzeichnis

Abbildungsverzeichnis

1 Einleitung

1.1 Einführung

Diese Arbeit handelt von einem Testdatengenerator, welcher Daten erzeugt, die realen XML-basierten Produktdaten für einen Webshop entsprechen sollen. Die erzeugten Daten müssen sich somit auf der einen Seite an das Format und die Struktur der Produktdaten halten, welche in dem Dateiformat XML dargestellt werden. Auf der anderen Seite müssen die erzeugten Daten den Anforderungen der Tester genügen, damit diese den Shop mit den erzeugten Daten testen können. Für ihre Tests benötigen sie Daten, die in ihrer Struktur und in ihrem Verhalten im Shop, Daten aus der Realität entsprechen. Gewisse Details wie Texte oder Bilder haben dabei keinen Anspruch auf Sinnhaftigkeit

Um Testdaten zu generieren muss man sich zunächst damit auseinandersetzen, was und wie überhaupt getestet wird und was dafür benötigt wird. Wobei es bei der Auseinandersetzung erst einmal unwesentlich ist, ob dies manuell oder automatisiert erfolgt.

Bei der Generierung der Testdaten muss in diesem Szenario darauf geachtet werden, dass die erzeugten Daten genügend Varianz, also Vielfalt beinhalten. Damit die Daten für jeden unterstützten Shop angepasst, beziehungsweise anpassbar sind und dass der Benutzer, welcher Daten benötigt, genügend Einflussmöglichkeiten auf die Generierung der Daten hat.

1.2 Motivation

Die Produktdatensätze für Webshops werden immer größer und vielfältiger. Zudem basieren mehrere Shops oft auf derselben Programmgrundlage aber nicht auf den gleichen Produktdatensätzen, welche sich sogar erheblich unterscheiden können. Diese Produktdatensätze basieren zudem auf real existierenden Daten. Zum Testen der Webshops werden oftmals auch Daten benötigt, die nicht in den realen Produktdaten verfügbar sind, sondern erst später zu den Daten hinzugefügt werden.

Eine manuelle Generierung solcher Daten ist durch die Größe und Vielfalt der Produktdaten sehr zeitaufwändig. Zudem sind die Produktdatensätze oft sehr groß und können auf dem lokalen Rechner des Entwicklers nicht sinnvoll genutzt werden, da entweder der Shop sehr lange zum Hochfahren benötigt oder nicht hochfährt, weil es zu Speicherproblemen kommen würde. Es werden in vielen Fällen zwei Datensätze benötigt, ein Datensatz für das lokale Testen und einer für das Testen auf einem Server, wobei der lokale auf dem Produktdatensatz des Servers basieren sollte.

Eine automatisierte Testdaten Generierung und eine automatisierte Reduzierung der Produktdaten, auf eine lokal ausführbare Größe, ist somit weitaus wirtschaftlicher. Jede Änderung im Code sollte getestet werden und kann einen oder mehrere Shops betreffen. Diese Shops benötigen einen Produktdatensatz, der den Test unterstützt oder zu einer neuen Funktionalität im Shop passt.

1.3 Problemstellung

Die Mengen an Produktdaten in Webshops werden immer größer und vielfältiger. Zudem beinhalten Webshops immer mehr Funktionalitäten, die unter anderem Informationen aus den Produktdaten benötigen. Die Produktdaten der Webshops basieren allerdings meist auf realen Daten. Weswegen das manuelle Erstellen von Produktdaten in dem meisten Fällen zu aufwändig ist, vor allem bei Webshops, die bereits aktiv und für Kunden zugänglich sind. Ebenso gibt es auch Webshops, die anteilig oder in Gänze auf derselben Implementierung basieren aber jeweils andere Produktdatensätze nutzen.

Es gibt Funktionalitäten, die auf existenten Inhalten aus den Produktdaten basieren, aber es gibt auch Funktionalitäten, die eingebaut werden sollen, wo es noch keine entsprechenden Inhalte in den Produktdaten gibt. Das Letztere kann zum Beispiel der Fall sein, wenn die Inhalte noch nicht eingepflegt wurden oder aber das Sortiment in dem Webshop in der Zukunft erweitert wird und die Funktionalität vorher schon integriert werden soll.

Ebenso wachsen die Produktdaten stetig an und können dabei so groß werden, dass sie auf einem lokalen Rechner nicht mehr nutzbar sind. Die Daten können dabei eine Größe erreichen, die entweder die Zeiten zum Hochfahren des Shops so weit erhöhen, dass es für viele Tests einfach zu lange dauern würde. Oder die Produktdaten erreichen eine Größe, die es dem Webshop unmöglich macht hochzufahren, weil es zu Speicherproblemen kommt.

Mit einer automatisierten Generierung lassen sich menschliche Fehler bei der Erzeugung von Testdaten reduzieren. Zudem wird bei der automatisierten Erzeugung der Testdaten, im Vergleich zur manuellen Erzeugung, auch deutlich Zeit eingespart. Darüber hinaus lassen sich, durch eine automatisierte Reduzierung von Produktdaten, stets aktuelle Datensätze für lokale Tests erzeugen. Diese Produktdaten passen dann zu den Produktdatensätzen, die auf dem Server zu Testzwecken abgelegt werden.

Das Ziel dieser Arbeit ist die exemplarische Entwicklung und Implementierung eines Testdatengenerators, welcher die Subanwendungen eines Datengenerators und eines Datenreduzierers (Minifizierers) enthält. Der Produktdatengenerator hat die Aufgabe Produktdaten zu generieren und somit existente Produktdatensätze zu erweitern . Der Minifizierer hat die Aufgabe Produktdatensätze in ihrer Masse zu reduzieren, damit diese Datensätze lokal ausführbar sind.

Die Subanwendung des Produktdatengenerators soll als Basis für die Datengenerierung, reale und existente Produktdaten nutzen und dann die existenten Datensätze, um generierte Daten zu erweitern. Die Subanwendung des Minifizierers soll ein vorgegebenen Datensatz in seiner Menge reduzieren können, damit Datensätze, die für die lokale Anwendung zu groß sind, lokal nutzbar gemacht werden. Bei beiden Subanwendungen muss darauf geachtet werden, dass die, für die spezifischen Anwendungsfälle benötigte Struktur und Inhalte der Daten erhalten bleiben.

1.4 Gliederung

Die Arbeit gliedert sich in Neun Bestandteile, wobei der erste Teil die Einleitung darstellt. Hier wird ein kurzer Überblick in die notwendigen Themengebiete gegeben und die Zielsetzung der Arbeit definiert. Danach folgt der Abschnitt zwei mit den Grundlagen und Abschnitt drei mit den verwandten Arbeiten. Den vierten Abschnitt der Arbeit bildet die Analyse. Hier wird der in dieser Arbeit entwickelte Testdatengenerator hinsichtlich der Anforderungen untersucht, mit existenten und vergleichbaren Testdatengeneratoren verglichen und der Aufbau, der von diesem Testdatengenerators erzeugten Test- beziehungsweise Produktdaten erklärt. Im fünften Abschnitt folgt dann, basierend auf Abschnitt vier, die Konzeption des Testdatengenerators. Zur Konzeption gehören mitunter die System-Architektur und die Eingabesystematik des Testdatengenerators. Der sechste Abschnitt besteht aus der Entwicklung des Testdatengenerators, wie dieser Daten generiert und des Produktdaten Minifizierers, welcher einen speziellen Teil des Testdatengenerators darstellt. Im siebten Abschnitt folgt, basierend auf Abschnitt sechs, die Optimierung, hier wird erläutert wie sich der Testdatengenerator hinsichtlich der Performance, der Speicherausnutzung und der Laufzeit verhält. Abschnitt acht beinhaltet die Diskussion und die Evaluation der Erfüllung des Zieles, welches im Laufe dieser Arbeit angestrebt wurde und abstrahiert den Generierungsprozesses, welcher innerhalb dieser Arbeit entwickelt wird. Abschnitt neun beinhaltet die Zusammenfassung und beinhaltet ergänzend zu Abschnitt acht, was der Testdatengenerator bisher noch nicht kann und welche Anforderungen noch hinzukommen könnten, zudem werden noch weitere mögliche Erweiterungen dargestellt.

2 Grundlagen

2.1 Testen

Das Testen ist die Methodik zur Sicherstellung der Qualtität und der Sicherheit eines Systems. Durch das Prüfen und Überprüfen lassen sich Fehler und Mängel aufdecken, deren Behebung die Qualität und die Zuverlässigkeit des Systems verbessert.

2.1.1 Motivation zum Testen

Testen ist eine Form der Qualitätssicherung und der Sicherheitsüberprüfung. Mit Hilfe von Tests lässt sich sicherstellen, dass das entwickelte System keinen Schaden verursacht. Ebenso kann man durch das Testen mögliche Mängel und Fehler aufdecken, die die Funktionalität beeinträchtigen können.

”Software ist allgegenwärtig! Es gibt kaum noch Geräte, Maschinen oder Anlagen, in denen die Steuerung nicht über Software bzw. Softwareanteile realisiert wird.” [Sch83]

Durch diese Allgegenwärtigkeit der Software resultiert eine hohe Abhängigkeit des Menschen, von eben dieser Software. Dementsprechend will man sich auch stets auf die Zuverlässigkeit der Software verlassen können. Software wird hingegen von Menschen entwickelt, die nicht fehlerfrei sind und diese Fehler finden sich auch in der entwickelten Software wieder.

Softwarefehler lassen sich nicht ausschließen und können in einigen Fällen sogar Sach- oder Vermögensschäden verursachen. Als Verursacher solcher Schäden entstehen dann weitreichende Risiken und Schäden an der eigenen Reputation. Eine Software ist niemals fehlerfrei und selbst die besten Tests werden niemals alle Fehler aufzeigen können. Allerdings muss man auch nicht alle Fehler ausbessern, es genügt jene Fehler zu finden, die zu Schäden oder Einschränkungen bei der typischen Produktanwendung oder naheliegenden Fehlanwendung, führen können.

2.1.2 Prinzipien des Testens

Das Testen wird von bestimmten Prämissen und Formen des Denkens bestimmt und begleitet. Diese Prämissen lassen sich zu Prinzipien zusammenfassen. Die folgenden Prinzipien sind eine beispielhafte Auswahl und haben keinen Anspruch auf Vollständigkeit.

Die Prinzipien des Testens lassen sich wie folgt zusammenfassen:

1. Tests zeigen Fehler auf und sind kein Nachweis für die Absenz von Fehlern.
2. Erschöpfendes Testen ist nicht möglich, außer in Ausnahmefällen bei trivialen Testobjekten.
3. Testaktivitäten sollte man so früh wie möglich starten.
4. Fehler sind nicht gleichmäßig über ein System verteilt, sondern sammeln sich oft an bestimmten Stellen.
5. Testfälle müssen regelmäßig geprüft, erweitert und angepasst werden.
6. Tests müssen an den Kontext, die Einsatzumgebung und die Rahmenbedingungen der Anwendung angepasst werden.
7. Nur weil man keine Fehler findet bedeutet es noch nicht, dass die Anwendung oder das System auch sinnvoll operiert.

Die oben stehenden Prinzipien wurden aus folgenden Quellen heraus erarbeitet und zusammengefasst: [Cle10, Lig09, Sch83, Mü11, Sch05]

2.1.3 Psychologische und ethische Aspekte des Testens

Ein Tester muss die richtige Distanz wahren können, er muss den Entwicklern gegenüber eine unterstützende und faire Rolle einnehmen, aber genug Abstand wahren, um Fehler konstruktiv aufdecken zu können. Ein Tester kümmert sich nicht um die Korrektur der aufgedeckten Fehler, er konzentriert sich stattdessen nur auf die Aufdeckung und das Erfassen von Fehlern. Entwickler werden oft als konstruktives Element betrachtet und Tester als destruktives Element, dabei erfordert das Testen genauso wie das Entwickeln ein hohes Maß an Kreativität und beides stellt eine intellektuelle Herausforderung dar. Wenn man das Testen und Entwickeln konstruktiv miteinander verbindet, verbessert dies signifikant die Qualität des resultierenden Produktes. Ein Tester sollte stets die korrekte Funktionalität eines zu testenden Systems und die Sicherheit der potenziellen Anwender im Blick behalten.

”Menschen machen Fehler, aber sie geben es nur sehr ungern zu! Ein Ziel des Testens von Software ist die Aufdeckung von Abweichungen gegenüber der Spezifikation bzw. den Kundenwünschen.” [Sch83]

2.1.4 Das Testen im Software Lebenszyklus

Der Grundgedanke des Testens während des Software Lebenszyklus, ist es nach jedem Schritt des Entwicklungsprozesses das entwickelte System, aber auch die Herangehensweise zu prüfen. Durch diese Form des Testens lassen sich nicht nur funktionale Fehler eines System aufdecken, sondern auch systematische Denkfehler oder Fehler hinsichtlich der Sicht auf das System aufdecken.

"Jedes Softwareentwicklungsvorhaben wird in aller Regel orientiert an einem im Voraus ausgewählten Vorgehensmodell (auch Entwicklungsmodell, Softwareentwicklungs oder Softwareentwicklungslebenszyklus-Modell genannt) geplant und durchgeführt."[Sch83]

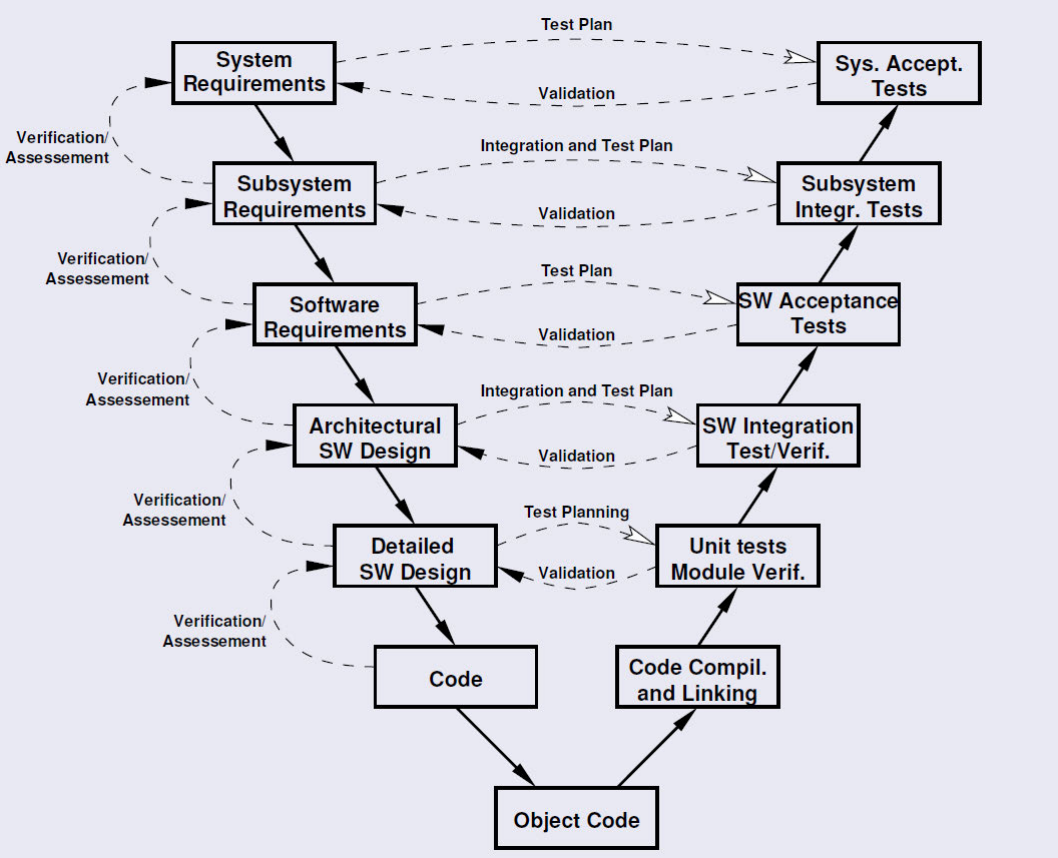

Abbildung 2.1: V-Modell des Bundes und der Länder

nach Scholz, Abb.7-2 [Sch05]

Die Abbildung 2.1 beschreibt prinzipiell ein Vorgehensmodell hinsichtlich der Entwicklung und der dazugehörigen Testaktivitäten. Auf der linken Seite wird der Entwicklungsprozess dargestellt, beginnend mit dem Sammeln der Anforderungen (Requirements) bis hin zur Implementation (Code). Auf der rechten Seite werden die zu den Entwicklungsschritten korrespondierenden Testaktivitäten dargestellt.

Für einen tieferen Einblick in die einzelnen Schritte siehe "3.1.2 Das allgemeine V-Modell" ab Seite 55 von [Sch83]

2.1.5 Der fundamentale Testprozess

Der fundamentale Testprozess besteht aus den folgenden Hauptaktivitäten. Erstens der Testplanung und Steuerung, zweitens der Testanalyse und dem Testentwurf, drittens der Testrealisierung und der Testdurchführung, viertens der Bewertung von Endekriterien und dem Bericht und zuletzt fünftens dem Abschluss der Testaktivität. Die oben aufgezählten Aktivitäten sind zwar sequenziell aufgelistet, können und werden sich in der Praxis allerdings häufig überschneiden oder parallel stattfinden.

Die Testdurchführung ist dabei der Teil, der am sichtbarsten ist. Wenn die Tests effizient ablaufen sollen muss genug Zeit für alle Teile eingeplant werden.

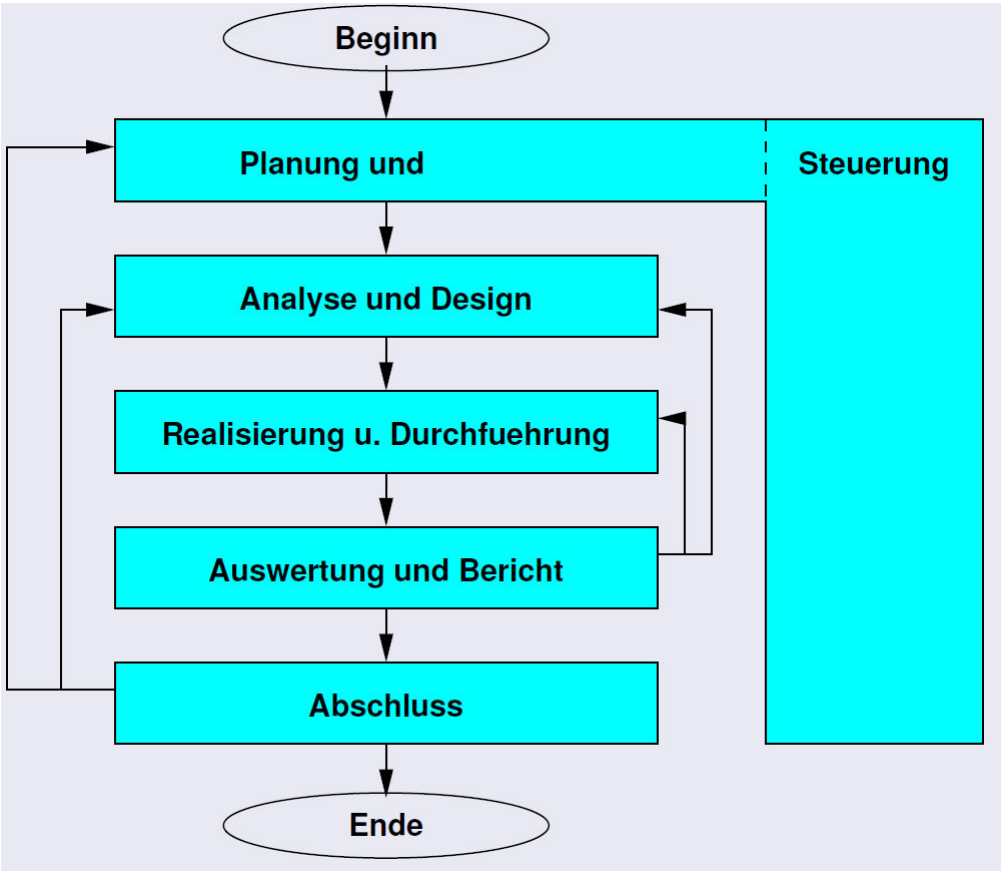

Abbildung 2.2: Der fundamentale Testprozess

nach Spillner, Abb.2-4 [Sch83]

Testplanung und Steuerung

Die Testplanung beinhaltet die Planung und Abschätzung der benötigten Ressourcen, die Definition der Testintensität für Subsysteme und die Planung und Definition der Testaspekte. Zu dieser Planung gehören zudem das Definieren der Testziele, aber auch die Festlegung der Testaktivitäten, die benötigt werden, um die Testziele zu erreichen. Weiterhin gehört zu der Planung die Festlegung der eigentlichen Teststrategie, beziehungsweise der Teststrategien. Zudem müssen die Priorisierungen der Tests festgelegt werden und die Testwerkzeuge ausgewählt oder angeschafft und schlussendlich auch evaluiert werden. Aber auch etwaige Trainingsmaßnahmen und Schulungen der Mitarbeiter müssen hier mit eingeplant werden.

Die Teststeuerung ist eine fortlaufende Aktivität, welche alle weiteren Aktivitäten begleitet. Die Teststeuerung dient zur Beobachtung der Testaktivitäten, welche hierbei mit dem, aus der Testplanung resultierenden, Plan abgeglichen werden. Sollten bei dem Abgleichen mit dem Plan Unstimmigkeiten auftauchen, so muss entweder der Plan aktualisiert oder die Aktivität angeglichen werden. Die Teststeuerung übernimmt somit auch die Rolle der Administration und der Instandhaltung hinsichtlich des Testprozesses, der Testinfrastruktur und der benötigten Mittel. Darüber hinaus muss innerhalb der Teststeuerung auch darauf geachtet werden, dass die relevanten Informationen zwischen den Mitarbeitern kommuniziert werden.

Testanalyse und Testdesign

Die Testanalyse, oder auch die Analyse der Testbasis, ist die Spezifikation, dessen was getestet werden soll. Diese Basis wird in dieser Aktivität erfasst und ausformuliert. Zur Testanalyse gehört auch die Untersuchung auf Testbarkeit, also welche Anteile des Systems oder Subsystems testbar sind. Im Zuge der Testanalyse kann es auch dazu kommen, dass die eigentlichen Anforderungen an das zu testende Objekt geändert werden müssen.

Das Testdesign, beziehungsweise der Testentwurf, umfasst die Definition logischer und konkreter Testfälle, basierend auf der ausgearbeiteten Testbasis. Ein logischer Testfall ist die direkte Ableitung eines einzelnen Testes aus der Testbasis und beinhaltet Eingabebereiche und abstrakte Sollwerte. Ein konkreter Testfall ist die spezifische Ausarbeitung

eines jeweiligen logischen Testfalles und beinhaltet konkrete Eingabedaten und konkrete Sollwerte.

Testrealisierung und Testdurchführung

Die Testrealisierung und Testdurchführung ist die Aktivität, bei der die Testabläufe und Testskripte spezifiziert werden. Darüber hinaus werden in dieser Aktivität die Testfälle ausgeführt, die Reihenfolge der jeweiligen Testfälle bestimmt und die Testdaten für die Testfälle werden erstellt und festgelegt.

Die Testrealisierung oder auch die Testimplementierung ist die direkte Überführung von Testbedingungen und den logischen Testfällen in konkrete Testfälle. Zudem werden hier auch die Testfälle, hinsichtlich ihrer Rahmenbedingungen, Vor- und Nachbedingungen detailliert beschrieben. Weitere Aspekte der Testrealisierung sind das Aufsetzen der Testumgebung und der Vorbereitung des Loggings.

Die Testdurchführung beinhaltet die Ausführung der logischen Testfälle, aber auch die Sicherstellung der Funktion der Testfälle innerhalb der Testumgebung. Die Testausführung erfolgt gemäß der, in der Testplanung getroffenen, Prioritäten und wird begleitet von einer aktiven Protokollierung des Testobjektes.

Testauswertung und Bericht

Die Auswertung oder auch die Bewertung der Endekriterien, erfolgt gemäß der festgelegten Kriterien und Prioritäten, welche dabei mit den vorhandenen Ressourcen abgeglichen werden. Bewertet wird hierbei die Abdeckung der Tests, was mit welchem Aufwand getestet wurde und ob weitere Aufwände gerechtfertigt sind. Die Bewertung ist somit ein fortlaufender Prozess der Überwachung und der Auswertung der Ergebnisse der Testdurchführung.

Der Testbericht ist der abschließende Bericht der Ergebnisse, nach Beendigung der Testaktivitäten. Der Testbericht beinhaltet die Ergebnisse der finalen Testdurchführungen, im Zusammenhang mit den getroffenen Prioritäten. Der Testbericht beinhaltet zudem auch die Auflistung der genutzten Ressourcen. Darüber hinaus wird in dem Bericht auch aufgelistet welche Testendekriterien erfüllt wurden und welche nicht.

Abschluss der Testaktivitäten

Der Abschluss der Testaktivitäten ist erreicht, wenn der finale Bericht steht, die Testaktivitäten eingestellt wurden und in diesem Testzyklus keine weiteren Korrekturen des zu testenden Systems mehr erfolgen. Der Abschluss beinhaltet die Abgabe des verfassten Berichtes, aber auch eine Evaluierung des durchgeführten Testzyklus. Die Ergebnisse der Evaluierung stellen eine weitere Planungs- und Analysebasis für kommende Testzyklen dar und bereichern künftige Testaktivitäten.

Sollte eine Korrektur, während der Testaktivitäten, innerhalb des zu testenden Systems erfolgen, so müssen die entsprechend betroffenen Aspekte und Teile des Systems neu getestet werden und die Tests gegebenenfalls angepasst werden, da jede Korrektur zu neuen Fehlern führen könnte, oder Fehler maskiert beziehungsweise demaskiert. Bei größeren Korrekturen kann es auch sein, dass man vorhergehende Aktivitäten neu absolvieren muss. (siehe Abbildung 2.2)

2.1.6 Testarten: Eine kurze Übersicht

Eine Testart ist eine bestimmte Maßnahme beim Testen und dient dem Aufdecken von Fehlern bestimmter Arten. Bei der Auswahl einer Testart wählt man eine bestimmte Blickrichtung hinsichtlich des zu testenden Objektes.

Es folgt eine Auflistung von gebräuchlichen Testarten, welche eine potenzielle Auswahl für das Testen darstellen. Diese Testarten werden innerhalb der vorher genannten Teststufen (2.1.4 und 2.1.5) genutzt und angewandt, die explizite Auswahl ist dabei stark Kontextabhängig.

1. Funktionaler Test
2. Nicht-Funktionaler Test
3. Test der Software-Struktur
4. Testen im Zusammenhang mit Veränderung und Regressionstest
5. Statisches Testen
    a) Strukturierte Gruppenprüfung
    b) Statische Analyse
6. Dynamisches Testen
    a) Black Box Tests
    b) White Box Tests
    c) Intuitive Tests und erfahrungsbasierte Testfallermittlung

Eine kurze Erläuterung der einzelnen Testarten folgt auf den nächsten Seiten.

Funktionaler Test

Mit einem funktionalem Test prüft man das Verhalten des zu testenden (Teil-)Systems, hinsichtlich seines Eingabe und Ausgabe Verhaltens. Dieses Verhalten wird dann mit den funktionalen Anforderungen abgeglichen.

Nicht Funktionaler Test

Bei einem nicht funktionalem Test prüft man das Verhalten, des zu testenden (Teil-)Systems, in Hinblick auf die Nicht-Funktionalen Anforderungen.

Test der Software-Struktur

Beim Testen der Software-Struktur wird die interne Struktur und die Architektur, des zu testenden (Teil-)Systems, geprüft. Hierbei ist das Ziel alle Strukturelemente mit Tests abzudecken.

Testen im Zusammenhang mit Veränderung und Regressionstest

Der sogenannte Regressionstest oder das Testen im Zusammenhang mit Veränderung bedeutet, dass man nach jeder Behebung eines Problems, innerhalb des zu testenden (Teil-)Systems, das ganze nochmal testen und prüfen muss, um neue Probleme oder vorher maskierte Probleme aufzudecken.

Statisches Testen

Bei dem statischen Testverfahren handelt es sich um analysierende Testverfahren. Es handelt sich somit um Testverfahren bei denen der Code nicht ausgeführt, sondern optisch und manuell geprüft wird.

Strukturierte Gruppenprüfung

Bei der strukturierten Gruppenprüfung oder dem Review handelt es sich um eine optische Prüfung des Quellcodes durch mehrere Personen, welche nicht am Schreiben des Quellcodes beteiligt waren. Diese Reviews verlaufen gemäß IEEE1028 folgendermaßen ab:

1. Planung; 2. Einführung; 3. Vorbereitung; 4. Treffen zum Review; 5. Überarbeitung des Codes; 6. Nachfolgesitzung

Zusätzlich sieht die IEEE1028, vor das folgende Rollen an die Teilnehmer des Reviews verteilt werden:

Manager, Moderator, Autor(des Quellcodes), Gutachter, Protokollführer.

Statische Analyse

Die statische Analyse ist ein Verfahren, bei welchem der verfügbare Compiler zur Analyse des Codes genutzt wird, ohne den Code dabei auszuführen. Darüber hinaus gibt es noch folgende manuelle Methoden dieser Testart:

Kontrolle hinsichtlich der Coding-Konventionen und der geltenden Standards, Kontrolle in Bezug auf Datenflussanomalien, die Kontrolle in Bezug auf Kontrollflussanomalien und die Kontrolle hinsichtlich möglicher Sicherheitsprobleme (wie zum Beispiel Puffer-Überlauf und Grenzkontrollen bezüglich der jeweiligen Eingaben oder Eingabemöglichkeiten)

Dynamisches Testen

Beim dynamischen Testen kontrolliert man das zu testende (Teil-)System während seiner Laufzeit.

Black Box Tests

Bei den Black Box Tests kontrolliert man das Ausgabe-Verhalten, des zu testenden (Teil-)Systems und gleicht die Ausgaben mit den erwarteten Werten ab. Hierbei wird der Quellcode selbst nicht betrachtet. Man sieht sich das Innere des (Teil-)Systems nicht an, man schaut lediglich auf die eingegebenen Daten und die vom (Teil-)System erzeugten Ausgabedaten.

White Box Tests

Bei den White Box Tests betrachtet man den Code und erstellt manuell oder automatisiert einen Kontrollflussgraphen. Mit Hilfe des Kontrollflussgraphen wird dann ermittelt, wie effizient der Quellcode aufgebaut und strukturiert ist. Mit dieser Testart lässt sich auch ungenutzter, also toter Code, ermitteln.

Intuitive Tests und Erfahrungsbasierte Testfallermittlung

Intuitive Tests und Erfahrungsbasierte Testfallermittlung sind Tests, die auf der Erfahrung der jeweiligen Tester basieren. Hierbei kontrolliert der jeweilige Tester das (Teil-)System auf für ihn typische oder häufig auftauchende Fehler und sortiert diese Tests basierend auf seiner Erfahrung.

Für einen tieferen Einblick, einer genaueren Erläuterung der Auswahl der Testarten und weitergehender Erläuterungen, bezüglich des Themas Testen, siehe [Cle10, Sch83, SL19]

2.2 XML-Dokumente und Parsen

2.2.1 Datenformat und Struktur XML

Die Abkürzung XML steht für extensible Markup Language, es handelt sich hierbei um ein Dokumentenformat, welches auf einem regelgebundenem Markup basiert. XML hat das Ziel „[...]Interoperabilität zwischen den verschiedenen Webanwendungen zu erleichtern[...]“ [Dob99].

XML stellt somit eine standardisierte und definierte Struktur dar, mit welcher sich Daten austauschen lassen. Ein XML-Dokument besteht aus einem optionalem Prolog und einer essenziellen Dokumenteninstanz. Die Dokumenteninstanz beinhaltet die eigentlichen Daten, welche hierarchisch, genauer gesagt baumartig, strukturiert sind. Innerhalb dieser Dokumenteninstanz kommen XML-Elemente (Entities) vor, welche durch umschließende Tags dargestellt werden. Für jedes Start-Tag(„<...>“) gibt es auch ein End-Tag(„</...>“) .

Entities besitzen immer einen Namen, das ist der Text innerhalb eines Tags, welcher zwischen Start und End-Tag identisch sein muss. Entities besitzen zudem optional noch einen Wert(Value), der sich zwischen den Tags befindet. Gibt es keinen solchen Value, so kann man Start- und End-Tag auch direkt verbinden(„<.../>).

Ein Value kann entweder leer sein, einfache Zeichenfolgen oder weitere Entities beinhalten. Darüber hinaus kann eine Entity auch XML-Attribute(Attributes) beinhalten, welche innerhalb des Start-Tags oder des Verbundenen-Tags stehen („<... AttributName=“Attributwert“ ...>“ oder „<... AttributName=“AttributWert“ .../>“).

Für einen detaillierten Einblick in die Struktur und die Ziele eines XML-Dokumentes siehe [BM98, Dob99].

2.2.2 Parsen eines XML-Dokumentes in Java

Für das Parsen eines XML-Dokumentes stellt Java verschiedene Bibliotheken zur Verfügung. Diese lassen sich primär in folgende Gruppen einteilen, die Highlevel-Parser und die Lowlevel-Parser.

Lowlevel-Parser

Die Lowlevel-Parser sind eine simple Java API, genauer gesagt eine Abstraktionsschicht. Die Lowlevel-Parser stellen keine weitere Verarbeitungslogik zur Verfügung, außer dass sie das simple Lesen des XML-Dokumentes ermöglichen. Jede weitere Verarbeitungs-Logik, wie das Abstrahieren oder Verwalten der gelesenen Daten wird komplett dem Entwickler überlassen. Lowlevel-Parser können daher im Vorteil gegenüber Highlevel-Parsern sein, wenn ein XML-Dokument spezielle Verarbeitungsanforderungen besitzt.

Highlevel-Parser

Highlevel-Parser basieren jeweils auf einem oder mehreren Lowlevel-Parsern und erweitern diese um eine eigene, vorgefertigte Verarbeitungslogik. Ein Highlevel-Parser stellt somit die Funktionalität des Einlesens eines XML-Dokumentes zur Verfügung und verarbeitet die eingelesenen Daten direkt. Die Verarbeitung folgt voreingestellten Strukturen, welche dann auch das Format und die Struktur der Eingabedaten (das XML-Dokument) bestimmen. Die resultierenden Objekte, die ein Highlevel-Parser erzeugt, folgen somit auch dem Format des Parsers. Diese Parser bieten beschränkte Möglichkeiten die Ein- und Ausgabedaten zu konfigurieren.

Highlevel-Parser sind dann im Vorteil, wenn die Eingabe- und Ausgabedaten einem bestimmten XML-Schema folgen, an welches die spezifischen Parser angelehnt sind.

Für einen tieferen Einblick in das Thema Java und XML-Parser siehe [ME06]

3 Verwandte Arbeiten

Dieses Kapitel soll einen Einblick in verwandte Arbeiten geben, diese Arbeiten beschäftigen sich dabei entweder mit der Generierung von Testdaten oder der Generierung und Verarbeitung von XML basierten Datensätzen.

3.1 Entwicklung eines Tools zur Generierung von Testdaten für Data Warehouses

Diese Arbeit beschäftigt sich mit der Generierung von Datensätzen, die den Anspruch haben Kundendaten und Geschäftsprozessdaten abzubilden, die so realitätsnah wie möglich sein sollen.

Der Autor orientiert sich bei der Erzeugung seiner Daten an der Struktur von realen Daten. Dabei sollen die für ihn als typisch angesehenen Datenfelder, wie Kundenvorname, Kundennachname und weitere Felder unterstützt werden. Zusätzlich zu den von dem Autor als typisch bezeichneten Datenfeldern sollen noch weitere Datenfelder befüllt werden, welche mit zufallsbasiertem Inhalt befüllt werden.

Die in dieser Bachelorarbeit beschriebene Anwendung soll zudem auch eine Kommunikation mit einem Datenbankserver aufbauen können um dort neue Tabellen anlegen oder vorhandene Tabellen abändern oder löschen zu können. Der Fokus der Anwendung liegt hierbei auf der Funktionalität und nicht auf der Performance.

Die Anwendung in dieser Arbeit erhält ihre Eingangsdaten über Datensätze im XML-Datenformat, welche dann geparst, ausgewertet und intern gespeichert werden. Diese Daten müssen nur jeweils einmal beim Starten der Anwendung geparst und verarbeitet werden und stehen dem Programm danach jederzeit zur Verfügung.

Für tiefere Einblicke in diese Arbeit siehe [Sch07]

3.2 Erzeugung von Testdaten für automatisiertes Fahren auf Basis eines Open Source Fahrsimulators

Diese Arbeit zeigt wie mit virtuellen Sensoren, eines Fahrsimulators, simulierte Testdaten generiert werden können. Ebenso wird geprüft, ob der Simulator genutzt werden kann um spezielle Testdaten zu erzeugen.

Der Anspruch beim automatisiertem Fahren ist die Gewährleistung einer hohen Zuverlässigkeit. Diese Zuverlässigkeit soll durch Tests erreicht werden. Damit die Reaktionen und möglichen Fehler reproduzierbar sind, wird eine Simulation benötigt, welche dann die Testdaten generiert. Innerhalb der Simulation werden dann Testszenarien absolviert.

Der Fahrsimulator dient in dieser Arbeit allein zur Generierung der Testdaten, wobei diese Testdaten nicht in Echtzeit ausgewertet werden müssen. Die erzeugten Testdaten, beziehungsweise die Daten der simulierten Sensoren werden anschließend an die entsprechenden Algorithmen zur Verarbeitung und Auswertung weitergereicht.

Für tiefere Einblicke in diese Arbeit siehe [LBPH07]

3.3 Modellierung und Generierung von Testdaten für datenbankbasierte Anwendungen

Diese Arbeit beschäftigt sich mit dem Ziel die Spezifikationen von Testdaten für Datenbank-basierte-Java-Anwendungen zu vereinfachen. Um diese Testdaten zu vereinfachen wurde eine eigene Sprache, die Domänen-spezifische Sprache (DSL), entwickelt.

Das in dieser Arbeit geschilderte Problem ist das umfangreiche Spezifikationsspektrum von Testdaten für Datenbank-basierte Anwendungen. Die entwickelte domänenspezifische Sprache soll dabei als Schnittstelle zur Erstellung der spezifischen Testdaten dienen.

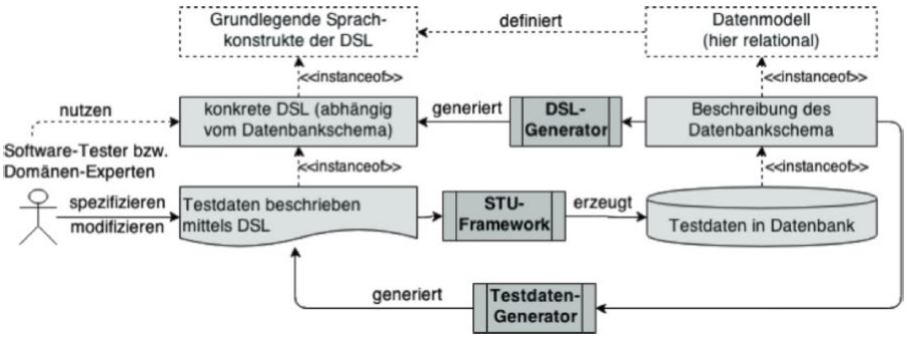

Abbildung 3.1: Überblick über den gewählten Ansatz

[MBFW14] S. 152; Abbildung 1: Überblick über den gewählten Ansatz

Der Fokus bei der Arbeit lag dabei auf der Generierung von möglichst kleinen Testdaten-Sets, welche eine hohe Testabdeckung bieten sollten. Somit sollten die Testdaten für möglichst viele fachliche Tests eingesetzt werden können. Der Autor der Arbeit weist zudem darauf hin, dass es vorher keine entsprechende Lösung gab. Auf Grund dessen wurde die Entwicklung eines Algorithmus zur entsprechenden Testdatengenerierung auch Teil dieser Arbeit.

Für tiefere Einblicke in diese Arbeit siehe [MBFW14]

3.4 Synthetic Data Generation for Big Data

Diese Arbeit beschäftigt sich mit der Problemstellung der Erzeugung großer Datenmengen, zum Validieren und Verifizieren, für Informations- und Kommunikationssysteme. Der Autor schildert warum derartige Datenmengen nicht mehr manuell generierbar sind und warum eine synthetische und automatisierte Generierung nötig ist.

Das Ziel dieser Arbeit ist somit die Generierung von großen Datenmengen und Datensätzen, welche als Testgrundlage dienen sollen. Der Autor erklärt zudem die Rolle und auch die Probleme, die mit diesen Datenmengen, der Big Data, einhergehen.

Der Autor erklärt darüber hinaus auf welchen Systemen seine Arbeit basiert und welche Frameworks und Strukturen er für seine Anwendung benötigt und benutzt. Die in dieser Arbeit erzeugte Anwendung ist für den Anwendungsfall innerhalb der Arbeit spezifiziert, hat aber auch den Anspruch für andere Anwendungsfälle konfigurierbar zu sein.

Für tiefere Einblicke in diese Arbeit siehe [For16]

3.5 Vergleichende Einordnung

In diesem Abschnitt werden die verwandten Arbeiten miteinander verglichen und hinsichtlich ihrer Themengebiete eingeordnet. Anschließend folgt zudem ein Fazit, welches zeigen soll, welche Bereiche die verwandten Arbeiten nicht abdecken und welche thematische Lücken diese Bachelorarbeit schließen soll.

Vergleichstabelle

Die folgende Tabelle vergleicht die verwandten Arbeiten. Dabei werden die Zielgebiete, Problemstellungen und erzeugten Datenmedien der verwandten Arbeiten betrachtet.

Arbeit	Format/Sprache	Ziel	Quelldaten
V1	XML	Data Warehouses	Anwendereingaben
V2	XML	Automatisiertes Fahren	existente Daten
V3	DSL	Datenbank-Anwendungen	Anwendereingaben
V4	XML,SCXML	Big Data	existente Daten
BA	XML	Webshop	Anwendereingaben, existente Daten

Die Abkürzungen innerhalb der Tabelle entsprechen den Verwandten Arbeiten wie folgt:

- V1) Entwicklung eines Tools zur Generierung von Testdaten für Data Warehouses
- V2) Erzeugung von Testdaten für automatisiertes Fahren auf Basis eines Open Source Fahrsimulators
- V3) Modellierung und Generierung von Testdaten für Datenbank-basierte Anwendungen
- V4) Synthetic Data Generation for Big Data
- BA) Die Anwendung, die in dieser Arbeit erzeugt wird

3.5.1 Fazit und Legitimation der zu entwickelnden Anwendung

Wie man in der Tabelle in 3.5.1 sieht gibt es schon einige datengenerierungs Werkzeuge die Daten im XML-Datenformat erzeugen, allerdings erzeugen diese keine Daten für einen Webshop. Zudem basieren die erzeugten Daten hinsichtlich ihrer Quelldaten, primär auf existenten Daten oder Anwenderangaben, wobei die in dieser Arbeit entwickelte Anwendung dahingehend eine gute Balance benötigt.

Die Anwendung in dieser Bachelorarbeit muss nämlich zwei grundlegende Anwendungs-Gruppen abdecken, die Reduktion von existenten Produktdaten, basierend auf Spezifikation seitens des Anwenders und die Generierung von Produktdaten, um einen existenten Datensatz zu erweitern.

4 Analyse

Ziel dieser Analyse ist es die Anforderungen und Anwendungsfälle, in Kommunikation mit den zukünftigen und potenziellen Anwendern, zu definieren und zu systematisieren. Hierfür werden in dem ersten Abschnitt zunächst einige Anwendungsfälle vorgestellt und deren Anforderungen anschließend zusammengefasst. In dem zweiten Abschnitt wird dann der Kontext, der in dieser Arbeit entwickelten Anwendung, genauer betrachtet. In dem dritten und letzten Abschnitt dieses Kapitels werden dann die Anforderungen zusammengefasst und miteinander in Verbindung gebracht.

4.1 Anwendungsfälle

Die folgenden Anwendungsfälle sind das Ergebnis von mehreren Interviews und Besprechungen mit den Entwicklern und deren Vorgesetzten. Diese Entwickler und deren Vorgesetzten sind die zukünftigen Nutzer des Entwicklerwerkzeuges, welches das Ergebnis dieser Arbeit sein wird.

Hinweise:

- Wenn Anforderungen, die den Anwendungsfällen entnommen wurden, sich doppeln würden, werden diese nur jeweils einmal in den "Entnommenen Anforderungen" aufgelistet.
- Die Beteiligten Akteure in den folgenden Anwendungsfällen sind stets Entwickler (Anwender).

Anwendungsfall Diagramm

Die folgende Abbildung 4.1 zeigt die nachfolgenden Anwendungsfälle in gesammelter Form anhand eines Anwendungsfalldiagrammes. Der Anwender beziehungsweise der Entwickler wählt sich vor der der eigentlichen Eingabe aus welche Sub-Anwendung er nutzen möchte und benutzt dann die entsprechende graphische Nutzeroberfläche (GUI).

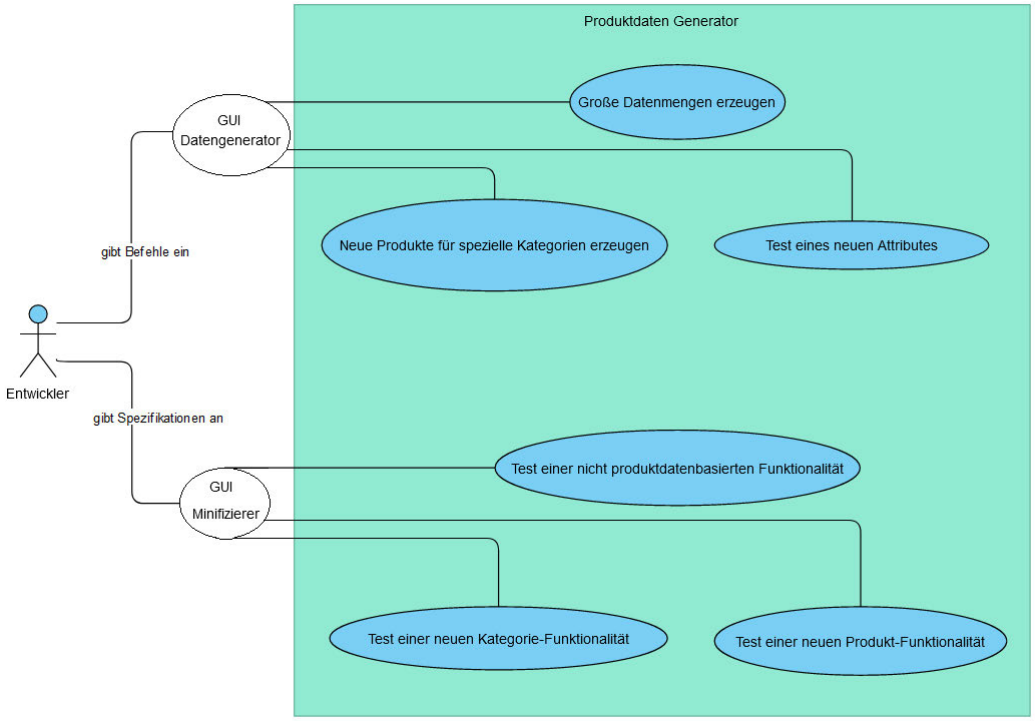

Abbildung 4.1: Anwendungsfall Diagramm

4.1.1 Große Datenmengen erzeugen

ID: UC:G001

Beschreibung:

Große Datenmengen erzeugen

Vorhaben:

Der Anwender möchte eine große Menge an Produktdaten erzeugen, um diverse Belastungstests an einem neuen Shop ausführen zu können.

Vorbedingungen:

Der Shop verfügt über eine kleine, aber vielfältige Auswahl an Produktdaten. Der Anwender übermittelt die notwendigen Instruktionen an den Produktdaten-Generator.

Nachbedingungen:

Der Produktdaten-Generator gibt eine Produktdatendatei aus, welche die neu-generierten Daten, als auch die bisherigen Daten beinhaltet. Die bisherigen Daten entsprechen inhaltlich den ursprünglichen Eingangsdaten. Die bisherigen Daten wurden zudem hinsichtlich ihrer Relationen und Attribute manipuliert. Die Manipulation der bisherigen Daten beschränkt sich auf Verweise und Verwebung der Daten, die nötig sind, um die neu generierten Produktdaten einfließen zu lassen.

Wenn man zum Beispiel ein neues Item für ein existentes Produkt erzeugt, muss das existente Produkt eine Relation erhalten, welche auf das neue Item verweist.

Entnommene Anforderungen:

- E-1: Die Anwendung muss dem Entwickler eine Schnittstelle für Eingaben bieten.
- E-2: Die Anwendung muss in der Lage sein vorhandene Produktdaten einzulesen, um diese dann verarbeiten zu können.
- E-3: Die Anwendung muss, basierend auf wenigen vorhandenen Daten, große Datenmengen mit genug Varianz erzeugen können.
- E-4: Die Anwendung muss, an den jeweiligen Shop angepasste Daten generieren können. Sie muss somit einstellbar und/oder konfigurierbar sein.

4.1.2 Neue Produkte für spezielle Kategorien erzeugen

ID: UC:G002

Beschreibung:

Neue Produkte für spezielle Kategorien erzeugen

Vorhaben:

Der Anwender möchte eine jeweils geringe Menge an Produkten für ausgewählte Kategorien erzeugen.

Vorbedingungen:

Der Shop verfügt über eine vielfältige Auswahl an Produktdaten. Der Anwender übermittelt die notwendigen Instruktionen an den Produktdaten Generator.

Nachbedingungen:

siehe UC:G001

Entnommene Anforderungen:

- E-5: Die Anwendung muss unterscheiden können welche Anteile der Daten in welchem Kontext essenziell oder optional sind. Es gibt Daten, die sind für bestimmte Kategorien essenziell und für andere Kategorien sind diese Daten optional oder irrelevant.
- E-6: Die Anwendung muss Strukturen und Relationen in den Produktdaten erkennen, wahren und erzeugen können.
- E-7: Die Anwendung muss neue Daten-Elemente (zum Beispiel Produkte), basierend auf vorhandenen Strukturen erzeugen können.
- E-8: Die Anwendung muss konfigurierbar sein, damit bestimmt werden kann welche Attribute für welche Kategorie essenziell und/oder optional sind.
- E-9: Der Anwender muss angeben können, was für Produkte in welchen Mengen zu den entsprechenden Kategorien hinzugefügt werden sollen.

4.1.3 Test eines neuen Attributes

ID: UC:G003

Beschreibung:

Test eines neuen Attributes, welches noch nicht in den realen Produktdaten existiert.

Vorhaben:

Der Anwender möchte eine neue Funktionalität im Shop testen. Diese Funktionalität benötigt ein bestimmtes Attribut innerhalb der Produktdaten. Dieses Attribut existiert allerdings noch nicht in den realen Produktdaten.

Vorbedingungen:

Der Shop verfügt über eine breite und vielfältige Auswahl an bereits existierenden Produktdaten.

Nachbedingungen:

siehe UC:G001

Entnommene Anforderungen:

- E-10: Die Anwendung muss eingelesene Daten nach Art unterscheiden können. Sie muss zum Beispiel ein Produkt von einer Attribut Definition unterscheiden können.
- E-11: Die Anwendung muss Instruktionen für neue Attribute erkennen können.
- E-12: Die Anwendung muss basierend auf Instruktionen, neue Attribute erzeugen und mit den restlichen Daten verknüpfen können.
- E-13: Der Anwender muss angeben können, welche Daten-Elemente mit dem Attribut versehen werden sollen.

4.1.4 Test einer nicht produktdatenbasierten Funktionalität

ID: UC:M001

Beschreibung:

Lokaler Test einer neuen Funktionalität des Shops, welche nicht auf den Produktdaten basiert.

Vorhaben:

Der Anwender möchte eine neue Funktionalität im Shop testen, die nicht direkt auf den Produktdaten basiert. Die Produktdaten werden allerdings benötigt, um den Shop starten zu können, ebenso sollte der Datensatz entsprechend klein sein, damit der Start des Shops schnell ausgeführt werden kann.

Vorbedingungen:

Der Shop verfügt über eine breite und vielfältige Auswahl an bereits existierenden Produktdaten. Die Produktdaten beinhalten zudem die für den Test erforderlichen Daten.

Nachbedingungen:

Der Produktdaten-Generator gibt eine Produktdaten Datei aus, welche eine reduzierte Menge der ursprünglichen Produktdaten beinhaltet. Die reduzierte Produktdaten-Menge soll dabei dennoch eine gute Verteilung und Varianz an Daten beinhalten. (Genug Produkte pro Kategorie, genug Items pro Produkt und einige wenige Bewertungen pro Item)

Entnommene Anforderungen:

- E-14: Die Anwendung muss den Befehl zur Reduktion vorhandener Datenmengen von Instruktionen zur Generierung von Produktdaten unterscheiden können.
- E-15: Die Anwendung muss hinsichtlich ihrer Untermengen einstellbar sein, aber auch einen guten Standardwert besitzen. (Mengenangabe durch den Anwender für Anzahl Produkte pro Kategorie, Items pro Produkt und Bewertungen pro Item).

4.1.5 Test einer neuen Kategorie-Funktionalität

ID: UC:M002

Beschreibung:

Lokaler Test einer neuen Funktionalität des Shops. Diese Funktionalität basiert auf bestimmten Daten innerhalb der Kategorien in den Produktdaten.

Vorhaben:

Der Anwender möchte eine neue Funktionalität im Shop testen, die auf existenten Attributen basiert. Diese Attribute befinden sich schon in den Produktdaten und kommen bei bestimmten Produkten und Items vor. Die Produkte und Items, die dieses Attribut beinhalten, liegen alle innerhalb einer bestimmten Kategorie.

Vorbedingungen:

siehe UC:M001

Nachbedingungen:

siehe UC:M001

Entnommene Anforderungen:

- E-16: Die Anwendung muss den Befehl zur Reduktion von bestimmten Datenmengen, als sowohl den Befehl für den Ausschluss bestimmter Daten von dieser Reduktion, verstehen können.
- E-17: Der Anwender muss angeben können, welche Kategorie(-n) er von der Reduktion ausschließen möchte. Diese Kategorien beinhalten somit alle Produkte aus der Ursprungsdatei.
- E-18: Der Anwender muss die Möglichkeit haben bestimmte Datenmengen herausfiltern zu können. (Ihn interessieren nur bestimmte Kategorien, also sollte er die Möglichkeit haben andere Kategorien ausschließen zu können, damit diese nicht in der reduzierten Produktdatenmenge vorkommen. Er setzt diese ausgewählten Kategorien somit auf eine ”schwarze Liste” (Black-List))

4.1.6 Test einer neuen Produkt-Funktionalität

ID: UC:M003

Beschreibung:

Lokaler Test einer neuen Funktionalität des Shops, welche auf bestimmten Anteilen innerhalb spezieller Produkte in den Produktdaten basiert.

Vorhaben:

Der Anwender möchte eine neue Funktionalität im Shop testen die auf existenten Attributen basiert. Diese Attribute befinden sich schon in den Produktdaten und kommen bei wenigen bestimmten Produkten und damit verbundenen Items vor.

Vorbedingungen:

siehe UC:M001

Nachbedingungen:

siehe UC:M001

Entnommene Anforderungen:

- E-19: Der Anwender muss die Möglichkeit haben bestimmte Produkte und Items angeben zu können, welche dann für die reduzierte Produktdatenmenge ausgewählt werden.
- E-20: Die Anwendung muss spezifisch angegebene Produkte und Items in die Auswahl, für die reduzierten Produktdaten, inkludieren können.

4.2 Kontextbetrachtung

4.2.1 Automatisiertes Generieren

Bei dem automatisierten Generieren geht es darum Zeit zu sparen, da das manuelle Generieren sehr zeitaufwändig sein kann. Das manuelle Generieren von Daten bedeutet, dass man ein kleiner Datensatz per Hand oder expliziten Konsolenbefehlen aus den Ursprungsdaten entnimmt oder neue Daten händisch einpflegt. Eine manuelle Generierung von Hand ist somit auch sehr fehleranfällig und die Pflege eines solchen Datensatzes ist dementsprechend auch aufwändiger, als wenn die gewünschten Daten automatisch erzeugt werden würden, gemäß der Eingaben und Anforderungen des Anwenders.

Dabei muss man genau analysieren, ob das automatisierte Generieren möglich ist und welchen Beschränkungen es unterliegt. Ebenso muss man darauf achten, dass die erzeugbaren Daten zu allen unterstützten Systemen passen. Damit die erzeugten Daten zu den unterstützten System passen muss man untersuchen, wo ihre Gemeinsamkeiten und ihre Unterschiede sind und was sich da in Zukunft ändern kann. Ein automatisierter Test-Daten-Generator muss also konfigurierbar sein hinsichtlich der unterstützten Systeme und auch dem Nutzer die Möglichkeit geben Einfluss auf die zu erzeugenden Daten zu nehmen.

Des Weiteren muss man sich die zu erzeugenden Daten genau ansehen und analysieren, ob alles davon generierbar ist, welche Entscheidungen der Test-Daten-Generator allein treffen kann, wo er Hilfe oder Informationen benötigt oder wo man die Entscheidung an den Nutzer auslagern muss.

Das bedeutet der Test-Daten-Generator muss mit folgenden drei Fällen umgehen können: Komplett generierbare Daten, Daten die er nur mit Hilfestellung und Informationen generieren kann und Daten die er nicht generieren kann. Die Daten, die ohne Hilfe generiert werden können, stellen dabei das kleinste Problem dar. Die Daten, die mit Hilfe generiert werden können, benötigen eine Kommunikationsmöglichkeit, damit die benötigten Informationen von außen bereitgestellt werden können. Die Daten, die nicht automatisiert generiert werden können stellen dabei das größte Problem hinsichtlich der Arbeitslogik dar. Daten die nicht automatisiert generiert werden können, müssen also von woanders beschafft werden und der Benutzer muss auch hier wieder Einfluss nehmen können.

4.2.2 Entwicklertool

Ein Entwicklertool ist ein Werkzeug, welches Entwickler als Anwender-Zielgruppe hat und diese in ihrem Arbeitsalltag unterstützen soll. Bei der Zielgruppe der Entwickler setzt man zudem voraus, dass sich diese eigenständig und fachlich mit den Funktionalitäten des Entwicklertools auseinandersetzen. Zudem lässt sich voraussetzen, dass ein Entwickler die Rolle des Entwicklertools einschätzen kann, um zu wissen welche Ergebnisse dieses Tool erzeugt und was der Entwickler an Eingaben tätigen muss, um die für ihn gewünschten Ergebnisse zu erhalten.

Ein Entwicklertool wird somit zwar von einem beschränktem Klientel benutzt, allerdings gibt es auch hier mehrere Gruppierungen, die man beachten muss, was bedeutet das man sowohl auf ihre Anforderungen und ihre Denkweise eingehen muss. Auf diese Gruppierungen einzugehen bedeutet sich zu informieren, was sie an Anforderungen haben, aber auch welche Eigenheiten man bedenken muss und in welche Richtungen man das Programm absichern muss. Ein Entwicklertool für einen geschlossenen Kreis hat zudem den Vorteil, dass man die Anwender anlernen und einweisen kann, bevor diese das Programm benutzen.

4.2.3 Einflussmöglichkeiten des Anwenders

In dem Kontext eines Entwicklertools ist es auch oft nötig, dass der Anwender anstelle des Programmes oder in Zusammenarbeit mit dem Programm Entscheidungen trifft oder erarbeitet. Damit diese Entscheidungen in Zusammenarbeit getroffen werden können ist ein Austausch zwischen Programm und Anwender essenziell.

In dem Gebiet, in dem der Testdatengenerator bei der Firma intern eingesetzt werden soll, ist es spezifisch von Nöten, dass der Entwickler dem Entwicklertool Vorgaben mitgeben kann. Diese Vorgaben betreffen vor allem die Struktur der zu generierenden Daten. Die Vorgaben betreffen zudem die Auflistung und Auswahl spezifischer Inhalte, aber auch das Ausschließen und Hinzufügen bestimmter Daten.

4.2.4 Eingabesprache

Damit der Anwender dem Programm Strukturen und Daten mitgeben kann, wird eine Eingabesprache benötigt, also eine gemeinsame Sprache, mit der der Entwickler dem Programm seine Ansprüche mitteilen kann. Diese gemeinsame Sprache wird im Zuge dieser Arbeit festgelegt und entwickelt.

4.2.5 Konfigurationen

Für die Fälle, in denen der Daten-Generator keine alleinigen Entscheidungen treffen kann oder wo er besondere Informationen benötigt, muss man dem Programm die Möglichkeit geben den Anwender abzufragen, dies lässt sich durch Konfigurationsdateien und Formulare zum Abfragen realisieren. Es handelt sich hierbei also um Dateien, die das Programm zu bestimmten Prozess Abschnitten abfragt oder einliest. Diese Dateien können bei jedem Vorgang entweder neu abgefragt, oder auch fest hinterlegt sein.

4.3 Zusammenfassung der Anforderungen

Die Anforderungen aus den einzelnen Anwendungsfällen wurden zunächst ermittelt. Nach dem Ermitteln wurden erneut Interviews mit den Entwicklern (den zukünftigen Anwender) und deren Vorgesetzten geführt, um die Vollständigkeit der gesammelten Anforderungen sicherzustellen.

4.3.1 Funktionale Anforderungen aus den Anwendungsfällen

Die Anforderungen entsprechen den Anforderungen, die in den Anwendungsfällen aufgelistet sind.

- E-1: Die Anwendung muss dem Entwickler eine Schnittstelle für Eingaben bieten.
- E-2: Die Anwendung muss in der Lage sein vorhandene Produktdaten einzulesen, um diese dann verarbeiten zu können.
- E-3: Die Anwendung muss, basierend auf wenigen vorhandenen Daten, große Datenmengen mit genug Varianz erzeugen können.
- E-4: Die Anwendung muss, an den jeweiligen Shop angepasste Daten generieren können. Sie muss somit einstellbar und/oder konfigurierbar sein.
- E-5: Die Anwendung muss unterscheiden können welche Anteile der Daten in welchem Kontext essenziell oder optional sind. Es gibt Daten, die sind für bestimmte Kategorien essenziell und für andere Kategorien sind diese Daten optional oder irrelevant.
- E-6: Die Anwendung muss Strukturen und Relationen in den Produktdaten erkennen, wahren und erzeugen können.
- E-7: Die Anwendung muss neue Daten-Elemente (zum Beispiel Produkte), basierend auf vorhandenen Strukturen erzeugen können.
- E-8: Die Anwendung muss konfigurierbar sein, damit bestimmt werden kann welche Attribute für welche Kategorie essenziell und/oder optional sind.
- E-9: Der Anwender muss angeben können, was für Produkte in welchen Mengen zu den entsprechenden Kategorien hinzugefügt werden sollen.

- E-10: Die Anwendung muss eingelesene Daten nach Art unterscheiden können. Sie muss zum Beispiel ein Produkt von einer Attribut Definition unterscheiden können.
- E-11: Die Anwendung muss Instruktionen für neue Attribute erkennen können.
- E-12: Die Anwendung muss basierend auf Instruktionen, neue Attribute erzeugen und mit den restlichen Daten verknüpfen können.
- E-13: Der Anwender muss angeben können, welche Daten-Elemente mit dem Attribut versehen werden sollen.
- E-14: Die Anwendung muss den Befehl zur Reduktion vorhandener Datenmengen von Instruktionen zur Generierung von Produktdaten unterscheiden können.
- E-15: Die Anwendung muss hinsichtlich ihrer Untermengen einstellbar sein, aber auch einen guten Standardwert besitzen. (Mengenangabe durch den Anwender für Anzahl Produkte pro Kategorie, Items pro Produkt und Bewertungen pro Item).
- E-16: Die Anwendung muss den Befehl zur Reduktion von bestimmten Datenmengen, als sowohl den Befehl für den Ausschluss bestimmter Daten von dieser Reduktion, verstehen können.
- E-17: Der Anwender muss angeben können, welche Kategorie(-n) er von der Reduktion ausschließen möchte. Diese Kategorien beinhalten somit alle Produkte aus der Ursprungsdatei.
- E-18: Der Anwender muss die Möglichkeit haben bestimmte Datenmengen herausfiltern zu können. (Ihn interessieren nur bestimmte Kategorien, also sollte er die Möglichkeit haben andere Kategorien ausschließen zu können, damit diese nicht in der reduzierten Produktdaten-Menge vorkommen. Er setzt diese ausgewählten Kategorien somit auf eine Schwarzeliste (Black-List))
- E-19: Der Anwender muss die Möglichkeit haben bestimmte Produkte und Items angeben zu können, welche dann für die reduzierte Produktdatenmenge ausgewählt werden.
- E-20: Die Anwendung muss spezifisch angegebene Produkte und Items in die Auswahl, für die reduzierten Produktdaten, inkludieren können.

4.3.2 Weitere Funktionale Anforderungen

Bei den weiteren Anforderungen handelt es sich um Anforderungen, die im Zuge der Interviews und Rücksprachen mit den zukünftigen Anwendern gesammelt wurden.

- W-1: Der Anwender soll die Möglichkeit haben die angegebenen Instruktionen und Eingaben abzuspeichern.
- W-2: Der Anwender soll die Möglichkeit haben einen eigenen Namen für die Ausgabedatei angeben zu können.
- W-3: Der Anwender soll die Möglichkeit haben die Ein- und Ausgabe-Pfade selber zu bestimmen.
- W-4: Die Anwendung soll mehrere aufeinander folgende Anfragen behandeln können.
- W-5: Die Anwendung muss vorhandene Daten direkt und ohne Seiteneffekte manipulieren können.

4.3.3 Weitere Nichtfunktionale Anforderungen

Im Folgenden werden nichtfunktionale Anforderungen aufgelistet, welche im Zuge der Interviews und Rücksprachen mit den zukünftigen Anwendern gesammelt wurden.

- N-1: Die Verarbeitungszeit einer einzelnen Generierung sollte möglichst gering gehalten werden.
- N-2: Die Ausgabedateien sollten möglichst klein gehalten werden bei der Datenreduzierung (bei der Minifizierung).
- N-3: Die Anwendung solle wenn möglich den Sekundärspeicher und nur wenn es notwendig ist den Arbeitsspeicher nutzen.

4.4 Analyse der zu generierenden Daten

Bei den zu generierenden Daten handelt es sich um Produktdaten, im XML-Dateiformat, welche für einen Webshop konzipiert sind. Diese Daten folgen den grundlegenden XML-Strukturen, besitzen darüber hinaus aber auch eigene Systematiken, Verweise und beinhalten auch andere gängige und eigene Datenstrukturen, welche gewahrt werden müssen.

4.4.1 Grundlegende Struktur

Die Produktdaten lassen sich in zwei übergeordnete XML-Elemente unterteilen, die Objekt-Deklaration und die Objekt-Definition. Ein zusammenhängendes Paar aus Objekt-Deklaration(o) und Objekt-Definition(d) bilden dabei einen Knoten. Jeder Knoten lässt sich über eine eindeutige ID identifizieren, diese ID steht dabei an beiden übergeordneten XML-Elementen dieses Knotens.

Die Knoten lassen sich weiterhin in folgende Kategorien unterteilen, Kategorie(N), Produkt(P), Item(I), Bewertung(R), Attribut-Definition(AD), Attribut-Wert(AV) und Anderes(O). Die Knoten ”Attribut Definition” und die ”Attribut Wert” bilden dabei ein spezifisches Shop-Attribut(a).

Die Knoten besitzen dann eine spezifische Auswahl aus folgenden XML-Attribut Mengen, Relation(r), Shop-Attribut(a) und Asset(asset).

4.4.2 Hierarchie der Knoten

Die Knoten werden in Hierarchieform organisiert und strukturiert, diese Strukturierung erfolgt über Verweise(Referenzen) innerhalb der XML-Attribute.

Die Abhängigkeit und Hierarchie der Knoten sieht dabei wie folgt aus:

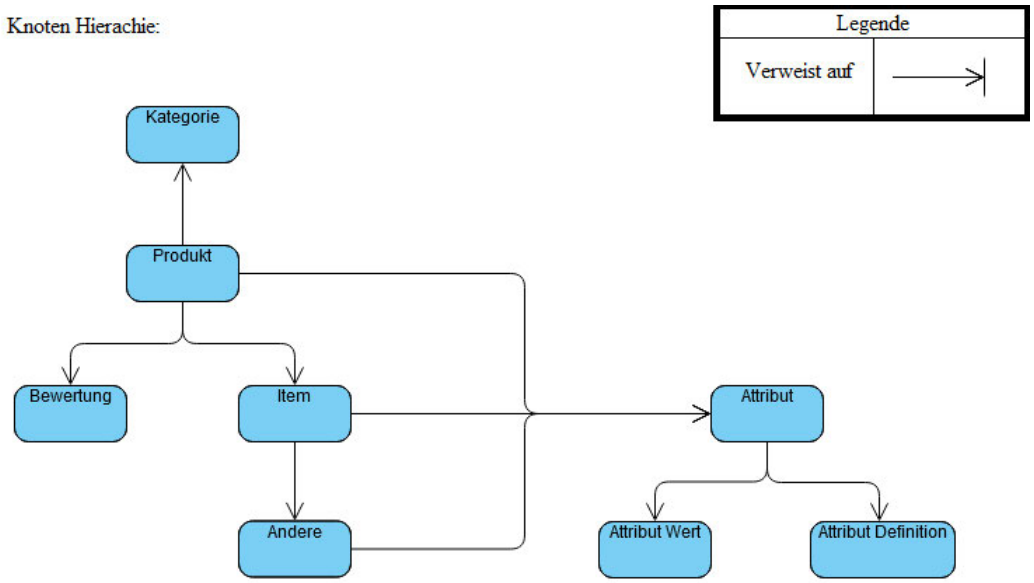

Abbildung 4.2: Knoten Hierarchie

Die Abbildung 4.2 zeigt wie die Knoten miteinander in Verbindung stehen und welcher Knoten auf welche Knoten zeigt. Der Pfeil stellt dabei die Verweisrichtung dar. Der Verweis steht bei dem Knoten, der den Ursprung des Pfeils angebunden hat und der Knoten auf den verwiesen wird steht am Pfeilspitzen-Ende. Ein Attribut hat da die Sondereigenschaft, dass mehrere Knoten auf diese verweisen können. Ein Attribut selbst ist dabei kein Knoten, sondern besteht immer aus einem Attribut-Definitions Knoten und optional aus einem Attribut-Wert-Knoten.

4.4.3 Aufbau der einzelnen Knoten

Jede Knoten-Art folgt bestimmten Aufbauregeln und Strukturen, welche in diesem Abschnitt dargestellt werden. Jeder Knoten beinhaltet zudem eine unikate ID und seinen spezifischen Knoten-Typ.

Kategorie Knoten

Kategorieknoten sind baumartig strukturiert und bestehen aus Relationen und wenigen Attributen. Die Relationen zeigen auf andere Kategorien und die Attribute beinhalten Daten wie den Namen der Kategorie oder strukturelle Informationen, die der Shop benötigt. Eine besondere Kategorie stellt dabei die Wurzel des Kategoriebaumes dar, da diese auch wenige aber essenzielle Informationen zu dem Shop als solchen beinhaltet (zum Beispiel den Namen des Shops).

Produkt Knoten

Produkt Knoten bestehen aus Relationen auf Items und beinhalten verschiedene Attributen. Innerhalb dieser Attribute befinden sich auch die Verweise auf die Kategorien, in denen sich das Produkt befindet. Ebenso beinhalten die Attribute die Bewertungen zu diesem Produkt und den darunter liegenden Items. Die restlichen Attribute beinhalten verschiedene Informationen über das Produkt.

Item Knoten

Item Knoten bestehen aus Relationen auf "Andere Knoten" (O-Knoten) und diversen Attributen. Die Attribute beinhalten verschiedene Informationen über das jeweilige Item.

Bewertungs Knoten

Bewertungs Knoten bestehen aus Attributen und beinhalten keine Relationen. Die Attribute beinhalten die eigentliche Bewertung, die ID's des Produktes und Items für das diese Bewertung gedacht ist. Optional beinhalten die Bewertungen innerhalb der Attribute auch einige wenige personenbezogene Informationen, wie den Namen und den Wohnort des Bewertenden.

Attribut Definitions Knoten

Attribut Definitions Knoten bestehen aus Attributen und beinhalten keine Relationen. Attribut Definitions Knoten sind strukturell zwar sehr ähnlich aufgebaut aber sie beinhalten in ihren Attributen Daten, die sehr stark von realen Kontexten, vom jeweiligen Shop und spezifischen Kategorien abhängig sind. Sie verhalten sich je nach Kontext somit sehr unterschiedlich.

Attribut Wert Knoten

Attribut Wert Knoten bestehen aus Attributen und beinhalten keine Relationen. Attribut Wert Knoten stellen besondere Einträge oder Inhalte dar, die im Zusammenhang mit den jeweiligen Attribut Definitions Knoten stehen. Die Attribut Wert Knoten beinhalten nicht viele XML-Attribute, basieren aber in ihrer Abhängigkeit auf den Attribut Definitionen, und haben eine ebenso große Abhängigkeit zum realem Kontext, innerhalb des Webshops.

Andere Knoten

Andere Knoten bestehen aus Attributen und beinhalten keine Relationen. Andere Knoten stellen am häufigsten Service Funktionalitäten des Shops im Zusammenhang mit Produkten und Items dar, sind allerdings nicht darauf beschränkt. Andere Knoten haben die größten Unterschiede untereinander im Vergleich zu den anderen Knotenarten.

4.4.4 Aufbau bestimmter XML-Attribute

Relation

Eine Relation ist ein direkter Verweis innerhalb eines Knotens und beinhaltet die ID, den Knotentyp und optional auch einen Relationstyp vom dem jeweiligen Kindknoten, auf den verwiesen wird.

Shop Attribut

Ein Shop Attribut ist ein Verweis auf eine Attribut Definition und optional auch auf einen Attribut Wert. Ein Shop Attribut stellt eine Eigenschaft oder eine shopspezifische Funktionalität dar.

5 Konzeption

In diesem Kapitel wird das Konzept zur Erstellung eines automatisierten Testdaten-Generators für XML-basierte Produktdaten im E-Commerce beschrieben.

Das Konzept beinhaltet die Architektursicht auf das Gesamtsystem, der beiden Teilsysteme des Datengenerators und des Minifizierers und deren Systemverhalten. Eine Komponentensicht mit Komponentendiagramm und Erläuterung zu den einzelnen Komponenten und deren Schnittstellen. Danach die Erläuterung der Eingabesystematik mit einer Übersicht der Eingabemöglichkeiten für den Minifizierer und einer Erläuterung der Eingabesystematik für den Datengenerator. Zuletzt wird dargestellt wie der Datengenerator die gewünschten Datenmengen erzeugt.

5.1 Architekturentwurf

In diesem Abschnitt werden die Architektur- und Entwurfsentscheidungen zur Erstellung eines Produktdaten Generators erläutert und veranschaulicht.

5.1.1 Übersicht

Die Anwendung wird in zwei Subanwendungen unterteilt, dem Datengenerator und dem Minifizierer. Diese Unterteilung wird gemacht, da sich die Anwendungsfälle in zwei Gruppen aufteilen lassen, das Erzeugen von neuen Daten und das Reduzieren eines Datensatzes, damit sich Tests auf einem lokalen Rechner durchführen lassen, da die vollen Datensätze der Produktdaten sehr groß sind.

Diese beiden Subanwendungen des Generierens und des Minifizierens überschneiden sich nur wenn ein Anwender lokal mit generierten Daten testen möchte. Im Falle einer solchen Anwendungsfallüberschneidung kann der Anwender die beiden Subanwendungen hintereinander ausführen.

Diese Trennung erlaubt es die entsprechende Anwendungslogiken besser zu trennen und zu gruppieren, um die Gesamtanwendung übersichtlicher und wartungsfreundlicher zu gestalten.

5.1.2 Minifizierer

Der Minifizierer wird die Aufgabe haben große Produktdatensätze zu kleinen Datensätzen zu reduzieren, dieser Vorgang wird innerhalb dieser Arbeit auch als "Minifizierung" bezeichnet. Der Anwender kann dabei entweder die Standard-Einstellungen beibehalten oder individuelle Angaben, entsprechend der in das System integrierten Anwendungsfälle, machen.

5.1.3 Datengenerator

Der Datengeneratror wird die Aufgabe haben zu Produktdatensätzen neue, generierte Daten hinzuzufügen. Der Anwender wird dabei dem Datengenerator die benötigten Anweisungen und Konfigurationen mitliefern müssen. Die Anweisungen werden dabei bei jedem Vorgang eingegeben, wobei die Konfigurationen entsprechend hinterlegt und gepflegt werden müssen, damit ein Shop vom Datengenerator unterstützt werden kann.

5.1.4 Architekturmuster

Das System wird in einzelne Komponenten aufgeteilt, welche nach Aufgabenbereichen gruppiert und sortiert werden. Die Komponenten und die darin enthaltenen Verarbeitungsschritte sind logisch getrennt und entsprechend ihrer Zuständigkeiten sortiert und strukturiert. Zudem werden die Verarbeitungsschritte so organisiert, dass es an den entscheidenden Stellen, innerhalb und zwischen den Komponenten, entsprechende logisch bedingte Abzweigungen gibt.

Prinzipieller Verarbeitungsablauf

Die Anwendung folgt einem prinzipiellem Arbeitsablauf, der wie folgt aussieht:

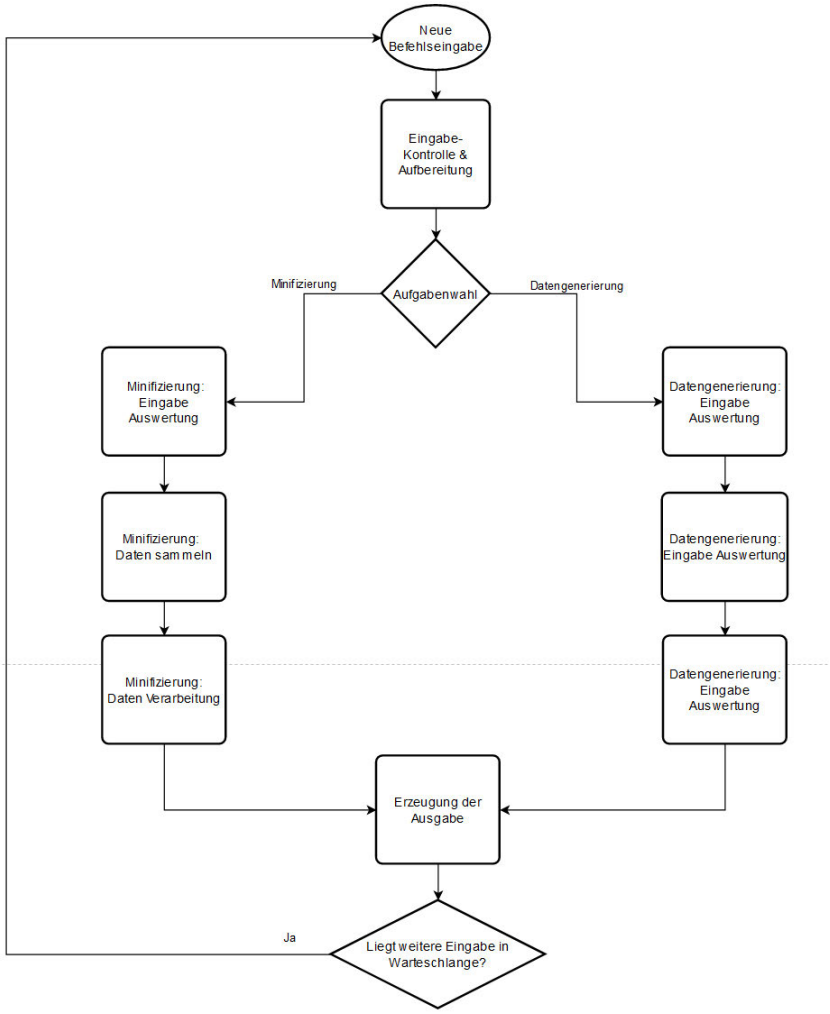

Abbildung 5.1: Prinziplieller Verarbeitungsablauf

Die Abbildung 5.1 stellt den prinzipiellen Verarbeitungsablauf dar. Als Erstes erfolgt die Befehlseingabe (seitens des Anwenders), danach folgt die Aufbereitung und Kontrolle der Eingabe, gefolgt von der Entscheidung welche Subanwendung diese Aufgabe übernehmen soll. Die entsprechende Anwendung erhält die aufbereitete Eingabe, welche sie dann zunächst auswertet, danach holt sie sich die von ihr benötigten Daten aus den Produktdaten. Nachdem die Subanwendung alle Daten gesammelt hat, beginnt sie mit ihrer eigentlichen Verarbeitung, nachdem die Anwendung mit der Verarbeitung fertig ist wird die Ausgabedatei erzeugt und das System wartet auf eine neue Eingabe oder verarbeitet die nächste, auf die Verarbeitung wartende, Eingabe.

Verarbeitungsreihenfolge der Produktdaten

Die Verarbeitungsreihenfolge wird durch die Abhängigkeiten und Verweise innerhalb der Produktdaten bestimmt. In Kapitel 4.4.3 "Aufbau der einzelnen Knoten" werden die Relationen und Verweise dargestellt. Basierend auf diesen Verweisen wird in diesem Abschnitt die gewählte Abfolge geschildert und erklärt. Eine Übersicht zu der Struktur der Verweise findet sich zudem im Abschnitt 4.4.2 "Hierarchie der Knoten".

1.) Kategorien:
Zuerst werden die Kategorien verarbeitet. Die Kategorien beinhalten nicht viele aber essenzielle Daten und werden für alle weiteren Verarbeitungsschritte benötigt. Die Kategorien besitzen eine Baumstruktur, bei der eine Kategorie beliebig viele Kindknoten besitzt.

Bei der Minifizierung ist diese Baumstruktur essenziell, da sie eine schnelle und strukturierte Verarbeitung ermöglicht. Es wird bei der Minifizierung von der Wurzel ausgehend der Baum bearbeitet, um so alle Kategorien abzudecken. Im Standardfall bleiben alle Kategorien aus der Eingabedatei in der Ausgabe erhalten.

Der Datengenerator benötigt einen schnellen Zugriff über die IDs auf die Kategorien. Um dies zu realisieren wird ein Set gewählt, da zudem alle IDs und somit auch die Kategorien einzigartig sind (Dies trifft auf alle Knotenarten zu). Aus Nutzersicht befinden sich die Kategorien auch hier über den Produkten, auch wenn die Produkte auf die Kategorien zeigen und nicht umgekehrt. Darüber hinaus gibt es Einstellungen innerhalb der Kategorien oder damit verbundene Shopfunktionalitäten, die gewisse Eigenschaften in den Produkten und Items bedingen oder erfordern.

2.) Produkte:
Als zweites werden die Produkte verarbeitet. Die Produkte beinhalten, im Vergleich zu den anderen Knotenarten, die meisten Verweise und Relationen, auf andere Knotenarten. Darüber hinaus gibt es kaum Verweise, innerhalb der anderen Knotenarten, welche auf die Produkt Knoten zeigen.

Bei der Minifizierung werden beim Durchlaufen des Kategoriebaumes die entsprechenden Mengen an Produkten ausgewählt, die in der Ausgabe vorkommen sollen. Dabei werden alle entsprechenden Verweise ausgelesen. Die Verweise auf die Items und Bewertungen werden dabei gesammelt und entsprechend der Eingabe in ihrer Menge begrenzt.

Bei der Datengenerierung werden die Produkte als zweites verarbeitet. Dies ist nötig da Kategorien bestimmte Attribute innerhalb der Produkte bestimmen können. Die Produkte beinhalten zudem die meisten Verweise. Diese Verweise müssen dann entsprechend sortiert, eingefügt oder gegebenenfalls manipuliert werden. Je nach Eingabe, was dann die Verarbeitung der Knoten bestimmen kann auf die verwiesen wird, mit Ausnahme der Kategorien.

3.) Items:
Als drittes werden die Items verarbeitet. Die Items sind hinsichtlich ihrer Struktur und ihres Aufbaus sehr ähnlich zu den Produkten und es gibt nur wenig strukturelle Unterschiede. Allerdings gibt es Attribute und Eigenschaften innerhalb der Produkte, die die Attribute und Eigenschaften bei den zugehörigen Items bedingen.

Bei der Minifizierung verläuft die Auswahl der Items simpel. Während der Auswahl der Produkte wurden schon alle IDs der auszuwählenden Items gesammelt. Es müssen nur noch die Items rausgesucht werden deren IDs für die Ausgabedatei gesammelt wurden.

Bei der Datengenerierung benötigen die Items Informationen aus den Kategorien genauso wie bei den Produkten, allerdings benötigen die Items gegebenenfalls auch attributbezogene Informationen aus den Produkten. Die Items selbst stellen zudem Kindknoten der Produkte dar, weil direkt auf diese referenziert wird.

4.) Bewertungen:
Als viertes werden die Bewertungen verarbeitet. Die Bewertungen sind sehr simpel aufgebaut, beinhalten allerdings stark realitätsgebundene Daten.

Bei der Minifizierung verläuft die Auswahl der Bewertungen genauso simpel wie bei den Items. Während der Auswahl der Produkte wurden schon alle IDs der auszuwählenden Bewertungen gesammelt. Es müssen nur noch die Bewertungen rausgesucht werden, deren IDs für die Ausgabedatei gesammelt wurden.

Bei der Datengenerierung benötigen die Bewertungen Informationen aus den Produkten und Items. Die Bewertungen selbst stellen zudem Kindknoten der Produkte dar, weil direkt auf diese referenziert wird. Da Bewertungen auch oft personenbezogene Details (Name und Wohnort des Bewertenden) beinhalten könnten müssen diese bei der Datengenerierung entsprechend anonymisiert werden.

5.) Attribute (AD- und AV-Knoten):
Attribute sind Daten, die an reale Kontexte gebunden sind. Zudem hängt der logische Aufbau und Inhalt eines Attributes von vielen Shop- und Kategorie-Abhängigen Faktoren ab. Verweise auf die Attribute lassen sich in fast allen anderen Knoten vorfinden.

Bei der Minifizierung wird die Sammlung der Attribute sehr einfach gehalten, da die Attribute hinsichtlich des Speicherverbrauchs sehr klein sind, aber in ihrer Vielfalt und Verzweigungsmöglichkeiten sehr groß, werden diese einfach direkt und ohne Aussiebung in die Ausgabedatei übernommen.

Bei der Datengeneration verhält es sich genauso, nur das hier bei Bedarf noch die neuen Attribute hinzugefügt werden. Aufgrund der Komplexität und der potenziell großen Unterschiede von Shop zu Shop werden die Attribute, entsprechend der Anwenderangaben, direkt übernommen. Attribute beinhalten viele Details, die einen reellen Bezug haben, und es gibt wenig Systematik innerhalb der Attribute um sinnvolle Daten automatisiert zu generieren.

6.) Andere (O-Knoten):
Unter dem Begriff "Andere Knoten" steckt eine sehr grobe und weitreichende Sammlung diverser Knoten. Diese Knoten beinhalten viele realitätsgebundene Daten. Im Vergleich zu den Attributen sind sie schwerer zu systematisieren. Ihr Bezug zu realen Kontexten und Daten ist dabei vielfältig und weitreichend.

Bei der Minifizierung und der Datengenerierung verhalten sich die Anderen (O-Knoten) genauso wie die Attribute und werden daher ebenso behandelt hinsichtlich der Auswahl und Verarbeitung.

5.2 Komponentenentwurf

In dieser Sektiuon werden die Komponenten im Einzelnen aufgeführt, beschrieben und mittels eines Komponentendiagrammes dargestellt.

Komponentendiagramm

Die folgende Abbildung 5.2 zeigt die Komponenten der Anwendung und wie diese miteinander in Verbindung stehen. Die Struktur und Hilfskomponenten werden von allen Kernkomponenten genutzt und sind der Übersicht halber optisch getrennt. Die einzige Komponente, die nicht aufgeführt wurde, ist die Graphische Oberfläche (GUI), da die für diese Anwendung geschaffene GUI nur eine lose Kopplung zum Rest des System besitzt und von rudimentärer Natur ist.

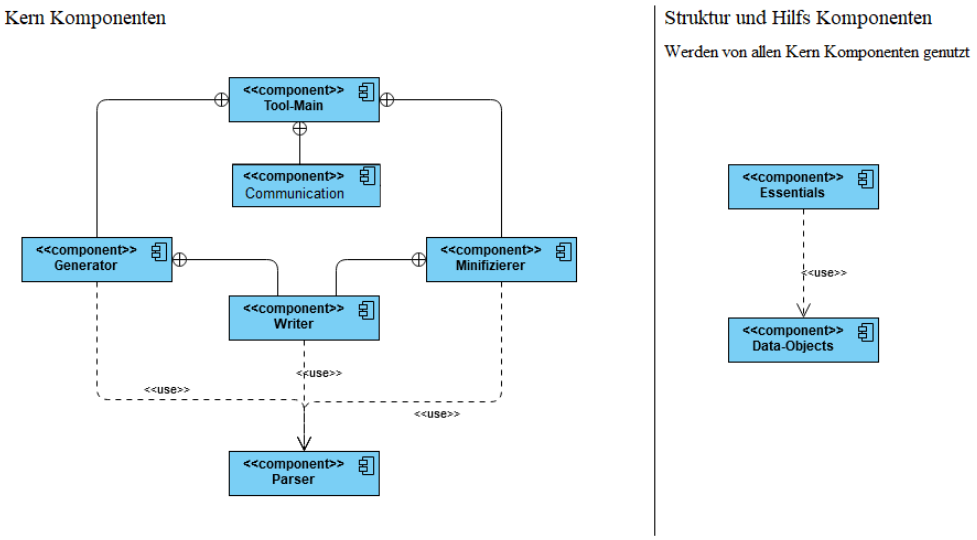

Abbildung 5.2: Komponentendiagramm

Die Struktur der Komponenten und das Verhältnis der Komponenten zueinander ist auf technischer und auf fachlicher Ebene gleich. Auf die jeweiligen technischen Einzelheiten und die Implementation wird in dem Kapitel Entwicklung eingegangen.

5.2.1 Komponenten

Dieser Abschnitt stellt die einzelnen Komponenten, aus Abbildung 5.2, deren Funktion und deren Aufgabenmodule mit deren Funktionen dar.

Tool-Main

Die Tool-Main Komponente beinhaltet alle Module und Klassen die nötig sind um eine übergeordnete Steuerung zu ermöglichen.

Dazu gehören die Aufgabenmodule der Systemsteuerung, der Eingabe-Kontrolle und der Operationsauswahl.

Die Systemsteuerung ist das Zentrum der Tool-Main und koordiniert die anderen Module und hat Zugriff auf die Eingabedaten, die Warteschlange und alle nötigen Datenstrukturen.

Die Eingabe-Kontrolle prüft die Eingabedaten und validiert alle darin angegebenen Verweise, darüber hinaus erzeugt sie auch die Standardwerte für leere Felder in der Eingabedatei. Die Eingabe-Kontrolle prüft auch das Dateiformat der Eingabedaten, es handelt sich hierbei um das JSON-Dateiformat. Die von der Eingabe-Kontrolle geprüfte und aufbereitete Eingabedatei wird im Anschluss an die Systemsteuerung zurückgegeben.

Die Operationsauswahl prüft die aufbereiteten Eingabedaten hinsichtlich der gewählten Aufgabe und gibt diese dann an die entsprechende Komponente weiter. Die wählbaren Aufgaben entsprechen dabei den Modulen. Wenn man eine Datei mit neuen Knoten generieren will übernimmt die Komponente des Generators. Wenn man eine reduzierte, also eine minifizierte Datei erhalten möchte, übernimmt die Komponente des Minifizierers.

Communication

Die Communication Komponente beinhaltet die Module und Klassen die nötig sind, um Eingabebefehle zu empfangen und an die Anwendung weiter zu leiten. Diese Komponente beinhaltet unter anderem die Dateiüberwachung und stellt die Schnittstelle für die Eingabe dar. Die Dateiüberwachung beobachtet einen ausgewählten Dateipfad, und sobald eine neue Eingabedatei von außen abgelegt wurde, wird diese Eingabedatei erkannt. Danach wird der Pfad zu dieser Datei in der Warteschlange abgelegt.

Minimizer

Die Minimizer Komponente beinhaltet alle Funktionalitäten um gemäß Anwender Spezifikationen existente Produktdaten in ihrer Menge strukturiert zu reduzieren. Die Minimizer Komponente beinhaltet Arbeitsmodule, welche eine Trennung der Arbeitsschritte darstellen.

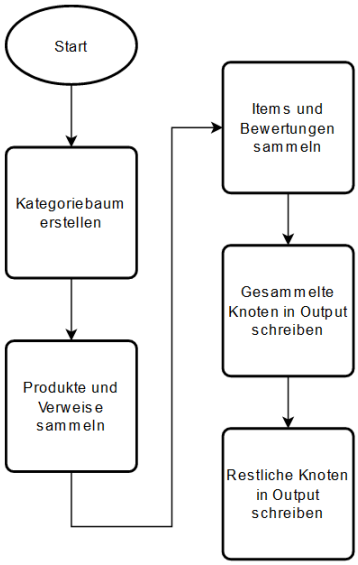

Abbildung 5.3: Module des Minifizierers

In der Abbildung 5.3 kann man die Module des Minifizierers und die damit verbundenen Arbeitsschritte sehen. Als erstes werden die Kategorien eingelesen, verarbeitet und in eine Baumstruktur arrangiert. Als zweites werden die Produkte eingelesen und verarbeitet, dabei werden die entsprechenden Verweise auf die Kategorien beachtet. Zudem werden die Verweise auf die Items und Bewertungen ausgewählt und gesammelt. Während der Verarbeitung der Produkte werden auch die Verteilungswerte beachtet, die der Anwender vorgegeben hat. Bei den Verteilungswerten handelt es sich um die Anzahl der Produkte pro Kategorie, die Anzahl der Items und die Anzahl der Bewertungen pro Produkt. Danach werden als drittes die Items und Bewertungen rausgesucht, für die die Verweise gesammelt wurden. Als viertes werden dann die Kategorien, die Produkte, die Items und Bewertungen, die gesammelt wurden, in die Ausgabedatei (Output) geschrieben. Zuletzt werden dann die restlichen Knoten aus der Quelldatei, ohne Reduzierung der Datenmengen, in den Output übertragen, dabei handelt es sich um die folgenden Knotenarten: Attribut Definitionen, Attribut Werte und Andere Knoten (O-Knoten)

Generator

Die Generator Komponente beinhaltet alle Funktionalitäten, um gemäß Anwendereingaben Produktdaten zu generieren und diese zu existenten Daten hinzuzufügen. Die Generator Komponente beinhaltet Arbeitsmodule, welche eine Trennung der Arbeitsschritte darstellen.

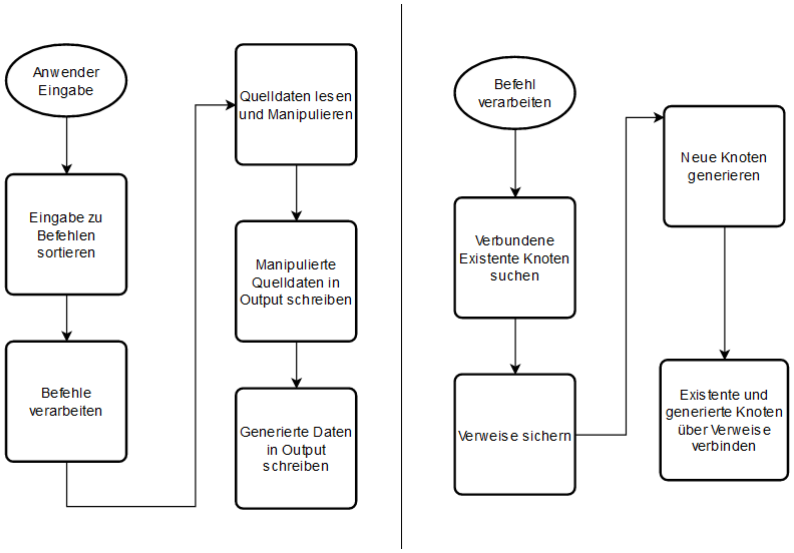

Abbildung 5.4: Module des Datengenerators

In der Abbildung 5.4 kann man die Module des Datengenerators und die damit verbundenen Arbeitsschritte sehen. Zuallererst werden die Eingaben des Benutzers eingelesen, verarbeitet, sortiert und zu internen Befehlen beziehungsweise Variablen geformt.

Danach werden als zweites diese Befehle verarbeitet. Dabei werden zunächst alle Befehle für die Kategorien verarbeitet, danach für alle Produkte, Items, Bewertungen, Attribut Definitionen, Attribut Werte und zuletzt für alle Anderen (O-Knoten). Die Arbeitsschritte innerhalb der Befehlsverarbeitung sehen dann folgendermaßen aus.

Der erste Verarbeitungsschritt ist das Einlesen von existenten Knoten, auf die der Anwender in seinen Anweisungen verwiesen hat, zum Beispiel eine existente Kategorie, in die ein neues Produkt hineingeneriert werden soll. Diese existenten Knoten werden eingelesen und gespeichert. Danach werden als zweiter Schritt alle Verweise von den gespeicherten Knoten gesammelt und, um einen schnelleren Zugriff zu gewährleisten, separat gesichert.

Der dritte Verarbeitungsschritt ist dann das eigentliche Generieren. Der letzte Verarbeitungsschritt, innerhalb der Befehlsverarbeitung ist dann das Verbinden der existenten Knoten, die im Verarbeitungsschritt eins eingelesen wurden, mit den neugenerierten Knoten. Diese Verbindung erfolgt über die Verweise innerhalb der Knoten, welcher Knoten dabei in welcher Form manipuliert werden muss ist der Hierachie der Knoten zu entnehmen.

Nach der Verarbeitung der Befehle erfolgt der eigentliche dritte Arbeitsschritt, das Lesen und Manipulieren der Quelldaten. Dabei müssen nur die Daten manipuliert werden, von denen im vorherigen Schritt die existenten Knoten eingelesen wurden, anhand der ID werden diese Knoten dann erkannt und mit den entsprechenden Informationen erweitert. Als viertes werden dann alle originalen und manipulierten Quellknoten in die Ausgabedatei geschrieben. Zuletzt werden dann im fünften Arbeitsschritt alle generierten Knoten in die Ausgabedatei geschrieben.

Writer

Die Writer Komponente beinhaltet die Module, die zum Schreiben in XML-Dateien benötigt werden.

Parser

Die Parser Komponente beinhaltet die Module, die zum Einlesen und Parsen von XML-Dateien, benötigt werden.

Essentials

Die Essentials Komponente beinhaltet alle Pfade, Verweise und Strukturdaten, die die Anwendung benötigt. Die Essentials Komponente beinhaltet zudem auch alle internen Daten- und Verwaltungsobjekte.

XML-Data-Objects

Die XML-Datenobjekte (XML-Data-Objects) Komponente beinhaltet alle Datenelemente, die zum Lesen, Schreiben und Verarbeiten der XML-Daten benötigt werden.

5.3 Eingabesystematik

In dieser Arbeit wird für die Anwendung eine rudimentäre graphische Oberfläche (GUI) bereitgestellt, diese besitzt eine lose Kopplung zu der eigentlichen Anwendung, da die GUI nur die Eingaben annimmt und diese in eine Datei im JSON-Dateiformat abspeichert. Die Eingabedatei wird dann in dem Ordner abgelegt, welcher von der Dateiüberwachung beobachtet wird.

5.3.1 Eingabemöglichkeiten Minifizierer

Im Folgenden werden die Eingabemöglichkeiten für den Minifizierer zusammengefasst aufgelistet und kurz erläutert. Eingabemöglichkeiten sind die optionalen und essenziellen Befehle, die man dem Minifizierer übermitteln kann.

- Auswahl des Shops:
  Bei der Eingabe des Shops handelt es sich um eine essenzielle Eingabe, bei der man aus einer Auswahl von unterstützten Shops auswählen kann.
- Pfad für die eingehenden Daten:
  Die Pfad-Auswahl für die eingehenden Produktdaten ist eine optionale Eingabe. Sollte der Anwender nichts angeben, wird der Standardpfad für die Eingabedaten ausgewählt.
- Pfad für die ausgehenden Daten:
  Die Pfad-Auswahl für die ausgehenden Produktdaten ist eine optionale Eingabe. Sollte der Anwender nichts angeben, wird der Standardpfad für die Ausgabedaten ausgewählt.
- Name der Ausgabe-Datei:
  Die Angabe für den Namen für die Ausgabe-Datei, welche die ausgehenden Produktdaten beinhaltet, ist eine optionale Eingabe. Sollte der Anwender nichts angeben, wird ein Standard-Name für die Ausgabe-Datei generiert.
- Menge der Produkte pro Kategorie:
  Mit diesem Feld wird bestimmt wie viele Produkte pro Kategorie ausgewählt und in die Ausgabedatei mit übernommen werden. Hier wird dem Anwender ein Standardwert vorgeschlagen, den dieser bei Bedarf abändern kann. Dieser Wert wird entsprechend nach oben und unten begrenzt.

- Menge der Items pro Produkt:
  Mit diesem Feld wird bestimmt wie viele Items pro Produkt ausgewählt und in die Ausgabedatei mit übernommen werden. Hier wird dem Anwender ein Standardwert vorgeschlagen, den dieser bei Bedarf abändern kann. Dieser Wert wird entsprechend nach oben und unten begrenzt.

- Menge der Bewertungen pro Produkt:
  Mit diesem Feld wird bestimmt wie viele Bewertungen pro Produkt ausgewählt und in die Ausgabedatei mit übernommen werden. Hier wird dem Anwender ein Standard-Wert vorgeschlagen, den dieser bei Bedarf abändern kann. Dieser Wert wird entsprechend nach oben und unten begrenzt. Im Standard-Fall ist dieses Feld leer.

- Folgende Kategorien als Ursprung betrachten:
  Die Kategorien in den Shops sind wie ein Baum aufgebaut. Wenn der Anwender in diesem Feld eine oder mehrere Kategorien angibt, werden nur diese Kategorien und deren Kindknoten ausgewählt, alle anderen Kategorien werden rausgefiltert.

- Für folgende Kategorien die Limitierung aufheben:
  Wenn der Anwender hier eine oder mehrere Kategorien angibt, werden alle Produkte aus diesen Kategorien übernommen, ungeachtet der Zahl die bei ”Produkte pro Kategorie” steht. Im Standardfall ist dieses Feld leer.

- Bestimmte Produkte:
  In diesem Feld kann der Anwender eine beliebige Anzahl von Produkten angeben, die dieser explizit in den ausgehenden Produktdaten enthalten haben möchte. Bei der Auswahl dieser Produkte werden dann die Limitierungen aus ’Produkte pro Kategorie” ignoriert.

- Bestimmte Items:
  In diesem Feld kann der Anwender eine beliebige Anzahl von Items angeben, die dieser explizit in den ausgehenden Produktdaten enthalten haben möchte. Diese Items benötigen allerdings entsprechend vorhandene Produkte innerhalb der ausgehenden Produktdaten. Bei der Auswahl dieser Items werden dann die Limitierungen aus ”Items pro Produkt” ignoriert.

5.3.2 Eingabesystematik Datengenerator

Im Folgenden wird die Eingabesystematik der Datengenerator Subanwendung dargestellt. Der zukünftige Anwender hat bei der Generierung von Daten eine andere Sichtweise und andere Ansprüche als bei der Minifizierugn der Daten. Diese Ansprüche wurden mittels Interviews und Gesprächen ermittelt und in Form der gesammelten Anforderungen gesichert.

Was bei beiden Subanwendungen gleich bleibt, ist die Shop-Auswahl die bei dem Datengenerator also wie folgt lautet. Bei der Eingabe des Shops handelt es sich um eine essenzielle Eingabe, bei der man aus einer Auswahl von unterstützten Shops auswählen kann.

Der Datengenerator benötigt eine Angabe, der vom Anwender erwarteten Struktur, der Daten. Zudem muss der Generator über besondere Eigenarten des Shops informiert werden. Diese Eigenarten des Webshops werden in der Form von Konfigurationen gespeichert und der Datengenerator kann, während des Generierungsprozesses, auf diese zugreifen. Die Konfigurationen beschreiben Regeln, die den ganzen Webshop oder spezielle Anteile des Shops betreffen betreffen. Es gibt in einigen Shops Kategorien, die besondere Regeln für ihre Produkte haben, diese Regeln müssen auch in der Form von Konfigurationen angelegt werden. Diese Konfigurationsdateien werden einmalig angelegt, wenn ein Shop, in die Liste der unterstützten Shops, eingebunden wird. Die Konfigurationen müssen nach der Einbindung eines Shops regelmäßig geprüft und aktualisiert werden, sofern der Shop strukturelle Veränderungen durchlebt hat.

Darüber hinaus benötigt der Datengenerator die eigentlichen Anweisungen des Anwenders, diese Anweisungen können vielfältig sein. In den Anwendungsfällen lassen sich die funktional getrennten Exemplare solcher Anweisungen wiederfinden, es ist aber auch jegliche Kombination denkbar. Darüberhinaus muss der Datengenerator auch eine stabile Erweiterbarkeit aufweisen, um neue Anwendungsfälle mit abdecken zu können. Es wird dabei davon ausgegangen, dass der Anwender vertraut genug mit den Produktdaten ist, um sinnvolle Eingaben und Ansprüche an die Anwendung zu stellen.

Aufgrund der im vorherigen Absatz geschilderten Gründe wurde XML als Eingabesprache gewählt. Die XML basierte Eingabesystematik basiert auf der Form und den Regeln der Produktdaten und wurde um spezielle Anweisungen erweitert. Beispiele für die Form und den Aufbau der Eingabesystematik finden sich im Anhang, in dem Abschnitt für Testfälle.

Im Anhang werden die entsprechenden Befehle, mit einem kurzem Text, hinsichtlich des erwarteten Ziels, erläutert.

5.4 Generierung der Zufallsdaten bei der Datengenerierung

Die Generierung der Zufallsdaten basiert auf dem Prinzip des Mischens von Anteilen aus realen Daten.

Der Datengenerator kann Kategorien, Produkte, Items und Bewertungen durch Zufall aus existenten Daten generieren. Der Zufall folgt dabei vorbestimmten Strukturen und Regeln. Attribut Definitionen, Attribut Werte und Andere Knoten können nicht generiert werden, da ihr Bezug zur Realität und zu den Shops so explizit ausgeprägt ist, dass es nicht möglich ist diese Daten per geregeltem Zufall zu generieren. Bei den nicht generierbaren Knoten muss der Anwender diese explizit angeben damit diese Knoten direkt übernommen werden können.

Bei den generierbaren Knoten gibt es zwei Mengen, die unspezifischen Knoten und die spezifischen Knoten. Bei den unspezifischen Knoten gibt der Anweder lediglich die Informationen an, die der Generator benötigt, um zu wissen was er in welcher Menge und wie die Verweise hinsichtlich dieser Knoten aussehen. Bei den spezifischen Knoten werden die selben Informationen wie bei den unspezifischen Knoten benötigt, allerdings beinhalten sie noch attributbezogene und relationsbezogene Spezifikationen oder Eigenheiten die der Anwender explizit einfordert.

Sobald die Subanwendung des Datengenerators die Eingabedaten für die Generierung erhalten hat sortiert er diese nach Knotenart und arbeitet sie entsprechend der Hierachie ab, wobei die Bewertungen nach den Items bearbeitet werden. Wie diese Generierung im Detail abläuft folgt im Kapitel Entwicklung.

6 Optimierung

Dieses Kapitel beschreibt die Optimierung der Konzeption. Hier werden die Anpassungen und Verbesserungen, den die Konzepte durchlaufen haben, geschildert. Die Gründe und die argumentative Basis für die Anpassungen werden ebenfalls in diesem Kapitel dargestellt. Darüber hinaus werden die Verbesserungen erklärt und den Problemen, die sie lösen gegenübergestellt.

Die Optimierungen sind in mehrere Abschnitte gruppiert und unterteilt. Als Erstes kommt die LookUp Systematik. Als Zweites folgt die Speicherausnutzung, danach als Drittes das Laufzeitverhalten und abschließend folgt, mit einer zusammenfassenden Gegenüberstellung, die Performance.

6.1 LookUp Systematik

Aufgrund der Größe der Produktdaten und somit auch der Größe des jeweiligen XML-Dokumentes, ist es nicht möglich und nicht sinnvoll das XML-Dokument direkt in den Arbeitsspeicher zu laden und von dort aus zu verarbeiten. Ebenso wenig ist es möglich, die Objekte aus dem Produktdaten XML-Dokument als Java-Objekte im Arbeitsspeicher zu halten, da dies auf einem lokalen Rechner den Arbeitsspeicher überlasten würde und auch bei Servern wäre das eine Verschwendung von Ressourcen. Das Parsen eines einzelnen großen XML-Dokumentes mit den Produktdaten dauert auch sehr lange und verursacht große Verarbeitungs- und somit Wartezeiten, da ein Dokument mehr als einmal durchlaufen werden muss. Ohne eine entsprechende Systematik und Verwaltung der Daten kann somit die Anforderung W-5 (aus dem Abschnitt 4.3.2) nicht eingehalten werden.

Um die Anforderungen einhalten zu können und auch um die Verarbeitungszeit zu verkürzen, werden LookUp XML-Dokumente erstellt. Die einzelnen LookUps beinhalten nach Knotentyp gefilterte und spezifisch gruppierte XML-Daten. Es gibt für jede Knotenart

zwei LookUps, einmal die gesammelten Knoten selbst und die gesammelten ID's. Zudem werden LookUps für spezielle Aufgaben erstellt, die die Auswahl von Daten beim Parsen erleichtern können.

Die Erstellung der LookUps ist zeitaufwändig. Allerdings müssen diese LookUps nicht bei jedem Verarbeitungsvorgang ausgeführt werden, sondern nur wenn der Anwender es wünscht. Eine Erstellung der LookUps ist sinnvoll, wenn sich die Produktdaten inhaltlich verändert haben. Nach Rücksprachen mit den Entwicklern hat sich herausgestellt, dass es genügt, wenn die LookUps einmal pro Woche neu erstellt werden.

Die LookUps sind in Summe, durch die Redundanz der beinhalteten Daten, größer als die eigentlichen Produktdaten, was beim Ablegen der LookUps und dem bereitgestellten Speicher zu beachten ist. Das Parsen von vielen kleineren Daten, die in Summe größer als die eigentlichen Produktdaten sind geht allerdings erheblich schneller.

So würde der Verarbeitungsvorgang, am Beispiel des Minifizierens mit einer 12GB großen Produktdatendatei, welche zu einer 100 bis 300MB großen Ausgabedatei reduziert wird. Ohne LookUps mehrere Stunden benötigen. Mit der LookUps Systematik dauert ein Verarbeitungsvorgang beim Minifizieren hingegen nur circa 40-70 Minuten mit Erstellung der LookUps oder 10-20 Minuten ohne Erstellung der LookUps. Die Schwankung bei den Zeiten mit der LookUp Systematik hängen von den Eingaben des Anwenders und der somit resultierenden Größe der Ausgabe zusammen. Zur Veranschaulichung des genannten Beispieles siehe auch Abbildung 6.1.

Am häufigsten würde der Anwender also Daten mit bereits vorhandenen LookUps generieren oder minifizieren, was in Summe zu einer großen Zeitersparnis führt und direkt mögliche Problematiken hinsichtlich des Arbeitsspeichers umgeht.

6.2 Speicherausnutzung

Durch die Einführung der LookUp Systematik folgt auch eine notwendige Verwaltung des Speichers. Während eines Verarbeitungsvorganges benötigt die Anwendung dank der LookUps, nicht viel Arbeitsspeicher, was bezüglich der Speicherausnutzung nicht mehr beachtet werden muss. Die Ausnutzung des Speichers auf der Festplatte ist wegen der LookUps allerdings signifikant angestiegen. Die Produktdaten selbst befinden sich auf den Servern, wo auch die Anwendung später laufen wird. Ebenso liegen die LookUps später in einem entsprechenden Verzeichnis auf dem entsprechenden Server. Der Anwender muss nur die entsprechenden Ausgabedateien vom Server herunterladen, nachdem ein Verarbeitungsdurchgang fertig gestellt wurde.

Die LookUps benötigen allerdings das Anderthalbfache bis Doppelte an Speicher im Vergleich zu den originalen Produktdaten und werden diese nicht ersetzen. Das Bedeutet, dass der entsprechende Speicher für jeden unterstützten Shop zugesichert werden muss. Zudem muss die Anwendung sicherstellen, dass bei einem Vorgang der LookUp-Erstellung die alten LookUps gelöscht und durch die Neuen ersetzt werden. In Rücksprache mit den Entwicklern und den Betreibern der später genutzten Server, hat sich ergeben das diese Speicherausnutzung kein Problem darstellt und es ihnen wichtiger ist sparsam mit dem Arbeitsspeicher umzugehen. Das Argument, dass die Minifizierung und Generierung im Alltag der Entwickler sehr viel Zeit ersparen wird, rechtfertigt seitens der Entwickler und deren Vorgesetzten die erhöhte Speicherausnutzung durch die LookUps.

Daürber hinaus befinden sich die LookUps zusammen mit der Anwendung auf einem Server, welcher genug Speicherplatz bietet. So kann der Anwender die Anwendung ohne Belastung seines lokalen Speichers starten und die fertige Datei jederzeit herunterladen.

6.3 Laufzeitverhalten

Das Laufzeitverhalten wird durch die Bearbeitungsreihenfolge der Knoten und die LookUp Systematik bestimmt. Es werden nur wenige bestimmte Daten im Arbeitsspeicher gehalten. Zu diesen Daten gehören primär ID's und Werte, die benötigt werden, um Daten aus den LookUps auszulesen.

Die Anwendung wird während einer Verarbeitungsoperation (Minifizieren oder Datengenerierung) die LookUps häufiger parsen und durchlaufen müssen. Parsing ist allerdings zeitaufwändig und sollte sparsam genutzt werden. Die Balance zwischen den Daten, die im Arbeitsspeicher gehalten werden und die die in LookUps abgelegt werden, wird somit experimentell bestimmt und kann sich in der Zukunft, basierend auf Erfahrungswerte auch ändern.

Das Parsen wird zunächst immer dann ausgeführt, wenn eine neue Knotenart bearbeitet wird, oder wenn ein neuer Arbeitsschritt ausgeführt wird. Dabei werden dann nur die Daten im Arbeitsspeicher gehalten, die während eines Arbeitsschrittes oder für einen folgenden Arbeitsschritt benötigt werden, dies betrifft nur Daten mit strukturellen Informationen für die Knoten, wie ID's oder Verweise. Die eigentlichen Inhalte der Knoten werden dabei in temporäre Dateien oder entsprechende LookUps abgelegt.

6.4 Performance

In diesem Abschnitt werden die getroffenen Optimierungen zusammengefasst und dem nicht optimierten Konzeptionen gegenübergestellt.

6.4.1 Neuer Arbeitsablauf einer Subanwendung

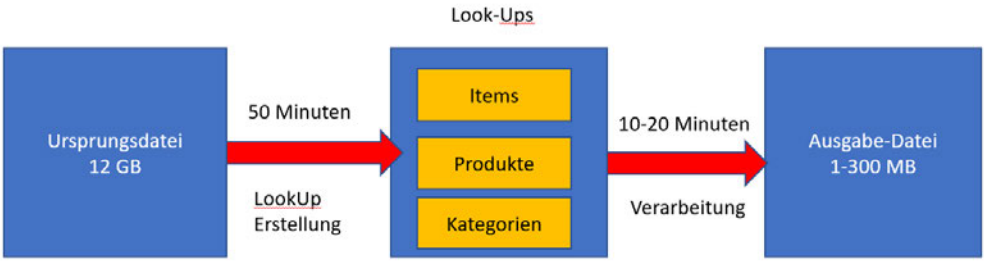

Abbildung 6.1: Arbeitsablauf

Die Abbildung 6.1 stellt schematisch einen vereinfachten und prinzipiellen Arbeitsablauf dar. Zudem beinhaltet die Abbildung die Zeiten, die in dem Beispiel aus dem Abschnitt 6.1, aufgetaucht sind. In dieser Abbildung wird dargestellt, wie der Arbeitsablauf aus dem Kapitel Konzeptionierung erweitert wurde. Die Erstellung der LookUps ist somit ein optionaler Teil des Arbeitsschrittes der Verarbeitung der Daten.

6.4.2 Gegenüberstellung

In diesem Abschnitt werden die Erweiterungen und Veränderungen dargestellt, die durch die Optimierung erfolgt sind. Die Optimierung erfolgte dabei primär durch die LookUp Systematik und die damit verbundene Verwaltung der Datenmengen innerhalb der XML-Dokumente.

Optimiertes Komponentendiagramm

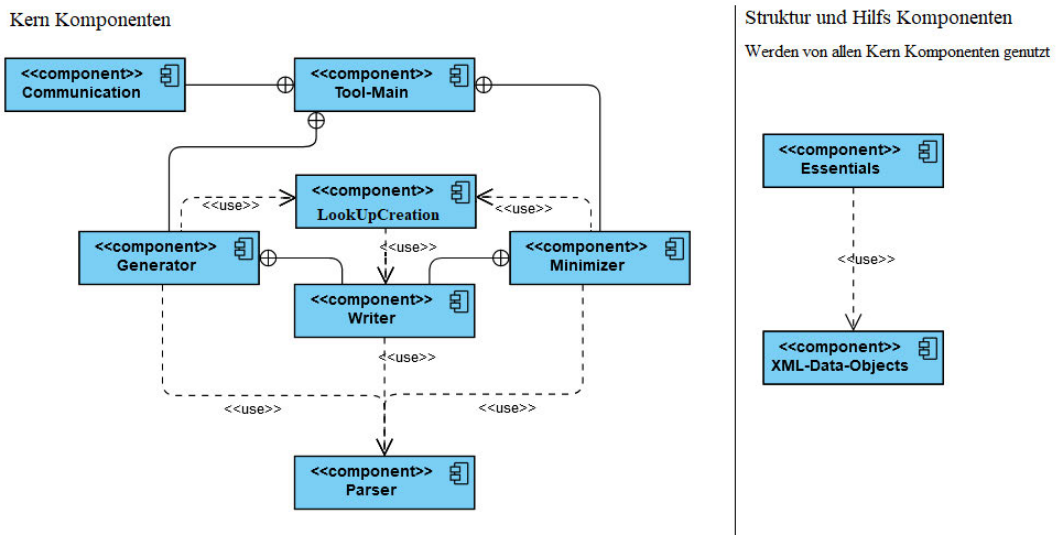

Abbildung 6.2: Komponentendiagramm mit LookUps

(eigene Abbildung)

Die Abbildung Abbildung 6.2 entspricht der Abbildung 5.2, mit dem Unterschied der hinzugefügten ”LookupCreation” Komponente

LookupCreation

Durch das hinzufügen der LookUp Systematik, um das Konzept zu optimieren, folgte auch das hinzufügen einer weiteren Komponente. Die LookupCreation Komponente beinhaltet alle Funktionalitäten, um die LookUps zu generieren und zu verwalten.

7 Entwicklung und Testen

Dieses Kapitel beschreibt die Umsetzung der Konzeption, mit der integrierten Optimierung, in Form der Implementierung und erläutert den funktionalen Ablauf innerhalb des Minifizierungs-Moduls und des der Generierungs-Moduls.

7.1 Implementierung

Dieser Abschnitt beschreibt die Implementierung des automatisierten Testdaten-Generators für XML-basierte Produktdaten im E-Commerce. Zuerst wird die Implementierung der Komponenten dargestellt. Danach wird dargestellt, wie die Anwendungsfälle auf Implementationsebene realisiert wurden.

7.1.1 Implementierung der Komponenten

In diesem Abschnitt wird die Implementierung der Komponenten realisiert. Die Komponenten basieren auf den in Kapitel 5.2 präsentierten Komponentenentwurf.

Reihenfolge der Komponentenerzeugung

Die Erzeugung der Komponenten folgt der Hierarchie der Knoten innerhalb der Produktdaten. Um das Verhalten des Systems stets unter Kontrolle zu haben und um Fehler während der Entwicklung sicher detektieren und lokalisieren zu können, wurden zuallererst die grundlegenden Verarbeitungskomponenten, schrittweise erzeugt und getestet. Diese Vorgehensweise war sinnvoll, um so die einzelnen Komponenten und enthaltenen Module schrittweise und einzeln zu testen, damit das System am Ende stabil läuft.

Zunächst lag der Fokus auf der Subanwendung des Minifizierers, die Subanwendung des Datengenerators wurde zuletzt erstellt. Danach wurde das Gesamtsystem und alle mit

dem Datengenerator verbundenen Komponenten entsprechend erweitert. Die Main Komponente wurde zu Beginn rudimentär gehalten.

Als Erstes wurde die Parser Komponente erzeugt und mit dem eingelesenem Datensatz abgeglichen. Das zu Beginn genutzte Datensatz von Produktdaten war bewusst klein gewählt, um die Daten schnell und einfach überblicken zu können. Hierbei wurden die Knoten entsprechend ihrer Hierarchie bearbeitet und kontrolliert.

Synchron zu der Parser Komponente wurden die Datenobjekte erstellt, die benötigt wurden, um den XML-Basierten Datensatz speichern zu können. Die Daten Objekt Komponente wird weiterhin synchron zu den weiteren Implementationen gepflegt.

Als Zweites wurde die Writer Komponente erzeugt, um sicherzustellen, dass die eingelesenen Daten identisch zu den ausgegebenen Daten sind. Es wurden danach mehrere Tests zum Abgleich der Daten ausgeführt.

Als Drittes wurde die Essentials Komponente erstellt, um wichtige Dateipfade und Begriffe für die Anwendung bereit zu stellen. Ebenso wurde das JSON-Objekt erzeugt um Befehle, und somit auch Tests, für die Anwendung schreiben zu können.

Als Viertes wurde die Minifizierer Komponente erstellt. Bei der Erzeugung der Minifizierer Komponente wurde die Verarbeitungsreihenfolge der Knoten durch deren Hierarche bestimmt. Synchron zur Erzeugung des Minifierers wurde auch eine rudimentäre GUI erzeugt, um das Verhalten des Minifizierers testen zu können.

Als Fünftes wurde die Tool Main Komponente erzeugt, um die Auswahl der Subanwendungen sicherzustellen.

Als Sechstes wurde der Datengenerator erzeugt. Die vom Datengenerator erzeugten Daten konnten direkt in den Shops lokal getestet werden, da der Minifizierer die richtige Größe der Daten sicherstellen konnte und die Daten vom Minifizierer auch lokal positiv getestet wurden.

Zuletzt wurde als siebtes die rudimentäre GUI für den Datengenerator erzeugt.

Die Struktur und Hilfskomponenten wurden, während des Implementationsprozesses, iterativ verbessert und in ihrer Struktur optimiert.

Realisierung der Komponenten

In diesem Abschnitt wird die Implementierung der Komponenten dargestellt und geschildert.

Parser:
Die Parser Komponente beinhaltet verschiedene Parser, die auf spezielle Verarbeitungen der Produktdaten spezialisiert sind. Einige lesen die Produktdaten und schreiben diese selektiert in andere Dateien andere lesen die Produktdaten und sichern Informationen in Form von Java Objekten, die dann von der Anwendung verarbeitet werden.

Auf technischer Ebene basieren die einzelnen Parser auf dem SAX2 XML-Parser. Die Parser erweitern (extends) dabei den DefaultHandler aus der Java SAX Bibliothek. Darüber hinaus sind die einzelnen Parser gemäß der Vererbungslehre aufgebaut, um so Doppplungen im Code zu vermeiden. Durch diese Strategie lassen sich schnell weitere spezielle Parser hinzufügen und bereits implementierte Parser lassen sich somit schnell und einfach warten.

Writer:
Die Writer Komponente beinhaltet Klassen die notwendig sind, um das Schreiben von Knoten in XML-Dokumente zu ermöglichen. Das betrifft LookUps, Temporäre Dateien und auch die eigentlichen Ausgabedaten.

Auf technischer Ebene handelt es sich hierbei um XML als sowohl JSON Writer und Serialisierer aus der FasterXML Jackson Bilbliothek. Dabei werden mitunter die die JSONGenerator-, JSONSerializer-, ToXmlGenerator-, XmlMapper-Objekte aus der FasterXML Bibliothek genutzt. Auf diese Art lassen sich die entsprechenden Datenobjekte direkt in ein XML-Dokument übertragen. Ebenso lassen sich bei Bedarf eigene Serialisierungen und somit eigene Mapper zwischen schalten, um bestimmte Datenobjekte nach eigenen Angaben zu verarbeiten. Die eigenen Mapper werden dabei in Fällen genutzt in denen die Datenstruktur innerhalb der Anwendung nicht der Struktur der Daten, in den Produktdaten, gleicht.

Minimizer:
Die Minimizer Komponente beinhaltet die Klassen die notwendig sind um Produktdaten koordiniert minifizieren, also in ihrer Menge, auf den gewünschten Umfang, reduzieren zu können.

Generator:
Die Generator Komponente beinhaltet die Klassen die notwendig sind, um Produktdaten zu generieren, per Verweise mit Existenten Produktdaten zu verknüpfen und das Ergebnis zusammenzufügen.

Tool-Main:
Die Tool-Main Komponente beinhaltet die notwendigen Klassen, die zur Steuerung des Gesamtsystems benötigt werden und koordiniert das Zusammenspiel aus Anwender Angaben und Wahl der Subanwendung.

Communications:
Die Communications Komponente beinhaltet die Klassen, die für das Überwachen des spezifischen Ordners zuständig sind, in der die JSON-Dateien mit den Anwendereingaben abgelegt werden. Wenn eine Datei abgelegt wird leitet diese Klasse das entsprechend weiter.

Auf technischer Ebene besteht die Communications Komponente primär aus einer Ordner- beziehungsweise Dateiüberwachung und einer Liste (ArrayList) für die einzelnen Eingaben. Die Dateiüberwachung basiert dabei auf der Java NIO File Bibliothek. Die Kernklassen, aus der NIO Bibliothek, für die Überwachung der Eingabedaten im JSON Format, sind dabei die SimpleFileVisitor-, WatchEvent-, WatchKey- und WatchService-Klassen. Die Dateiüberwachung beobachtet den entsprechenden Ordner und überprüft, in voreingestellten Zeitintervallen, ob eine Eingabedatei abgelegt wurde. Wenn eine neue Eingabedatei erkannt wurde wird der Pfad für diese Datei in einer Liste abgelegt.

Essentials:
Die Essentials Komponente beinhaltet die Klassen und Interfaces, die zur Verwaltung der Daten und zur internen Kommunikation, zwischen der Komponenten, genutzt werden. Sie beinhaltet keine Arbeitslogik.

Data-Objects:
Die Data-Objects Komponente beinhaltet die nötigen Mapping Klassen damit die Produktdaten aus dem jeweiligen Dokument auf Java Objekte projiziert werden können.

Auf technischer Ebene sind diese Datenobjekte für die Produktdaten im XML-Dokumentenformat optimiert.

7.2 Ablauf Produktdaten Minifizierung

In diesem Abschnitt wird die Minifizierung der Produktdaten dargestellt und beschrieben.

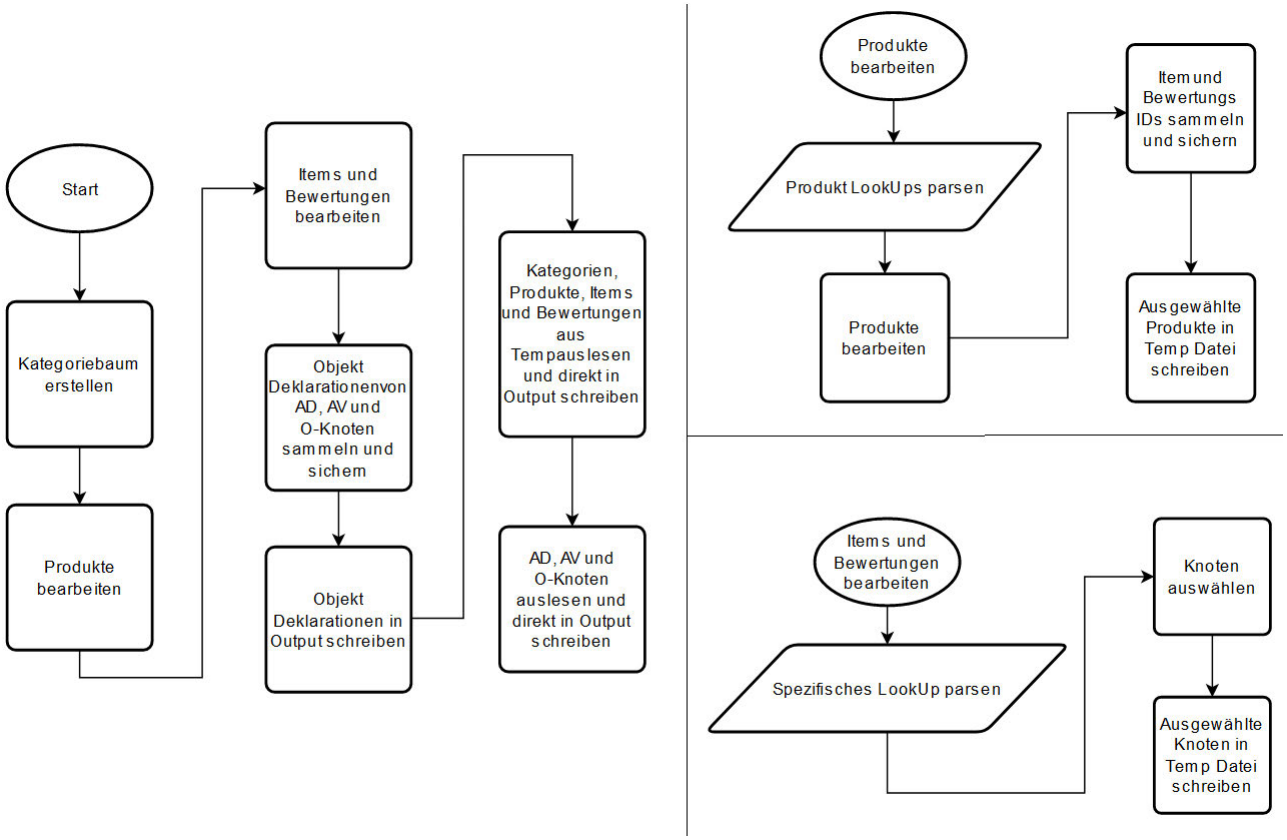

Abbildung 7.1: Ablauf Minifizierung

In der Abbildung 7.1 wird der Ablauf der Minifizierung innerhalb der Anwendung dargestellt. Als erstes wird die LookUp Datei mit den Kategorien geparst, um daraus den Kategorie-Baum zu erstellen. Danach werden als zweites die Produkte bearbeitet, dafür wird das entsprechende LookUp geparst und die Produkte, gemäß der vom Anwender angegebenen Mengen pro Kategorie, ausgewählt. Bei jedem der ausgewählten Produkte wird dann, entsprechend der Anwenderangaben, eine bestimmte Menge an Bewertung und Item IDs ausgelesen und gespeichert. Danach werden die ausgewählten Produkte in eine Temporären XML-Datei geschrieben. Als drittes und viertes werden nach der Verarbeitung der Produkte die Items und die Bewertungen bearbeitet. Dafür wird zunächst die entsprechende LookUp Datei geparst und gemäß der vorher gespeicherten IDs werden

die Knoten ausgewählt und abschließend gesammelt in die entsprechende XML-Datei für Bewertungen oder Items geschrieben. Als fünftes werden die Objekt Deklarationen der Attribut Definitionen (AD-Knoten), der Attribut Werte (AV-Knoten) und der Anderen-Knoten (O-Knoten) gesammelt und abgespeichert. Als sechstes werden dann die Objekt Deklarationen aller Knoten in den Output geschrieben. Als siebtes werden die Kategorien, Produkte, Items und Bewertungen aus den Temporären Dateien geparst und direkt in die Outputdatei geschrieben. Zuletzt werden als achtes die restlichen Knoten direkt und ohne Reduzierung in der Menge geparst und in den Output geschrieben.

7.3 Ablauf Produktdaten Generierung

In diesem Abschnitt wird die Generierung der Produktdaten dargestellt und beschrieben. Es folgt zunächst eine allgemeine Übersicht der Datengenerierung, gefolgt von der exemplarischen Darstellung wie ein einzelner Knoten generiert wird.

7.3.1 Allgemeiner Generierungsablauf

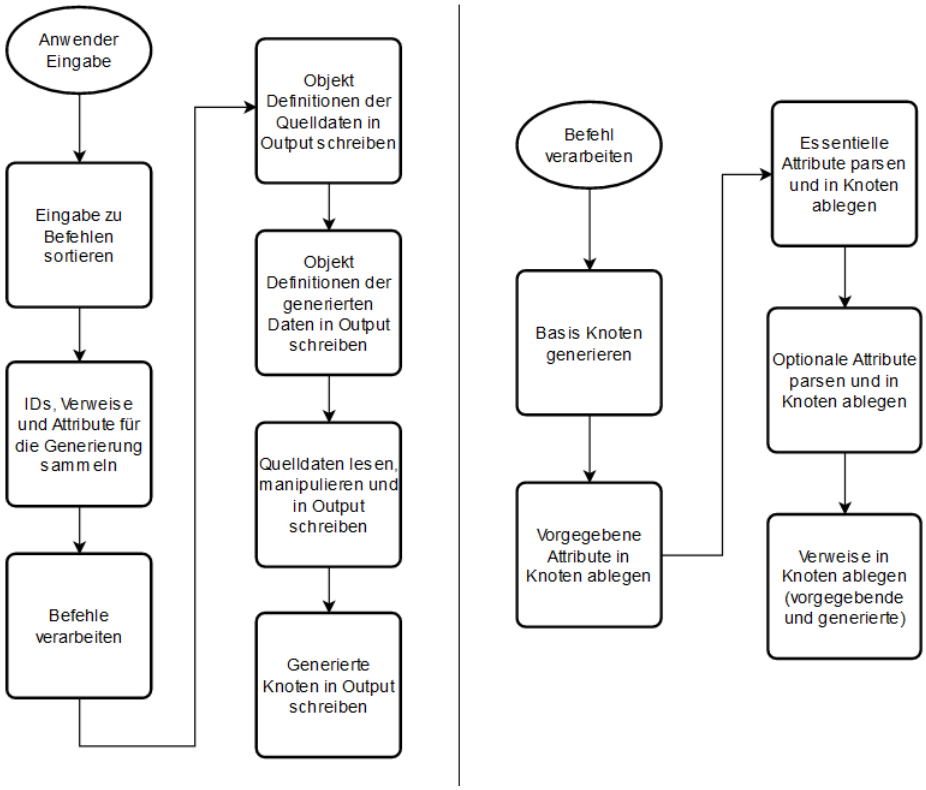

Abbildung 7.2: Ablauf Datengenerierung

In der Abbildung 7.1 wird der Ablauf der Datengenerierung innerhalb der Anwendung dargestellt. Als erstes werden die Anwender Angaben sortiert und die Datenstrukturen mit den Angaben befüllt, um so Befehle zu formieren. Als Zweites werden die IDs, Verweise und Attribute aus den Anwender Angaben gesammelt und entsprechend in die Befehls-Datenstruktur einsortiert. Danach werden schrittweise die Befehle, sofern zutreffend, verarbeitet. Als erstes werden die Kategorien, dann die Produkte, die Items, die Bewertungen, die Attribut Definitionen, die Attribut Werte und zuletzt die Anderen

Knoten verarbeitet. Die Verabreitungsabfolge sieht dabei folgender maßen aus. Als erstes werden simple Basis Knoten erstellt und mit den angegebenen IDs versehen. Danach werden die vorgegebenen Attribute gesammelt und in die entsprechenden neu generierten Basisknoten abgelegt. Danach werden die Essenziellen Attribute geparst, selektiert und in dem entsprechenden Knoten abgelegt, gefolgt von dem Parsen und Ablegen der Optionalen Attribute. Zuletzt werden innerhalb der Verarbeitung die weiteren vorgegebenen und neu generierten Verweise und Relationen in die entsprechenden Knoten abgelegt. Nach der Verarbeitung werden die Objekt Definitionen der Quelldaten gefolgt von den Objekt Definitionen der neu Generierten Knoten in die Ausgabedatei geschrieben. Danach wird die die Quelldatei erneut geparst und die Knoten in die Ausgabedatei übertragen. Bei der Übertragung der Knoten werden diese, sofern notwendig, entsprechend manipuliert, damit die Relationen und Verweise zu den neu generierten Daten passen. Abschließend werden dann die neu generierten Knoten in die Ausgabedatei geschrieben.

7.3.2 Generierung eines Knotens

Der Ablauf der Generierung eines Knotens sieht dabei wie folgt aus (Exemplarisch am Beispiel eines Produkt Knotens, da diese die meisten Daten beinhalten):

Vor der Generierung neuen Produkt Knotens wurden bereits die Kategorien bearbeitet und entsprechend abgelegt. Ebenso wurden von dem Anwender die IDs und einige bestimmte Shop-Attribute und Relationen, für die zu generierenden Produkte, mitgegeben. Darüber hinaus wurden auch die LookUps erstellt und passen zu den aktuellen Quelldaten.

Zu Beginn werden die Kategorien ausgewählt und deren IDs gesammelt in das zu generierende Produkt eingebunden werden soll. Diese Liste wird dann in den entsprechenden Generator, dem Produkt Generator, hinein gereicht und die Generierung beginnt.

Als erstes wird bestimmt um welche Art von zu generierendem Produkt es sich handelt. Da der Anwender mehrere Shop-Attribute mitgegeben hat handelt es sich um ein spezifisches Produkt, das bedeutet das die Shop-Attribute und Relationen, die vom Anwender mitgegeben wurden, nicht überschrieben oder verändert werden sollen.

Danach werden als zweites die Informationen an den Datengenerator für Produkte und Items weitergereicht, da sich die nächsten Schritte für beide Knoten gleichen. Innerhalb des Generators für Items und Produkte werden zunächst die Pfadnamen und Dateinamen vorbereitet, um die entsprechenden Daten auslesen zu können.

Als drittes wird dann das Datenobjekt für das Produkt vorbereitet und die Produkt ID hineingeschrieben. Ebenso werden auch alle vom Anwender mitgegebenen Shop-Attribute und Relationen in das Produkt Datenobjekt geschrieben.

Als viertes werden dann alle essenziellen Shop-Attribute für das Produkt gesammelt. Hierfür wird zunächst Konfigurations XML-Dokumente geparst und die nötigen Informationen gesammelt. In diesen XML- Dokumente stehen alle Attribute die nötig sind, damit das Produkt die minimalen Informationen beinhaltet, um im Shop angezeigt werden zu können.

Als fünftes werden dann die optionalen Attribute gesammelt. Das Sammeln der optionalen Shop-Attribute läuft dabei folgender maßen ab, als erstes wird ein LookUp geparst welches beinhaltet welches Attribut, wie oft in den Quelldaten vorkommt. Zur Erzeugung dieses LookUps werden nur die IDs der Attribute betrachtet aber nicht deren Inhalt oder

XML-Attribute. Danach werden die Informationen über die Anzahl der Attribute genutzt um in einem weiterem LookUp, alle Attribute aus den Quelldaten zu durchsuchen.

Die Auswahl eines einzelnen Attributes während des Parsens läuft dabei folgender Maßen ab. Wenn ein Attribut vom Parser ausgewählt wurde wird per Zufall bestimmt, ob diese Attributart betrachtet und übernommen werden soll, dabei werden alle essenziellen und im Produkt vorhandenen Attribute ignoriert. Ebenso werden alle Attribute ignoriert, die auf die Schwarze Liste gesetzt wurden. Wenn diese Attributart übernommen werden soll wird die Anzahl des Auftretens des Attributes betrachtet und eine Zufallszahl erstellt die kleiner oder gleich der Anzahl ist. Während des Parsens wird somit für jede betrachtete Attributart mitgezählt, ob dieses Attribut ausgewählt werden soll, wenn der Zähler null erreicht wird der entsprechende Vertreter dieser Attributart in das Produkt geschrieben. Es gibt zudem ein Limit an möglichen optionalen Attribute die Ausgewählt werden können, dieses Limit wird basierend auf Erfahrungswerten im Code eingestellt.

Wenn alle ausgewählten optionalen Attribute in das Produkt geschrieben wurden, folgt der sechste Schritt. Das Produkt wird zurück an den Produktdatengenerator übergeben, welcher die Relation zu den, vom Anwender angegebenen, Kategorien, Items und Bewertungen in das Produkt ablegt und die Generierung somit abschließt.

7.4 Testen

Das Testen des Tools erfolgte synchron zur Implementierung und Entwicklung. Immer wenn ein Arbeitsschritt oder Arbeitsmodul fertiggestellt wurde, wurde dieses getestet.

Das Testen des Minifizierers lässt sich in fünf Stufen unterteilen, wobei viele Anteile der Tests auch die Basis der gesamten Anwendung überprüft haben. In der ersten Stufe wurde während der Entwicklung der Lesenden Module, die Abdeckung getestet. Hierfür wurden zunächst alle potenziell möglichen Tags gesammelt und geprüft das diese in der richtigen Struktur erkannt werden. Nachdem alle Tags erkannt wurden, wurde optisch geprüft, ob alle XML-Attribute und Values richtig eingelesen werden. Danach wurden in der zweiten Stufe die Schreibenden Module in Verbindung mit den Lesenden geprüft. Hierbei wurde sichergestellt das alle Inhalte, die gelesen werden, auch direkt geschrieben werden können. Für diese Tests wurden unterschiedlich große Produktdatensets aus verschiedenen Shops ausgewählt und verarbeitet. Diese Daten wurden während des Parsens direkt in neue Dateien geschrieben und danach zunächst optisch und mit Hilfe von Konsolenbefehlen verglichen und geprüft. In der dritten Stufe wurden dann direkt vor und während der Implementierung kleinere Produktdatensätze verarbeitet und innerhalb der jeweiligen Shops geprüft. Während dieser Prüfung wurde sichergestellt das der Shop zunächst keine Fehler entdeckt, die Daten annimmt und damit auch fehlerlos funktioniert. Darüber hinaus wurde während der Prüfung im lokalen Webshop auch sichergestellt, dass die vom Minifizierer erzeugten Daten sich genauso verhalten wie die Quelldaten. In der fünften Stufe wurden die Testfälle in mehreren Variationen ausgeführt und geprüft. Dafür wurden die Ausgabe Daten hinsichtlich ihrer Struktur optisch geprüft und im Webshop ausgeführt und dort hinsichtlich ihres Verhaltens geprüft, da der Webshop eigene Mechanismen zur Prüfung von Produktdaten besitzt. So konnte die Allgemeine Funktionalität der Subanwendung des Minifizierers geprüft und kontrolliert werden. Zudem hatte die frühe Fertigstellung der Subanwendung eine fünfte Stufe des Testens ermöglicht, das Black-Box testen durch andere Anwender im Hause, welche die Subanwendung für ihren Alltag nutzen konnten. Hierbei wurde die Subanwendung des Minifizierers, in Form einer Ausführbaren Datei, mehreren Entwicklern im Hause zur Verfügung gestellt. Diese Entwickler haben dann konstruktives Feedback gegeben und sich bei etwaigen Problemen gemeldet, was iterative Optimierungen und Verbesserungen ermöglichte.

Während der Entwicklung der Subanwendung des Datengenerators stand schon ein getestetes Grundgerüst hinsichtlich der Lesenden und Schreibenden Arbeitsmodule zur Verfügung, was das Testen geradliniger gemacht hat. Das Testen der Subanwendung des

Datengenerators verlief ebenso synchron zur Entwicklung und war wesentlich kleinschrittiger als das Testen der Subanwendung des Minifizierers. Zunächst wurden die einzelnen Module, welche die jeweiligen Knoten generieren , mittels optischer Kontrollen, getestet. Wobei diese Knoten in sortierte Ausgaben geschrieben wurden, um die Sichtprüfungen und optischen Vergleiche zu vereinfachen. Danach wurden als zweites die möglichen Verbindungen zwischen den Knoten gesammelt und wieder optisch geprüft ob diese Verweise an den richtigen Stellen landen. Nachdem alle Knoten und Verweise die optischen Prüfungen überstanden hatten wurden die Daten manuell in existente und funktionierende Produktdaten kopiert und innerhalb des Webshops geprüft. Diese Daten waren dabei noch frei von Zufallselementen und bestanden aus Platzhalterwerten, welche automatisiert aus XML-Datensätzen ausgelesen wurden. Als drittes kam, nachdem die vorherigen Stufen erfolgreich absolviert wurden, die Funktionalität der Zufallsgenerierung hinzu. Um diese zu testen wurden wieder diverse Sichtprüfungen, Tests im Shop und sehr genaues Debugging genutzt. Nachdem die Ergebnisse zufriedenstellend waren wurden Testfälle für die Subanwendung des Datengenerators erstellt und ausgeführt. Die durch die Testfälle erzeugten Daten wurden dann optisch und im Shop geprüft.

Eine Sichtprüfung oder eine optische Prüfung der erzeugten Daten sah dabei wie folgt aus. Zunächst wurde das erwartete Ergebnis zusammengefasst, notiert und analysiert. Danach wurden die erzeugten Daten mit der Erwartung geprüft und analysiert. Diese Analyse beinhaltete die Prüfung der Strukturen und Verweise in den erzeugten XML-Daten. Dafür musste in jedem Knoten als erstes geprüft werden ob die Verweise auf diesen Knoten stimmen und ob der geprüfte Knoten richtig auf andere Knoten verweist. Als zweites wurde geprüft, ob der Knoten alle essenziellen Attribute enthält und dass er alle zu ignorierenden Attribute nicht enthält.

Die Prüfung im Webshop bediente sich der Prüfmechanismen des jeweiligen Shops. Da der Webshop nur Daten annehmen und richtig interpretieren wird, die einem bestimmten Format, beziehungsweise einer bestimmten Struktur, entsprechen. Wenn generierte Daten nicht den Anforderungen des Webshops genügen wird der Shop einen Fehler ausgeben oder die betroffenen Datenanteile nicht anzeigen können. Hierbei ist allerdings auch die Korrekte Konfiguration des Shop und der betroffenen Kategorien zu beachten und es lässt sich nur durch ausgiebige Sichtprüfung feststellen, ob der Fehler in der Konfiguration für den Datengenerator oder in der Generierung selbst liegt.

Bei den Testfällen handelt es sich in dieser Arbeit um bestimmte, auf der Realität, basierende Szenarien welche für die aktiv ausgeführten Tests soweit erweitert wurden, dass jede

erfasste Kombination abgedeckt wurde. Diese Szenarien bieten keine hundertprozentige Abdeckung, werden allerdings während der aktiven Benutzung der Gesamtanwendung erweitert und immer weiter ausgebaut.

Das aktive Testen der Gesamtanwendung hat das Ziel eine fehlerfreie und stabile Funktionalität sicherzustellen, wobei sichergestellt wurde, dass das Programm selbst bei einer Fehlfunktion oder einem Absturz keine Schäden anrichten wird. Dafür wurden besonders die schreibenden und lesenden Komponenten streng begrenzt und es wurde sichergestellt das diese Komponenten nur innerhalb eines vorgegebenen Bereiches agieren. Die Anwendung setzt allerdings einen gewissen Grad an Eigenverantwortung voraus, denn wenn ein Anwender Eingaben tätigt, die keinen Sinn ergeben wird, die Ausgabe auch keinen Sinn ergeben.

8 Diskussion

Im Rahmen dieser Arbeit wurde ein automatisierter Testdaten-Generator für XML-basierte Produktdaten im E-Commerce an einem Exempel konzipiert und implementiert. Der implementierte Testdatengenerator konzentriert sich in seiner Funktionalität auf zwei primäre Subanwendungen. Die erste Subanwendung ist der Datengenerator. Der Datengenerator generiert, basierend auf Konfigurationen und Eingaben zufallsbasierte Produktdaten. Die vom Datengenerator erzeugten Produktdaten werden zu einem existenten Datensatz hinzugefügt um diesen um die neugenerierten Daten zu erweitern. Die zweite Subanwendung ist der Minifizierer, welcher die Aufgabe hat existente Datensätze, die zu groß für den lokalen Gebrauch sind, in ihrer Menge und Größe soweit zu reduzieren, dass der Datensätze lokal ausführbar ist und alle für den jeweiligen Test benötigten Daten beinhaltet.

8.1 Rückblick auf die Entwicklung

Die Erstellung des automatisierten Testdaten-Generators erfolgte mehrschrittig. Zunächst wurden die einzelnen Anforderungen gesammelt. Es wurden Interviews mit den zukünftigen Anwendern und deren Vorgesetzten geführt, um so einen sinnvollen Überblick zu erhalten. Danach wurde aus Gründen der Testbarkeit zunächst die Subanwendung des Minifizierers in einzelnen Schritten konzipiert und implementiert.

Der Minifizierer wurde zunächst, hinsichtlich seines prinzipiellen Arbeitsablaufes und seiner einzelnen Arbeitsschritte, konzipiert. Diese Arbeitsschritte wurden dann schrittweise implementiert und getestet, um so etwaige Fehler möglichst gering zu halten und die Menge an potenziellen Fehlerquellen stets im Blick behalten zu können. Als erstes wurde das Modul für den Arbeitsschritt des Einlesen und Parsen von XML-Daten implementiert, danach das Modul des Schreibens und zuletzt die einzelnen verarbeitenden Module. Es wurde zudem auch eine rudimentäre GUI erstellt und die Subanwendung wurde einigen wenigen Entwicklern zur Verfügung gestellt, um die Stabilität direkt testen zu können.

Während der Entwicklung und Implementierung der Subanwendung des Minifizierers kam das Problem der Speicherverwaltung auf. Die Produktdaten, welche als Quelldaten dienen sollten, waren teilweise zu groß und konnten nicht sinnvoll im Arbeitsspeicher abgelegt und verarbeitet werden. Basierend auf diesem Problem wurden weitere Interviews und Besprechungen mit ausgewählten Entwicklern und Administratoren geführt. Im Zuge dieser Interviews wurde das, in dieser Arbeit konzipierte, Modell der LookUp Systematik diskutiert und besprochen. Zudem wurde auch besprochen welche Grenzen und Bedingungen auf den lokalen Rechnern und den Servern existieren. Die Ergebnisse aus diesen Interviews und Besprechungen wurden zusammengetragen und basierend darauf, wurde die finale Version der LookUp Systematik konzipiert und anschließend implementiert. Nach der Fertigstellung der LookUp Systematik konnte dann die Subanwendung des Minifizierers fertiggestellt werden.

Nachdem der Minifizierer fertig war, wurde die Subanwendung des Datengenerators zu Ende konzipiert und anschließend implementiert. Diese Arbeitsreihenfolge hatte den Vorteil, dass bereits praktische Erfahrungen in der Verarbeitung von XML-Daten gesammelt werden konnten. Basierend auf dieser Erfahrung wurde die Konzeption des Datengenerators angepasst und optimiert. Die Implementierung erfolgte bei dem Datengenerator ebenfalls schrittweise. Zunächst wurden die einzelnen Generierungsschritte, also die Generierung der einzelnen verschiedenen Knoten, realisiert und getestet. Nach jedem vollendetem Schritt wurde das System erneut getestet und etwaige Ergebnisse und Ausgaben überprüft. Während der Generierung der einzelnen Knoten wurde zunächst sichergestellt, dass erst diese Knotenart als solches erstellt werden kann und danach wurde sichergestellt, dass die Verweise und Verbindungen der Knoten untereinander korrekt sind.

Während der Implementierung der Subanwendung des Datengenerators wurde auch die Eingabesystematik und die Arbeitsweise des geregelten Zufalls konzipiert und implementiert. Die Entwicklung der Eingabesystematik wurde ebenfalls von Interviews und Absprachen begleitet, um zu verstehen was das System, aus Sicht der Anwender leisten muss und welche Daten die Anwender in den erzeugten Dateien erwarten. Die Implementierung der zufallsgesteuerten Generierung der Daten wurde neben der Generierung der einzelnen Knoten implementiert, aber auch stetig erweitert und überarbeitet.

Als letztes wurde das Problem der Konfigurierbarkeit für den Datengenerator genauer betrachtet. Jeder Shop benötigt bestimmte Vorgaben hinsichtlich essenzieller und optionaler Informationen in den Produktdaten. Diese Vorgaben müssen im späteren Verlauf von den Entwicklern benannt und zur Verfügung gestellt werden, da diese stark von der

Implementierung der jeweiligen Webshop Logik abhängig sind. Ebenso hat jeder Shop, unter Umständen, mehrere Kategorien, welche besondere und zusätzliche Konfigurationen benötigen.

Im Rahmen dieser Bachelorarbeit wurde nur ein Shop und eine darin enthaltene Kategorie vorkonfiguriert. Die Konfiguration eines Shops und dessen Kategorien für den Datengenerator benötigt die aktive Hilfe eines Entwicklers, welcher mit dem jeweiligen Shop vertraut sein muss. Dies bedeutet, dass der helfende Entwickler genug Zeit mitbringen muss, was in dem Rahmen der Bachelorarbeit leider nicht möglich war. Daher wurde der Fokus auf eine Kategorie in einem Shop gelegt, da es ansonsten zu erheblichen Wartezeiten gekommen wäre, die das Projekt ausgebremst hätten. Der eigentliche Plan, am Ende der Bachelorarbeit einen kompletten Shop, hinsichtlich der Datengenerierung, zu unterstützen war somit nicht umsetzbar, in der gegebenen Zeit. Basierend auf dem bisherigen Konzept der Datengenerierung, wird es allerdings genügen eine Konfigurationsdatei hinzuzufügen, um so eine weitere Kategorie zu unterstützen. Selbstverständlich muss dies dann auch nochmal getestet werden, um eine stabile Testgrundlage garantieren zu können. Die neu generierten Daten wurden ausgiebig geprüft, im laufenden Shop getestet und mit reellen Produktdaten verglichen. Durch diese Tests wurde sichergestellt das erstens die generierten Daten den Ansprüchen genügen und zweitens, dass die Konfigurationsdateien die Generierung wie gewünscht beeinflussen.

Auf der anderen Seite bekam die Subanwendung des Minifizierers bereits produktive und positive Rückmeldungen. Diese Rückmeldungen und die darauf basierenden Verbesserungen waren möglich, da die Subanwendung des Minifizierers während der Entwicklung des Datengenerators, ausgewählten Entwicklern zur Verfügung gestellt wurde. Diese ausgewählten Entwickler haben somit als Black Box Tester für die Subanwendung gedient.

Das Konzept des Minifizierers war wesentlich einfacher zu konzipieren als das des Datengenerators. Der Minifizierer basiert im Kern auf einem einstellbaren Auswahlmechanismus und muss auf die entsprechenden Datenstrukturen der XML-basierten Daten angepasst werden. Dabei stellt der Minifizierer dem Anwender Einstellmöglichkeiten zur Verfügung, damit dieser bestimmte Mengengrößen einstellen kann oder bestimmte Inhalte vorgeben und ausschließen kann. Der Datengenerator war in seiner Konzeptionierung komplexer, da die generierten Daten nicht einfach aus den Ursprungsdaten kopiert oder in ihrer Menge reduziert wurden.

Der Datengenerator musste neue Inhalte generieren können, wobei die Varianz und die Vielfalt wichtiger war als die Sinnhaftigkeit. Es musste hierbei nicht darauf geachtet wer-

den, dass zum Beispiel die Kombination aus dem ausgewählten Bild und dem beschreibenden Text zu einem neu generierten Produkt passen. Die Probleme bei der Generierung lagen primär in zwei Bereichen, der erste und kleinere Bereich war die Konfiguration. Die Konfigurationen mussten für den Shop und für die entsprechenden Kategorien erst einmal extrahiert, dann zusammengetragen und anschließend in passender Form und Struktur abgelegt werden, damit diese bei der Generierung betrachtet werden können. Eine Konfiguration stellt aus Sicht des Datengenerators eine Anleitung dar, im XML-Dokumentenformat. Die Konfiguration beschreibt welche Inhalte in dem jeweils aktuell generierten Knotenessenziell sind, also auftauchen müssen, optional auftauchen können oder nicht vorhanden sein dürfen. Das Verarbeiten einer solchen Konfiguration war somit einfach, verglichen mit dem Aufwand der Erstellung. Der zweite herausfordernde Bereich ist der Ablauf des geregelten Zufalls. Die Regeln des Zufalls basieren auf der Struktur und dem potenziellen Inhalt des zu generierenden Knotens und auf den jeweils relevanten Konfigurationen.

Basierend auf den Regeln für den Zufall, welcher für die Generierung essenziell ist, wurde eine weitere LookUps erstellt. Diese spezifischen LookUps beinhalten die möglichen Inhalte, beziehungsweise die Attribute der Quelldaten in sortierter und für den Zugriff optimierter Form. Für die Knotenart des Produktes oder Items zum Beispiel gibt es daher jeweils zwei LookUps. Der erste LookUp beinhaltet alle potenziellen Attribute für die jeweilige Knotenart, ohne Doppelungen. Aus diesen Einträgen wird später das spezifische Attribut ausgewählt. Der zweite LookUp beinhaltet die Namen und ID's der potenziellen Attribute und ist somit wesentlich kleiner und schneller zu parsen als der erste LookUp. Zudem beinhaltet der zweite LookUp eine Mengenangabe, welche beinhaltet wie oft das jeweilige Attribut in dem ersten LookUp vorkommt. Dank dieser Mengenangabe kann eine simple Zufallsregelung realisiert werden. Es wird zunächst der zweitgenannte LookUp, der Namens-LookUp, und danach der erstgenannte, der Attribut-LookUp, LookUp geparst. Beim Parsen des Namens-LookUps wird entschieden, ob dieses Attribut in den Knoten mit aufgenommen werden soll. Bei essenziellen Attributen erfolgt diese Entscheidung entsprechend der Konfiguration, bei optionalen Attributen wird dies durch den Zufall entschieden. Wenn ein Attribut ausgewählt wurde, wird danach eine Zufallszahl bestimmt, die innerhalb der Mengenangabe liegt, dies bildet dann den Auswahlmechanismus für das zweite Parsen. Danach wird der Attribut-LookUp geparst und es werden nur die vorher ausgewählten Attribute in den Knoten übernommen.

Diese Zufallsregelungen durchlebten viele Änderungen und Optimierungen, damit die Generierung stabil abläuft und das Ergebnis vielfältig ist. Während der Konzeptionierung

waren diese Aspekte nicht detailliert genug planbar und wurden synchron zur Implementierung iterativ ausgearbeitet. Ebenso war das System der LookUp Systematik während der Konzeptionierung nicht geplant und wurde zu einem, immer essenzieller werdenden, Stützpfeiler des Gesamtsystems. Ohne die LookUp Systematik wäre die aktuell genutzte Form des geregelten Zufalls nicht möglich gewesen.

Das Parsen der XML-Dokumente stellte oftmals eine starke Limitierung dar, da es nicht möglich war während des Parsens in die später folgenden Daten zu blicken. Bei der Arbeit mit den XML-basierten Daten musste stets abgewägt werden, welche Daten in einem XML-Dokument gespeichert werden und welche im Arbeitsspeicher, für einen schnellen Zugriff, gehalten werden. Zudem mussten die Daten sinnvoll in einem XML-Dokument abgelegt werden, um die Häufigkeit des Parsens zu reduzieren, das gleiche gilt für die damit verbundene Arbeitslogik. Es war daher stets nötig zu analysieren welche Information die Anwendung an welcher Stelle benötigt, um das Parsen und die Struktur dahingehend zu optimieren. Die Arbeit stellt hinsichtlich dieser Optimierungen somit kein perfektes Ergebnis dar. In der Zukunft wird es an dieser Stelle noch Möglichkeiten für Verbesserungen geben.

8.2 Evaluation der Zielerfüllung

Die Evaluation macht deutlich, dass das Ziel einen exemplarischen, automatisierten Testdaten-Generators, für XML-basierte Produktdaten im E-Commerce, zu entwickeln erfüllt wurde. Das System wurde bereits mit mehreren Eingaben getestet und die Ergebnisse konnten erfolgreich ausgeführt werden. Die bisher getätigten Eingaben decken, die bereits gesammelten Anwendungsfälle, ab und das System arbeitet, innerhalb dieser Grenzen, wie gewünscht.

Innerhalb des gegebenen Zeitrahmens war es nur möglich das System so weit zu konfigurieren, dass ein Shop und eine darin enthaltene Kategorie für den praktischen Gebrauch unterstützt wird. Um weitere Webshops und Kategorien unterstützen zu können, müssen lediglich Konfigurationsdateien erstellt und abgelegt werden. Das Erstellen neuer Konfigurationsdateien stellt nur einen zeitlichen und keinen entwicklungsbezogenen Aufwand dar. Aus Sicht des Testdaten-Generators ist eine Konfigurationsdatei lediglich eine Anleitung und stellt für die Verarbeitung innerhalb des Systems kein Hindernis dar. Die Anwendung und deren Ausgaben wurden mit Blick auf zukünftig mögliche Konfigurationen und deren Verarbeitung geprüft. Die Ausgaben der Anwendung entsprechen in ihrer Form und Struktur den Ursprungsdaten. Die generierten Knoten beinhalten die gewünschte zufällig erzeugte Vielfalt. Zudem stimmen die Verweise der Knoten untereinander und somit wurde die Hierarchie der Knoten auch eingehalten. Die Anwendung verhält sich hinsichtlich ihrer Arbeitsspeicher Auslastung minimal, auch wenn es hierbei in der Zukunft Verbesserungspotential gibt.

Erfüllte Anforderungen

In diesem Kapitel wird die Erfüllung der einzelnen Anforderungen dargestellt, wobei "✓"bedeutet das diese Anforderung erfüllt wurde. Besonderheiten und Eigenarten werden unter Anmerkungen aufgelistet.

E-1	✓	E-11	✓
E-2	✓	E-12	✓
E-3	✓	E-13	✓
E-4	✓	E-14	✓
E-5	✓	E-15	✓
E-6	✓	E-16	✓
E-7	✓	E-17	✓
E-8	✓	E-18	✓
E-9	✓	E-19	✓
E-10	✓	E-20	✓

Entnommene funktionale Anforderungen (E-n)

W-1	✓	N-1	✓
W-2	✓	N-2	✓
W-3	✓	N-3	✓
W-4	✓	—	—
W-5	✓	—	—

Weitere funktionale Anforderungen (W-n)
und Nichtfunktionale Anforderugnen (N-n)

Anmerkungen

Bezüglich E-4 und E-8 (anteilig somit auch bezüglich E-5 und E-9) ist die Anwendung zwar dazu in der Lage konfiguriert zu werden und derartige Konfigurationen Problemlos lesen zu können, allerdings hatte der Zeitliche Rahmen keine Möglichkeit geboten mehr als einen Shop oder mehr als eine Kategorie konfigurieren und dementsprechend ausgiebig testen zu können.

Bezüglich E-11, E-12 und E-13 ist anzumerken das die Anwendung selbstständig keine Webshop Attribute erzeugen kann, sondern sich dabei auf die korrekten Eingaben des Anwenders verlassen muss.

Aufgrund der Baumstruktur ist E-18 umsetzbar allerdings ergibt sich ein White-List und kein Black-List Format, es werden also keine Kategorien explizit ausgegrenzt, sondern sich auch bestimmte Kategorien fokussiert.

Die nichtfunktionalen Anforderungen N-1 und N-3 wurden eingehalten, aber an diesen Stellen gibt es dennoch Potential für weitere Verbesserungen und Optimierungen. Die nichtfunktionale Anforderung N-3 kann vom Anwender umgangen werden, wenn dieser mit Adminrechten die entsprechenden Grenzkontrollen der Mengenangaben umgehen möchte. Wird die Subanwendug des Minifizierers aber standardmäßig genutzt wird die Anforderung N-2 erfüllt.

8.3 Abstraktion des Generierungsprozesses

Der in dieser Arbeit entwickelte Testdatengenerator wurde auf technischer Ebene exemplarisch für eine Anwendungsumgebung entwickelt und beinhaltet hinsichtlich seiner Funktionalität zwei zentrale Subanwendungen. Die Subanwendungen orientieren sich dabei an den Anwendungsfällen und wie diese gruppiert werden. Die fachlichen und die technischen Komponenten sind durch den exemplarischen und somit begrenzten Anwendungszweck identisch. Allerdings lassen sich die technischen Komponenten einfach erweitern und ordnen sich dann den entsprechenden fachlichen Komponenten unter.

Es ist denkbar, dass neue Anwendungsfälle dazu kommen können. Diese neuen Anwendungsfälle würden entweder die existenten Subanwendungen erweitern, kombinieren oder eine neue Gruppe an Anforderungsfällen bilden und somit den Bedarf nach einer neuen Subanwendung darstellen. In dem letzteren Fall müssten die existenten Komponenten entweder erweitert oder neue Komponenten erstellt werden, welche die neuen Subanwendungen darstellen würden.

Des Weiteren basiert die entwickelte Anwendung auf bestimmten Datenformaten (XML, JSON) und bestimmten Strukturen innerhalb der Daten. Es besteht theoretisch auch die Möglichkeit das sich diese grundlegenden Datenformate und Datenstrukturen ändern oder der Testdatengenerator auch in anderen Bereichen eingesetzt werden soll. Sollte dieser Fall eintreten, ist es bei einer Änderung des Datenformates notwendig die Parser- und Writer-Komponenten entweder auszutauschen oder entsprechend neue Schreib- und Lese-Komponenten zu entwickeln, beziehungsweise zu implementieren und in diese fachlichen Komponenten einzugliedern. Sollte sich die Struktur innerhalb der Daten hingegen ändern, muss die Komponente der Datenelemente angepasst oder erweitert werden. Darüber hinaus muss, bei einer strukturellen Änderung der Daten, die Arbeitslogik innerhalb der Subanwendungen gegebenenfalls angepasst oder erweitert werden.

9 Zusammenfassung und Ausblick

9.1 Fazit

Das Ziel dieser Arbeit war es einen Testdatengenerator für Webshops zu implementieren, welcher automatisiert, gemäß Angaben und Spezifikationen des Anwenders, Produktdaten generiert oder in ihrer Menge reduzieren kann. Diese Testdaten werden dann zum Testen in einer lokalen oder Server Umgebung genutzt. Für die Erzeugung dieser Testdaten dienen reale Produktdaten als Grundlage.

Um diese Generierung realisieren zu können, musste ein Verwaltungs- und Verarbeitungssystem konzipiert werden, damit die großen XML-Dokumente, mit geringem Arbeitsspeicherbedarf, verarbeitet werden können. Ebenso musste die Balance hinsichtlich des Aufwandes, für den späteren Anwender, gewahrt werden. Das System dient der Entlastung des Anwenders und soll ihn weder beschränken noch zu viel Arbeit bei der Generierung der Daten auflasten. Die möglichen Angaben und Anwendungsfälle mussten somit in Kommunikation und Rücksprache mit den zukünftigen Anwendern sorgfältig gewählt werden. Zudem musste die Struktur der Produktdaten gewahrt und die Eigenheiten des jeweiligen Shops beachtet werden.

Die Implementierung und Konzeption wurden schrittweise absolviert, um so ein stabiles System zu realisieren. Zu Beginn wurden die Produktdaten analysiert, um darauf aufbauend die schreibenden und lesenden Komponenten zu konzipieren, zu testen und um potenzielle Ansätze zu vergleichen. So konnte auch direkt die Verarbeitung der Daten simpel getestet werden, da die ausgehenden Daten den eingehenden entsprechen mussten. Danach wurde die Subanwendung des Minifizierers konzipiert und implementiert, welche zunächst kleine Produktdatensätze reduziert hat. Danach wurde die LookUp Systematik konzipiert und realisiert, um große Datensätze verarbeiten zu können. Nachdem alle Tests hinsichtlich der Subanwendung des Minifizierers abgeschlossen waren, wurde der

Datengenerator konzipiert. Im Laufe der Entwicklung wurde die LookUp Systematik entsprechend erweitert, um die Generierung der Produktdaten schrittweise realisieren und testen zu können. Während der jeweiligen Konzeptionsphase wurden immer wieder Interviews, mit den potenziellen zukünftigen Anwendern geführt, um so die Anwendungsfälle und Benutzerschnittstellen sinnvoll auszuarbeiten. Als Resultat dessen wurde die Subanwendung des Datengenerators und somit die Gesamtanwendung fertig gestellt.

9.2 Ausblick

9.2.1 Weitere mögliche Anwendungsfälle und Erweiterungen

Übertragung fremder Webshop Daten

Ein potenziell möglicher Anwendungsfall ist das Übertragen von Produktdaten aus einem Webshop in einen anderen Webshop. Davon ausgehend, dass beide Shops von der Anwendung unterstützt werden und das einer der Shops sehr viele Daten hat und der andere Shop nicht genug, um darauf basierende Testdaten zu generieren. Um das zu realisieren müssen die Strukturen der beiden Shops erkannt werden und die Anwendung muss diese Strukturen übersetzen können.

Der Anwender möchte zum Beispiel, für einen neu entwickelten Shop, mit so gut wie keinen Produktdaten aber genug strukturellen Informationen, Daten generieren, um den Shop auf einem Server testen zu können. Um realitätsnahe Tests durchführen zu können braucht er also große Mengen an Produktdaten. Er benötigt also Daten aus anderen Quellen, da sein Shop nicht genug Daten liefert, um genug Vielfalt zu schaffen.

Generieren und Minifizieren in einem Durchgang

Ein weiterer möglicher Anwendungsfall wäre das gleichzeitige Generieren und Minifizieren von Produktdaten. Ein Anwender möchte neue Daten generieren und als Ausgabe einen Datensatz haben, den er direkt lokal testen kann, ohne dass er die Subanwendungen hintereinander ausführen muss. Oder der Anwender möchte einen minifizierten und einen großen Datensatz haben, damit er lokal und synchron auf einem Server testen kann, wobei der minifizierte auf dem großen Datensatz basiert.

9.2.2 Mögliche Verbesserungen und Optimierungen

Subanwendungen weiter modularisieren

Eine potenzielle Optimierung der Anwendung wäre es die Arbeitsprozesse der Subanwendungen noch weiter zu modularisieren, um die Flexibilität und die internen Prozesse zu optimieren. So würde man zudem auch die Wartbarkeit erhöhen und man könnte besser auf potenzielle Veränderungen in der XML-Struktur der Produktdaten reagieren. Zudem würde diese Optimierung die Integration neuer Webshops erleichtern.

LookUp Erstellung optimieren

Eine mögliche Verbesserung wäre es die Erstellung der LookUps zu optimieren, indem man die Erstellung automatisch und gruppiert startet. Dies lässt sich durch das Auslagern der LookUp Erstellung in einen eigenen Prozess realisieren, welcher in voreingestellten Zyklen die LookUps für ausgewählte Webshops erstellt.

Prozesse parallelisieren

Eine weitere Optimierung und Verbesserung wäre die Parallelisierung der Module in der Anwendung. Auf die Art könnten mehrere Aufträge zur Datengenerierung und Minifizierung zeitgleich bearbeitet werden.

9.2.3 Blick auf zukünftige Forschungen

Diese Arbeit fokussiert sich auf eine exemplarische Entwicklung, von daher ist es sehr gut möglich weitere Kernaspekte zu extrahieren und allgemeingültige Datengenerierungssystematiken zu erstellen. Ebenso werden nicht alle potenziell möglichen Anwendungsfälle behandelt, sondern nur die bisher gesammelten. Während der Entwicklung kamen bereits neue potenzielle Ideen auf was die Systematiken leisten könnten und wie man die Gesamtanwendung erweitern kann.

Im Vergleich zu bereits existierenden Arbeiten hat diese Anwendung die Besonderheit, dass sie Daten zunächst primär erweitert, aber auch für die Zukunft das Potenzial hat komplett eigene Daten zu generieren oder zu übersetzen. Zudem behandelt die Anwendung in dieser Arbeit zwei Testumgebungen, die auf dem Server und die auf dem lokalen Rechner. Wobei dieselbe Datei hier als Testgrundlage dienen kann und für den lokalen Testgebrauch lediglich auf den benötigten Kern an Daten reduziert wird. Die Einflussmöglichkeiten des Anwenders sind dabei ebenfalls sehr groß, was aber auch den Nachteil hat, dass unerfahrene Anwender für sie nicht praktische Daten erzeugen können.

Mit Hinblick auf zukünftige Arbeiten, gibt es genug Spielraum die Subanwendungen dieser Arbeit zu erweitern, zu kombinieren oder in andere Bereiche zu transferieren, um sie da in andere Gesamtsysteme einzubetten. Ein Beispiel für die Zukunft wäre eine Parallelisierung der einzelnen Arbeitsschritte und damit verbundene Analysen hinsichtlich der Performanz. Ein weiteres Beispiel für eine zukünftige Arbeit wäre eine automatisierte Erstellung der Konfigurationsdateien, welche die entsprechenden Informationen aus derselben Quelle wie der Programmcode ausliest.

Literaturverzeichnis

[BM98] Henning Behme and Stefan Mintert. *XML in der Praxis.* Addison-Wesley, 1998.

[Cle10] Torsten Cleff. *Basiswissen Testen von Software Vorbereitung zum Certified Tester (Foundation Level) nach ISTQB-Standard.* Herdecke Witten W3L-Verlag, 2010.

[Dob99] Susanne Dobratz. Xml - extensible markup language. HU-Berlin, 1999.

[For16] Alexander Foril. Synthetic data generation forbig data. Universität Stuttgart, 2016.

[LBPH07] Julius Lochbaum, René Bergelt, Time Pech, and Wolfram Hardt. Erzeugung von testdaten für automatisiertes fahren auf basis eines open source fahrsimulators. Technische Universität Chemnitz, 2007.

[Lig09] Peter Liggesmeyer. *Software-Qualität: Testen, Analysieren und Verifizieren von Software.* Springer Spektrum, 2009.

[MBFW14] Nikolaus Moll, Christian Baranowski, Thomas Fox, and Juergen Waesch. Modellierung und generierung von testdaten fur datenbank-basierte anwendungen. Hasso Plattner Institut der Universität Potsdam, 2014.

[ME06] Brett McLaughlin and Justin Edelson. *Java and XML.* O'Reilly Media, Inc., 2006.

[Mü11] Thomas Müller. Certified tester foundation level syllabus. International Software Testing Qualification Board, 2011.

[Sch83] Paul Schmitz. *Software-Qualitätssicherung — Testen im Software-Lebenszyklus.* Vieweg+Teubner Verlag, 1983.

[Sch05] Peter Scholz. *Vorgehensmodelle und Standards der Entwicklung.* Springer, 2005.

[Sch07] Thomas Peter Schulenberg. Entwicklung eines tools zur generierungvon testdaten für data warehouses. Institut für Informatik, 2007.

[SL19] Andreas Spillner and Tilo Linz. *Basiswissen Softwaretest : Aus- und Weiterbildung zum Certified Tester – Foundation Level nach ISTQB®-Standard.* dpunkt.verlag, 2019.

A Anhang

Es folgt eine Auflistung des Anhanges, der sich auf der beigelegten CD befindet.

A.0.1 Bachelorarbeit_Steven_Beyermann.pdf

Die Bachelorarbeit_Steven_Beyermann.pdf beinhaltet die Bachelorarbeit in digitaler Form.

A.0.2 README.txt

Die README.txt beinhaltet eine kurze Anleitung zum Ausführen des Produktdatengenerators und gibt zudem eine zusammengefasste Übersicht über den Inhalt der CD.

A.0.3 GeneratorTool.zip

Die GeneratorTool.zip beinhaltet die GeneratorTool.jar Datei zum Ausführen des Produktdatengenerators, welcher im Rahmen dieser Arbeit entwickelt wurde.

A.0.4 GeneratorGUI.zip

Die GeneratorGUI.zip beinhaltet die GeneratorGUI.jar Datei zum Ausführen der GUI für den Produktdatengenerator.

A.0.5 paths.properties

Die paths.properties Datei beinhaltet Informationen über Pfade, die der Produktdatengenerator benötigt und welche vom Nutzer an die Umgebung angepasst werden müssen.

A.0.6 startTool.sh

Die Datei startTool.sh beinhaltet das BashScript zum Ausführen der GeneratorTool.jar Datei mit den entsprechenden Optionen.

A.0.7 startGUI.sh

Die Datei startGUI.sh beinhaltet das BashScript zum Ausführen der GeneratorGUI.jar Datei mit den entsprechenden Optionen.

A.0.8 Quellcode.zip

Die Quellcode.zip Datei beinhaltet den Quellcode für den Produktdatengenerator und die dazugehörige GUI.

A.0.9 ExampleInput.json

Die Datei ExampleInput.json beinhaltet eine beispielhafte Inputdatei für den Produktdatengenerator. Diese Datei dient nur der Veranschaulichung und ist nicht für die Verarbeitung gedacht.

A.0.10 Testcases.zip

Die Testcases.zip beinhaltet *.json Dateien, welche den Testfällen aus dem Anhang Teil-C entsprechen und als Eingabedateien für den Produktdatengenerator dienen und dazu passende exemplarische Ausgabedateien. Diese Datei ist in 2 Parts aufgeteilt: Testcases.zip.001 und Testcases.zip.002

A.0.11 Productdata.xml

Die Datei Productdata.xml enthält einen vorbereiteten und nutzbaren Produktdatensatz für den Produktdatengenerator. Diese Datei befindet sich innerhalb von "Datengenerator.zip".

A.0.12 ExampleStructure.zip

Die Datei ExampleStructure.zip beinhaltet eine beispielhafte Ordnerstruktur zur Ausführung des Produktdatengenerators. Zudem sind auch an die Productdata.xml angepasste Konfigurationen enthalten.

A.0.13 Abbildungen.zip

Die Datei Abbildungen.zip beinhaltet die original Versionen der Abbildungen, die für diese Arbeit erstellt wurden. Zudem enthält die Abbildungen.zip auch noch weitere Abbildungen und eine entsprechende SSummary.txt"Datei welche die Bilder in Kurzform beschreibt.

B Testfälle

B.1 Minimale Generierung von jedem Knotentyp

ID: TC:G001

Deckt folgende Fälle ab: UC:G002, UC:G003,

Beschreibung:

Es wird von jedem Knotentyp genau ein Exemplar erzeugt. Diese Daten sind miteinander verbunden und werden in eine existente Kategorie eingehangen. Es handelt sich explizit um ein neues Produkt einer neuen Kategorie, welches ein neues Item mit einer entsprechend neuen Bewertung beinhaltet. Zu dem Produkt wird ein neues Attribut (die Kombination aus neuem AD-Knoten und neuem AV-Knoten) hinzugefügt und das Item erhält einen neuen O-Knoten (eine Servicefunktionalität). Dieser neu generierte Datensatz wird in die angegebene, existente Kategorie eingehangen.

Instruktionen für die Anwendung:

```
    Anwendungsoption: Generieren

#XML:(
  <d id="Y_YH50111" type="N" keywords="#Existing:(true)">
    <r id="Y_Y12345678" type="N" keywords="#Existing:(false)"/>
  </d>
  <d id="Y_Y12345678" type="N" keywords="#Existing:(false)">
    <a n="CATEGORYNAME" l="de">Generierte Teppiche</a>
  </d>

  <d id="55541099007" type="P" keywords="#Existing:(false)">
```

```
    <r id="55541099008" type="I" relationType="child"
        keywords="#Existing:(false)"/>
    <a n="CATEGORYPATH" b="Yourhome">Y_Y12345678</a>
    <a n="generiertesFeld" l="de" adref="A123456789"
        avref="AV123456789">generierter AV Begriff</a>
  </d>

  <d id="55541099008" type="I" keywords="#Existing:(false)">
    <r id="8cd61123-334a-1234-8b36-c21c4eb12345" type="O"
        relationType="Yourhome_Service"/>
    <a n="name" adref="A1884685095">Gestreifte Fransen</a>
  </d>

  <d id="55541099009" type="B" keywords="#Existing:(false)">
    <a n="productId" adref="A273981979">55541099007</a>
    <a n="articleNumber" adref="A273981984">55541099008</a>
  </d>

  <d id="A123456789" type="AD">
    <a n="identifier">SpecialDescription</a>
    <a n="displayName" l="de">GenerierteBeschreibung</a>
    <a n="sequenceNo"/>
    <a n="attributeScope">ALL</a>
    <a n="attributeType">S</a>
    <a n="attributeGroup" adref="TT66" avref="AV158">IPIMCORE</a>
    <a n="attributeDataLevel">productLevel</a>
    <a n="ATTRIBUTION_USE_PRD_ATTR_FOR_ITEM">true</a>
    <a n="elastic_search_exclude_attribute">true</a>
    <a n="ishop_exclude_identifier">true</a>
    <a n="DC_IS_SOURCE_ATTR">true</a>
    <a n="ishop_export_name">geneerierterName</a>
  </d>

  <d id="AV123456789" type="AV">
    <a n="identifier">generierter AV Begriff</a>
    <a n="displayName" l="de">generierter AV Begriff</a>
```

```
    <a n="sequenceNo"></a>
  </d>

  <d id="8cd61123-334a-1234-8b36-c21c4eb12345" type="O">
    <a n="name">2 Monate Garantie</a>
    <a n="dc_delivery24h" t="boolean">true</a>
    <a n="dc_desiredDate" t="boolean">true</a>
    <a n="ServiceGroup" l="de">2 Monate Garantie</a>
    <a n="oc_orderableBeforeInvoicing" t="boolean">true</a>
    <a n="serviceType">4</a>
    <a n="service_validFrom" b="Yourhome">28.08.2020</a>
    <a n="selectionType">1</a>
    <a n="oc_maximumOrderQuantity">1</a>
    <a n="oc_daysOrderableAfterInvoicing">0</a>
    <a n="ServicePrice#relatedRetailPrice_20000" b="Yourhome"
        t="price" currency="EUR" vf="2020-08-28T15:28:02.000">50</a>
    <a n="ServiceText" l="de">
        2 Monate Garantie... nicht mehr und nicht weniger</a>
  </d>
)XML#
```

B.2 Minimaler Belastungstest

ID: TC:G002

Deckt folgende Fälle ab: UC:G001

Beschreibung:

Es werden mehrere neue Kategorien mit mehreren neuen Produkten generiert. Der genaue Inhalt dieser Daten ist dabei nicht relevant. Der Datensatz wird dann bei einer existenten Kategorie eingehangen.

Instruktionen für die Anwendung:

```
    Anwendungsoption: Generieren

#XML:(
  <d id="Y_YH50111" type="N" keywords="#Existing:(true)">
    <r id="Y_Y1234" type="N" keywords="#Existing:(false)"/>
    <r id="Y_Y2345" type="N" keywords="#Existing:(false)"/>
    <r id="Y_Y3456" type="N" keywords="#Existing:(false)"/>
    <r id="Y_Y4567" type="N" keywords="#Existing:(false)"/>
    <r id="Y_Y5678" type="N" keywords="#Existing:(false)"/>
  </d>

  <d id="Y_Y1234" type="N" keywords="#Existing:(false)">
    <a n="CATEGORYNAME" l="de">Generierte Teppiche1</a>
  </d>
  <d id="Y_Y2345" type="N" keywords="#Existing:(false)">
    <a n="CATEGORYNAME" l="de">Generierte Teppiche2</a>
  </d>
  <d id="Y_Y3456" type="N" keywords="#Existing:(false)">
    <a n="CATEGORYNAME" l="de">Generierte Teppiche3</a>
  </d>
  <d id="Y_Y4567" type="N" keywords="#Existing:(false)">
    <a n="CATEGORYNAME" l="de">Generierte Teppiche4</a>
  </d>
  <d id="Y_Y5678" type="N" keywords="#Existing:(false)">
```

```
    <a n="CATEGORYNAME" l="de">Generierte Teppiche5</a>
  </d>

  <d id="" type="P" keywords="#Existing:(false)#Amount:(2)">
    <r id="" type="I" relationType="child"
        keywords="#Existing:(false)#Amount:(2)"/>
    <a n="CATEGORYPATH" b="Yourhome">Y_Y1234</a>
  </d>
  <d id="" type="P" keywords="#Existing:(false)#Amount:(2)">
    <r id="" type="I" relationType="child"
        keywords="#Existing:(false)#Amount:(2)"/>
    <a n="CATEGORYPATH" b="Yourhome">Y_Y2345</a>
  </d>
  <d id="" type="P" keywords="#Existing:(false)#Amount:(2)">
    <r id="" type="I" relationType="child"
        keywords="#Existing:(false)#Amount:(2)"/>
    <a n="CATEGORYPATH" b="Yourhome">Y_Y3456</a>
  </d>
  <d id="" type="P" keywords="#Existing:(false)#Amount:(2)">
    <r id="" type="I" relationType="child"
        keywords="#Existing:(false)#Amount:(2)"/>
    <a n="CATEGORYPATH" b="Yourhome">Y_Y4567</a>
  </d>
  <d id="" type="P" keywords="#Existing:(false)#Amount:(2)">
    <r id="" type="I" relationType="child"
        keywords="#Existing:(false)#Amount:(2)"/>
    <a n="CATEGORYPATH" b="Yourhome">Y_Y5678</a>
  </d>
)XML#
```

B.3 Existente Produkte erweitern

ID: TC:G003

Deckt folgende Fälle ab: UC:G003

Beschreibung:

Es soll ein neues Attribut zu bestimmten, bereits existierenden Produkten hinzugefügt werden.

Instruktionen für die Anwendung:

```
    Anwendungsoption: Generieren

#XML:(
  <d id="10043239948" type="P" keywords="#Existing:(true)">
    <a n="generiertesFeld" l="de" adref="A123456789"
        avref="AV123456789">generierter AV Begriff</a>
  </d>
  <d id="10058754169" type="P" keywords="#Existing:(true)">
    <a n="generiertesFeld" l="de" adref="A123456789"
        avref="AV123456789">generierter AV Begriff</a>
  </d>
  <d id="10072039934" type="P" keywords="#Existing:(true)">
    <a n="generiertesFeld" l="de" adref="A123456789"
        avref="AV123456789">generierter AV Begriff</a>
  </d>

  <d id="A123456789" type="AD">
    <a n="identifier">SpecialDescription</a>
    <a n="displayName" l="de">GenerierteBeschreibung</a>
    <a n="sequenceNo"/>
    <a n="attributeScope">ALL</a>
    <a n="attributeType">S</a>
    <a n="attributeGroup" adref="TT66" avref="AV158">IPIMCORE</a>
    <a n="attributeDataLevel">productLevel</a>
```

```
    <a n="ATTRIBUTION_USE_PRD_ATTR_FOR_ITEM">true</a>
    <a n="elastic_search_exclude_attribute">true</a>
    <a n="ishop_exclude_identifier">true</a>
    <a n="DC_IS_SOURCE_ATTR">true</a>
    <a n="ishop_export_name">generierterName</a>
  </d>

  <d id="AV123456789" type="AV">
    <a n="identifier">generierter AV Begriff</a>
    <a n="displayName" l="de">generierter AV Begriff</a>
    <a n="sequenceNo"></a>
  </d>
)XML#
```

B.4 Rudimentärer Minifizierungstest

ID: TC:M001

Deckt folgende Fälle ab: UC:M001

Beschreibung:

Es soll eine einfache reduzierte Menge an Produktdaten erzeugt werden, basierend auf den realen Produktdaten eines Shops.

Instruktionen für die Anwendung:

```
Anwendungsoption: Minifizieren

Produkte pro Kategorie: default (5)
Items pro Produkt: default (4)
Bewertungen pro Item: default (1)

Besondere Kategorien von Limitierung ausgeschlossen: keine
Spezifisch ausgewählte Produkte: keine
Spezifisch ausgewählte Items: keine
```

B.5 Test einer einzelnen befüllten Kategorie

ID: TC:M002

Deckt folgende Fälle ab: UC:M002

Beschreibung:

Es soll eine einfache reduzierte Produktdatenmenge erzeugt werden, basierend auf den realen Produktdaten eines Shops. Nur eine Kategorie soll ihre ursprünglichen Produkte und Items beinhalten. Alle nicht ausgewählten Kategorien sollen keine Produkte und Items beinhalten (Ausgenommen von dieser Regel sind alle Kategorien, in die die ausgewählte Kategorie eingehangen ist)

Instruktionen für die Anwendung:

```
Anwendungsoption: Minifizieren

Produkte pro Kategorie: default (5)
Items pro Produkt: default (4)
Bewertungen pro Item: default (1)

Folgende Kategorie als neue Wurzel betrachten: Y_YH30108104
Besondere Kategorien von Limitierung ausgeschlossen: Y_YH30108104
Spezifisch ausgewählte Produkte: keine
Spezifisch ausgewählte Items: keine
```

B.6 Bestimmte Produkte und Items auswählen

ID: TC:M003

Deckt folgende Fälle ab: UC:M003

Beschreibung:

Es sollen nur bestimmte Produkte und dazugehörige Items ausgewählt werden, die IDs dieser Produkte und Items ist bekannt. Diese Produkte und Items sind über mehrere Kategorien verteilt. Es sollen nur die ausgewählten Daten in der Ausgabedatei enthalten sein.

Instruktionen für die Anwendung:

```
Anwendungsoption: Minifizieren

Produkte pro Kategorie: 0
Items pro Produkt: 0
Bewertungen pro Item: 0

Besondere Kategorien von Limitierung ausgeschlossen: keine

Spezifisch ausgewählte Produkte:
10058754169, 601980162, 10016496989

Spezifisch ausgewählte Items:
10073446105, 602089506, 10018244032, 10018247414
```

B.7 Test einer von der Limitierung befreiten befüllten Kategorie

ID: TC:M004

Deckt folgende Fälle ab: UC:M002

Beschreibung:

Es soll eine einfache reduzierte Produktdatenmenge erzeugt werden, basierend auf den realen Produktdaten eines Shops. Nur eine Kategorie soll ihre ursprünglichen Produkte und Items beinhalten. Alle nicht ausgewählten Kategorien sollen wenige Produkte beinhalten (Ausgenommen von dieser Regel sind alle Kategorien, in die die ausgewählte Kategorie eingehangen ist)

Instruktionen für die Anwendung:

```
Anwendungsoption: Minifizieren

Produkte pro Kategorie: 1
Items pro Produkt: 1
Bewertungen pro Item: 0

Besondere Kategorien von Limitierung ausgeschlossen: Y_YH30108104
Spezifisch ausgewählte Produkte: keine
Spezifisch ausgewählte Items: keine
```

B.8 Neues Item und neue Bewertung

ID: TC:G010

Beschreibung:

Testfall um realistische Kombinationen abzudecken, ohne das es einen spezifischen Fall gibt. Es wird ein neues Item und eine passende Bewertung an ein existentes Produkt eingehangen.

Instruktionen für die Anwendung:

```
    Anwendungsoption: Generieren

#XML:(
  <d id="Y_YH50111" type="N" keywords="#Existing:(true)">
  </d>

  <d id="10066080754" type="P" keywords="#Existing:(true)">
    <r id="55541099008" type="I" relationType="child"
        keywords="#Existing:(false)"/>
  </d>

  <d id="55541099008" type="I" keywords="#Existing:(false)">
    <a n="name" adref="A1884685095">Gestreifte Fransen</a>
  </d>

  <d id="55541099009" type="B" keywords="#Existing:(false)">
    <a n="productId" adref="A273981979">1132167429</a>
    <a n="articleNumber" adref="A273981984">55541099008</a>
  </d>
)XML#
```

B.9 Neues Datenset mit AD-Knoten

ID: TC:G011

Beschreibung:

Testfall um realistische Kombinationen abzudecken, ohne das es einen spezifischen Anwendungsfall gibt. Es wird eine neue Kategorie, ein neues Produkt, ein neues Item, eine neue Bewertung an ein existentes Produkt eingehangen und ein neuer AD-Knoten (Attribut Definition) eingehangen.

Instruktionen für die Anwendung:

```
    Anwendungsoption: Generieren

#XML:(
  <d id="Y_YH50111" type="N" keywords="#Existing:(true)">
    <r id="Y_Y12345678" type="N" keywords="#Existing:(false)"/>
  </d>

  <d id="Y_Y12345678" type="N" keywords="#Existing:(false)">
    <a n="CATEGORYNAME" l="de">Generierte Teppiche</a>
  </d>

  <d id="55541099007" type="P" keywords="#Existing:(false)">
    <r id="55541099008" type="I" relationType="child"
        keywords="#Existing:(false)"/>
    <a n="CATEGORYPATH" b="Yourhome">Y_Y12345678</a>
    <a n="generiertesFeld" l="de" adref="A123456789"
        avref="AV13">(1) Sehr gut</a>
  </d>

  <d id="55541099008" type="I" keywords="#Existing:(false)">
    <a n="name" adref="A1884685095">Gestreifte Fransen</a>
  </d>

  <d id="55541099009" type="B" keywords="#Existing:(false)">
```

```
    <a n="productId" adref="A273981979">55541099007</a>
    <a n="articleNumber" adref="A273981984">55541099008</a>
  </d>

  <d id="A123456789" type="AD">
    <a n="identifier">SpecialDescription</a>
    <a n="displayName" l="de">GenerierteBeschreibung</a>
    <a n="sequenceNo"/>
    <a n="attributeScope">ALL</a>
    <a n="attributeType">S</a>
    <a n="attributeGroup" adref="TT66" avref="AV158">IPIMCORE</a>
    <a n="attributeDataLevel">productLevel</a>
    <a n="ATTRIBUTION_USE_PRD_ATTR_FOR_ITEM">true</a>
    <a n="elastic_search_exclude_attribute">true</a>
    <a n="ishop_exclude_identifier">true</a>
    <a n="DC_IS_SOURCE_ATTR">true</a>
    <a n="ishop_export_name">generierterName</a>
  </d>
)XML#
```

B.10 Neues Datenset mit AV-Knoten

ID: TC:G012

Beschreibung:

Testfall um realistische Kombinationen abzudecken, ohne das es einen spezifischen Fall gibt. Es wird eine neue Kategorie, ein neues Produkt, ein neues Item, eine neue Bewertung an ein existentes Produkt eingehangen und ein neuer AV-Knoten (Attribut Wert) eingehangen.

Instruktionen für die Anwendung:

```
    Anwendungsoption: Generieren

#XML:(
  <d id="Y_YH50111" type="N" keywords="#Existing:(true)">
    <r id="Y_Y12345678" type="N" keywords="#Existing:(false)"/>
  </d>

  <d id="Y_Y12345678" type="N" keywords="#Existing:(false)">
    <a n="CATEGORYNAME" l="de">Generierte Teppiche</a>
  </d>

  <d id="55541099007" type="P" keywords="#Existing:(false)">
    <r id="55541099008" type="I" relationType="child"
        keywords="#Existing:(false)"/>
    <a n="CATEGORYPATH" b="Yourhome">Y_Y12345678</a>
    <a n="generiertesFeld" l="de" adref="A20"
        avref="AV123456789">generierter AV Begriff</a>
  </d>

  <d id="55541099008" type="I" keywords="#Existing:(false)">
    <a n="name" adref="A1884685095">Gestreifte Fransen</a>
  </d>

  <d id="55541099009" type="B" keywords="#Existing:(false)">
```

```
    <a n="productId" adref="A273981979">55541099007</a>
    <a n="articleNumber" adref="A273981984">55541099008</a>
  </d>

  <d id="AV123456789" type="AV">
    <a n="identifier">generierter AV Begriff</a>
    <a n="displayName" l="de">generierter AV Begriff</a>
    <a n="sequenceNo"></a>
  </d>
)XML#
```

B.11 Neues Datenset mit O-Knoten

ID: TC:G013

Beschreibung:

Testfall um realistische Kombinationen Abzudecken, ohne das es einen spezifischen Anwendungsfall gibt. Es wird eine neue Kategorie, ein neues Produkt, ein neues Item, eine neue Bewertung an ein existentes Produkt eingehangen und ein neuer O-Knoten (Other Knoten, eine spezielle Kundenorientierte Funktionalität) eingehangen.

Instruktionen für die Anwendung:

```
    Anwendungsoption: Generieren

#XML:(
  <d id="Y_YH50111" type="N" keywords="#Existing:(true)">
    <r id="Y_Y12345678" type="N" keywords="#Existing:(false)"/>
  </d>

  <d id="Y_Y12345678" type="N" keywords="#Existing:(false)">
    <a n="CATEGORYNAME" l="de">Generierte Teppiche</a>
  </d>

  <d id="55541099007" type="P" keywords="#Existing:(false)">
    <r id="55541099008" type="I" relationType="child"
        keywords="#Existing:(false)"/>
    <a n="CATEGORYPATH" b="Yourhome">Y_Y12345678</a>
    <a n="generiertesFeld" l="de" adref="A4"
    avref="AV13">(1) Sehr gut</a>
  </d>

  <d id="55541099008" type="I" keywords="#Existing:(false)">
    <r id="8cd61123-334a-1234-8b36-c21c4eb12345" type="O"
        relationType="Yourhome_Service"/>
    <a n="name" adref="A1884685095">Gestreifte Fransen</a>
  </d>
```

```
  <d id="55541099009" type="B" keywords="#Existing:(false)">
    <a n="productId" adref="A273981979">55541099007</a>
    <a n="articleNumber" adref="A273981984">55541099008</a>
  </d>

  <d id="8cd61123-334a-1234-8b36-c21c4eb12345" type="O">
    <a n="name">2 Monate Garantie</a>
    <a n="dc_delivery24h" t="boolean">true</a>
    <a n="dc_desiredDate" t="boolean">true</a>
    <a n="ServiceGroup" l="de">2 Monate Garantie</a>
    <a n="oc_orderableBeforeInvoicing" t="boolean">true</a>
    <a n="serviceType">4</a>
    <a n="service_validFrom" b="Yourhome">28.08.2020</a>
    <a n="selectionType">1</a>
    <a n="oc_maximumOrderQuantity">1</a>
    <a n="oc_daysOrderableAfterInvoicing">0</a>
    <a n="ServicePrice#relatedRetailPrice_20000" b="Yourhome"
        t="price" currency="EUR" vf="2020-08-28T15:28:02.000">50</a>
    <a n="ServiceText" l="de">
        2 Monate Garantie... nicht mehr und nicht weniger</a>
  </d>
)XML#
```

Erklärung zur selbstständigen Bearbeitung einer Abschlussarbeit

Gemäß der Allgemeinen Prüfungs- und Studienordnung ist zusammen mit der Abschlussarbeit eine schriftliche Erklärung abzugeben, in der der Studierende bestätigt, dass die Abschlussarbeit „— bei einer Gruppenarbeit die entsprechend gekennzeichneten Teile der Arbeit [(§ 18 Abs. 1 APSO-TI-BM bzw. § 21 Abs. 1 APSO-INGI)] — ohne fremde Hilfe selbständig verfasst und nur die angegebenen Quellen und Hilfsmittel benutzt wurden. Wörtlich oder dem Sinn nach aus anderen Werken entnommene Stellen sind unter Angabe der Quellen kenntlich zu machen."

Quelle: § 16 Abs. 5 APSO-TI-BM bzw. § 15 Abs. 6 APSO-INGI

Erklärung zur selbstständigen Bearbeitung der Arbeit

Hiermit versichere ich,

Name: _____

Vorname: _____

dass ich die vorliegende Bachelorarbeit – bzw. bei einer Gruppenarbeit die entsprechend gekennzeichneten Teile der Arbeit – mit dem Thema:

Entwicklung eines automatisierten Testdaten-Generators für XML-basierte Produktdaten im E-Commerce

ohne fremde Hilfe selbständig verfasst und nur die angegebenen Quellen und Hilfsmittel benutzt habe. Wörtlich oder dem Sinn nach aus anderen Werken entnommene Stellen sind unter Angabe der Quellen kenntlich gemacht.

Ort	Datum	Unterschrift im Original