Projektarbeit 1

Entwicklung eines Testfallgenerators für verschiedene Eingangsformate in der Grundsteuer

Fakultät Technik

Studiengang Informatik / Angewandte Informatik

Im Rahmen der Prüfung zum Bachelor of Science

von

Ante Babic

Abgabedatum 29. September 2025

Sperrvermerk

Matrikelnummer:	5810399
Kurs:	TINF24B2
Bearbeitungszeitraum:	13 Wochen
Behörde:	Oberfinanzdirektion Baden-Württemberg
Abteilung:	Landeszentrum für Datenverarbeitung
Betrieblicher Betreuer:	Sven Binnig
Studiengangsleitung:	Prof. Dr. Sebastian Ritterbusch

Selbstständigkeitserklärung

Ich versichere hiermit, dass ich meine Projektarbeit mit dem Thema: „Entwicklung eines Testfallgenerators für verschiedene Eingangsformate in der Grundsteuer“ selbstständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel benutzt habe. Ich versichere zudem, dass die eingereichte elektronische Fassung mit der gedruckten Fassung übereinstimmt.

Karlsruhe, den 24. September 2025

Ort Datum

Unterschrift

Sperrvermerk

Der Inhalt dieser Arbeit darf weder als Ganzes noch in Auszügen Personen außerhalb des Prüfungs- und Evaluationsverfahrens zugänglich gemacht werden, sofern keine anders lautende Genehmigung des Dualen Partners vorliegt.

Inhaltsverzeichnis

Vorwort

Zur besseren Lesbarkeit sind häufig verwendete Abkürzungen und organisationsspezifische Bezeichnungen in kursiver Schrift dargestellt und im Abkürzungsverzeichnis aufgeführt.

Abbildungsverzeichnis

Abkürzungsverzeichnis

AE	Anwendungsentwicklung
LZfD	Landeszentrum für Datenverarbeitung
OFD BW	Oberfinanzdirektion Baden-Württemberg
XML	Extensible Markup Language
JAXB	Java Architecture for XML Binding
DOM	Document Object Model
SITiF	Sicherheitszentrum IT in der Finanzverwaltung Baden-Württemberg
Steuer-ID	Steuerliche Identifikationsnummer
CSV	Comma-Separated Values
DTO	Data Transfer Objects

1 Einleitung

Dieses Kapitel dient dem Einstieg in die vorliegende Projektarbeit. Es erläutert das fachliche sowie organisatorische Umfeld, die Motivation für die Themenwahl und beschreibt die Zielsetzung der Arbeit.

1.1 Betriebliches Umfeld

Die Oberfinanzdirektion Baden-Württemberg (OFD BW) ist die Landesmittelbehörde der Steuerverwaltung. Sie nimmt die Dienst- und Fachaufsicht über alle 65 Finanzämter in Baden-Württemberg wahr und koordiniert deren organisatorische und fachliche Ausrichtung. Zusätzlich ist sie für die Landesoberkasse zuständig sowie für die staatlichen Hochbauämter, die im Bereich des Bundesbaus tätig sind. Ein weiterer Aufgabenbereich umfasst die Steuerung von Organisation, Personal, Haushalt innerhalb der Steuerverwaltung. Die Behörde spielt außerdem eine wichtige Rolle bei der Weiterentwicklung und Digitalisierung von Prozessen, Standards und IT-Systemen in der Finanzverwaltung [Oberfinanzdirektion Baden-Württemberg, 2025a].

Das Landeszentrum für Datenverarbeitung (LZfD) ist eine Abteilung innerhalb der OFD BW und fungiert als zentraler IT-Dienstleister der Steuerverwaltung in Baden-Württemberg. Es betreibt und betreut eine Vielzahl von IT-Anwendungen, die in den Finanzämtern eingesetzt werden, etwa Systeme zur Steuerveranlagung oder zur elektronischen Kommunikation mit Bürgerinnen und Bürgern. Darüber hinaus unterstützt das LZfD Fachbereiche bei der Umsetzung neuer Softwarelösungen, stellt die technische Infrastruktur bereit und ist für Themen IT-Sicherheit und Datenpflege zuständig [Landeszentrum für Datenverarbeitung Baden-Württemberg, 2025]. Das LZfD ist in sieben verschiedene EDV Abteilungen aufgeteilt.

- EDV 1 Zentrale Dienste, Querschnittsaufgaben
- EDV 2 Anwendungsentwicklung (AE)
- EDV 3 Applikationsmanagement

- EDV 4 Systembetrieb und Basisdienste
- EDV 5 Service
- EDV 6 KONSENS, Architektur Projekte und Test
- EDV 7 Sicherheitszentrum IT in der Finanzverwaltung Baden-Württemberg (SITiF)

Die Abteilung EDV 2 der Oberfinanzdirektion Baden-Württemberg entwickelt und betreut IT-Anwendungen für die Finanzämter des Landes. Diese unterstützen die Verwaltung bei der korrekten Erfassung, Verarbeitung und Auswertung von Steuerdaten zur effizienten Erfüllung steuerlicher Aufgaben. Ein zentrales Projekt ist das länderübergreifende Vorhaben KONSENS, das eine einheitliche Softwarelösung zur Optimierung der Arbeitsprozesse in den Finanzämtern bereitstellt [Oberfinanzdirektion Baden-Württemberg, 2025c].

Die Unterabteilung EDV 211 entwickelt und pflegt die Software zur Bearbeitung der Grundsteuererklärungen nach den aktuellen gesetzlichen Vorgaben und sorgt für eine reibungslose Einführung der neuen Regelungen [Oberfinanzdirektion Baden-Württemberg, 2025b]. Zudem passt EDV 211 die Anwendungen konstant an geänderte rechtliche Anforderungen an und arbeitet eng mit anderen IT-Abteilungen, insbesondere im Bereich IT-Sicherheit, zusammen, um den Schutz sensibler Daten und die Systemstabilität zu gewährleisten.

1.2 Motivation

Mit der Einführung der neuen Grundsteuerregelung in Deutschland müssen zahlreiche Grundsteuererklärungen digital erfasst und verarbeitet werden. Die dafür eingesetzte Software muss zuverlässig arbeiten und in der Lage sein, eine große Bandbreite unterschiedlicher Eingabekombinationen korrekt zu verarbeiten. Die bisherige manuelle Erstellung von Testfällen ist in diesem Zusammenhang sehr zeitaufwendig und risikoanfällig, so dass wichtige Szenarien unberücksichtigt bleiben. Bereits geringe Abweichungen in der Berechnung können zu fehlerhaften Steuerbescheiden führen und damit zusätzlichen Aufwand für Verwaltung sowie Bürgerinnen und Bürgern verursachen. Ein automatisierter Testfallgenerator stellt hierfür eine effiziente Lösung dar, um die Software ausreichend zu prüfen und Fehler bei den Berechnungen frühzeitig zu erkennen. Er kann in kurzer Zeit eine Vielzahl von Testfällen erzeugen. Auf diese

Weise lassen sich Fehler frühzeitig erkennen, die Testabdeckung erhöhen und der gesamte Testprozess deutlich beschleunigen.

1.3 Zielsetzung der Arbeit

In dieser Projektarbeit wird eine Software entwickelt, die automatisch Testfälle erstellt um die korrekte Verarbeitung von Steuererklärungen zu überprüfen. Oft werden solche Testfälle noch manuell angelegt. Dies ist zeitaufwendig, erfordert spezielles Fachwissen und kann zu Fehlern führen. Die geplante Lösung soll diesen Prozess vereinfachen und beschleunigen. Die Software nutzt strukturierte Eingabedaten, zum Beispiel aus Extensible Markup Language (XML) oder Excel Dateien.[1] Diese enthalten alle wichtigen Informationen, erwartete Ausgaben und besondere Bedingungen. Das Programm liest die Daten ein, prüft sie auf Vollständigkeit und wandelt sie in ein einheitliches Format um. Dieses Format kann direkt in gängigen Testumgebungen verwendet werden, ohne dass zusätzliche Anpassungen nötig sind. Ein Schwerpunkt liegt auf der einfachen Bedienung. Die Anwendung soll auch von Personen genutzt werden können, die keine Programmierkenntnisse haben. Über definierte Schnittstellen, wie etwa eine API, lässt sich das Programm problemlos in bestehende Testsysteme einbinden und nahtlos in vorhandene Arbeitsabläufe integrieren.

Ziel ist es, den Aufwand für die Testfallerstellung deutlich zu verringern, den Testprozess effizienter zu gestalten und die Qualität der Steuerungssoftware langfristig zu sichern. Durch die Automatisierung werden Fehlerquellen reduziert, Entwicklungszeiten verkürzt und die Testergebnisse zuverlässiger.

[1] Im Folgenden wird häufig von „Excel Dateien“ gesprochen. Damit sind Dateien gemeint, die mit Microsoft Excel verarbeitet werden können, insbesondere die Formate .xlsx (Excel-Arbeitsmappen) sowie .csv (Comma-Separated Values). Diese Definition umfasst sowohl native Excel-Dateien als auch tabellarische Textdateien, die häufig für den Datenaustausch verwendet werden.

2 Grundlagen

In diesem Kapitel werden die notwendigen technischen Vorkenntnisse vermittelt, um ein besseres Verständis späterer Schritte zu ermöglichen.

2.1 Java

Java ist eine objektorientierte Programmiersprache, die auf der sogenannten Java Virtual Machine (JVM) ausgeführt wird [Rentrop, 2017a]. Dies ermöglicht es, Java-Programme plattformunabhängig zu nutzen, das heißt, sie laufen auf verschiedenen Betriebssystemen wie Windows, Linux oder macOS ohne Änderungen am Programmcode [Innowise Group, 2024]. Diese Plattformunabhängigkeit ist einer der Hauptvorteile von Java [Rentrop, 2017a].

Die Programmiersprache Java verfügt über eine umfangreiche Standardbibliothek, die viele wichtige Funktionen bereitstellt, beispielsweise für die Datenverarbeitung, die Netzwerkkommunikation oder die Benutzeroberflächenentwicklung [Ullenboom, 2015]. Java ist so aufgebaut, dass Programme in Klassen und Objekten organisiert werden. Dieses Prinzip der Objektorientierung erleichtert die Entwicklung von gut strukturierten, wartbaren und erweiterbaren Anwendungen. Durch Konzepte wie Vererbung und Schnittstellen können Programmteile wiederverwendet und flexibel gestaltet werden [Innowise Group, 2024]. Java bietet viele Möglichkeiten zur Verarbeitung verschiedener Datenformate wie XML und Excel [Innowise Group, 2024]. Dank der Plattformunabhängigkeit, der klaren Struktur und der großen Auswahl an Bibliotheken eignet sich Java besonders gut für die Entwicklung stabiler und flexibler Programme [Rentrop, 2017a] [Ullenboom, 2015]. Die aktive Entwickler-Community sorgt zudem dafür, dass Java regelmäßig weiterentwickelt und an neue Anforderungen angepasst wird [Innowise Group, 2024].

2.2 XML

XML ist eine textbasierte Sprache zur strukturierten Speicherung und zum Austausch von Daten zwischen verschiedenen Systemen [Ausbildung in der IT, 2025]. XML Daten werden in Elementen dargestellt, die durch Tags gekennzeichnet sind, zum Beispiel <name>...</name>. Durch diese Verschachtelung entsteht eine klare hierarchische Struktur [Microsoft Support, o. D.[b]]. Ein XML Dokument beginnt mit einer Deklaration, die Informationen zur XML Version und Zeichencodierung enthält [Microsoft Support, o. D.[b]]. XML Daten können sowohl in Elementen als auch in Attributen gespeichert werden. Elemente eignen sich gut für umfangreiche oder hierarchisch strukturierte Informationen, da sie selbst weitere Unterelemente enthalten können. Attribute hingegen speichern einfache, meist metadatenähnliche Informationen in Form von Schlüssel Wert Paaren direkt im öffnenden Tag, zum Beispiel <person alter="30»Max</person>. Ein Vorteil von Attributen ist, dass sie den Aufbau kompakter machen, jedoch können sie keine komplexen Datenstrukturen abbilden. Ein wichtiger Vorteil von XML ist seine Selbstbeschreibbarkeit. Die Tags geben Auskunft darüber, welche Daten enthalten sind, sodass Menschen und Maschinen sie gut verstehen können [Ausbildung in der IT, 2025].

XML wird von nahezu allen Programmiersprachen und vielen Anwendungen unterstützt, wodurch eine hohe Flexibilität entsteht. Dies ermöglicht den Einsatz in Bereichen wie Webservices, Konfigurationsdateien oder dem Dokumentenaustausch.

Aufgrund seiner strukturierten und standardisierten Form eignet sich XML besonders als Format für die Weiterverarbeitung und Speicherung der gesammelten Daten. Die klare Datenorganisation ermöglicht eine systematische Beschreibung und maschinelle Auswertung. Zusätzlich ist XML plattformunabhängig und unterstützt die Validierung durch definierte Schemata, was den sicheren Austausch und die langfristige Archivierung von Daten gewährleistet. Dadurch wird XML bevorzugt als „Speicher- und Austauschformat“ eingesetzt, das die zuverlässige Verwaltung und Übermittlung von Daten zwischen unterschiedlichen Systemen ermöglicht [World Wide Web Consortium (W3C), 2008].

2.3 Excel

Microsoft Excel ist ein Programm aus der Microsoft Office Reihe, das vor allem dazu genutzt wird, Daten übersichtlich in Tabellen zu erfassen, zu bearbeiten und auszuwerten. Es gehört zur Gruppe der Tabellenkalkulationsprogramme und findet breite Anwendung in Bereichen wie Wirtschaft, Wissenschaft oder auch im Alltag [Vogt, 2015]. Die Grundlage von Excel bildet das Arbeitsblatt, das aus vielen kleinen Zellen besteht. Diese sind in Spalten (alphabetisch) und Zeilen (nummerisch) organisiert, sodass jede Zelle eine eindeutige Adresse wie A1 erhält. In den Zellen lassen sich verschiedene Inhalte eintragen, etwa Zahlen, Texte oder Datumsangaben [Vogt, 2015].

Ein zentraler Vorteil von Excel ist die tabellarische Darstellung der Daten, die eine klare Struktur und eine einfache Orientierung ermöglicht. Darüber hinaus bietet Excel verschiedene Werkzeuge zur visuellen Aufbereitung und Analyse von Daten, wie zum Beispiel Filter, Diagramme, Pivot Tabellen oder bedingte Formatierungen. Diese helfen dabei, Muster, Auffälligkeiten oder Abweichungen in größeren Datenmengen schnell zu erkennen und gezielt auszuwerten [Microsoft Support, o. D.[a]].

Ein weiterer wichtiger Aspekt ist die Möglichkeit zur individuellen Zellformatierung. Excel unterscheidet verschiedene Datentypen wie Zahlen, Texte, Datumsangaben, Uhrzeiten, Prozentwerte oder Währungen [Vogt, 2015]. Diese Formate beeinflussen sowohl die Darstellung als auch das Verhalten der Daten, zum Beispiel beim Sortieren oder bei der Berechnung von Summen [Microsoft Support, o. D.[a]]. Excel erkennt viele dieser Typen automatisch, was den Arbeitsaufwand bei der Datenerfassung reduziert.

Excel Dateien werden häufig zur Datenerfassung eingesetzt, da die benutzerfreundliche Oberfläche eine intuitive Eingabe und Strukturierung ermöglicht, auch ohne tiefgehende technische Kenntnisse. Die breite Verfügbarkeit, die einfache Bedienbarkeit sowie die integrierten Auswertungs und Darstellungsmöglichkeiten machen Excel zu einem beliebten Werkzeug für die initiale Datensammlung. Es dient daher oft als „Eingabewerkzeug“, das eine direkte und flexible Interaktion mit den Daten erlaubt [Thomas H. Davenport and Jeanne G. Harris, 2007].

2.4 XML Verarbeitung in Java mit JAXB und DOM

Um XML Dateien mit Java zu verarbeiten, werden zwei Java Technologien, Java Architecture for XML Binding (JAXB) und Document Object Model (DOM) benutzt, diese werden im folgenden näher erläutert.

2.4.1 JAXB

JAXB ist eine Java Technologie, mit der sich XML Daten direkt in Java Objekte umwandeln lassen und umgekehrt [Horn, 2010]. Dieser Vorgang wird Marshalling genannt, wenn Java Objekte in XML geschrieben werden, und Unmarshalling, wenn XML in Java Objekte eingelesen wird. Dadurch können XML Daten strukturiert verarbeitet werden, ohne dass Entwickler selbst komplexe Parser schreiben müssen. Das spart Zeit und verringert den Programmieraufwand [Oracle Corporation, 2024]. Für die Umsetzung werden Java Klassen mit speziellen JAXB Annotationen versehen, zum Beispiel @XmlElement, @XmlAttribute oder @XmlRootElement. Diese legen fest, wie Felder und Methoden der Klasse mit den XML Elementen und Attributen verknüpft sind. Alternativ kann JAXB aus einer XML Schema Datei (XSD) automatisch passende Java Klassen erzeugen, was besonders bei umfangreichen oder komplexen Datenstrukturen hilfreich ist. Ein weiterer Vorteil ist, dass der Code übersichtlich und gut wartbar bleibt, da direkt mit Java Objekten gearbeitet wird [Horn, 2010]. Änderungen an der XML Struktur lassen sich durch Anpassungen an den Klassen oder am Schema schnell umsetzen [Oracle Corporation, 2024]. JAXB unterstützt außerdem Namespaces, komplexe Datentypen und Listen, sodass auch große und verschachtelte XML Dokumente verarbeitet werden können. Ein Nachteil von JAXB ist, dass es das gesamte XML Dokument im Speicher hält, was bei sehr großen Dateien zu einem erhöhten Speicherverbrauch führen kann. JAXB wird vor allem in Anwendungen eingesetzt, die regelmäßig XML Daten lesen oder schreiben, zum Beispiel bei Webservices, Konfigurationsdateien, Datenaustausch zwischen Systemen oder standardisierten Dokumentformaten [Rentrop, 2017b].

2.4.2 DOM

Das DOM ist ein standardisiertes Modell, mit dem XML oder HTML Dokumente als Baumstruktur dargestellt und bearbeitet werden [Ausbildung in der IT, 2025]. Über DOM können

einzelne Elemente gezielt gelesen, geändert, gelöscht oder ergänzt werden. Das gesamte Dokument wird dabei in den Arbeitsspeicher geladen und als Hierarchie aus Knoten dargestellt, wobei jedes Element, Attribut oder Textstück einen eigenen Knoten bildet [Wende, o. D.] So kann man gezielt auf bestimmte Teile des Dokuments zugreifen und diese bearbeiten. Ein Vorteil ist, dass Entwickler das Dokument wie ein Objektmodell behandeln können [Microsoft Support, o. D.[b]]. Über Programmierschnittstellen (APIs) lassen sich Knoten suchen, verschieben, kopieren oder neu anlegen. DOM ist sprachunabhängig und wird in vielen Programmiersprachen wie Java, JavaScript, Python oder C# unterstützt. DOM wird oft genutzt, wenn komplexe Dokumente verarbeitet werden müssen, zum Beispiel bei der Anpassung von Webseiten im Browser, beim Auslesen bestimmter Daten aus XML Dateien oder beim Erstellen neuer Dokumente aus vorhandenen Strukturen [Wende, o. D.] Es eignet sich besonders, wenn man den kompletten Überblick über die Struktur braucht oder an mehreren Stellen Änderungen vornehmen will. Ein Nachteil ist, dass immer das gesamte Dokument im Speicher gehalten wird. Bei sehr großen Dateien kann das viel Speicher verbrauchen und zu längeren Ladezeiten führen. In solchen Fällen nutzt man oft Alternativen wie SAX (Simple API for XML) oder StAX (Streaming API for XML), die das Dokument Schritt für Schritt verarbeiten und weniger Speicher benötigen [Ausbildung in der IT, 2025]. Trotzdem bleibt DOM wegen seiner klaren Struktur, der einfachen Navigation und der breiten Unterstützung ein wichtiges Werkzeug für die Arbeit mit XML und HTML Dokumenten [Wende, o. D.]

2.5 Excel Verarbeitung in Java mit Apache POI

Zur Verarbeitung von Excel-Dateien wurde die Java-Bibliothek Apache POI verwendet. Diese frei verfügbare Bibliothek ermöglicht es, Microsoft-Office-Dokumente, vor allem Excel Dateien im Format .xls und .xlsx direkt im Programm zu öffnen, auszulesen, zu bearbeiten und auch neue Dateien zu erstellen [Apache Software Foundation, 2025].

Mit Apache POI kann gezielt auf Inhalte von Tabellen zugegriffen, einzelne Zellen oder ganze Bereiche verändert und neue Arbeitsblätter angelegt werden Ergebnisse, die während der Ausführung entstehen, lassen sich problemlos wieder in eine Excel Datei schreiben, etwa zur Dokumentation oder für spätere Auswertungen [Apache Software Foundation, 2025][Baeldung, 2024]. Die Bibliothek besteht aus verschiedenen Teilen. HSSF (Horrible Spreadsheet Format) ist zuständig für ältere .xls Dateien, während XSSF (XML Spreadsheet Format) mit dem

neueren .xlsx Format arbeitet. Darüber hinaus bietet Apache POI auch Funktionen, mit denen man Inhalte formatieren, Formeln einfügen oder sogar Diagramme erstellen kann, ganz ohne Excel manuell zu öffnen [Baeldung, 2024]. Durch den Einsatz von Apache POI lässt sich die Arbeit mit Excel-Dateien vollständig automatisieren. Das spart Zeit, reduziert Fehler und macht Testabläufe einfacher und zuverlässiger. Außerdem können Excel-Daten direkt in Java-Anwendungen eingebunden werden, was die Entwicklung deutlich erleichtert [Apache Software Foundation, 2025; Baeldung, 2024].

3 Anforderungen

Dieses Kapitel behandelt die Ergebnisse der Anforderungserhebung, bestimmt die Anforderungen an den Testfallgenerator und stellt geeignete Lösungsansätze zur Umsetzung dieser Anforderungen vor.

3.1 Anforderungserhebung

Bevor mit der Entwicklung des Testfallgenerators begonnen wurde, mussten zunächst die Anforderungen festgelegt werden. Dabei wurde festgelegt, welche Funktionen der Generator später unterstützen soll. Außerdem wurden die erforderlichen Eingabe und Ausgabeformate bestimmt, um eine reibungslose Nutzung und Weiterverarbeitung der Testfalldaten zu gewährleisten. Ein wichtiger Punkt war auch die Struktur der Testfalldaten. Diese musste so gestaltet sein, dass sie von anderen Systemen oder Anwendern problemlos weiterverarbeitet werden kann. Zusätzlich wurde geprüft, wie die Testfälle später verwendet werden, um sicherzustellen, dass alle notwendigen Regeln berücksichtigt werden.

Diese Erkenntnisse bildeten die Grundlage für die technische Umsetzung des Testfallgenerators. Durch die klare Definition der Anforderungen konnte die Entwicklung zielgerichtet und effizient erfolgen.

3.2 Anforderungen an den Testfallgenerator

Der Testfallgenerator soll automatisch Testfälle aus strukturierten Eingaben wie XML oder Excel erzeugen können. Dabei muss er die Daten richtig einlesen, die Testfälle im gewünschten Format ausgeben und bestimmte Regeln bei der Erzeugung beachten. Das Tool soll außerdem so gebaut sein, dass es leicht angepasst oder erweitert kann, zum Beispiel für andere Datenformate oder neue Anforderungen. Die Eingabe- und Ausgabedatenstruktur soll klar aufgebaut und nachvollziehbar sein.

Die folgenden Anforderungen sind in funktionale und nicht funktionale Anforderungen aufgelistet.

Funktionale Anforderungen:

- Korrekter Datenimport der Eingabedaten

  Der Testfallgenerator muss in der Lage sein, Testfalldaten aus Excel beziehungsweise XML Dateien fehlerfrei zu importieren, wobei alle relevanten Ergebnisse korrekt übernommen werden

- Automatische Erzeugung von Testfällen basierend auf den eingelesenen Daten

  Das Programm soll automatisch strukturierte Testfälle generieren, sobald die Eingabedaten aus der Excel beziehungsweise XML Datei erfolgreich importiert wurden

- Unterstützung mehrerer Ausgabeformate

  Der Testfallgenerator soll die erzeugten Testfälle in verschiedenen Formaten exportieren können

Nicht Funktionale Anforderungen:

- Effiziente Verarbeitung großer Daten

  Der Testfallgenerator soll auch bei sehr umfangreichen Eingabedateien, eine zügige Verarbeitung gewährleisten, um die Benutzerfreundlichkeit nicht zu beeinträchtigen

- Zuverlässigkeit der Korrektheit der Testfälle

  Der Testfallgenerator soll sicherstellen, dass alle erzeugten Testfälle inhaltlich korrekt und vollständig sind, sodass sie ohne manuelle Nachkorrektur in Testsystemen verwendet werden können

- Wartbarkeit des Generators

  Der Quellcode des Generators soll so strukturiert, dokumentiert und modular aufgebaut sein, dass zukünftige Änderungen oder Erweiterungen mit minimalem Aufwand möglich sind

- Einfache Bedienung für die Anwender

  Der Testfallgenerator soll über klar dokumentierte und verständliche Schnittstellen oder Befehle verfügen, sodass Anwender ohne großen Aufwand Testfälle importieren und exportieren können

- Möglichkeit zur Erweiterbarkeit neuer Datenformate

  Der Testfallgenerator soll so gestaltet sein, dass künftig problemlos neue Eingabe und Ausgabeformate hinzugefügt werden können, ohne den bestehenden Code stark ändern zu müssen

In einem Projekt ist es wichtig, die verschiedenen Anforderungen nach ihrer Bedeutung zu ordnen, dafür existieren verschiedene Methoden zur Einteilung, diese werden im Folgenden näher erläutert.

Das Kano Modell unterscheidet Anforderungen in Basisfaktoren, Leistungsfaktoren und Begeisterungsfaktoren, um zu zeigen, wie ihre Erfüllung die Zufriedenheit von Nutzern beeinflusst. Es macht ersichtlich, welche Merkmale als selbstverständlich erwartet werden, welche direkt die Zufriedenheit steigern und welche einen überraschend positiven Effekt haben [Kano, Noriaki and Seraku, Nobuhiku and Takahashi, Fumio and Tsuji, Shinichi, 1984].

Die RICE Kategorisierung bildet einen nummerischen Score aus Reach × Impact × Confidence / Effort, sodass Anforderungen anhand von Reichweite, Wirkung, Vertrauenswürdigkeit der Schätzung und Aufwand vergleichbar werden. Die Methode erlaubt ein quantifiziertes Ranking, das hilft, viele Ideen systematisch zu ordnen [Melissa Perri, 2018].

Die WSJF Kategorisierung priorisiert nach dem Verhältnis Cost of Delay zur geschätzten Dauer, sodass Aufgaben mit dem höchsten wirtschaftlichen Nutzen pro Zeiteinheit zuerst bearbeitet werden. Die Methode fokussiert Time to Market und ökonomische Trade offs, indem sie Verzögerungskosten in Relation zur Aufwandsdauer setzt [Leffingwell, Dean, 2018].

Die MoSCoW Methode teilt Anforderungen in vier Kategorien ein, Must (Muss), Should (Soll), Could (Kann) und Won’t (wird nicht sein). Muss Anforderungen sind unverzichtbar, ohne sie kann das Projektziel nicht erreicht werden. Soll Anforderungen sind ebenfalls wichtig, tragen aber eher zur Verbesserung oder Optimierung bei. Kann Anforderungen sind "nice to have"Funktionen, die umgesetzt werden, wenn Zeit und Ressourcen verfügbar sind. Won’t

Anforderungen werden für das betrachtete Projekt bewusst ausgeschlossen [Agile Business Consortium, 2014].

Zur Kategorisierung der Folgenden Anforderungen wird die MoSCoW Methode verwendet, diese überzeugt durch ihre einfache Struktur und klare Priorisierung, wodurch sie sehr verständlich ist und eine einfache, klare Grundlage für die Entscheidungen schafft.

Die Anforderungen sind in den Kategorien Muss-, Soll- und Kann Anforderungen zugeordnet. Eine Übersicht bietet die folgende Tabelle.

Kategorie	**Beschreibung der Anforderung**
Muss	Korrekter Datenimport der eingegebenen Daten
Muss	Zuverlässigkeit der Korrektheit der Testfälle
Soll	Möglichkeit zur Erweiterbarkeit neuer Datenformate
Soll	Automatische Erzeugung von Testfällen basierend auf den eingelesenen Daten
Soll	Effiziente Verarbeitung großer Datenmengen
Soll	Wartbarkeit des Generators
Kann	Unterstützung mehrerer Ausgabeformate
Kann	Einfache Bedienung für die Anwender

Zu den **Muss Anforderungen** zählt der korrekte Import der eingegebenen Daten. Diese Funktion ist wesentlich, da ohne einen fehlerfreien und vollständigen Datenimport keine Testfälle erstellt werden können. Ebenso unverzichtbar ist die Gewährleistung der Korrektheit der Testfälle, da nur fehlerfreie Ergebnisse eine zuverlässige Überprüfung der Logik in der Steuererklärung Software ermöglichen. MoSCoW klassifiziert diese Anforderungen als Must da ohne diese der Testfallgenerator seine Funktion nicht erfüllen kann.

Zu den **Soll Anforderungen** gehört die Möglichkeit, neue Datenformate zu unterstützen, um die Software flexibel an zukünftige Anforderungen anpassen zu können, da diese Funktion für zukünftige Erweiterungen wichtig ist und für den Anfang nicht relevant ist. Auch die automatische Erzeugung von Testfällen auf Basis der eingelesenen Daten ist ein wesentlicher Bestandteil, der den Prozess deutlich beschleunigt, im Bedarfsfall jedoch auch manuell umgesetzt werden könnte. Die effiziente Verarbeitung großer Datenmengen trägt zu einer hohen Leistungsfähigkeit bei, ist jedoch nicht zwingend für die Grundfunktion erforderlich. Die

Wartbarkeit des Generators ist ebenfalls von Bedeutung, um spätere Anpassungen und Fehlerbehebungen mit geringem Aufwand vornehmen zu können. Die MoSCoW Methode ordnet diese Anforderungen als Should ein, da sie die Qualität und Erweiterbarkeit des Testfallgenerators deutlich erhöhen aber nicht zwingend für die Grundfunktionalität erforderlich sind.

Zu den **Kann Anforderungen** zählen die Unterstützung mehrerer Ausgabeformate, die den Anwendern zusätzliche Möglichkeiten bei der Weiterverarbeitung der Testergebnisse bietet, sowie eine besonders einfache Bedienung, die den Einstieg erleichtert und die Nutzererfahrung verbessert. MoSCoW klassifiziert diese Anforderungen als Could, da sie den Bedienkomfort und die Flexibilität des Testfallgenerators erhöhen, für die erste funktionsfähige Version des Testfallgenerators jedoch nicht zwingend erforderlich sind.

Damit ist die Priorisierung abgeschlossen und die Anforderungen sind klar nach Wichtigkeit geordnet. Die Einordnung bildet die Grundlage für die Planung und zeigt, welche Funktionen zuerst umgesetzt werden müssen und welche bei Bedarf verschoben werden könne. Prioritäten sollten regelmäßig übergeprüft und bei neuen Erkenntnissen oder geänderten Rahmenbedingungen angepasst werden.

4 Implementierung und Laufzeit

Dieser Abschnitt beschreibt das Einlesen und Verarbeiten von XML und Excel Dateien, die Logik zur Testfallgenerierung, die Wahl der Werkzeuge sowie beispielhafte Testfallausgaben und beschreibt die Laufzeit.

4.1 Lösungsansatz zur Umsetzung der Anforderungen

Um die gestellten Anforderungen effizient umzusetzen, wurde der Testfallgenerator in der Programmiersprache Java entwickelt. Das grundlegende Konzept besteht darin, unterschiedliche Eingabedatenformate, wie zum Beispiel XML oder Excel Dateien, einzulesen und diese anschließend intern weiterzuverarbeiten. Auf Basis dieser verarbeiteten Daten werden automatisch Testfälle generiert.

Zuerst werden die eingelesenen Daten in Java Objekte umgewandelt, damit sie im Programm übersichtlich und gut organisiert verarbeitet werden können. Danach werden die Testfälle nach festgelegten Regeln erstellt, sodass die Ergebnisse zuverlässig und nachvollziehbar sind. Zum Schluss werden die fertigen Testfälle in einem passenden Format ausgegeben, das für die weiteren Schritte oder die Testumgebung verwendet werden kann.

Besonders wichtig bei der Entwicklung des Testfallgenerators war der strukturierte Aufbau des Codes. Dieser wurde bewusst so gestaltet, dass er in Zukunft leicht erweitert und an neue Anforderungen angepasst werden kann. Zur Sicherstellung der Erweiterbarkeit basiert die Architektur auf klar definierten Reader und Writer Interfaces, sodass neue Eingabe und Ausgabeformate (z. B. CSV oder Datenbanken) als separate Implementierungen ergänzt werden können, ohne die Kernlogik des Testfallgenerators tiefgreifend zu verändern. Dadurch wird die Integration weiterer Formate oder die Anpassung an geänderte Datenstrukturen möglich, ohne bestehende Funktionen zu beeinträchtigen.

Ein zentrales Ziel der Lösung war es, eine einfache und zuverlässige Funktion sicherzustellen, die gleichzeitig flexibel genug ist, um auch künftigen Anforderungen gerecht zu werden. So

kann der Testfallgenerator langfristig und effizient in verschiedenen Anwendungsszenarien eingesetzt werden.

4.2 Implementierung

In diesen Abschnitt, wird die Implementierung des Testfallgenerators genauer definiert.

4.2.1 Einlesen und Verarbeiten von XML/Excel Dateien

Der Einlese und Verarbeitungsprozess von XML und Excel Dateien bildet die Grundlage für die Arbeit des Testfallgenerators. Beide Formate unterscheiden sich in ihrer Struktur, XML nutzt eine hierarische Baumdarstellung, während Excel tabellarisch aufgebaut ist. Im ersten Schritt werden die Dateien geöffnet und die relevanten Daten wie laufende Nummer, Eingabewerte und optionale Beschreibungen ausgelesen.

Bei XML erfolgt dies über die Navigation durch die Knoten, dazu erstellt man zunächst eine *DocumentBuilderFactory*, die den Parser konfiguriert. Mit einem *DocumentBuilder* wird die XML Datei eingelesen und ein DOM Baum im Speicher aufgebaut, der das gesamte Dokument abbildet. Über Methoden wie getDocumentElement(), lässt sich das Root Element auswählen, während getElementByTagName() gezielt bestimmte Elemente anspricht. Attribute können mit getAttribute() ausgelesen werden, und die Inhalte der Elemente lassen sich über ihre Unterelemente verarbeiten.

In Excel werden Daten in Tabellenform gespeichert, und bestehen aus mehreren Tabellenblättern die Sheet genannt werden, die jeweils Zeilen und Zellen enthalten. Ein Workbook stellt die ganze Datei dar und kann mit WorkbookFactory.create() geöffnet werden. Um die Daten auszulesen, durchläuft das Programm die einzelnden Tabellenblätter und liest die Daten aus den Zeilen und Zellen nacheinander. Dabei ist es wichtig die verschiedenen Zelltypen zu beachten, wie Text STRING, Zahlen NUMERIC oder Wahrheitswerte BOOLEAN. Apache POI bietet Methoden wie zum Beispiel getStringCellValue() um die Daten aus den einzelnden Zellen auszulesen. Beim auslesen von XML als auch Excel Dateien ist es wichtig auf die Fehlerbehandlung zu achten zum Beispiel durch die Nutzung von try-catch Anweisungen. Das Ziel ist es die gleichen Daten zu erfassen unabhängig vom Format.

Im nächsten Schritt werden die Daten in eine gemeinsame interne Struktur gebracht. Die eingelesenen Daten werden zur Laufzeit in Java Objekten gespeichert, die im Arbeitsspeicher (Heap) der Java Virtual Machine abgelegt sind, diese Java Objekte werden Data Transfer Objects (DTO) genannt. Diese sind einfache Java Klassen, die Daten strukturiert und ohne Logik kapseln. Erweitert wird der Abschnitt durch eine Beschreibung des internen Datenmodells und der Architektur. Eingelesene Werte werden in Java Klassen vom Typ Feld abgelegt, die Klasse Feld speichert die E Nummer[1] und den dazugehörigen Eingabewert. Mehrere Feldobjekte bilden ein Formular, zum Beispiel GW1,[1] GW2[1] oder GW4,[1] außerdem bilden mehrere Formulare zusammen eine Erklärung, diese enthält die Daten der Steuererklärung. Zur klaren Trennung der Verantwortlichkeiten existieren ein gemeinsames Reader Interface und ein gemeinsames Writer Interface. Konkrete Implementierungen XMLReader, ExcelReader, XMLWriter und ExcelWriter implementieren diese Schnittstellen und übernehmen jeweils das formatspezifische Parsen. Der Generator setzt die DTOs zu Testfällen zusammen und Writer geben die Testfälle in den gewünschten Formaten aus.

4.2.2 Ablauf zur Generierung der Testfälle

Bei der Generierung von Testfällen im Steuerumfeld geht es darum, aus vorhandenen Daten systematisch Testfälle zu erstellen. Die Daten sind in XML oder Excel Dateien vorhanden und enthalten zum Beispiel die Steuerliche Identifikationsnummer (Steuer-ID), Einkommenswerte oder den Familienstand. Damit die Testfälle einheitlich aufgebaut sind, bekommen sie eine feste Struktur mit einer laufenden Nummer, den Eingabewerten und dem erwarteten Ergebnis. Auf dieser Basis erzeugt der Testfallgenerator die Testfälle. Am Ende werden alle Testfälle geprüft, durchnummeriert und gespeichert. Dadurch entsteht eine einheitliche Sammlung, die für den Testverlauf genutzt werden kann.

4.2.3 Wahl und Begründung der Werkzeuge

Für die Verarbeitung der Daten wurde die Programmiersprache Java gewählt. Java bietet eine plattformunabhängige Umgebung sowie viele verfügbare Bibliotheken, die die Entwicklung erleichtern.

[1] Die Begriffe "E Nummer", "GW1", "GW2" und "GW3" werden im Kapitel 4.2.4 weiter Erläutert

Zur Verarbeitung von XML Dateien kam die DOM API zum Einsatz. Die DOM API bildet das XML Dokument als Baumstruktur im Arbeitsspeicher ab, wodurch einzelne Elemente und Attribute einfach zugänglich und veränderbar sind. Dies ist besonders hilfreich, wenn umfangreiche Manipulationen oder mehrfache Zugriffe auf verschiedene Teile der XML Struktur nötig sind. Alternativ wären SAX oder StAX möglich gewesen, die eher eventbasiert arbeiten und weniger Speicher benötigen. Allerdings sind diese APIs eher für sequentielle Lesevorgänge geeignet und weniger intuitiv bei komplexeren Bearbeitungen. Daher wurde die DOM API bevorzugt, da sie eine einfachere und übersichtlichere Handhabung für diesen Anwendungsfall bietet. Zudem ist die DOM API direkt in Java integriert, sodass keine externen Abhängigkeiten erforderlich sind. Zusätzlich ermöglicht es eine flexible Bearbeitung der XML Daten und erleichtert spätere Anpassungen oder Erweiterungen der XML Struktur.

Für die Verarbeitung von Excel Dateien wurde die Bibliothek Apache POI verwendet. Apache POI unterstützt umfassend das Lesen und Schreiben von Excel Dateien in verschiedenen Formaten (.xls und .xlsx). Die Bibliothek bietet eine Vielzahl von Methoden, um Tabellenblätter, Zeilen und Zellen zu lesen oder zu bearbeiten. Im Vergleich zu Alternativen wie JExcelAPI oder OpenCSV ist Apache POI aktueller, unterstützt mehr Exce Formate und wird von einer großen Entwicklergemeinschaft gepflegt. Dadurch ist die Integration in Java Projekte einfach und die langfristige Wartbarkeit gesichert.

Bibliotheken mit guter Dokumentation und aktiver Entwicklergemeinschaft unterstützen die Wartbarkeit, da Fehler schnell behoben und Anpassungen leicht umgesetzt werden können. Durch den modularen Aufbau und der klaren Trennung zwischen Einlesen, Datenmodellierung und Testfallgenerierung ergibt sich eine flexible Lösung, wodurch Erweiterungen ohne großen Aufwand möglich sind. Durch die Kombination dieser Technologien entstand eine leistungsfähige, flexible und wartbare Lösung, die Daten aus beiden Formaten zuverlässig auslesen, in Java Objekte überführen und für die Testfallgenerierung verwenden kann.

4.2.4 Beispielhafte Testfallausgaben

Zur besseren Verständlichkeit der Funktionsweise des Testfallgenerators wird im Folgenden der Aufbau der Grundsteuererklärung erläutert. Diese gliedert sich in drei Formulare: GW1, GW2 und GW4.

Das Formular GW1 dient als Hauptvordruck für die Grundsteuererklärung. Es erfasst grundlegende Informationen zur wirtschaftlichen Einheit, einschließlich Angaben zum Eigentümer, zur Lage des Grundstücks, zum Aktenzeichen sowie zum Grund der Feststellung. Dieses Formular bildet die Basis für die weiteren Angaben und ist für alle Grundstücke erforderlich [ELSTER – Elektronische Steuererklärung Baden-Württemberg, 2025].

Die Anlage GW2 enthält spezifische Daten zum Grundstück selbst. Hier werden Informationen wie die Gemarkung, die Flurstücksnummer, die Grundstücksfläche, der Bodenrichtwert sowie die Nutzung des Grundstücks abgefragt. Bei Miteigentum sind zudem die jeweiligen Eigentumsanteile anzugeben. Dieses Formular ist notwendig, wenn es sich nicht um ein land- und forstwirtschaftliches Grundstück handelt [ELSTER – Elektronische Steuererklärung Baden-Württemberg, 2025].

Das Formular GW4 wird verwendet, wenn das Grundstück ganz oder teilweise von der Grundsteuer befreit oder eine Ermäßigung der Steuermesszahl in Anspruch genommen wird. Hier sind die Art der Befreiung oder Vergünstigung sowie die entsprechenden gesetzlichen Grundlagen anzugeben. Dieses Formular ist insbesondere relevant für Grundstücke, die beispielsweise gemeinnützigen Zwecken dienen oder unter Denkmalschutz stehen [ELSTER – Elektronische Steuererklärung Baden-Württemberg, 2025].

Die eingegebenen Daten in der Grundsteuererklärung werden elektronisch gespeichert. Dabei sind die einzelnen Eingabefelder jeweils mit sogenannten E Nummern versehen. Diese E Nummern dienen der internen Verarbeitung und Zuordnung der Daten durch die Finanzbehörden.

E Nummern sind interne Kennzeichnungen für Eingabefelder in den elektronischen Grundsteuerformularen, die über das ELSTER Portal eingereicht werden. Sie erleichtern den Finanzämtern die strukturierte Verarbeitung und Zuordnung der Daten. Jede E Nummer wird mit einem E und einer 7 stelligen Zahl definiert zum Beispiel "E1234567". Informationsfelder wie Laufendenummer, Beschreibung oder ähnliche besitzen keine Feldkennnummer und bekommen ihren Namen als eindeutige Feldkennung. Die Feldkennnummer wird innerhalb der Datei mit "nr" und der Eingabewert mit "wert" abgekürzt. Für Bürgerinnen und Bürger sind diese Nummern meist unsichtbar und richten sich hauptsächlich an Softwareentwickler, Steuerberater und Finanzbeamte [Ministerium der Finanzen Baden-Württemberg, 2025].

eine beispielhafte Datei der erzeugten Testfälle gezeigt. Diese verdeutlicht den Aufbau der Testfälle. Jeder Testfall hat dabei eine eindeutige laufende Nummer, eine kurze Beschreibung und die dazugehörigen Eingabedaten.

Zur Veranschaulichung beispielhafter Testfallausgaben wurde XML verwendet, um den Aufbau und die Struktur darzustellen. Hierfür wurde das Formular GW2 als Teil einer Erklärung am folgenden Beispiel gezeigt.

```
<DatenTeil>
<Erklaerung LfdNr="4">
<GW2>
<Feld nr="E7421322" wert="1"/>
<Feld nr="E7403010" wert="71"/>
<Feld nr="E7403011" wert="730"/>
</GW2>
</Erklaerung>
</DatenTeil>
```

Quelltext 4.1: Beispielhafte Testfallausgaben in XML

4.3 Laufzeit

In diesem Abschnitt wird die Laufzeit der Konvertierung von einer Excel Datei in eine XML Datei des Testfallgenerators untersucht.

Das System, auf dem die Messungen durchgeführt wurden, besteht aus einem Intel(R) Core(TM) i7 10750H (2,60 GHz), 32 GB RAM, einer NVIDIA Quadro T1000 (4 GB) sowie integrierter Intel UHD Graphics (128 MB). Das Betriebssystem leuft unter Windows 10 Enterprise und verwendet zwei SSDs (Toshiba KXG6AZNV512G 477 GB, Samsung MZVLB1T0HBLR 000L7 954 GB), wobei die Messläufe lokal auf einer SSD ausgeführt wurden.

Für jede Dateigröße (50 KB, 100 KB, 150 KB, 200 KB, 250 KB) wurde die Konvertierung von Excel nach XML fünfmal ausgeführt. Der arithmetische Mittelwert dieser Daten bildet die Messgröße der Gesamtverarbeitungszeit vom Programmstart bis zur fertigen XML Datei pro Größe. Die Läufe wurden lokal auf einer SSD unter ruhenden Systembedingungen durchgeführt.

Ergebnisse der Messwerte

- 50 KB in 937 ms $\rightarrow \approx 50.36$ KB/s
- 100 KB in 1167 ms $\rightarrow \approx 85.69$ KB/s
- 150 KB in 1456 ms $\rightarrow \approx 103.05$ KB/s

- 200 KB in 1771 ms $\rightarrow \approx 112.87$ KB/s
- 250 KB in 1985 ms $\rightarrow \approx 125.94$ KB/s

Die Messgrößen zeigen, dass die Laufzeit mit der Dateigröße linear wächst und sich durch die Formel T(n) = k·n + c beschreiben lässt, wobei n die Dateigröße, c ein fester Startaufwand und k die Zeit pro Daten Einheit bezeichnet. Damit ist die asymptotische Laufzeit linear, also $T(n) \in$ O(n).

Die folgende Grafik zeigt die Messwerte und veranschaulicht das Laufzeitverhalten.

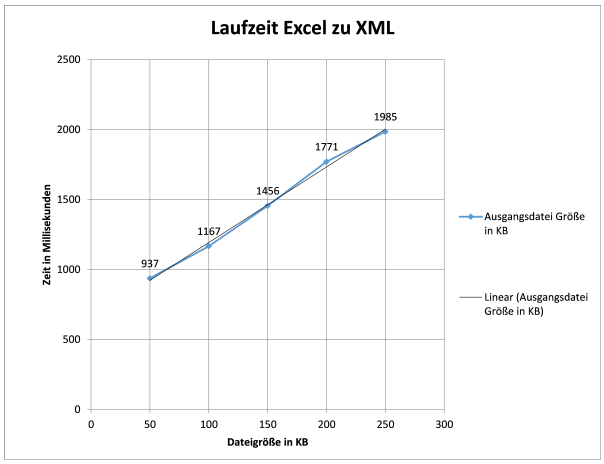

Abbildung 4.1: Laufzeit Analyse Excel zu XML

5 Fazit und Ausblick

Dieses Kapitel fasst die Ergebnisse der Arbeit zusammen und zeigt, welche Möglichkeiten sich für die Weiterentwicklung des Generators anbieten.

5.1 Zusammenfassung der Arbeit

In dieser Arbeit wurde ein Testfallgenerator für steuerliche Anwendungen entwickelt. Ziel war es, ein Programm zu entwickeln, das automatisch Testfälle aus vorhandenen XML und Excel Dateien erzeugt. Damit sollte die Testfallerstellung schneller, einfacher und weniger fehleranfällig werden, da manuelle Eingaben oft aufwendig sind und zu Fehlern führen können.

Am Anfang der Arbeit wurden die Anforderungen an den Generator festgelegt. Dazu gehörte, dass die Daten aus den Dateien korrekt übernommen werden, dass die erzeugten Testfälle immer gleich aufgebaut sind und dass auch Sonderfälle wie fehlende Eingaben oder falsche Werte berücksichtigt werden. Die Umsetzung erfolgte mit Java. Für die Verarbeitung von XML Dateien wurden die Werkzeuge JAXB und DOM eingesetzt, während Excel Dateien mit der Bibliothek Apache POI gelesen wurden. Alle Testfälle werden in einer einheitlichen Struktur gespeichert, sodass sie später problemlos für Tests genutzt werden können.

Die Logik des Generators ist klar aufgebaut. Zuerst werden die Dateien eingelesen und überprüft. Danach werden daraus Testfälle erstellt. Diese werden durchnummeriert und in einem passenden Format gespeichert. Auf diese Weise entstehen viele Testfälle in kurzer Zeit, die für verschiedene Szenarien genutzt werden können.

Der Testfallgenerator erfüllt die wesentlichen Anforderungen und unterstützt die schnellere sowie fehlerfreie Erstellung von Testfällen. Die automatische Verarbeitung von XML und Excel Dateien spart Zeit und erleichtert die Arbeit.

Trotzdem gibt es auch Schwächen. Der Testfallgenerator kann keine fehlerhaften Testfälle erkennen oder gezielt erstellen. Das kann ein Nachteil sein, wenn man testen will, wie ein System auf falsche Eingaben reagiert. Außerdem unterstützt der Testfallgenerator nur XML und Excel, andere Formate wie CSV oder ähnliche fehlen. Auch bei sehr großen Dateien

könnte die Leistung besser und effizienter sein. Die Bedienbarkeit des Testfallgeneratos ist noch ausbaufähig, da die Benutzerführung teilweise unübersichtlich ist und dadurch die Bedienung erschwert wird.

Insgesamt ist der Testfallgenerator eine gute Unterstützung, doch vor allem in der Fehlerbehandlung und der Flexibilität, gibt es noch Verbesserungsmöglichkeiten.

5.2 Ideen für Weiterentwicklung

Der Testfallgenerator erfüllt die in dieser Arbeit gesetzten Anforderungen, dennoch gibt es verschiedene Möglichkeiten, das System in Zukunft zu erweitern und an neue Anforderungen anzupassen. Solche Weiterentwicklungen sind sinnvoll, um den praktischen Nutzen des Generators zu erhöhen und den Einsatzbereich zu vergrößern.

Ein erster Ansatz wäre die Unterstützung weiterer Eingabeformate. Derzeit verarbeitet der Generator XML und Excel Dateien. In vielen Unternehmen werden jedoch auch andere Formate wie Comma-Separated Values (CSV) Dateien oder Datenbanken genutzt. Wenn der Generator diese Quellen ebenfalls einlesen könnte, würde er deutlich flexibler einsetzbar sein. Auch eine Kombination verschiedener Datenquellen wäre denkbar, um Testfälle aus unterschiedlichen Systemen zusammenzuführen.

Ein weiterer wichtiger Punkt ist die Anbindung an automatisierte Testumgebungen. Aktuell erzeugt der Generator die Testfälle, die anschließend manuell in Testsysteme eingebunden werden müssen. Würde eine direkte Schnittstelle zu gängigen Test Frameworks bestehen, könnten die erzeugten Testfälle automatisch ausgeführt werden. Dadurch lässt sich der gesamte Testprozess beschleunigen und effizienter gestalten. Darüber hinaus könnte die Bedienbarkeit des Generators verbessert werden. Bislang läuft die Steuerung hauptsächlich über den Code. Eine grafische Benutzeroberfläche würde den Umgang deutlich vereinfachen, da auch Personen ohne tiefere Programmierkenntnisse den Generator nutzen könnten. Eine solche Oberfläche könnte zum Beispiel das Auswählen der Eingabedateien, das Festlegen von Parametern oder das Anzeigen der generierten Testfälle erleichtern. Auch die Logik zur Erzeugung von Sonder und Fehlerfällen könnte weiter ausgebaut werden. Momentan werden grundlegende Fälle berücksichtigt, wie fehlende Eingaben oder falsche Werte. In Zukunft wäre es möglich, komplexere Szenarien zu erzeugen, die realistische Fehlerbilder besser abbilden. So könnten

beispielsweise Abhängigkeiten zwischen Eingabefeldern geprüft oder seltene Spezialfälle berücksichtigt werden. Dies würde die Qualität der erzeugten Tests weiter erhöhen. Zusätzlich ließe sich überlegen, den Generator modular zu gestalten. So könnten einzelne Teile wie die Verarbeitung bestimmter Datenformate oder die Logik zur Testfallerstellung leicht ausgetauscht oder erweitert werden. Damit wäre der Generator langfristig besser anpassbar und könnte schrittweise an neue Anforderungen angepasst werden.

Insgesamt bieten sich damit mehrere sinnvolle Ansätze für die Weiterentwicklung. Die Unterstützung weiterer Formate, die Anbindung an automatisierte Tests, eine bessere Bedienbarkeit und ein Ausbau der Testlogik. Mit diesen Erweiterungen könnte der Generator nicht nur breiter eingesetzt werden, sondern auch einen noch größeren Beitrag zur Qualitätssicherung leisten.

Anhang

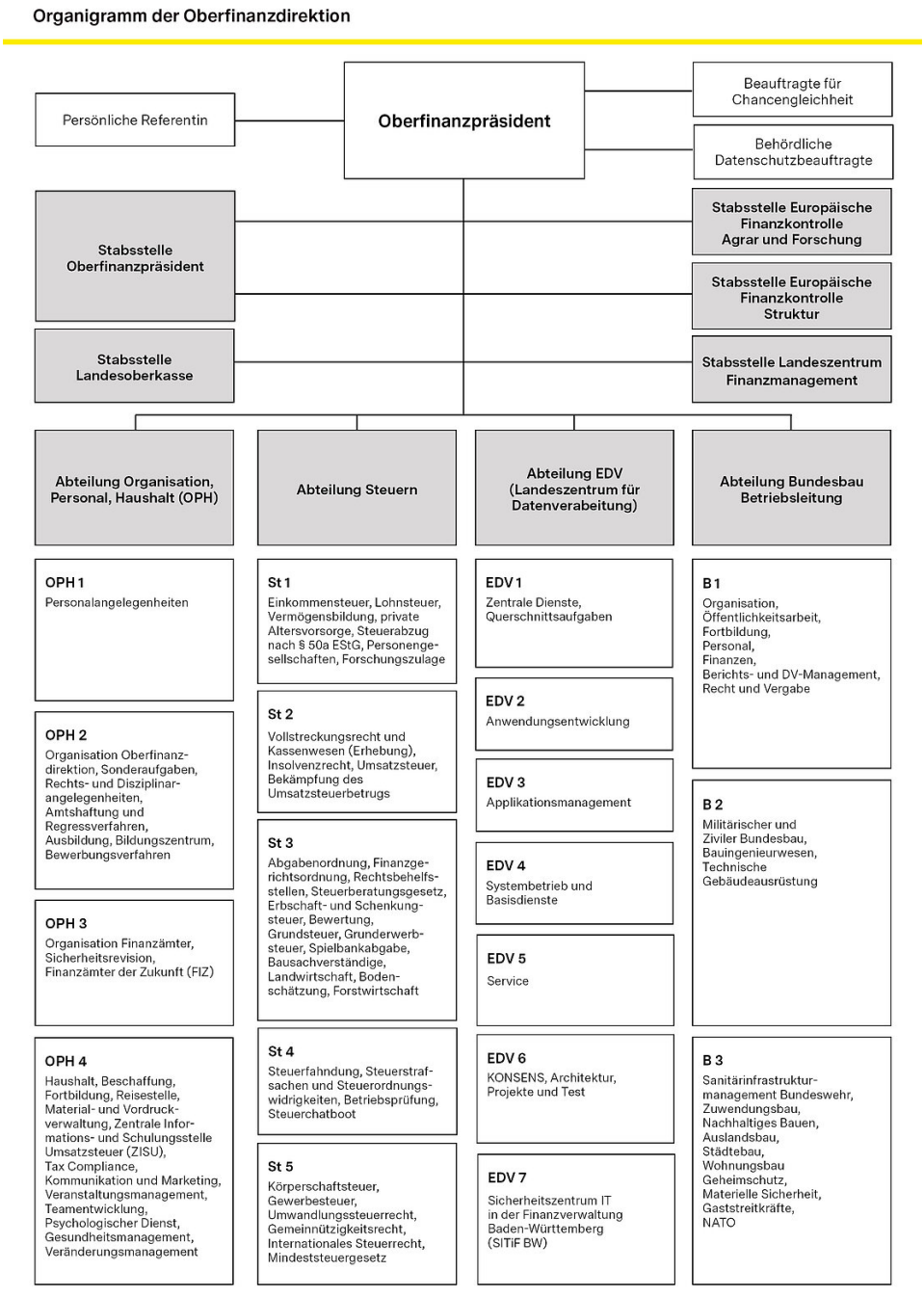

Abbildung .1: Oberfinanzdirektion Baden-Württemberg Organigramm

Literaturverzeichnis

Agile Business Consortium [2014]. *DSDM Agile Project Framework*. Agile Business Consortium.

Apache Software Foundation [2025]. *Apache POI - Java API for Microsoft Documents*. Zugriff am 3. September 2025. URL: https://poi.apache.org/.

Ausbildung in der IT [2025]. *Einfach erklärt: XML*. Abgerufen am 03.09.2025. URL: https://ausbildung-in-der-it.de/lexikon/xml.

Baeldung [2024]. *Working with Microsoft Excel in Java using Apache POI*. Abgerufen am 03.09.2025. URL: https://www.baeldung.com/java-microsoft-excel.

ELSTER – Elektronische Steuererklärung Baden-Württemberg [2025]. *Grundsteuerformulare GW1, GW2, GW4 – Erläuterungen und Ausfüllhilfe*. Abgerufen am 11.10.2025. URL: https://www.elster.de/eportal/start.

Horn, Torsten [2010]. *JAXB zur XML-Verarbeitung mit Java*. Abgerufen am 03.09.2025. URL: https://www.torsten-horn.de/techdocs/java-xml-jaxb.htm.

Innowise Group [2024]. *Vor- und Nachteile der Java-Entwicklung*. Abgerufen am 03.09.2025. URL: https://innowise.com/de/blog/benefits-and-drawbacks-of-java/.

Kano, Noriaki and Seraku, Nobuhiku and Takahashi, Fumio and Tsuji, Shinichi [1984]. „Attractive Quality and Must-Be Quality“. In: *The Journal of the Japanese Society for Quality Control*.

Landeszentrum für Datenverarbeitung Baden-Württemberg [2025]. *IT-Dienstleistungen für die Steuerverwaltung Baden-Württemberg*. Abgerufen am 04.09.2025. URL: https://www.lzfd-bw.de/.

Leffingwell, Dean [2018]. *SAFe 4.5 Reference Guide: Scaled Agile Framework for Lean Enterprises*. Addison-Wesley Professional.

Melissa Perri [2018]. *Escaping the Build Trap: How Effective Product Management Creates Real Value*. O'Reilly Media.

Microsoft Support [o. D.[a]]. *Erstellen eines PivotCharts*. Abgerufen am 03.09.2025. URL: https://support.microsoft.com/de-de/office/erstellen-eines-pivotcharts-c1b1e057-6990-4c38-b52b-8255538e7b1c.

Microsoft Support [o. D.[b]]. *XML für Anfänger*. Abgerufen am 03.09.2025. URL: https://support.microsoft.com/de-de/office/xml-f%C3%BCr-anf%C3%A4nger-a87d234d-4c2e-4409-9cbc-45e4eb857d44.

Ministerium der Finanzen Baden-Württemberg [2025]. *Grundsteuerreform in Baden-Württemberg – Elektronische Grundsteuererklärung*. Abgerufen am 11.10.2025. URL: https://fm.baden-wuerttemberg.de/de/steuern/grundsteuer/.

Oberfinanzdirektion Baden-Württemberg [2025a]. *Die Oberfinanzdirektion Baden-Württemberg – Aufgaben und Organisation*. Abgerufen am 04.09.2025. URL: https://www.ofd-bw.de/.

Oberfinanzdirektion Baden-Württemberg [2025b]. *Grundsteuerreform 2025 – Umsetzung durch EDV 211*. Abgerufen am 04.09.2025. URL: https://www.ofd-bw.de/grundsteuerneu.

Oberfinanzdirektion Baden-Württemberg [2025c]. *Projekt KONSENS – Einheitliche Softwarelösung für Finanzämter*. Abgerufen am 04.09.2025. URL: https://www.ofd-bw.de/konsens.

Oracle Corporation [2024]. *Java Architecture for XML Binding (JAXB)*. Zugriff am 3. September 2025. URL: https://docs.oracle.com/javase/tutorial/jaxb/.

Rentrop, Christian [2017a]. *Programmieren mit Java: Plattformunabhängig Entwickeln*. Abgerufen am 03.09.2025. URL: https://www.datacenter-insider.de/java-plattformunabhaengige-anwendungsentwicklung-a-381e431b5364e32f1e32c9d9a8c65bf3/.

Rentrop, Christian [2017b]. *Programmieren mit Java: Plattformunabhängig Entwickeln*. Zugriff am 3. September 2025. URL: https://www.datacenter-insider.de/java-plattformunabhaengige-anwendungsentwicklung-a-381e431b5364e32f1e32c9d9a8c65bf3/.

Thomas H. Davenport and Jeanne G. Harris [2007]. *Competing on Analytics*. Beschreibt Excel als Standard-Tool für Datenaufbereitung und -eingabe in Unternehmen. Harvard Business School Press.

Ullenboom, Christian [2015]. *Java SE 8 Standard-Bibliothek*. Online-Version. Rheinwerk Verlag. URL: https://openbook.rheinwerk-verlag.de/java8/.

Vogt, Dominik [2015]. *Excel-Grundlagen – Kursunterlagen*. Abgerufen am 03.09.2025. URL: https://userpage.fu-berlin.de/~vodo/excelkurs/Folien%20-%20Excel-Grundlagen.pdf.

Wende, Matthäus [o. D.] *XML: Definition, Struktur und Anwendungsbereiche*. Abgerufen am 03.09.2025. URL: https://www.matthaeus-wende.de/lexikon/artikel/xml/.

World Wide Web Consortium (W3C) [2008]. *Extensible Markup Language (XML) 1.0 (Fifth Edition)*. Offizielle Spezifikation von XML als plattformunabhängiges, strukturiertes Format für den Datenaustausch. URL: https://www.w3.org/TR/xml/.