Eine Spezifikation. Viele Artefakte.

ELSTER UStVA als Fallbeispiel für Specification Drift - und eine MORe®-Strategie zur Vermeidung

Nicht die Dokumentation ist die Spezifikation. Die Spezifikation ist das führende Artefakt, aus dem erzeugte Dokumentation, XSD, Validatoren, Tests und Implementierungsparameter konsistent hervorgehen. Die Implementierung wird daran entwickelt und nachweisbar geprüft.

Eine XML kann fachlich plausible Werte enthalten, im praktischen Übermittlungsweg akzeptiert werden und dennoch gegen ein veröffentlichtes XSD scheitern. Das Problem ist dann nicht XML. Das Problem ist eine Informationsarchitektur, in der Regeln, Formate, Dokumentation, Implementierung und Tests nicht nachweisbar aus derselben fachlichen Quelle entstehen.

Autor: Reiner Schindler
Organisation: MORe® Framework / projektkram
Version: v1.2 · August 2026
Fallstudie: ELSTER Umsatzsteuer-Voranmeldung 2026

Abgrenzung: Das Whitepaper bewertet nicht die ELSTER-Gesamtarchitektur. Es analysiert einen konkreten Integrationsfall anhand der veröffentlichten 2026er XSDs, der Jahresdokumentation und beobachteten Laufzeitverhaltens. Wo die Ursache einer Abweichung nicht aus den Artefakten belegbar ist, wird sie ausdrücklich offengelassen.

1. Management Summary

Der untersuchte UStVA-Fall zeigt auf kleinem Raum ein Problem, das in großen IT- und Transformationsprojekten regelmäßig auftritt: Die fachliche Wahrheit wird nicht an einer Stelle geführt, sondern in mehreren Artefakten reproduziert. Sobald diese Artefakte unabhängig gepflegt oder unterschiedlich interpretiert werden, entsteht Drift.

- Die 2026er UStVA-XSD definiert Struktur, Datentypen, Optionalität und eine feste Elementreihenfolge.
- Die Jahresdokumentation enthält zusätzliche fachliche Regeln, unter anderem gemeinsame Kennzahlenpaare und Nullwert-Hinweise.
- Kz10 ist als Kennzeichen für die berichtigte Anmeldung modelliert, optional und mit dem Wert 1 belegt.
- Im betrachteten Integrationsfall akzeptierte ELSTER über Monate eine XML-Reihenfolge, die eine lokale XSD-Validierung mit xmllint wegen xs:sequence zurückweist.
- Leere Werte und explizite Nullwerte sind fachlich nicht dasselbe. Wer beide zu 0 normalisiert, zerstört Information, bevor die fachliche Plausibilisierung überhaupt beginnt.

Das Kernproblem ist nicht, dass es mehrere Spezifikationen gibt. Das Kernproblem ist, wenn mehrere erzeugte Repräsentationen der Spezifikation so behandelt werden, als wären sie jeweils die Spezifikation.

MORe setzt genau an dieser Stelle an. Die fachliche Spezifikation wird als führendes Informationsmodell und damit als **maßgebliches Artefakt** geführt. XSD, Feldkatalog, dokumentierte Plausibilitätsregeln, Validatoren, Testfälle und Implementierungsparameter sind daraus erzeugte Repräsentationen der Spezifikation. Sie dürfen nicht unabhängig voneinander zu neuen Wahrheiten werden.

Für die Organisation bedeutet das einen Wechsel von dokumentenzentriertem Anforderungsmanagement zu einer modellzentrierten Informationsarchitektur: Jede fachliche Information wird genau einmal definiert, versioniert, entschieden und mit ihren Ableitungen verbunden.

Zielbild: Eine Änderung an einer fachlichen Regel verändert nicht fünf Dokumente manuell. Sie verändert die Spezifikation. Daraus werden die betroffenen Repräsentationen - etwa erzeugte Dokumentation, XSD, Validatoren, Testfälle und Implementierungsparameter - nachvollziehbar neu erzeugt. Der Anwendungscode selbst wird nicht generiert; er bleibt ein eigenständiges, refaktorierbares Engineering-Artefakt und wird über Akzeptanzkriterien, Validatoren, Tests und Traceability an die Spezifikation gebunden.

2. Das Fallbeispiel: UStVA 2026

Ausgangspunkt war eine vergleichsweise einfache Aufgabe: Daten aus einer Buchhaltungstabelle in eine ELSTER-konforme XML für die Umsatzsteuer-Voranmeldung überführen und vor dem Upload in ELSTER lokal zu validieren.

Der ursprüngliche Export erzeugte eine XML, die über Monate praktisch genutzt wurde. Im Zuge einer Generalisierung wurde die Implementierung gegen die im ERiC-Entwicklerpaket bereitgestellten XSDs und die Jahresdokumentation 2026 geprüft. Dabei wurden mehrere Ebenen sichtbar, die ein Integrator gleichzeitig beherrschen muss.

Erstens beschreibt das fachliche XSD die XML-Struktur. Zweitens enthält die Jahresdokumentation Regeln, die über reine Datentypen hinausgehen. Drittens existiert der reale Übermittlungsweg, dessen beobachtetes Verhalten nicht in jedem Detail mit der strikten lokalen XSD-Validierung übereinstimmen muss.

2.1 Die Artefaktbasis

- UStVA-2026.xsd - fachliches Schema für die Umsatzsteuer-Voranmeldung 2026.
- UStVA-2026-Nutzdaten.xsd - Einbettung der fachlichen Nutzdaten.
- elster11_UStVA_2026_extern.xsd sowie Header-/Basis-XSDs - technische ELSTER-v11-Hülle.
- Jahresdokumentation_UStVA_2026.ods - Felder, Kennzahlen, Texte und fachliche Plausibilitätsregeln.

Die Jahresdokumentation weist sich selbst für VZ 2026 / ERiC 44.2 (Versionslabel 6.0) aus.

2.2 Der erste wichtige Unterschied: leer ist nicht null

In einer Tabellen- oder API-Quelle können mindestens drei Zustände auftreten: kein Wert, der String “null” und ein expliziter numerischer Wert inkl. 0.00. In vielen Exporten werden diese Zustände früh vereinheitlicht. Aus leer wird 0, aus “null” wird 0, und anschließend wird 0.00 in die XML geschrieben.

Genau damit geht Information verloren. Für eine normale Voranmeldung kann ein Nullwert unnötig sein; bei einer Berichtigung kann ein expliziter Nullwert dagegen gerade die fachliche Aussage sein, dass ein zuvor vorhandener Wert auf 0.00 korrigiert wird.

Wer leer und 0.00 vor der Plausibilisierung gleichsetzt, hat die fachliche Entscheidung bereits verloren.

2.3 Der zweite Unterschied: Key-Value-Semantik und Serialisierungsreihenfolge

Kennzahlen wie Kz66, Kz67, Kz84 oder Kz85 sind semantisch durch ihren Schlüssel identifiziert. Fachlich ändert sich der Wert von Kz66 nicht dadurch, ob Kz84 davor oder danach steht.

Das XSD verwendet jedoch xs:sequence. Damit wird die Serialisierung positionsabhängig. Eine XML kann deshalb inhaltlich dieselben Key-Value-Paare enthalten und trotzdem strukturell ungültig sein, wenn die Reihenfolge von der XSD-Sequenz abweicht.

3. Was die veröffentlichten Artefakte tatsächlich ausdrücken

Für die Fallstudie ist entscheidend, nicht zu interpretieren, was ein Artefakt vermutlich meint, sondern zu lesen, was es tatsächlich spezifiziert.

3.1 Kz10: Berichtigte Anmeldung

```
<xs:element name="Kz10" type="Ja1BaseSType" minOccurs="0" maxOccurs="1">

<xs:documentation>Berichtigte Anmeldung</xs:documentation>

</xs:element>
```

Im XSD ist Kz10 optional. Der verwendete Basistyp erlaubt den Wert 1. Die Jahresdokumentation bezeichnet das Feld ebenfalls als „Berichtigte Anmeldung“ und führt es nicht als Pflichtfeld.

Noch aussagekräftiger sind die Regeln der Jahresdokumentation: Mehrere Nullwert-Hinweise prüfen explizit “FeldNichtAngegeben(Kz10)” und formulieren anschließend, dass es sich nicht um eine Berichtigung der Anmeldung handelt. Damit ist die Semantik des fehlenden Kennzeichens in diesen Regeln klar verwendet.

> Beispiel: Für Kz84/Kz85 wird ein Hinweis ausgelöst, wenn Kz10 fehlt, beide Felder angegeben sind und Kz84 = 0 sowie Kz85 = 0.00 sind. Die Dokumentation erklärt, dass Nullwerte bei einer nicht berichtigten Anmeldung nicht erforderlich sind.

3.2 Korrespondierende Kennzahlenpaare

Die Jahresdokumentation enthält Regeln, die fachliche Zusammenhänge ausdrücklich modellieren. Dazu gehören unter anderem:

- Kz35 und Kz36 sind gemeinsam anzugeben.
- Kz46 und Kz47 sind gemeinsam anzugeben.
- Kz73 und Kz74 sind gemeinsam anzugeben.
- Kz76 und Kz80 sind gemeinsam anzugeben.
- Kz84 und Kz85 sind gemeinsam anzugeben.
- Kz95 und Kz98 sind gemeinsam anzugeben.
- Kz94 und Kz96 sind gemeinsam anzugeben.

Diese Regeln sind keine Formatdetails. Sie sind fachliche Invarianten. Ein Validator, der nur das XSD prüft, kennt nicht automatisch alle diese Zusammenhänge. Ein Exporter, der nur jede befüllte Tabellenzelle in XML übersetzt, kennt sie ebenfalls nicht.

3.3 Datentypen sind kennzahlabhängig

- Kz84: Ganzzahl ohne Nachkommastellen.
- Kz85: Dezimalzahl mit genau zwei Nachkommastellen.
- Kz86: Ganzzahl ohne Nachkommastellen.
- Kz66 und Kz67: Dezimalzahlen mit genau zwei Nachkommastellen.

Ein generischer „Betrag = zwei Nachkommastellen“-Formatter ist damit fachlich unzureichend. Die Formatierung ist Teil der Feldspezifikation und muss aus derselben Quelle kommen wie die Dokumentation und die Tests.

3.4 Reihenfolge durch xs:sequence

Im relevanten Sequenzabschnitt stehen Kz61 bis Kz69 vor Kz81, Kz83, Kz84, Kz85 und Kz86. Eine XML, die Kz84/Kz85 vor Kz66/Kz67 serialisiert, kann deshalb gegen die XSD scheitern, obwohl jede Kennzahl für sich korrekt benannt und typisiert ist.

```
<Kz66>243.67</Kz66>

<Kz67>5.92</Kz67>

<Kz84>65</Kz84>

<Kz85>5.92</Kz85>
```

Ob diese Serialisierungsreihenfolge fachlich sinnvoll ist, ist eine andere Frage. Formal ist sie Bestandteil der veröffentlichten XSD.

4. Der Bruch zwischen Artefakt und Laufzeit

Der interessanteste Befund der Fallstudie ist nicht ein einzelner XSD-Fehler. Es ist die Beobachtung, dass eine praktisch akzeptierte XML und eine strikte lokale XSD-Validierung nicht zwingend dieselbe Aussage über Gültigkeit liefern.

Im betrachteten Integrationsfall wurde über Monate eine XML importiert, in der **Kz84/Kz85 vor Kz66/Kz67 standen.** Die Werte selbst waren korrekt. Bei der späteren lokalen Prüfung gegen UStVA-2026.xsd beanstandete xmllint die Reihenfolge, weil das Schema eine xs:sequence vorgibt.

Damit ist zunächst nur eines sicher: Das beobachtete Laufzeitverhalten und die strikte lokale Schemaauswertung sind in diesem Punkt nicht deckungsgleich. Warum das so ist, lässt sich aus den vorliegenden Artefakten nicht beweisen.

> Mögliche technische Erklärungen wären eine Vorverarbeitung, Normalisierung oder ein anderer Validierungspfad. Keine dieser Erklärungen ist durch die vorliegenden Unterlagen belegt und wird deshalb hier nicht als Ursache behauptet.

4.1 Was der Befund nicht bedeutet

- Er beweist nicht, dass ELSTER insgesamt falsch spezifiziert ist.
- Er beweist nicht, dass die XSD wirkungslos ist.
- Er beweist nicht, dass xmllint falsch validiert - im Gegenteil: xs:sequence wird standardkonform ausgewertet.

4.2 Was der Befund sehr wohl bedeutet

- Ein Integrator kann aus einem einzelnen Artefakt nicht sicher auf das gesamte Laufzeitverhalten schließen.
- Ein produktiv akzeptiertes Beispiel ist kein vollständiger Ersatz für eine Spezifikation.
- Eine veröffentlichte XSD ist kein Ersatz für die fachlichen Plausibilitätsregeln.
- Ohne eindeutige Herleitung ist später kaum feststellbar, welches Artefakt aus welcher fachlichen Entscheidung entstanden ist.

Wenn man erst im Fehlerfall herausfinden muss, welches Artefakt die Regel enthält, ist die Informationsarchitektur bereits Teil des Problems.

5. Das eigentliche Muster: Specification Drift

Der ELSTER-Fall ist deshalb relevant, weil er ein sehr verbreitetes Projektmuster in einer kleinen, überprüfbaren Form sichtbar macht.

5.1 Der klassische Ablauf

Fachliche Idee
→ **Excel-Datei oder Fachkonzept**
→ **Anhang im Jira-Ticket**
→ **Entwicklerinterpretation**
→ **Implementierung**
→ **Testfälle aus einer weiteren Quelle**
→ **Dokumentation nach Fertigstellung**

In diesem Ablauf wird Information nicht geführt, sondern kopiert. Jede Kopie ist eine neue Gelegenheit für Auslassungen, Erweiterungen und Interpretationen. Eine Anforderung kann in der Excel-Datei stehen und im Code fehlen. Eine technische Restriktion kann im Code entstehen und nie zurück in die Dokumentation gelangen. Ein Test kann das tatsächliche Verhalten prüfen, ohne noch auf die ursprüngliche fachliche Regel zu verweisen.

Das Problem verschärft sich, sobald erzeugte Repräsentationen der Spezifikation und manuell gepflegte Parallelbeschreibungen nebeneinander existieren. Wenn eine XSD aus der Spezifikation erzeugt wird, der Entwickler aber eine separate Excel-Datei als faktische Anforderungsquelle implementiert, ist nicht die Generierung das Problem. Der Bruch liegt davor: Die erzeugte Repräsentation und die Implementierung werden nicht mehr nachweisbar aus derselben führenden Spezifikation gesteuert.

5.2 Vier typische Drift-Arten

- Definitionsdrift: dasselbe Spezifikationselement wird in verschiedenen erzeugten Repräsentationen mit unterschiedlicher Bedeutung oder Optionalität wiedergegeben.
- Regeldrift: eine fachliche Regel ist in einer erzeugten Repräsentation oder in der Runtime wirksam, fehlt aber in einer anderen erzeugten Repräsentation wie Validator oder Test.
- Formatdrift: Datentyp, Dezimalstellen, Reihenfolge oder Namespace werden unterschiedlich umgesetzt.
- Nachweisdrift: niemand kann später zeigen, welche Entscheidung zu welcher Regel, Implementierung und Testabdeckung geführt hat.

Das Gegenmittel ist nicht mehr Dokumentation. Das Gegenmittel ist weniger doppelte Information.

6. MORe: eine Spezifikation, viele erzeugte Repräsentationen

MORe behandelt Dokumentation nicht als nachträgliches Produkt der Entwicklung. Die Spezifikation ist das führende Artefakt. Dokumentation, XSD, Validatoren, Tests und Implementierungsparameter sind erzeugte Repräsentationen dieser Spezifikation. Die Implementierung selbst bleibt davon bewusst getrennt: Sie wird nicht aus MORe generiert, sondern gegen die Spezifikation entwickelt, geprüft und nachgewiesen.

6.1 Das Prinzip

Fachlicher Bedarf
→ **kanonische Anforderung / Regel**
→ **Daten- und Prozessmodell**
→ **Akzeptanzkriterien**
→ **XSD / API-Schema / Feldkatalog**
→ **Validator / Plausibilitätsregeln**
→ **Testfälle**
→ **Implementierung**
→ **Abnahme**
→ **Produktionsnachweis**

Entscheidend ist die Herkunft der Information. MORe fordert nicht, dass jede Repräsentation dasselbe Format hat. Eine XSD bleibt eine XSD, ein Test bleibt ein Test, ein Feldkatalog bleibt lesbare Dokumentation. Aber diese Repräsentationen werden aus derselben Spezifikation erzeugt. Die Implementierung ist dagegen kein generiertes Repräsentationsartefakt: Sie bleibt eigenständig und muss über Schnittstellen, Akzeptanzkriterien, Validatoren, Tests und Traceability nachweisbar zur Spezifikation passen.

Für ein Feld wie Kz84 wäre deshalb weder die XSD noch die erzeugte Excel-/Feld-Dokumentation die führende Wahrheit. Führend ist das Spezifikationselement selbst: mit eindeutiger ID, Bedeutung, Datentyp, Optionalität, Serialisierungsregel, Abhängigkeiten und Akzeptanzkriterien. XSD, Dokumentation, Validator und Tests stellen diese Information für unterschiedliche Zwecke dar.

> MORe schreibt kein bestimmtes Dateiformat für die kanonische Spezifikation vor. Entscheidend sind Eindeutigkeit, Versionierung, Verantwortlichkeit, Traceability und die Regel: Jede Information genau einmal.

6.2 Was aus der Spezifikation erzeugt werden sollte

- XSD- oder JSON-Schema-Definitionen für technische Schnittstellen.
- Erzeugte Dokumentation für Menschen: Feldkataloge, Regelbeschreibungen und fachliche Sichten.
- Validatoren und ausführbare Validierungsregeln mit stabilen Regel-IDs.
- Testfälle und Grenzwerttests aus Typen, Bedingungen, Akzeptanzkriterien und Abhängigkeiten.
- Mapping- und Implementierungsparameter für Import, Export und technische Anbindung.
- Release- und Änderungsdokumentation aus Version und Entscheidungsverlauf.
- Traceability-Nachweise für Abnahme, Revision und Betrieb.

Generieren heißt nicht: mehr Automatisierung um jeden Preis. Generieren heißt: Eine erzeugte Repräsentation darf nicht manuell zur zweiten Wahrheit werden. Und es heißt ausdrücklich nicht, den Anwendungscode einem Generator zu unterwerfen.

6.3 Abgrenzung zu klassischem Model-Driven Development

Der model-driven Ansatz hatte einen richtigen Kern: Information wird einmal modelliert und daraus werden konsistente Artefakte erzeugt. Mit dem Übergang zu iterativen und agilen Entwicklungsweisen trat dieser Ansatz jedoch in den Hintergrund. Problematisch war nicht die Modellierung selbst, sondern der Anspruch, aus einem zentralen Modell bis hinunter zum Anwendungscode möglichst vollständig zu generieren.

Damit wurde der Anwendungscode faktisch Eigentum des Generators. Manuelle technische Verbesserungen und Refactorings ließen sich nur eingeschränkt durchführen oder wurden beim nächsten Generierungslauf überschrieben. Gleichzeitig blieb ein belastbarer Roundtrip vom Code zurück ins Modell und wieder in den Code praktisch schwer beherrschbar. Modell und Implementierung konnten sich dadurch gerade dort gegenseitig behindern, wo iterative Entwicklung technische Evolution verlangt.

MORe zieht die Grenze bewusst früher. Model-driven ist die Spezifikation: Aus ihr werden die Repräsentationen erzeugt, die deterministisch ableitbar sind - Dokumentation, XSD/API-Schema, Validatoren, Testfälle, Mappings und Implementierungsparameter. Der Anwendungscode wird nicht aus MORe generiert. Er bleibt refaktorierbar und technisch evolvierbar; seine Konformität zur Spezifikation wird durch Akzeptanzkriterien, Validatoren, Tests und Traceability nachgewiesen.

MORe ist model-driven bei der Spezifikation, nicht generator-driven bei der Implementierung.

7. Zielarchitektur für ausführbare Spezifikationen

Die MORe-Anwendung auf Schnittstellen wie UStVA führt zu einer Architektur, in der die Spezifikation maschinenlesbar genug ist, um konsistente Repräsentationen zu erzeugen, und zugleich fachlich erklärbar für Fachbereich, Entwicklung, Test und Revision bleibt. Maschinenlesbarkeit dient dabei der Ableitung - nicht der Generierung des Anwendungscodes.

7.1 Kanonisches Feldmodell

```
field: Kz84

meaning: Bemessungsgrundlage für andere Leistungen

type: integer

optional: true

pairedWith: Kz85

serializationOrder: 42

sourceRuleIds: [110085072, 110085206]
```

Das Beispiel ist bewusst technologieagnostisch. Ob die Spezifikation in einer Datenbank, einem Modellierungswerkzeug, YAML, JSON oder einer spezialisierten Plattform liegt, ist zweitrangig. Entscheidend ist, dass die Attribute nicht erneut in XSD, Excel, Wiki und Testfall von Hand gepflegt werden.

7.2 Kanonisches Regelmodell

```
rule: UStVA_Kz84_85_Gemeinsam

if: oneOf(Kz84, Kz85) is present

then: both(Kz84, Kz85) must be present

severity: error

test: generated
```

Ein zweites Regelobjekt kann den Nullwert-Hinweis modellieren: Fehlt Kz10 und sind Kz84/Kz85 mit 0 beziehungsweise 0.00 belegt, ist die Nullwertangabe nicht erforderlich. Die Regel selbst ist Teil der Spezifikation. Aus ihr entstehen die erzeugte Dokumentation dieser Regel, Validatorcode und Testfall.

7.3 Fachregel und Serialisierungsregel trennen

Die XSD-Reihenfolge ist ein gutes Beispiel dafür, warum fachliche und technische Regeln nicht vermischt werden sollten. Fachlich bleibt Kz84 ein Schlüssel mit Wert. Technisch kann die Serialisierung trotzdem eine Position verlangen.

Im kanonischen Modell wird die Reihenfolge deshalb als technische Ableitung geführt. Der Fachprozess muss nicht positionsabhängig denken. Der XML-Generator sortiert vor der Ausgabe deterministisch nach dem definierten serializationOrder.

Technische Restriktionen dürfen die Fachsemantik beeinflussen, aber sie dürfen sie nicht ersetzen.

8. Anwendung auf den UStVA-Fall

Aus der Fallstudie lässt sich eine konkrete Umsetzungsstrategie ableiten, die über diesen einen Export hinaus wiederverwendbar ist.

8.1 Eingabedaten normalisieren - ohne Information zu vernichten

- Leere Zelle bleibt absent.
- String null wird als absent behandelt, wenn die Quelle ihn als Nichtbelegung verwendet.
- Explizite 0 oder 0.00 bleibt ein Wert.
- Erst nach dieser Unterscheidung erfolgt numerische Normalisierung und Formatierung.

Diese Reihenfolge verhindert, dass eine Hilfsfunktion wie CleanAmount aus einem leeren Feld eine 0 erzeugt und der XML-Generator daraus anschließend ein fachlich nicht vorhandenes Element macht.

8.2 Berichtigungen explizit behandeln

Die Jahresdokumentation verwendet das Fehlen von Kz10 in mehreren Regeln als Kennzeichen dafür, dass keine Berichtigung vorliegt. Für den Export folgt daraus eine klare Strategie: In normalen Voranmeldungen werden unnötige Nullwerte vermieden; in einer Berichtigung muss ein expliziter Nullwert erhalten bleiben können, weil er eine Korrektur auf null ausdrücken kann.

8.3 Korrespondierende Felder vor der Serialisierung prüfen

Paare wie Kz35/Kz36, Kz84/Kz85 oder Kz95/Kz98 werden nicht erst durch eine Fehlermeldung im Zielsystem entdeckt. Die Beziehung gehört in die kanonische Regelbasis und wird vor Erzeugung der XML geprüft.

Der Exporter sollte daher nicht mehr „Zeile lesen und sofort XML schreiben“. Er sammelt zunächst Kennzahl und Wert, führt die Regeln aus und serialisiert erst anschließend.

Quelle einlesen
→ absent / null / 0 unterscheiden
→ fachliche Regeln prüfen
→ Datentyp und Format anwenden
→ nach XSD-Reihenfolge sortieren
→ XML erzeugen
→ XSD validieren
→ fachliche Plausis validieren
→ Übermittlung / Rückmeldung protokollieren

8.4 Datentypen nicht im Code verstreuen

Sonderfälle wie Ganzzahl für Kz84 und Kz86 oder zwei Nachkommastellen für Kz85, Kz66 und Kz67 sollten nicht als wachsende Select-Case-Liste zur zweiten Spezifikation im Exporter werden. Kurzfristig ist ein Case eine pragmatische Korrektur. Zielarchitektonisch muss die Information aus dem Feldmodell kommen.

> Pragmatischer Übergang: bestehende CASE-Logik nutzen, aber jeden Sonderfall auf eine Regel-/Feld-ID zurückführen. Zielzustand: Formatter und Validator lesen Typinformationen aus der Spezifikation.

9. Eine mehrstufige Validierungsstrategie

Ein einzelner Validator kann nicht alle Fehlerklassen sinnvoll abdecken. MORe trennt die Ebenen, verbindet sie aber über dieselben Spezifikationselemente und Regel-IDs.

Stufe 1 - Quell- und Mappingvalidierung

- Sind Pflichtinformationen vorhanden?
- Sind Quelldaten eindeutig als absent, null oder Wert klassifiziert?
- Ist jede Quellspalte auf genau ein fachliches Feld gemappt?

Stufe 2 - Feldvalidierung

- Datentyp, Länge, Wertebereich, Enum und Dezimalstellen.
- Kennzahlspezifische Formatierung.

Stufe 3 - Fachliche Plausibilitäten

- Gemeinsam anzugebende Felder.
- Bedingte Angaben und Ausschlüsse.
- Nullwertregeln in Abhängigkeit von Kz10.

Stufe 4 - Schema- und Serialisierungsvalidierung

- Namespace, Root-Element, Version.
- Elementreihenfolge nach xs:sequence.
- XSD-Datentypen und Kardinalitäten.

Stufe 5 - Integrationsvalidierung

- Tatsächliche Annahme durch ERiC/ELSTER.
- Rückmeldungen, Fehlercodes und gegebenenfalls Abweichungen zum lokalen Validator.

Stufe 6 - Regression und Golden Files

Für repräsentative Fälle werden bekannte gültige und ungültige XMLs versioniert. Jede Änderung an der Spezifikation erzeugt einen nachvollziehbaren Diff: Welche Artefakte ändern sich, welche Tests kippen, welche neuen Fälle werden erwartet?

Damit wird eine produktiv akzeptierte Beispieldatei nicht zur heimlichen Spezifikation. Sie wird zu einem Regressionstest, dessen Herkunft und Gültigkeitsbereich bekannt sind.

Validierung ist kein Endschritt. Sie ist eine erzeugte, ausführbare Repräsentation der Spezifikation.

10. Einführungsstrategie mit MORe

Die Umstellung muss nicht mit einem Big Bang beginnen. Entscheidend ist, zuerst die führende Information zu identifizieren und anschließend die manuellen Kopien Schritt für Schritt durch Ableitungen zu ersetzen.

Phase A - Artefakte inventarisieren

- Welche XSDs, Excel-Dateien, Wiki-Seiten, Jira-Tickets, Codekonstanten und Tests enthalten fachliche Regeln?
- Welche davon werden generiert, welche manuell gepflegt?
- Wo widersprechen sich Definitionen oder Laufzeitverhalten?

Phase B - kanonische Spezifikation aufbauen

- Eindeutige IDs für Felder, Regeln, Entscheidungen und Versionen.
- Bedeutung, Typ, Optionalität, Abhängigkeiten und technische Serialisierung trennen.
- Jede Information genau einmal als führend festlegen.

Phase C - Ableitungen automatisieren

- XSD und andere technische Schemata aus der Spezifikation erzeugen.
- Erzeugte Feld-/Regeldokumentation, Validatoren und Testfälle aus denselben Spezifikationselementen ableiten.
- Dokumentation nicht erneut schreiben, sondern als Repräsentation der Spezifikation rendern/generieren.
- Anwendungscode nicht generieren: Die Implementierung bleibt eigenständig, refaktorierbar und wird über Tests, Validatoren und Akzeptanzkriterien an die Spezifikation gebunden.

Phase D - Drift technisch verhindern

- Generierte Dateien erhalten einen Versions- und Herkunftsnachweis.
- Manuelle Änderungen an generierten Artefakten werden im Build erkannt.
- CI prüft Schema, Regeln, Regressionen und Artefaktdiffs.

Phase E - Betrieb und Änderungen rückkoppeln

- Produktive Fehler führen nicht direkt zu einer Code-Sonderregel.
- Zuerst wird geprüft, welches Spezifikationselement fehlt oder falsch ist.
- Die Korrektur erfolgt in der Spezifikation; erzeugte Repräsentationen werden daraus neu erzeugt. Erforderliche Änderungen am Anwendungscode werden anschließend umgesetzt und gegen dieselbe Spezifikation geprüft.

> Die wichtigste organisatorische Regel lautet: Kein Entwickler muss eine Excel-Anforderung interpretieren, wenn die entscheidende Information bereits strukturiert in der Spezifikation vorhanden sein kann.

11. Governance: Wer verantwortet die eine Spezifikation?

*Eine **Single Source of Truth** ist keine Dateiablage. Sie funktioniert nur, wenn Verantwortung, Änderung und Nachweis genauso eindeutig sind wie die Datenstruktur.*

11.1 Rollen

- Fachlicher Owner: verantwortet Bedeutung und fachliche Entscheidung.
- Specification Steward: verantwortet Konsistenz des kanonischen Modells und die Informationsarchitektur.
- Engineering: verantwortet Generatoren, technische Ableitung und Implementierung.
- Test/QA: verantwortet ausführbare Nachweise und Regelabdeckung.
- Betrieb/Integration: liefert Laufzeitfeedback und Annahme-/Fehlernachweise zurück.

11.2 Änderungsprozess

Änderungsbedarf
→ **fachliche Entscheidung**
→ **Spezifikation ändern**
→ **erzeugte Repräsentationen automatisch neu erzeugen**
→ **erforderliche Implementierungsänderung umsetzen**
→ **Tests automatisch/anlassbezogen ausführen**
→ **Review des Diffs**
→ **Freigabe**
→ **Release**
→ **Produktionsnachweis**

11.3 Definition of Done für Spezifikationsänderungen

- Die Regel oder das Feld besitzt eine eindeutige ID.
- Die fachliche Entscheidung ist nachvollziehbar.
- XSD, erzeugte Dokumentation, Validatoren und Tests stammen nachweisbar aus derselben Spezifikationsversion.
- Der Anwendungscode erfüllt die zu dieser Spezifikationsversion gehörenden Akzeptanzkriterien und Tests.
- Positive und negative Tests existieren.
- Auswirkungen auf bestehende Integrationen sind sichtbar.
- Der Produktionsnachweis verweist auf die freigegebene Spezifikationsversion.

Governance bedeutet hier nicht mehr Freigabeschritte. Governance bedeutet: Es ist jederzeit nachvollziehbar, welche Information warum gilt und wo sie wirksam wurde.

12. Nutzen über ELSTER hinaus

Der UStVA-Fall ist klein genug, um das Muster sichtbar zu machen. Das gleiche Problem skaliert jedoch in großen Programmen exponentiell.

Schnittstellen, regulatorische Meldungen, Zahlungsverkehr, Kernbankenmigrationen, Berechtigungskonzepte oder Datenhaushalte bestehen aus tausenden Feldern, Regeln, Abhängigkeiten und Entscheidungen. Dort führt jede zusätzliche manuelle Kopie derselben Information zu mehr Abstimmung, mehr Testaufwand und mehr Rekonstruktion.

Eine MORe-basierte Spezifikationsarchitektur reduziert nicht zwangsläufig die Zahl der Artefakte. Sie reduziert die Zahl der unabhängig gepflegten Wahrheiten. Das ist der entscheidende Unterschied.

Für Fachbereiche

- Fachliche Regeln bleiben lesbar und entscheidbar.
- Technische Ableitungen sind sichtbar, ohne dass der Fachbereich XSD lesen muss.

Für Entwicklung

- Weniger Interpretationsarbeit aus Anhängen und Freitext.
- Datentypen, Regeln und Abhängigkeiten sind maschinenlesbar, ohne den Anwendungscode einem Generator zu unterwerfen.
- Refactoring und technische Evolution bleiben möglich; die Spezifikationskonformität wird über Tests und Validatoren abgesichert.

Für Test und Audit

- Testfälle lassen sich auf Regel-IDs und Entscheidungen zurückführen.
- Nachweise entstehen im Lebenszyklus statt in einer Rekonstruktionsphase vor dem Audit.

Für Betrieb und Integration

- Laufzeitabweichungen werden zu reproduzierbaren Fällen.
- Ein akzeptierter Sonderfall wird als Testfall dokumentiert, nicht als stilles Wissen einzelner Personen.

13. Fazit

Die entscheidende Frage lautet nicht: Wo steht die Dokumentation? Sie lautet: Wo steht die Spezifikation, aus der alle erzeugbaren Repräsentationen nachweisbar entstehen und gegen die die Implementierung geprüft wird?

Das ELSTER-UStVA-Beispiel zeigt, wie schnell sich diese Frage konkret stellt. Das XSD kennt Datentypen und Reihenfolgen. Die Jahresdokumentation kennt zusätzliche fachliche Regeln. Der reale Integrationsweg liefert Laufzeitverhalten. Ein lokaler Exporter muss daraus eine konsistente Entscheidung treffen.

Die falsche Reaktion wäre, jede neu entdeckte Besonderheit als weitere Sonderregel in Code, Excel oder Wiki zu verteilen. Damit entsteht genau der nächste Drift.

Die MORe-Reaktion ist umgekehrt: Die neu erkannte Information wird an die führende Spezifikation zurückgeführt. Von dort werden XSD, erzeugte Dokumentation, Validatoren, Tests und Implementierungsparameter neu erzeugt. Der Anwendungscode wird nicht regeneriert, sondern - sofern erforderlich - gezielt geändert, refaktoriert und anschließend gegen dieselbe Spezifikation geprüft. Der Zusammenhang bleibt erhalten.

Eine Spezifikation. Viele erzeugte Repräsentationen. Eine eigenständig evolvierbare Implementierung. Aber keine zweite Wahrheit.

Das ist keine theoretische Forderung nach perfekter Dokumentation. Es ist eine praktische Strategie gegen vergessene Anforderungen, undokumentierte Implementierungsdetails, widersprüchliche Tests und teure Rekonstruktion.

Die Dokumentation entsteht nicht nachträglich. Sie entsteht während der Arbeit an der Anforderung - weil sie aus derselben Information entsteht.

www.more-framework.org · mail@more-framework.org
www.projektkram.de · hallo@projektkram.de

Anhang A. Quellen- und Artefaktbasis

Die technischen Aussagen dieser Fallstudie beruhen auf den im Projekt verwendeten ELSTER-Entwicklerartefakten für die Umsatzsteuer-Voranmeldung 2026 sowie auf dem beschriebenen Integrationsfall.

A.1 ELSTER-Schemata

- UStVA-2026.xsd
- UStVA-2026-Nutzdaten.xsd
- elster11_UStVA_2026_extern.xsd
- th000011_extern.xsd
- ndh000011.xsd
- headerelemente.xsd
- headerbasis000003.xsd
- headerbasis_datenarten.xsd
- headerbasis_datentypen.xsd
- headerbasis_verfahren.xsd

A.2 Jahresdokumentation

- Jahresdokumentation_UStVA_2026.ods
- Blatt „Anmeldungssteuern - Felder“: Felddefinitionen, darunter Kz10 „Berichtigte Anmeldung“.
- Blatt „Anmeldungssteuern - Regeln“: gemeinsame Kennzahlenpaare, Nullwert-Hinweise und weitere Plausibilitäten.
- Blatt „Allg. Information“: Dokumentation für VZ 2026 / ERiC 44.2 (Versionslabel 6.0).

A.3 Relevante nachprüfbare Beispiele

- Kz10: optionales Element, Typ Ja1BaseSType, zulässiger Wert 1.
- Kz84 und Kz86: Ganzzahltypen.
- Kz85, Kz66 und Kz67: Dezimaltypen mit zwei Nachkommastellen.
- Kz84/Kz85 sowie Kz95/Kz98: Regeln zur gemeinsamen Angabe.
- Mehrere Nullwert-Hinweise prüfen explizit das Fehlen von Kz10.
- Die XSD serialisiert Kennzahlen über xs:sequence; damit ist die Reihenfolge strukturell relevant.

A.4 Praxisbeobachtung

Im betrachteten Integrationsfall wurden XML-Dateien mit einer von der XSD-Sequenz abweichenden Reihenfolge praktisch über ELSTER importiert. Diese Beobachtung wird als Fallbeobachtung dokumentiert. Sie ist keine allgemeine Aussage darüber, welche Validierungsstufen ELSTER intern verwendet.

Die technische Ursache für diese Abweichung ist mit den vorliegenden Artefakten nicht belegt. Genau diese Grenze ist Teil der Aussage des Whitepapers: Wo Ableitung und Traceability fehlen, wird aus einer einfachen Regelabweichung schnell eine Rekonstruktionsaufgabe.

> Hinweis zur Reproduzierbarkeit: Die lokale XSD-Prüfung wurde mit xmllint gegen UStVA-2026.xsd durchgeführt. Die fachlichen Regeln wurden aus der Jahresdokumentation 2026 gelesen. Laufzeitbeobachtungen sind separat als empirische Evidenz gekennzeichnet.