Technische Universität München

TUM School of Engineering and Design

Lehrstuhl für Computergestützte Modellierung und Simulation

Architektur eines inkrementellen, bidirektionalen Kommunikationsservers für BCF-Issues

Bachelorthesis

für den Bachelor of Science Studiengang Umweltingenieurwesen

Autor: Carina Sailer

Matrikelnummer: [redacted]

Prüfer: Prof. Dr.-Ing. André Borrmann

Betreuer: M.Sc. Sebastian Esser

Ausgabedatum:

Abgabedatum: 15. Februar 2024

Vorwort

Diese Arbeit untersucht insbesondere die Übertragung von XML-Methodiken auf den Kontext des Building Information Collaboration Formats mit dem Ziel, inkrementelle Verfahren für den Informationsaustausch zu entwickeln und validieren. Der Hauptteil konzentriert sich auf die Übertragung von Diff- und Patch-Verfahren auf die markup-Datei eines BCF-ZIP-Ordners. Eine Fallstudie dient der Validierung der Ergebnisse und zeigt die praktische Anwendbarkeit der vorgeschlagenen Methodiken auf.

Abstract

Diese Arbeit untersucht die Anwendung inkrementeller Methoden zur Änderungserkennung in Building Information Modeling Collaboration Format-Dateien. Ziel ist es, bewährte Methoden aus dem Bereich der XML-Verarbeitung auf die spezifischen Anforderungen der Änderungserkennung in BCF-Dateien anzuwenden. So soll der Datenaustausch in BIM-Projekten optimiert werden. Die Arbeit gliedert sich in die Bereiche Grundlagen, Übertragung von XML-Methodiken auf den BCF-Kontext und eine Fallstudie zur praktischen Anwendung des diskutierten Konzepts.

Zusammenfassung

Die vorliegende Arbeit widmet sich der Untersuchung und Anwendung inkrementeller Methoden zur Änderungserkennung in Building Information Collaboration Format-Dateien. Das Ziel besteht darin, herauszufinden, wie bewährte Methoden aus dem Bereich der XML-Verarbeitung auf die spezifischen Anforderungen der Änderungserkennung in BCF-Dateien angewendet werden können. Durch ihre Anwendung soll der Datenaustausch in BIM-Projekten optimiert werden. Außerdem soll dadurch der Verlauf von Änderungen in einem Bauprojekt nachverfolgt und dokumentiert werden, um eine Versionierung und Revisionierung von BCF-Dateien zu ermöglichen.

Inhaltsverzeichnis

Abkürzungsverzeichnis

API	Application Programming Interface
BCF	Building Information Collaboration Format
BIM	Building Information Modelling
CAD	Computer Aided Design
CDE	Common Data Environment
GUID	Globally Unique Identifier
HTTP	Hypertext Transfer Protocol
iabi	Institut für angewandte Bauinformatik
IFC	Industry Foundation Class
ISG	International Implementation Support Group
JSON	JavaScript Object Notation
REST	Representational State Transfer
RFC	Request for Comments
URI	Uniform Resource Identifier
UTF	Unicode Transportation Format
UUID	Universally Unique Identifier
W3C	Word Wide Web Consortium
XML	Extensible Markup Language

1 Einführung und Motivation

In der heutigen rasanten und hochkompetitiven Wirtschaftslandschaft steht die Innovationsfähigkeit und Effizienzsteigerung im Zentrum des Unternehmenserfolgs. Die stetige Beschleunigung von Veränderungen und Entwicklungen in nahezu allen Lebensbereichen, einschließlich der Baubranche, geprägt von technologischem Fortschritt, neuen Bauvorschriften und sich wandelnden Kundenanforderungen, erfordert von Unternehmen und Fachleuten kontinuierliche Anpassungsfähigkeit. Dieser Anpassungsdruck wird durch die wachsende Konkurrenz in der Baubranche verstärkt, wodurch Unternehmen gezwungen sind, effizienter zu arbeiten und innovative Ansätze zu verfolgen, um sich von ihren Mitbewerbern abzuheben.

1.1 Einführung

Im Kontext dieser Herausforderungen gewinnt Building Information Modeling (BIM) zunehmend an Bedeutung. BIM ermöglicht die digitale Modellierung von Bauprojekten und liefert umfassende Informationen über das gesamte Projekt. Diese Modelle dienen nicht nur der Planung und Visualisierung, sondern auch der effizienten Zusammenarbeit zwischen den verschiedenen Akteuren der Baubranche. Ein zentraler Aspekt ist der Datenaustausch zwischen den verschiedenen Beteiligten eines Bauprojekts. Hier rückt das Building Information Modeling Collaboration Format (BCF) in den Fokus. Dieses ist ein spezielles Dateiformat, das entwickelt wurde, um die Kommunikation und den Austausch von BIM-relevanten Informationen zwischen verschiedenen Softwareanwendungen zu erleichtern. Obwohl der Inhalt eines einzelnen BCF-Containers normalerweise klein ist, kann durch die Anwendung inkrementeller Methoden ein effizienter Datenaustausch erzielt werden. Dabei soll lediglich die Modifikationsinformation zwischen zwei Dateiversionen übertragen werden. Eine solche Implementierung eröffnet die Möglichkeit, den Datenaustausch in BIM-basierten Projekten weiter zu optimieren. Zudem kann eine Änderungserkennung eine präzise Möglichkeit bieten, den Verlauf der Änderungen in einem Bauvorhaben nachzuverfolgen und zu dokumentieren. Dies könnte nutzbar gemacht werden für die Versionierung und Revisionierung. Bauprojekte durchlaufen oftmals eine Vielzahl von Änderungen, die über die Zeit hin-

weg erfasst werden müssen. Durch die Modifikationsinformationen können alle vorgenommenen Transformationen an den BCF-Dateien genau festgestellt werden. So können Projektbeteiligte ihren Verlauf genau verfolgen und verstehen, wie sich das Modell im Laufe der Zeit entwickelt hat. Außerdem kann durch die Versionierung auf frühere Versionen der BCF-Dateien zugegriffen werden. So können Veränderungen rückgängig gemacht werden. Der Verlauf von Änderungen wird damit auch transparent.

1.2 Ziel der Arbeit

Das Ziel dieser Arbeit besteht darin, die Anwendbarkeit inkrementeller Methoden zur Änderungserkennung in Building Information Modeling Collaboration Format (BCF)-Dateien zu untersuchen. Dabei sollen Erkenntnisse aus dem Bereich der XML-Verarbeitung auf die spezifischen Anforderungen der Änderungserkennung in BCF-Dateien übertragen werden. Konkret sollen bewährte Methoden zur Verarbeitung und Analyse von XML-Daten daraufhin untersucht werden, wie sie auf die Änderungserkennung in BCF-Dateien angewendet werden können. Die umfasst technische Aspekte sowie auch praktische Implikationen.

Durch die Untersuchung von Testbeispielen soll ermittelt werden, inwiefern die vorhandenen Methoden aus der XML-Verarbeitung auf BCFs anwendbar sind. Das Ziel ist es den Datenaustausch in BIM-Projekten weiter zu optimieren und die Effizient der Kommunikation zu steigern, insbesondere in Projekten, in denen es einer Vielzahl von Änderungen häufig vorkommt. Zudem soll die Änderungserkennung ermöglichen, den Verlauf der Modifikationen in einem Bauprojekt nachzuverfolgen und zu dokumentieren. So sollen das Ziel der Versionierbarkeit und Revisionierbarkeit von BCFs erreicht werden.

1.3 Aufbau der Arbeit

Zunächst sollen die Grundlagen für das Verständnis der Arbeit geschaffen werden. Daher wird die modellbasierte Planung und Kollaboration mit föderierten Fachmodellen, sowie das Building Information Collaboration Format als eine Methode zur modellbasierten Kommunikation, behandelt. Die Struktur von BCF und die daraus entwickelte Forschungslücke werden ebenfalls dargelegt. Im Anschluss wird die Übertragung von XML-Methodiken auf den BCF-Kontext beleuchtet. Dabei werden das Potential von XML und inkrementelle Verfahren zur Änderungserkennung und Versionierung von

XML-Dateien vorgestellt. Die Integration dieser Verfahren auf BCF-Daten und die damit verbundenen Anforderungen werden daraus abgeleitet. Als Abschluss dient eine Fallstudie dazu, die in den vorherigen Kapiteln diskutierten Konzepte praktisch anzuwenden und zu überprüfen. Dabei wird der Versuchsaufbau beschrieben, die Ergebnisse validiert und zusammengefasst.

2 Hintergrund & verwandte Arbeiten

Voraussetzung für die Planung, Erstellung und den Betrieb von Bauprojekten ist das Zusammenwirken zahlreicher Beteiligter, wobei kontinuierlich Informationen ausgetauscht werden müssen. Grund hierfür ist, dass dem Bauwesen schwierige Rahmenbedingungen zugrunde liegen. Die Prozess- und Wertschöpfungskette liegt nämlich nicht nur in der Verantwortung eines einzigen Unternehmens. Vielmehr ist ein Bauprojekt ist in der Regel äußerst komplex, denn es umfasst verschiedene Fachgebiete wie Architektur, Ingenieurwesen, Bauwirtschaft, Umweltschutz und viele mehr. Entwürfe müssen abgestimmt werden, Ressourcen wie Finanzmittel, Materialien, Arbeitskraft und Zeit müssen effizient eingesetzt werden, Qualitätssicherung muss erfolgen und Risiken frühzeitig erkannt, identifiziert und bewältigt werden. Zur Koordination und Integration all dieser Bereiche bedarf es der Zusammenarbeit zahlreicher Fachleute. Eine klare Kommunikation und transparenter Informationsaustausch sind unerlässlich. (Borrmann and König, 2021) Eine zentrale Frage lautet: Wie können alle Beteiligten reibungslos Informationen austauschen, gemeinsam an digitalen Modellen arbeiten und gleichzeitig die Integrität der Daten gewährleisten?

Die Antwort auf diese Frage liegt in der Etablierung einer Common Data Environment (CDE). Im nachfolgenden soll dieser Begriff definiert werden. Außerdem wird die Bedeutung und Notwendigkeit einer CDE für die modellbasierte Planung und Kollaboration mit föderierten Fachmodellen in der Bauindustrie untersucht. Es wird auf die Herausforderungen eingegangen, die sich aus der Vielfalt der beteiligten Unternehmen und Disziplinen ergeben. Besonders beleuchtet wird, warum Modell und Kommunikation getrennt werden.

2.1 Modellbasierte Planung und Kollaboration mit föderierten Fachmodellen

2.1.1 Definition Common Data Environment

Der Begriff „Common Data Environment“ (CDE) bezieht sich auf eine digitale Plattform oder Umgebung, die es den Projektbeteiligten ermöglicht, Informationen während des gesamten Lebenszyklus eines Bauwerks zu teilen, zu speichern und zu verwalten.

Eine CDE fungiert als zentraler Speicherort für alle relevanten Daten und Dokumente im Rahmen eines Bauprojekts.

2.1.2 Notwendigkeit eines Common Data Environment

Die Komplexität von Bauvorhaben liegt in der Vielfalt der involvierten Disziplinen und Unternehmen (Abbildung 2). Diese reichen von kleinen und mittleren Betrieben bis hin zu großen multinationalen Konzernen (Abbildung 1). Bei Abbildung 2 handelt es sich lediglich um eine exemplarische Auswahl der möglichen involvierten Disziplinen.

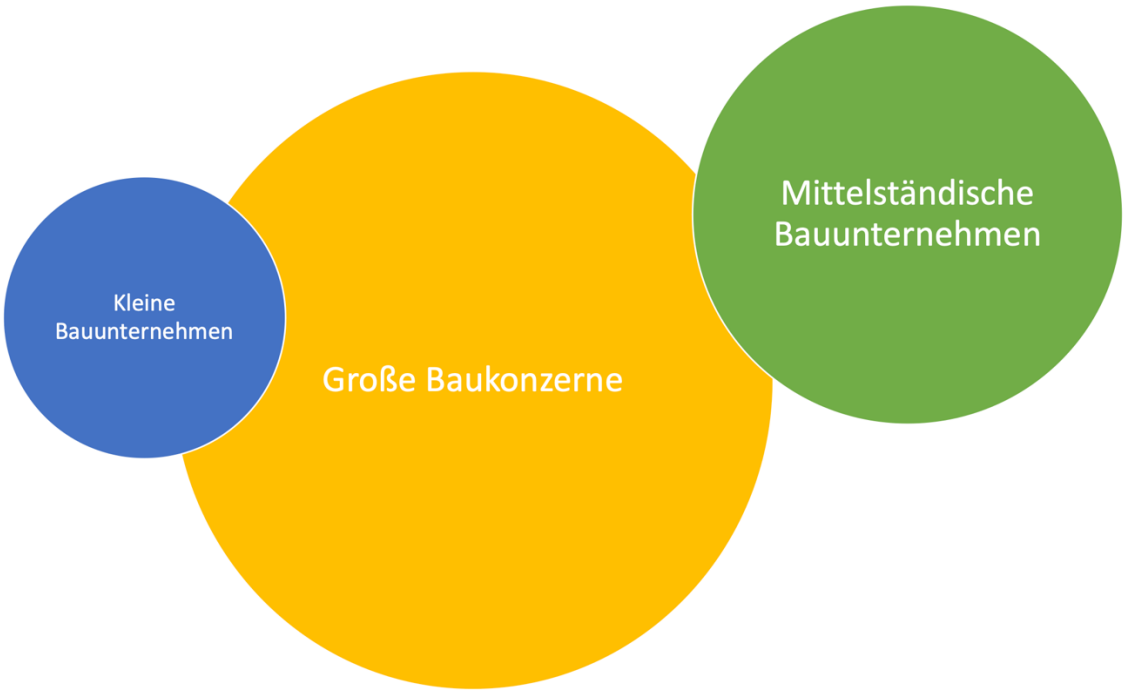

Abbildung 1 Beispiel der Unternehmensvielfalt

Abbildung 2 Vielfalt der involvierten Disziplinen

Jede Organisation nimmt zu unterschiedlichen Zeitpunkten am Bauprojekt teil und steuert verschiedene Mengen und Arten von Daten bei. So dominieren beispielsweise in der Planungsphase Architekten, wohingegen in der Bauphase Bauunternehmen, Ingenieurbüros und Handwerker aktiv sind. Während der Betriebsphase sind Facility-Management-Unternehmen und Instandhaltungsdienstleister beteiligt. Unterschiedliche Arten von Daten könnten sein:

- Geometrische Daten für Baupläne und Modelle
- Materialdaten, um Baustoffe zu identifizieren und ihre Eigenschaften zu beschreiben
- Terminpläne für die Projektplanung
- Kosteninformationen für die Budgetierung und Finanzierung

Zusätzlich verwenden die Projektbeteiligten unterschiedliche Werkzeuge, darunter sowohl BIM-basierte als auch nicht-BIM-basierte Autorenwerkzeige, um die notwendigen Informationen zu generieren. Zum Einsatz könnten zum Beispiel BIM-Software wie Autodesk Revit für 3D-Modellierung kommen oder im Gegensatz dazu CAD-Software

für traditionelle zweidimensionale Entwürfe. Diese Vielfalt stellt eine Herausforderung dar, die bei der Koordination von Bauprojekten berücksichtigt werden muss.

Aufgrund der Anatomie von Bauvorhaben und ihrer Unterteilung in unterschiedliche Leistungsphasen, ergibt sich eine vergleichsweise hohe Anzahl an Knotenpunkten, über welche Informationen ausgetauscht werden. Die verschiedenen Fachbereiche bringen unterschiedliche Sichtweisen mit und können ebenso verschiedene Softwareprodukte verwenden. Dadurch entstehen unterschiedliche Beschreibungen von Daten, die die Komplexität und Vielfalt der projektbezogenen Informationen widerspiegeln. (Preidel, 2020)

Bauherren nehmen verstärkt Einfluss auf Bauunternehmen und deren Subunternehmer, indem sie Richtlinien für Projekte festlegen sowie Standards einführen, die von ihren Partnern eingehalten werden müssen. Sie fordern vermehrt, eine gemeinsame Datenumgebung (CDE) bei Projekten. So soll sichergestellt werden, dass jederzeit eine digitale Aufzeichnung zur Verfügung ist, welche fortlaufend aktualisiert wird. (Hartmann, 2022)

2.1.3 Informationsaustausch mit einem Common Data Environment

In der heutigen Bauindustrie hat die modellbasierte Zusammenarbeit einen revolutionären Wandel eingeleitet und ist mittlerweile fest verankert im Konzept des Building Information Modeling (BIM). Ein wesentlicher Grundsatz der modellbasierten Kollaboration ist die Nutzung digitaler Modelle als Herzstück eines Bauvorhabens.

Die Praxis hat gezeigt, dass man sich nicht auf die Nutzung eines gemeinsamen Modells verlassen kann. Es führt zu Schwierigkeiten bei zeitgleicher und voneinander unabhängiger Bearbeitung von Aufgaben in der Planung. (Preidel and Borrmann, 2021)

In einem einzelnen Modell kann es zu Überschneidungen von Arbeitsbereichen der unterschiedlichen Fachgebiete kommen und so ist eine eindeutige Zuordnung der Zuständigkeiten nicht immer möglich (Preidel, 2020). Zudem sorgen rechtliche Gegebenheiten und wettbewerbliche Umstände, dass die beteiligten Parteien nicht dazu neigen, ihre aktuellen Fortschritte und Zwischenstände dauerhaft mit anderen zu teilen.

Die derzeitige Praxis und internationalen Standards implementieren die Zusammenarbeit auf der Grundlage der Modellföderation (Abbildung 3). Es bedeutet, dass die einzelnen Fachdisziplinen ihre eigenen Modelle erstellen und pflegen. Diese Modelle repräsentieren unterschiedliche Aspekte des Bauprojekts, wie beispielsweise Architektur oder HKLS (Heizung, Klima, Lüftung, Sanitär) und werden jedoch regelmäßig zu einem konsistenten Gesamtmodell zusammengeführt. (Esser et al., 2022)

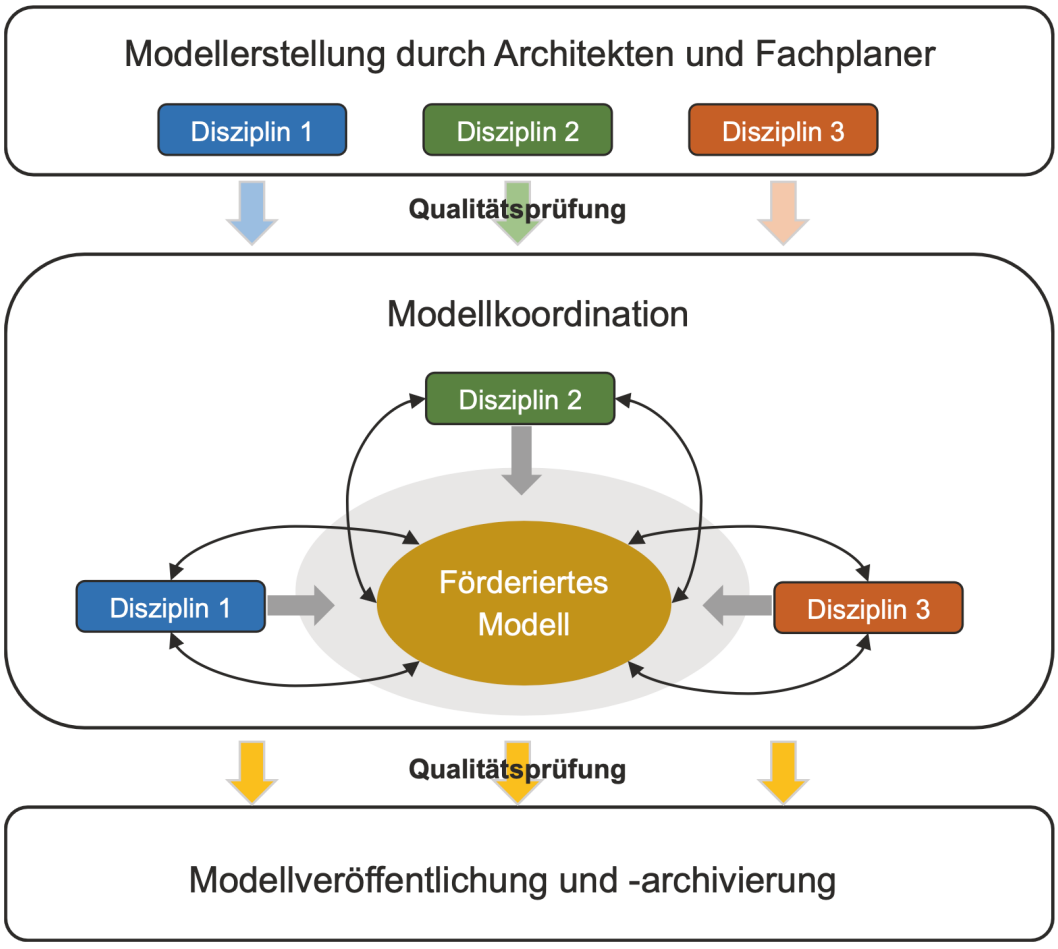

Abbildung 3 Prinzip der fachmodellbasierten Zusammenarbeit auf Basis eines föderierten Modells (Preidel and Borrmann, 2021)

Einheitlich aufgebaute Bauwerksmodelle ermöglichen eine direkte Verbesserung der Arbeits- und Kommunikationsprozesse der verschiedenen Parteien. Um diese Kooperation zu fördern, ist die Einrichtung eines gemeinsamen Datenraums entscheidend, in dem sämtliche Informationen der Projektbeteiligten zusammenfließen. Realisiert wird dies wie zuvor beschrieben durch ein Common Data Environment (CDE). Sie

dient als virtuelle Umgebung, um alle relevanten Informationen, Modelle und Daten im Verlauf eines Bauprojekts zu speichern, zu verwalten und auszutauschen. Damit wird das CDE als „Single Source of Truth“, zu Deutsch als einzige Quelle der Wahrheit verwendet (Rimskaia-Korsakova, 2022). Durch die Nutzung einer CDE ist gewährleistet, dass alle Beteiligten auf die gleichen aktuellen Informationen zugreifen können. Die zentrale Verwaltung der Modellinformationen dient damit als Basis für Kommunikation und Koordination, die Überprüfung der Informationsintegrität und die fortlaufende Aktualisierung des Projektstands.

Die Lösung komplexer und multidisziplinärer Probleme erfordert oft mehrere Iterationen in den Planungs- und Designaufgaben, um alle Anforderungen verschiedener Domänen zu erfüllen. Diese wiederholten Iterationen führen dazu, dass das Design schrittweise verfeinert wird. (Esser et al., 2022) In der Praxis hat sich gezeigt, dass dafür eine Trennung von Kommunikation und Modell erforderlich ist. Das Modell dient zwar als Grundlage für die Kommunikation, auf das jeder zugreifen kann, allerdings findet der Informationsaustausch über ein anderes Medium statt. Insbesondre, wenn die verschiedenen Fachplaner auch unterschiedliche Software nutzen. Folgendes Szenario soll das verdeutlichen: Ein spezifisches Bauteil wird mittels der CDE identifiziert und mit den beteiligten Partnern geteilt. Das Problem, das innerhalb des Bauteils besteht, muss jedoch zusätzlich durch ein Telefonat oder schriftlich per Mail gegebenenfalls mit Screenshots erläutert werden. (BIMcollab, 2024)

Eine Dokumentation der Probleme und Workflows wird parallel zur Arbeit am Modell benötigt, um auftretende Probleme strukturiert zu erfassen und zu verwalten. Nur so kann eine transparente Kommunikation gewährleistet werden. Daraus ergibt sich die Notwendigkeit eines Instruments zum effektiven Kommunikations- und Informationsaustausch.

2.2 Methoden zur modellbasierten Kommunikation

Im Folgenden wird das Building Information Collaboration Format als eine Methode zur modellbasierten Kommunikation in der Baubranche vorgestellt. Dafür wird BCF zunächst definiert und auf die Bedeutung im Bereich des Building Information Modeling eingegangen. Indem Herausforderungen des traditionellen Informationsaustauschs und deren Auswirkungen auf Bauprojekte betrachtet werden, wird die Notwendigkeit

von BCF für das Bauwesen erläutert. Verschiedene Anwendungsfälle von BCF über unterschiedliche Phasen eines Bauvorhabens werden genannt. Im weiteren Verlauf wird das BCF in einen breiteren Kontext eingeordnet und seine Rolle als Werkzeug der computergestützten Gruppenarbeit betrachtet. Abschließend wird darauf eingegangen, welche BIM-Produkte BCF unterstützen.

2.2.1 Definition des Building Information Collaboration Format

Das BIM-Kollaborationsformat (BCF) ist eine Methode, mit der verschiedene Anwendungen im Bereich des Building Information Modeling (BIM) miteinander kommunizieren können. Diese Kommunikation ermöglicht es Projektbeteiligten, modellbasierte Probleme oder Fragen zu identifizieren, zu dokumentieren und zu lösen. BCF nutzt dabei IFC-Modelle (Industry Foundation Classes), die zuvor zwischen den Projektbeteiligten ausgetauscht wurden. IFC ist ein offener, internationaler Standard für den Austausch von BIM-Daten zwischen verschiedenen Softwareanwendungen (buildingSMART International, o. J.-b). Die Entwicklung von BCF begann im Jahr 2009 durch eine Zusammenarbeit zwischen verschiedenen Mitgliedern der BuildingSMART International Implementation Support Group (ISG). Darunter waren Unternehmen, wie Solibri und Tekla sowie das Institut für angewandte Bauinformatik (iabi) in München. Ihr Ziel war es, eine offene Kommunikationstechnologie für den Austausch von BIM-Daten zu schaffen, was schließlich zur Entwicklung von BCF führte. (buildingSMART International, o. J.-a)

2.2.2 Notwendigkeit von BCF für das Bauwesen

Im traditionellen Informationsaustausch im Bauwesen stoßen häufig unterschiedliche Charaktere und Interessen aufeinander, und nur wenige Stakeholder sind von Anfang bis Ende eines Projekts involviert. Diese Dynamik erschwert einen ganzheitlichen Blick auf das Projekt über die gesamte Laufzeit. Ein Wechsel der Beteiligten kann dazu führen, dass Informationen verloren gehen oder, dass Wissen über bereits getroffenen Entscheidungen nicht ausreichend dokumentiert oder übergeben wird. Eine Folge können unvollständige oder veraltete Informationen sein, welche für das nicht einhalten von Kosten, Überschreiten von Termin oder Mängeln oder verantwortlich sein können. Durch die Nutzung von BCF können diese Herausforderungen besser bewältigt werden, indem Informationen über den gesamten Projektlebenszyklus hinweg verfügbar und zugänglich bleiben. Die gängige Praxis im Baubereich war lange Zeit geprägt von

analogen und papierbasierten Methoden. Bevor digitale Technologien sich verbreiteten, wurden Informationen zum Beispiel häufig über Papierbriefe per Post oder über persönliche Treffen ausgetauscht. Dies umfasste auch Zeichnungen und Pläne, die von Hand erstellt und in physischer Form weitergegeben wurden. Mit dem Aufkommen digitaler Medien hat sich zwar die Form der Kommunikation verändert, aber die grundlegenden Abläufe und Prozesse sind oft noch ähnlich wie zuvor geblieben. Papierbasierte Zeichnungen wurden durch digitale Versionen ersetzt, die beispielsweise im PDF-Format versendet werden. Obwohl die Informationen nun digital gespeichert und über Netzwerke übertragen werden können, fehlt es oft an effektiven Mechanismen zur Organisation, Verwaltung und Koordination dieser Daten. Dieses Phänomen wird auch als „digitales Chaos“(Hartmann, 2022) bezeichnet. In diesem digitalen Chaos kann es schwierig sein, den Überblick über die aktuellen Daten und ihren Bearbeitungsstand zu behalten (siehe Abbildung 4). Es gibt oft Unsicherheiten darüber, wer für die Prüfung und Sortierung der Informationen verantwortlich ist, wer in den Kommunikationsprozess eingebunden werden muss und wer letztendlich die Gesamtverantwortung trägt. Das Fehlen dieser klaren Struktur und Organisation kann zu Risiken führen, die sowohl Zeit als auch Geld kosten und die Produktivität senken können. Der Hauptunterschied zur analogen Kommunikation besteht darin, dass die Informationen nun digital verfügbar sind und schneller übertragen werden können. Dennoch bleiben die Herausforderungen in Bezug auf die effektive Zusammenarbeit und Koordination bestehen. Hier setzt das Konzept von BIM an, um die Organisation von Informationsflüssen und Arbeitsabläufen zu verbessern. So soll eine effizientere und produktivere Zusammenarbeit im Baubereich ermöglicht werden können. (Hartmann, 2022)

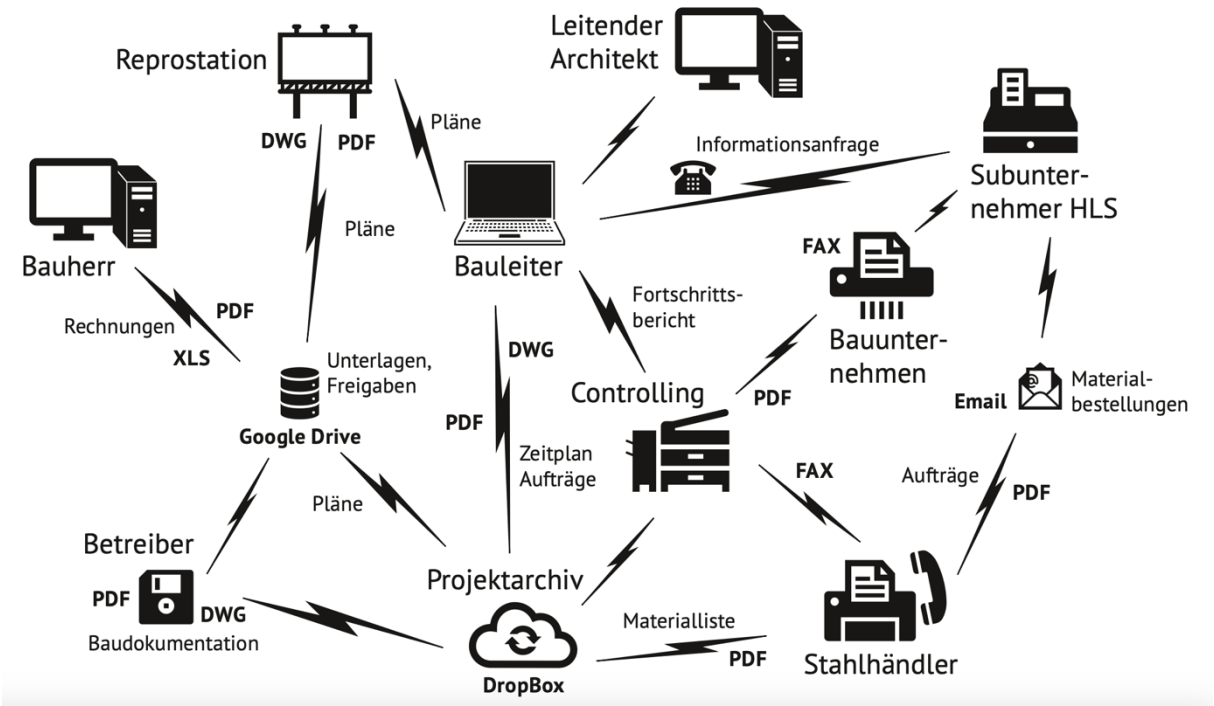

Abbildung 4 digitales Chaos (Hartmann, 2022)

2.2.3 Informationsaustausch mittels BCF

BCF wird verwendet, um die Kommunikation und Zusammenarbeit zwischen verschiedenen BIM-Softwareanwendungen zu erleichtern. Dies geschieht durch die Nutzung offener Standards wie Dateiformate und Datenkommunikationsprotokolle. Es gibt eine Reihe von Anwendungsfällen, die von BCF-fähigen Workflows profitieren können. Diese Anwendungsfälle erstrecken sich über verschiedene Phasen eines Bauprojekts: Beispielsweise können damit in der Designphase Designkoordinationsprobleme zwischen verschiedenen Bereichen (Clash Detection) identifiziert werden. In der Akquise kann BCF für die Koordinierung von Angebotsangaben und Klärungspunkten verwendet werden. Während der Bauphase kann das BCF zum Beispiel dazu dienen, Qualitätsprüfprotokolle von Installationen zu erstellen. In der Betriebsphase können Eigentümerhinweise zu benötigten Wartungsarbeiten dokumentiert werden. (buildingSMART International, o. J.-a)

2.2.4 Einordnung des BCF

Das BCF kann als ein Werkzeug der computergestützten Gruppenarbeit verstanden werden. Die Raum-/Zeit-Matrix bietet einen Kontext zu dem besseren Verständnis der Gruppenarbeitsdynamik. Die Interaktionsformen in Gruppenarbeiten werden anhand von zwei wesentlichen Kriterien strukturiert.

- Geographische Verteilung
- Zeitliche Verteilung

Dieses Modell sagt aus, ob die Kommunikation zeitgleich bzw. zeitlich versetzt stattfindet und ob sich die Teilnehmer am selben Ort befinden. Man differenziert zwischen benachbarter oder räumlich entfernter und zwischen asynchroner bzw. synchroner Kommunikation. (Teufel, 1995)

Auf den BIM-Kontext übertragen, ergeben sich aus dieser Differenzierung vier Kommunikationsformen nach (Schapke and Beetz, 2021), welche in folgender Abbildung verdeutlicht sind.

	gleichzeitig *synchron*	ungleichzeitig *asynchron*
örtlich zusammen *colocated*	**von Angesicht zu Angesicht** Besprechungsräume, gemeinsame Präsentationsflächen, Flipcharts etc.	**fortlaufend geteilter Arbeitsort** Projekträume, Baucontainer etc.
örtlich getrennt *remote*	**medial vermittelte Interaktion** Telefon, Videokonferenz, geteilte Anwendungen, (web-basierte) Mehrbenutzer-Editoren etc.	**Kommunikation und Koordination** Post, Email, Kalendersysteme, Modelserver, Versionskontrolle, Projektplattformen, Dokumentenmanagement Systeme etc.

Abbildung 5 Raum-Zeit-Matrix der Kommunikationsformen nach (Schapke and Beetz, 2021)

In diesem Schema kann das Building Information Collaboration Format als *asynchron* und *remote* eingeordnet werden. BCF unterstützt den asynchronen Austausch von BIM-bezogenen Kommentaren und Anmerkungen zwischen unterschiedlichen Projektbeteiligten. Bei der Kommunikation handelt es sich in der Regel um einen asyn-

chronen Prozess, bei dem Informationen zu diskreten Zeitpunkten ausgetauscht werden können. Akteure haben die Möglichkeit BIM-Modelle zu kommentieren, ohne dass alle gleichzeitig online oder am selben Ort sein müssen. So sind verschiedene Fachleute in der Lage, ungebunden von Zeit und Ort auf das Modell zugreifen und Feedback zu geben.

Ebenso verkörpert BCF das Schema der direkten und indirekten Kommunikation. BIM-Teams können unmittelbar miteinander kommunizieren, Rückmeldung geben, Diskussionen führen und spezifische Hinweise im Modell platzieren. Gleichzeitig bietet die gemeinsame Nutzung der Ressource des BIM-Modells einen zentralen Bezugspunkt für die Nachverfolgung von Lösungen und Entscheidungen. Somit handelt es sich um eine kollaborative Umgebung, welche eine strukturierte und kontextbezogene Interaktion fördert. Die Idee der direkten und indirekten Kommunikation basiert auf (Dix et al., 2004). Folgende Abbildung soll das eben genannte Konzept visualisieren.

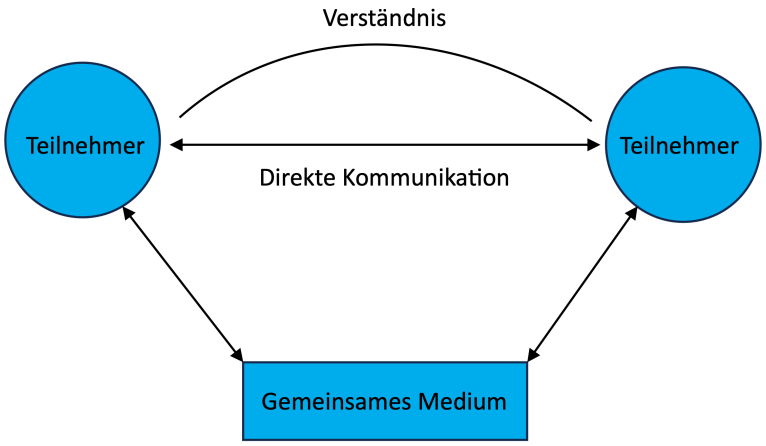

Abbildung 6 Direkte und indirekte Kommunikation nach (Dix et al., 2004)

Das BCF fungiert dabei als ein standardisiertes Protokoll für die Kommunikation und den Datenaustausch zwischen verschiedenen BIM-Softwareanwendungen. Insbesondere wird durch BCF die Übermittlung spezifischer Informationen wie Kollisionen, Probleme, Anforderungen und weitere relevante Themen im Zusammenhang mit dem BIM-Modell erleichtert. (Steinmann, 2021)

2.2.5 BCF-unterstützende Produkte

Heute ist BCF ein offener Standard von BuildingSMART International und steht allen Entwicklern zur Verfügung. Entwickler können BCF in ihren Produkten unterstützen und weitere Informationen über die offenen GitHub-Repositories von buildingSMART International erhalten. (buildingSMART International, o. J.-a)

Viele führende BIM-Produkte unterstützen bereits das BCF-Format, wodurch Benutzer von verbesserten Möglichkeiten zur Kollaboration und Fehlerverfolgung profitieren. Das Dateiformat ist in einer Vielzahl gängiger BIM-Authoring-Tools wie Revit, Archi-CAD, Allplan, Tekla, Plancal, DDS CAD usw., sowie Prüfprogrammen wie Solibri, Tekla BIM Sight, Naivsworks usw. integriert. Diese Plattformunabhängigkeit macht es zu einer fundamentalen Säule der Open BIM Methode. (BIMpedia, o. J.)

Die Nutzung von BCF variiert je nach Software und Version. Einige Softwareprodukte ermöglichen es den Benutzern, BCF für die Kommunikation von Anmerkungen und Problemen in 3D-Modellen zu verwenden. Beispielsweise ermöglicht Solibri es, BCF für die Erfassung und Diskussion von Qualitätsproblemen in BIM-Modellen zu nutzen, Andererseits bietet BIMcollab eine umfassende cloudbasierte Plattform, die die Verwendung von BCF für die Zusammenarbeit über verschiedene Disziplinen und Phasen eines Projekts ermöglicht.

Die Unterstützung von BCF kann von Softwareversion zu Softwareversion unterschiedlich sein. Es ist ratsam, die neuesten Versionen der Softwareprodukte zu verwenden, um von den verbesserten Funktionen und der bestmöglichen BCF-Integration zu profitieren. In einigen Fällen kann die Integration von BCF durch APIs (Application Programming Interfaces) erfolgen, während andere Softwareplattformen auf BCFXML setzen, wie in Tabelle 1 ersichtlich wird.

Softwareprodukt	Standards
Solibri Office	- bcfXML 2.0 - bcfXML 2.1 - bcfXML 3.0 - bcf API 2.1 - bcf API 3.0
BIM.works	- bcfXML 2.0 - bcfXML 2.1 - bcfXML 2.2 - bcfXML 3.0
BIMcollab Cloud	1. bcfXML 2.2 2. bcf API 2.2
ARCHICAD	3. bcfXML 2.2
Tekla BIMsight	4. bcfXML 2.2
Allplan Architecture	5. bcfXML 2.2 6. bcf API 2.2
Tekla Structures	7. bcfXML 2.2

Tabelle 1 jeweilige Standards der Softwareprodukte (buildingSMART International, o. J.-c)

2.3 Struktur von BCF

Bestandteile eines BCFs, sowie ihre Hierarchie und Funktionen sollen nun untersucht werden. Dafür werden die Grundbausteine eines BCFs erläutert. Besonders thematisiert wird die Markup-Datei. Auch wird erläutert, wie BCFs ausgetauscht werden können. So soll ein detailliertes Verständnis über die inneren Abläufe und Komponente des Building Information Collaboration Formats vermittelt werden.

2.3.1 Aufbau von BCFs

Das BCF-Datenmodell hat sich im Verlauf vieler Jahre durch die kollaborative Arbeit verschiedener internationaler Akteure kontinuierlich weiterentwickelt und ist mittlerweile als etablierter Standard anerkannt. Der Aufbau lässt sich in mehrere Schlüsselkomponenten gliedern, die im Folgenden gemäß der GitHub-Dokumentation (Kulbak, o. J.) näher erläutert werden.

Der Aufbau einer BCF-Datei entspricht einem ZIP-Archiv, das einen Ordner für jedes Thema (Topic) enthält. Hierbei war die Dateiendung „bcfzip“ für BCFv1.0 und BCFv2.0 gängig, während mit BCFv2.1 die Endung „bcf“ eingeführt wurde.

Die Grundbausteine bilden Projects (Projekte), Topics (Themen), Comments (Kommentare) und Viewpoints (Kameraeinstellungen). Zusammen formen diese ein Issue (Problem) und sind hierarchisch organisiert (siehe Abbildung 7). Dabei stellen Topics die Verbindung von Viewpoints und Comments dar. (Schulz et al., 2021) S.4

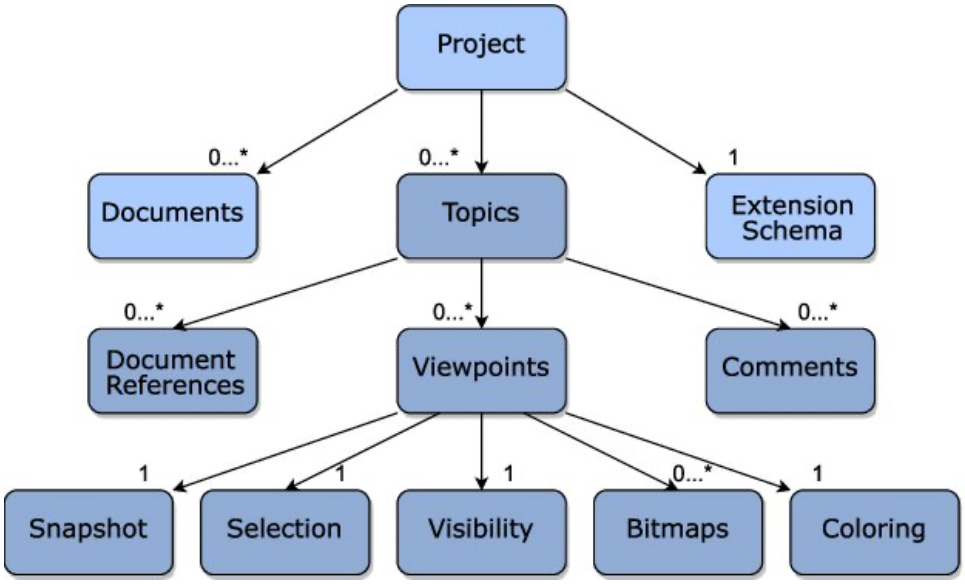

Abbildung 7 Allgemeine Struktur eines BCFs. Die dunkelblauen Rechtecke bilden das Issue. (Schulz et al., 2021)

Der Wurzelordner des BCF-Zips beinhaltet eine Reihe von Schlüsseldateien, die einen strukturierten Informationsaustausch ermöglichen.

- **extensions.xml**
  Diese XML-Datei definiert die Erweiterungen eines Projekts und folgt dem Schema extensions.xsd. Hier werden zusätzliche Projektaspekte oder spezielle Anforderungen festgelegt.
- **project.bcfp (optional)**
  Diese XML-Datei gibt detaillierte Informationen über das Projekt selbst an und folgt dem Schema projekt.xsd. Projektmanager können hier spezifische Details hinzufügen, um ein umfassenderes Bild des Projektkontexts zu erhalten. Diese Datei ist optional, aber dennoch mächtig.
- **documents.xml (optional)**

Diese optionale, wie auch nützliche XML-Datei definiert die Dokumente im Projekt und folgt dem Schema documents.xsd. Sie ermöglicht die Verknüpfung von projektbezogenen Dokumenten und erhöht die Nachvollziehbarkeit.

- **bcf.version**
  Diese XML-Datei folgt dem version.xsd-Schema. Sie gibt Auskunft über das verwendete BCF-Schema und ihr Inhalt ist für BCFv3.0 wie folgt definiert.

```
<?xml version="1.0" encoding="UTF-8"?>

<Version VersionId="3.0" xsi:noNamespaceSchemaLocation="https://raw.githubusercontent.com/buildingSMART/BCF-XML/release_3_0/Schemas/version.xsd"

    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">

</Version>
```

Formel 1 Inhalt der bcf.version-Datei für die Version 3.0 (Kulbak, o. J.)

Die BCF-Struktur ermöglicht auch die Speicherung zusätzlicher Dateien im BCF-Container als Dokumente. Diese Dokumente müssen sich in einem Ordner namens „Documents“ im Stammverzeichnis befinden und müssen von der documents.xml-Datei referenziert werden. Um die Eindeutigkeit sicherzustellen, muss der der Dateiname eines Dokuments im BCF die Dokument-GUID sein, während der tatsächliche Dateiname in documents.xml gespeichert wird.

Innerhalb des BCF-ZIP-Archivs befinden sich die Topics (siehe Abbildung 9). Topics müssen mit Daten gefüllt sein, die in Extension Schema (extensions.xml) festgelegt sind (Schulz et al., 2021). Die Benennung der jeweiligen Ordner der Topics entspricht der GUID (Globally Unique Identifier) des betreffenden Objekts im IFC-Modell und liegt im UUID-Format vor. Diese GUID darf ausschließlich aus Kleinbuchstaben bestehen. Sie dient als eindeutiges Merkmal für das betreffende Thema (Topic). Der zugehörige Ordner enthält die Datei markup.bcf, eine XML-Datei, die dem Markup.xsd-Schema folgt. In Abbildung 8 ist der eben erläuterten Aufbau eines BCF-Zips zusammengefasst schematisch illustriert.

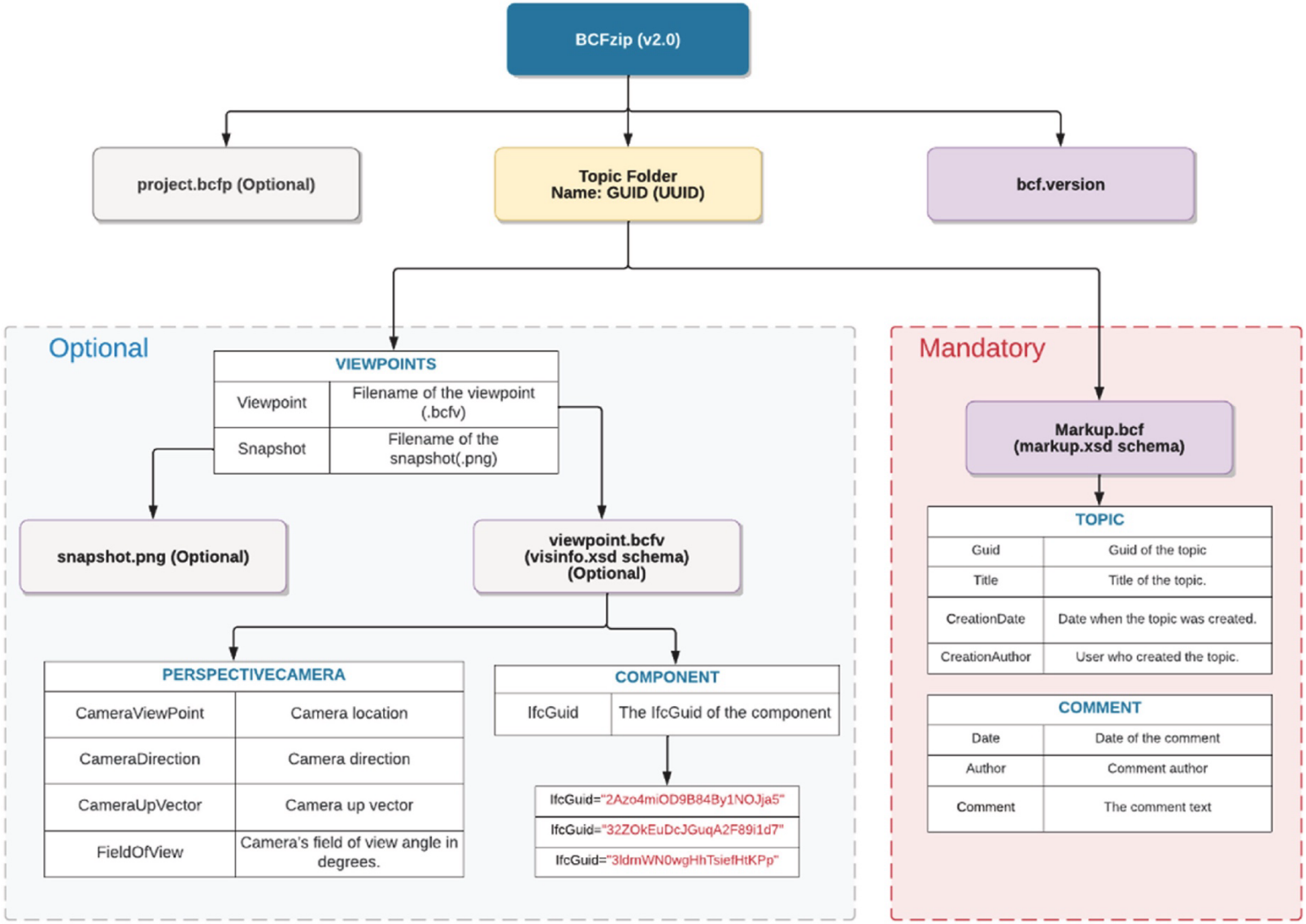

Abbildung 8 Bestandteile einer BCF-Zip-Datei aus (Khorchi & Boton, 2024)

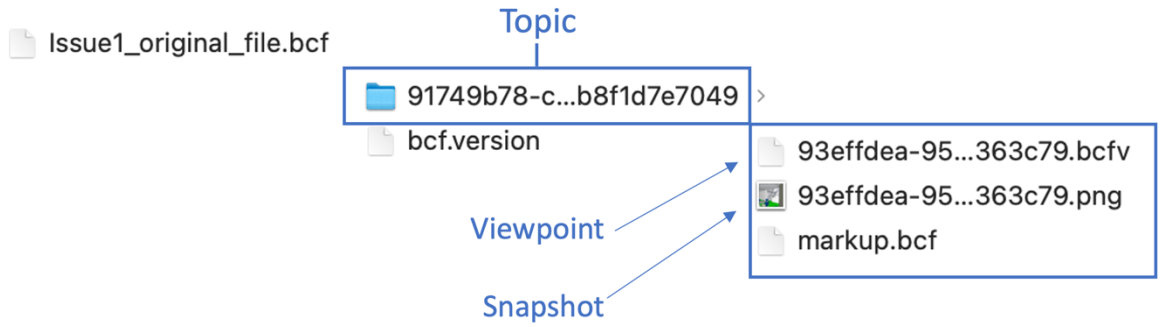

Abbildung 9 Nutzeransicht des Inhalts eines entpackten BCF-Zip-Ordners

Das Markup bildet das Herzstück, denn es enthält strukturierte textuelle Informationen zum Topic. Enthalten können Details sein, wie beispielsweise das Stockwerk oder die Ebene des betroffenen Bauteils, das Datum der Identifizierung, den Autor des Markups, die Priorität des Problems, die zugewiesene verantwortliche Partei und den aktuellen Status des Lösungsprozesses. Diese Details bieten ein umfassenderes Verständnis des Problems im breiteren Projektzusammenhang. Abbildung 10 visualisiert den hierarchischen aufgebauten Inhalt der „markup.bcf“-Datei. Die Knoten im Graph repräsentieren die verschiedenen Elemente, die in der Datei enthalten sein können. Diese Elemente, wie beispielsweise Topic, Comment, Title oder Guid, werden als Knoten im Graph dargestellt. Innerhalb dieser Hierarchie sind bestimmte Elemente untergeordnet oder übergeordnet.

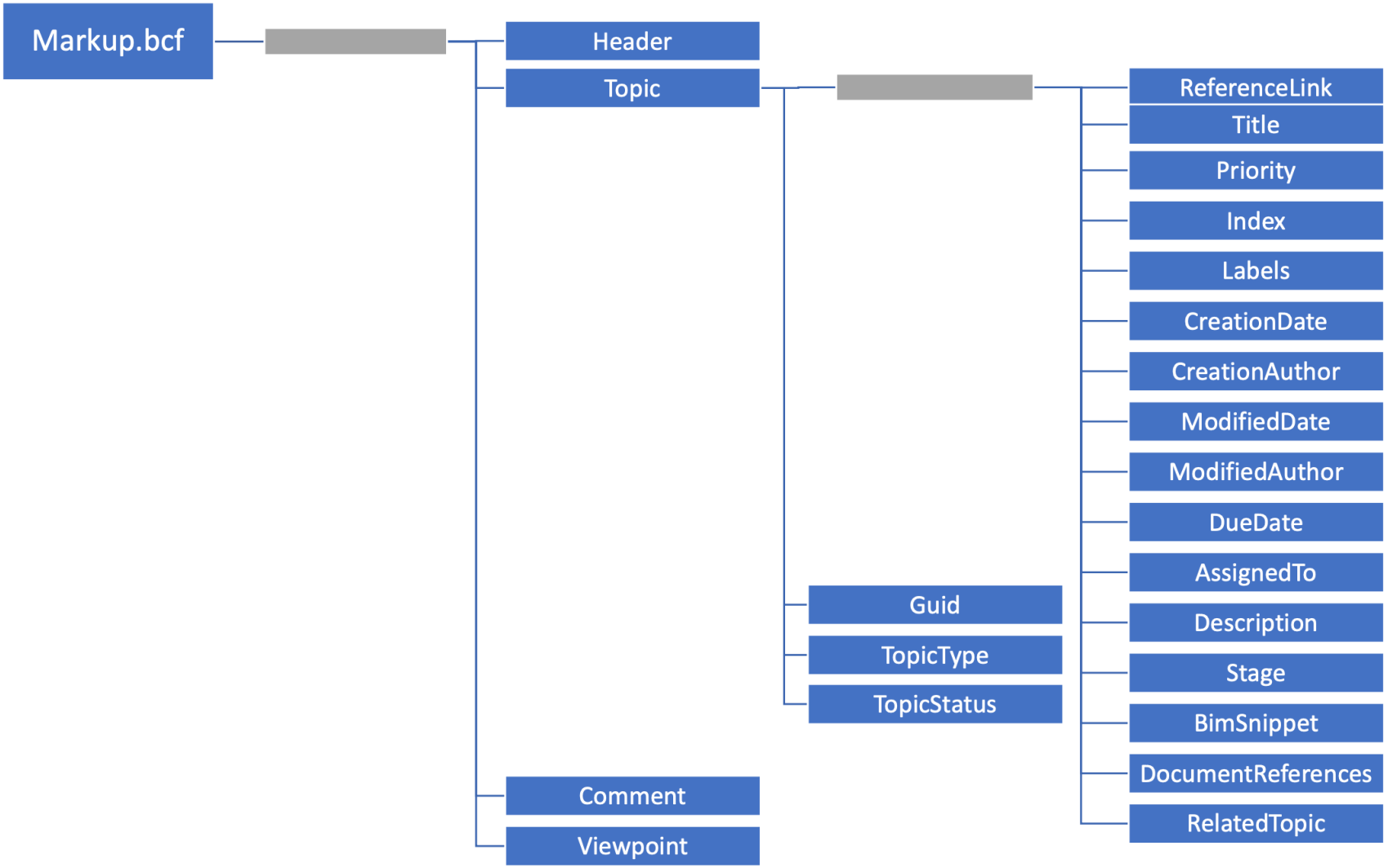

Abbildung 10 Markup als Graph dargestellt gemäß (Schatz, o. J.)

Vor allem aber integriert die Markup-Datei die GUID des Objekts im Modell der Industry Foundation Classes (IFC). Diese dient als eindeutiger Identifikator und verknüpft die BCF-Daten direkt mit dem spezifischen Element im IFC-Modell. Neben dieser Hauptdatei können im Topic-Ordner jeweils weitere Dateien enthalten sein.

- **Viewpoint-Dateien**
  Der Ordner kann zusätzliche XML-Dateien enthalten, die dem visinfo.xsd-Schema entsprechen. Diese Dateien werden als Viewpoint-Dateien bezeichnet und tragen die Erweiterung „.bcfv“. Ihre Namen sollten mit einem der Viewpoint-Elemente übereinstimmen. Diese Dateien definieren die Visualisierung des betroffenen Bauteils im 3D-Modell, einschließlich Kameraposition, -ausrichtung, Farben und geometrische Linien.
- **Snapshot-Dateien**
  Im Ordner sind sowohl PNG- als auch JPEG-Formate erlaubt. Dabei darf die Längendimension 1500 Pixel nicht überschreiten. Die Dateinamen müssen mit einem der Snapshot-Elemente übereinstimmen
- **Bitmaps**

Die Idee der Viewpoints stellt eine räumliche Verbindung zum BIM-Modell her und gibt Auskunft darüber, welche Teile des Gebäudes in der jeweiligen Kameraperspektive sichtbar sind oder ausgewählt wurden. Die Positionen und Rotationen werden dabei in einem kartesischen Koordinatensystem mit Vektoren X, Y und Z festgelegt. (Schulz et al., 2021)
Außerdem ermöglicht die Markup-Datei die Aufnahme von Hyperlinks oder Verweisen auf externe Dokumente, wie beispielsweise Mängelberichte oder Abnahmeprotokolle. Dies schafft eine nahtlose Verbindung zwischen modellbasierten und dokumentbasierten Systemen und verbessert die Zugänglichkeit und Nachverfolgbarkeit relevanter Informationen im Zusammenhang mit dem identifizierten Problem.
Die Organisation dieser eben aufgeführten Elemente innerhalb des Ordners gewährleistet eine klare Struktur und eine einheitliche Reihenfolge. Dies ist entscheidend, um die Konsistenz der Informationen sicherzustellen und die Zusammenarbeit zwischen verschiedenen BIM-Softwareanwendungen zu realisieren. Die Einhaltung der vorgegebenen Reihenfolge der Elemente ist zwingend und stellt sicher, dass die Daten korrekt interpretiert und verarbeitet werden können.

2.3.2 Ablauf eines BCF-Austauschs

Für den Austausch von BCFs gibt es zweit Hauptmöglichkeiten- dateibasiert und webbasiert, welche im Folgenden erläutert werden.

Der dateibasierte Austausch von BCF-Daten erfolgt durch den Transfer von Zip Foldern von einem Benutzer zum anderen. Nachdem eine eine BCF-Datei bearbeitet wurde, kann sie wieder zurückgesendet werden. Dabei kommen verschiedene Kanäle in Frage, zum Beispiel per E-Mail, Dateifreigabeplattformen oder Cloud-Speicherdienste. Es ist eine einfache und weit verbreitete Methode, erfordert jedoch manuellen Aufwand beim Versand, Empfang und Zusammenführen von Informationen. Ein wichtiger Aspekt beim dateibasierten Austausch ist die Möglichkeit des „Roundtripping“. Das bedeutet, dass die Daten hin und her zwischen den Beteiligten zirkulieren können, solange die Integrität der gemeinsam genutzten BCF-Datei gegeben ist und keine veralteten Kopien im Umlauf sind. Wie ein dateibasierter BCF-Austausch aussehen kann, illustriert folgende Abbildung. (buildingSMART International, o. J.-a)

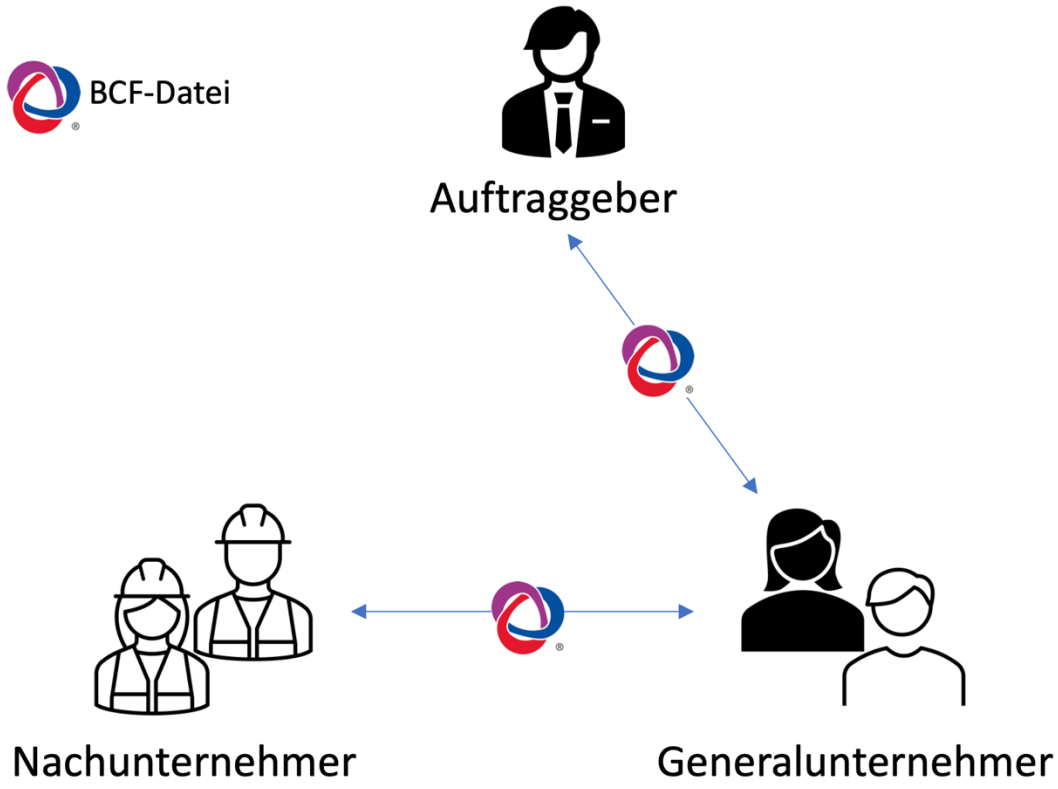

Abbildung 11 BCF-Austausch dateibasiert

Eine Alternative zum dateibasierten Workflow stellt der webbasierte (RESTful) API-Modus dar (siehe Abbildung 12) (Kulbak, o. J.). Dies beinhaltet die Implementierung

eines BCF-Servers, der als zentraler Speicher für alle BCF-Daten fungiert. Der BCF-Server kann dabei auch als BIM-Server dienen. Durch die Verwendung eines solchen Servers können Projektteilnehmer die Erstellung, Bearbeitung und Verwaltung von BCF-Themen an einem zentralen Ort synchronisieren. (buildingSMART International, o. J.-a)

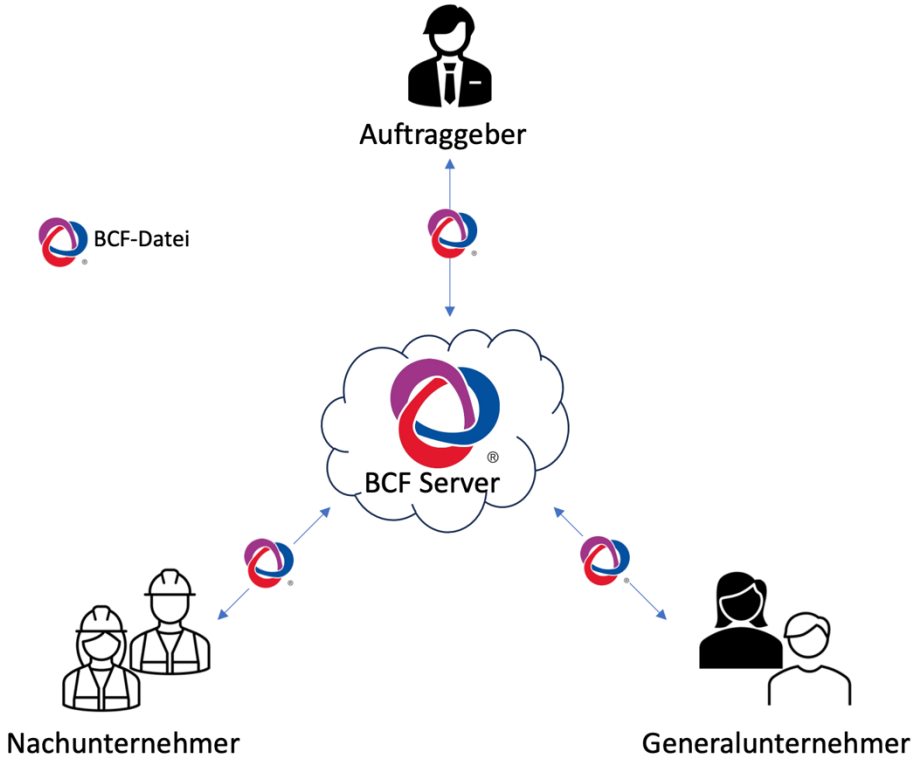

Abbildung 12 BCF-Austausch API-basiert

Eine RESTful API (Representational State Transfer Application Programming Interfaces) ist eine weit verbreitete Methode für den Austausch von Daten über das Web. Es handelt sich um eine Schnittstelle zweier Computersysteme, über die sicher Informationen transferiert werden können. (Amazon Web Services, o. J.)

APIs dienen somit als Vermittler von Daten. Sie initiieren Datenanfragen und führen Datenabfragen durch, prüfen Autorisierungen und stellen Daten bereit. Entscheidend ist dabei, dass zwischen den Anwendungen keine Interaktion der Benutzer erforderlich ist.REST APIs verwenden eindeutige URIs (Uniform Resource Identifiers), die auf dem HTTP(s)-Protokoll basieren und JSON (JavaScript Object Notation) als Datenformat nutzen. Durch die klaren Standards und Restriktionen gelten REST APIs als vergleichsweise einfach zu implementieren. (Bally and Brogini, 2020)

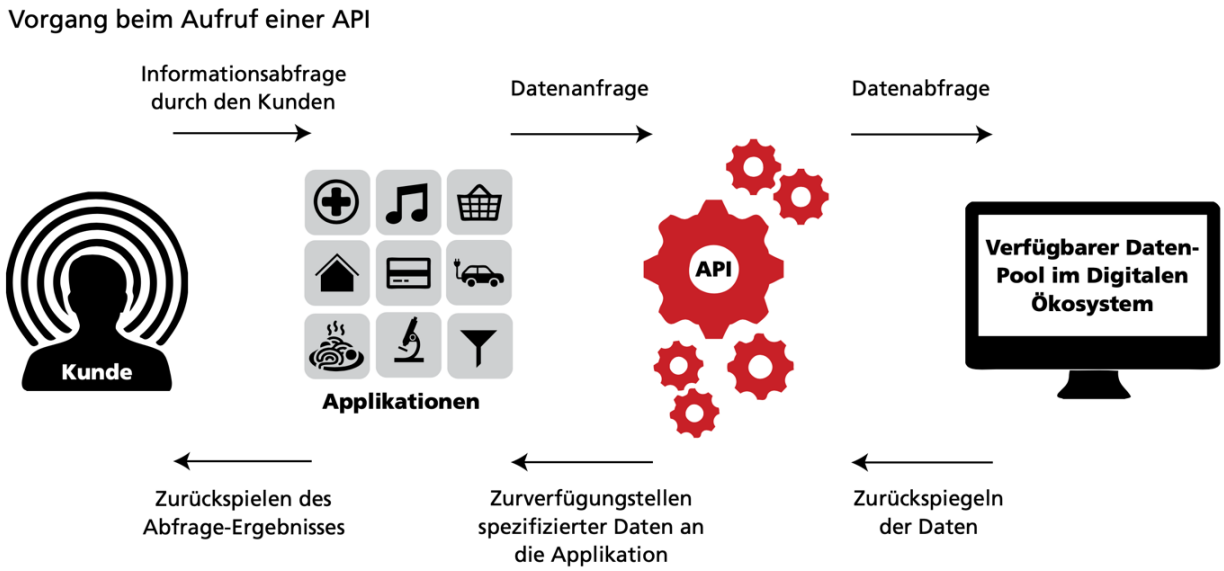

Abbildung 13 APIs als Datenvermittler ohne Benutzerinteraktion zwischen Anwendungen (Bally and Brogini, 2020)

Die Wahl zwischen diesen Methoden hängt von den Anforderungen und Präferenzen der Beteiligten im BIM-Projekt ab. Die dateibasierte Methode ist einfach und weit verbreitet, während die API-basierte Methode eine automatisierte und nahtlose Integration in bestehende Arbeitsabläufe ermöglicht.

Der Austausch von BCFs erfolgt in der Regel durch den Transfer der entsprechenden ZIP-Archive zwischen verschiedenen BIM-Softwareanwendungen oder Plattformen. Ein möglicher Vorgang kann wie folgt ablaufen:

1. **Erstellung der BCF-Datei:**

   Ein BCF wird erstellt, um spezifische Informationen zu einem bestimmten Thema, wie ein Designproblem oder ein bestimmtes Szenario, zu behandeln. Dies geschieht durch Erstellung einer ZIP-Datei, die die benötigten XML-Dateien und gegebenenfalls zusätzliche Dokumente enthält.

2. **Übertragung der BCF-Datei:**

   Die erstellte BCF-Datei wird anschließend zwischen den Projektbeteiligten übertragen. Es besteht die Möglichkeit eines dateibasierten oder eines webbasierten Austausches.

3. **Empfang und Entpacken:**

Der Empfänger erhält die BCF-Datei und entpackt das ZIP-Archiv. Dadurch werden die XML-Dateien und gegebenenfalls zusätzliche Dokumente sichtbar.

4. **Datenverarbeitung durch die BIM-Software:**

   Die BIM-Software des Empfängers verarbeiten die empfangenen XML-Dateien und integriert die Informationen in das bestehende BIM-Modell. Dies erlaubt es den Empfängern, die spezifischen Themen zu visualisieren und die darin enthaltene Mitteilung zu verstehen.

5. **Kollaboration und Aktualisierung:**

   Nach dem Import der BCF-Datei können die Projektbeteiligten die detaillierten Informationen über das Thema prüfen und darauf reagieren. Sie können zum Beispiel Kommentare hinzufügen, Aktualisierungen vornehmen oder Anmerkungen zu den dargestellten Szenen machen.

2.4 Identifizierte Forschungslücke

Der Austausch von Daten in Bauprojekten entwickelt sich kontinuierlich weiter. Insbesondere die Einführung von BIM hat Auswirkungen auf die Arbeitsweise in der Baubranche. BIM ermöglicht es Informationen eines Bauprojekts digital zu modellieren, verwalten und auszutauschen. Dadurch können Planung, Ausführung und Betrieb von Gebäuden effizienter gestaltet werden. In diesem Zusammenhang spielt der Datenaustausch eine entscheidende Rolle, um sicherzustellen, dass alle Beteiligten stets auf die aktuellen Informationen zugreifen können.

Ein Ansatz, der Modell und Kommunikation voneinander trennt ist das zuvor erläuterte BIM Collaboration Format (BCF). Als herstellerneutraler Datenaustauschstandard, unterstützt er die modellbasierte Kommunikation. Es ermöglicht den Import und Export von BCF-Daten, entweder dateibasiert oder über eine Webschnittstelle. BCF basiert strukturell auf XML, einem etablierten Standard für den Austausch und die Speicherung von Daten.

Obwohl der Inhalt eines einzelnen BCF-Containers in der Regel von geringem Umfang ist, kann durch die Anwendung inkrementeller Methoden ein effizienter Datenaustausch erzielt werden. Hierbei wird ausschließlich die Modifikationsinformation zwischen zwei Dateiversionen übertragen, anstelle des gesamten Containers.

Durch die XML-basierte Struktur von BCF-Dateien ist es möglich, das Wissen und die Erkenntnisse aus dem Bereich der XML-Verarbeitung auf die Änderungserkennung in BCF-Dateien zu übertragen. XML ermöglicht es Daten in einer hierarchischen Form zu organisieren und zu speichern.

Da XML eine etablierte und weit verbreitete Methode für den Austausch und die Speicherung von Daten ist, existieren bereits ein umfangreicher wissenschaftlicher Fundus und bewährte Methoden zur Verarbeitung und Analyse von XML-Daten. Diese sollen nun spezifisch in Hinblick auf die Anforderung und Herausforderungen der Änderungserkennung in BCFs untersucht werden.

Durch Testbeispiele soll ermittelt werden, inwieweit die bereits vorhandenen Erkenntnisse im Bereich der XML-Verarbeitung auf die spezifischen Anforderungen und Herausforderungen der Änderungserkennung in BCF-Dateien anwendbar sind. Dabei werden nicht nur die technischen Aspekte, sondern auch die praktischen Implikationen und mögliche Einschränkungen dieser Übertragung untersucht.

Die Implementierung inkrementeller Methoden eröffnet somit die Möglichkeit, den Datenaustausch in BIM-basierten Projekten weiter zu optimieren und die Effizienz der Kommunikation zu steigern. Insbesondere in Projekten, in denen eine Vielzahl von Änderungen häufig vorkommt, kann diese Methode von großer Relevanz sein.

Zudem kann eine Änderungserkennung eine präzise Möglichkeit bieten, den Verlauf der Modifikationen in einem Bauprojekt nachzuverfolgen und zu dokumentieren. Dies könnte entscheidend sein für die Versionierung und Revisionierung, da Bauprojekte oftmals eine Vielzahl von Änderungen durchlaufen, die über die Zeit hinweg dokumentiert werden müssen. Durch die Modifikationsinformationen könne alle vorgenommenen Änderungen an den BCF-Dateien genau erfasst werden, einschließlich Details wie hinzugefügte Elemente, gelöschte Elemente und geänderte Eigenschaften. Diese Informationen ermöglichen es den Projektbeteiligten, den Verlauf der Änderungen in einem Bauprojekt genau zu verfolgen und zu verstehen, wie sich das Modell im Laufe der Zeit entwickelt hat. Die Versionierung und Revisionskontrolle ermöglicht es den Projektbeteiligten, auf frühere Versionen der BCF-Dateien zuzugreifen und Änderungen rückgängig zu machen oder wiederherzustellen, falls erforderlich. Indem der Verlauf der Modifikationen transparent und nachvollziehbar wird, können potenzielle Konflikte vermieden werden.

3 Übertragung von XML-Methodiken auf den BCF-Kontext

Nachdem die identifizierte Forschungslücke hinsichtlich der Änderungserkennung in BCF-Dateien erörtert wurde, ist es nun notwendig, einen theoretischen Rahmen zu schaffen, der die Grundlagen von XML umfasst. Dieser Rahmen soll die Grundlage für die anschließende Analyse und Bewertung der Anwendbarkeit vorhandener XML-basierter Methoden auf die Änderungserkennung in BCFs bilden.

Zunächst wird auf die grundlegenden Konzepte und Strukturen von XML eingegangen, um ein Verständnis für die Funktionsweise dieses Datenformats zu schaffen. Anschließend werden spezifische XML-basierte Methoden zur Verarbeitung und Analyse von XML-Daten vorgestellt und erläutert.

3.1 XML

Grundlegende Konzepte und Ansätze zur Änderungserkennung in XML-Dateien werden vorgestellt, um ein Verständnis dafür zu vermitteln, wie diese Änderungen in einem strukturierten Datenformat wie XML effektiv erfasst und verarbeitet werden können.

3.1.1 Potential von XML

XML-basierte Dokumente spielen eine wichtige Rolle in heutigen Informationsstrukturen und den dazugehörigen Arbeitsabläufen. In heutigen Arbeitsumgebungen gehen die dynamischen Beziehungen zwischen Teams häufig über herkömmliche Hierarchien hinaus. Dass viele Personen Teil verschiedener Teams sind, resultiert in einer Vielzahl von Beziehungen. In der Praxis hat sich ergeben, dass Dokumente per Mail ausgetauscht werden und die verschiedenen Versionen manuell verglichen werden. Dies ist sehr zeitaufwändig und fehleranfällig. Darin liegt ein großes Potential zur Verbesserung der Effizienz im Informationsaustausch. (Rönnau and Borghoff, 2012) Die dynamischen Beziehungen zwischen Teams und die Notwendigkeit, Dokumente zwischen verschiedenen Personen auszutauschen, haben zu einem verstärkten Bedarf an einer effizienten Möglichkeit geführt, Änderungen in Dokumenten zu identifizieren und zu verfolgen.

Eine mögliche Lösung stellt das sogenannte „diff“ dar. Es identifiziert Unterschiede zwischen zwei unterschiedlichen Versionen eines Dokuments und stellt diese dar. Von gleicher Bedeutung ist auch das Zusammenführen zweier Dokumentversionen, welcher einer parallelen Bearbeitung entstammen. Dieses Zusammenführen wird als „patch“ bezeichnet. Der Vergleich zweier XML-Dokumente kann als spezieller Fall des Tree-to-Tree-Editierproblems betrachtet werden. (Rönnau and Borghoff, 2012)

Das Tree-to-Tree-Editierproblem (engl. tree-to-tree editing problem) wurde 1977 von Selkov definiert. Es beschäftigt sich mit der Herausforderung, zwei hierarchische Baumstrukturen miteinander zu vergleichen und zu editieren. Konkret geht es darum, wie man von einem Baum zu einem anderen Baum gelangt, indem man eine minimale Menge von Bearbeitungsoperationen durchführt. Beispiele für Bearbeitungsoperationen sind: Einfügen, Löschen oder Umbenennen von Knoten, um den ersten Baum in den zweiten Baum zu transformieren. Das Ziel besteht darin, eine „Delta“-Liste von Bearbeitungsoperationen zu erstellen. Angewendet auf den Ausgangsbaum soll diese den Zielbaum erzeugen. (Selkow, 1977) Übertragen auf den XML-Kontext handelt es sich bei der Erzeugung des Deltas um das „diff“ und beim Anwenden des Deltas um den „patch“.

Eine Umsetzung des „diffs“ birgt gemäß (Cobena et al., 2002) großes Potential hinsichtlich mehrerer Aspekte:

Versionierung und Abfragen der Vergangenheit: Indem die Ausgabe des „diffs“ als XML-Dokument gespeichert wird, kann dieses zur Versionierung als Aufzeichnung von Historie nutzbar gemacht werden. Später könnte man eine Abfrage zur Vergangenheit stellen wollen, z.B. nach dem Wert eines bestimmten Elements zu einem früheren Zeitpunkt fragen. Es handelt sich hierbei um eine reguläre Dokumentabfrage, da die Differentialausgabe als XML-Dokument gespeichert wird.

Lernen über Änderungen: Das „diff“ ermöglicht eine Beschreibung der vorgenommenen Änderung zwischen Versionen eines XML-Dokuments. So kann nicht nur die alte Version aktualisiert werden, sondern auch die Änderungen erläutert werden. In dem Fall, dass verschiedene Benutzer dasselbe XML-Dokument offline bearbeiten und anschließend synchronisieren wollen, kann das „diff“ die geschehenen Änderungen erkennen und beschreiben. So können etwaige Konflikte identifiziert und gelöst werden.

Überwachung von Änderungen: Das Delta kann für ein Abonnement System genutzt werden, das Änderungen in XML-Dokumenten erkennt und Benachrichtigungen über relevante Änderungen sendet.

Es gibt verschiedene Ansätze zur Umsetzung der Änderungserkennung, die sich hinsichtlich ihrer Geschwindigkeit und Genauigkeit bei der Erfassung von Änderungen zwischen den XML-Dokumentversionen unterscheiden. Aus dieser Notwendigkeit und dem wirtschaftlichen Interesse an diesem Verfahren ist ein großer Fundus an wissenschaftlichen Erkenntnissen in diesem Bereich entstanden. So beschäftigte sich Chawathe (Chawathe and Garcia-Molina, 1997a) bereits im Jahr 1997 mit der Erkennung von Änderungen in hierarchisch strukturierten Daten, indem Operationen wie das Verschieben oder Kopieren von Teilbäumen genutzt werden, um semantisch sinnvolle Änderungsbeschreibungen zu ermöglichen. Ein heuristischer Algorithmus wird vorgestellt, der Änderungen mit minimalen Beschreibungen liefert. Fraser stellt eine zustandsbasierte Methode zur Synchronisierung von Dokumenten vor, der eine Baumtopologie für die Synchronisation verwendet. Änderungen werden erkannt, indem der aktuelle Zustand mit dem vorherigen verglichen wird, um Unterschiede zu finden. Diese Unterschiede werden dann als Aktualisierungen weitergegeben.(Fraser, 2009)

Im Gegensatz zu BCF, das sich noch in der Entwicklungsphase befindet, ist XML ein langjährig etablierter Standard. Der große Vorteil dieser bereits vorhandenen Errungenschaften im XML-Bereich, ist ihr Potential für die Anwendung im BCF-Bereich. Wie in zuvor bereits erläutern, ist die Struktur von BCF-Dateien auf XML-basiert. Diese Tatsache eröffnet die Möglichkeit, dass Methoden und Techniken, die in der XML-Verarbeitung entwickelt wurden, auch auf die Verarbeitung von BCF-Dateien angewendet werden können. Um die Übertragbarkeit von XML-Wissen auf den BCF-Bereich zu untersuchen, müssen zuerst die Grundlagen von XML (Extensible Markup Language) behandelt werden. Dies legt den theoretischen Rahmen für die folgende Analyse und Bewertung der Anwendbarkeit vorhandener XML-basierter Methoden auf die Änderungserkennung in BCF-Dateien fest.

3.1.2 Struktur von XML

Das Verständnis der Struktur eines XML-Dokuments ist entscheidend, um darin zu navigieren. Ihr Aufbau soll nun gemäß (Becher, 2022a) erläutert werden. XML-Doku-

mente bestehen aus „Elementen“. Ihr Inhalt kann andere Elemente, Text, eine Kombination aus beiden oder leer sein. Jedes Element hat einen Namen. Ein Element ist aus einem Start-Tag, dem entsprechenden Ende-Tag und einem Inhalt aufgebaut, wie in Abbildung 14 verdeutlicht.

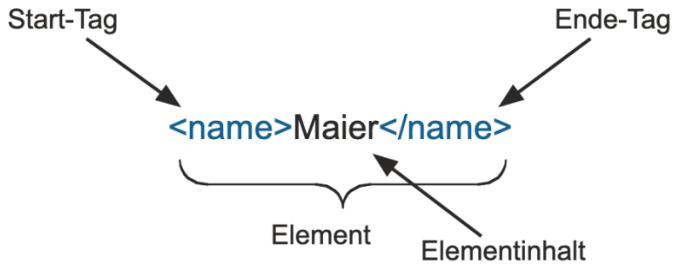

Abbildung 14 Ein einfaches Element (Becher, 2022a)

Als „Elternelement“ wird ein Element bezeichnet, das andere Elemente beinhaltet. Die untergeordneten Elemente bezeichnet man als „Kindelemente“ oder „Unterelemente“. Das erste Element eines Dokuments ist immer das Wurzelelement, welches alle anderen Elemente beinhaltet. Aus diesem Grund formt jedes XML-Dokument eine Baumstruktur von Elementen. Das Wurzelelement ist wie folgt definiert:

> Ein eindeutiger Knoten oder Vertex eines Graphen von dem aus jeder andere Knoten erreicht werden kann (Gardner, 2001).

Da XML immer verbreiteter genutzt wurde, ergab sich eine Nachfrage nach einer Sprache, die komplexe Abfragen an XML-Dokumente ermöglicht, ganz wie mit SQL bei Datenbanken. Aus diesem Grund wurde vom Word Wide Web Consortium (W3C) eine Arbeitsgruppe beauftragt, um eine solche Abfragesprache zu entwickeln.(Becher, 2022b) Das W3C ist eine international tätige, gemeinnützige Organisation. Sie befasst sich mit der Entwicklung und Förderung für Standards für das Word Wide Web(Word Wide Web Consortium, o. J.). Sie verabschiedeten im Jahr 1999 die Version 1.0 von XPath, einer Unterstützungssprache, um auf Strukturbestandteile eines XML-Dokuments zuzugreifen und diese zu adressieren (Becher, 2022b).

Die Navigation mittels XPath in einem XML-Dokument bezieht sich auf das Ansprechen verschiedener Knoten in Bezug auf ihre Beziehungen zueinander. Durch Baumstruktur von Elementen eines XML-Dokuments und das Verwenden von Begriffen wie „Elternelement“ und Kindelement“, liegt der Vergleich mit einem Familienstammbaum

nahe. Ein Stammbaum verfolgt die Eltern, Großeltern und andere Vorfahren. XML verwendet die gleiche familiäre Terminologie, um die Beziehungen zwischen den Knoten eines Dokuments zu beschreiben. (Gardner, 2001) Diese Analogie erleichtert das Verständnis der Struktur und Beziehungen innerhalb eines XML-Dokuments, indem sie vertraute Konzepte wie Familienbeziehungen auf das Verständnis von Dokumentstrukturen überträgt.

Der Name XPath wurde gewählt, um auf seine hauptsächliche Notationssyntax, nämlich den Pfad, hinzuweisen. Diese Pfade werden verwendet, um Knoten zu finden, auszuwählen, zu verändern oder zu zählen. Somit stellen Pfade das Grundgerüst von XPath dar.(Gardner, 2001)

XPath definiert sieben verschiedene Knotenarten, welche in nachfolgender Tabelle gemäß (Becher, 2022b) definiert werden.

Knotenart	Beschreibung
Dokumentknoten (document node)	Der Dokumentknoten ist ein namenloser Knoten. Sein Inhalt umfasst das ganze Dokument.
Elementknoten (element node)	Jedes Element des Dokuments wird durch einen Elementknoten im Baum repräsentiert. Ein Elementknoten enthält evtl. weitere Elementknoten als Kinder.
Textknoten (text node)	Ein Textknoten enthält ausschließlich Zeichendaten.
Attributknoten (attribute node)	Attributknoten entsprechen den Attributen eines Elementes. Sie haben einen Bezeichner und einen Wert. Ein Attributknoten wird nicht als Kind eines Elementknotens, sondern als „zum Element zugehörig“ betrachtet.
Namensraumknoten (namespace node)	Ein Namensraumknoten repräsentiert den Namensraum eines Elementes. Er ist kein Kindknoten des Elementes, sondern dem Element zugeordnet.
Verarbeitungsanweisungsknoten (processing instruction node)	Ein Verarbeitungsanweisungsknoten repräsentiert eine Ver- arbeitungsanweisung. Er ist ein Kindknoten des Dokument- knotens und somit ein Geschwisterknoten des Elementknotens des Wurzelelementes.
Kommentarknoten (comment node)	Kommentarknoten repräsentieren die Kommentare im XML-Dokument. Ihr Wert ist die in <!-- --> enthaltene Zeichenkette.

Tabelle 2 Knotenarten von XPath (Becher, 2022b)

In folgender Abbildung 15 werden die verschiedenen Knotenarten illustriert.

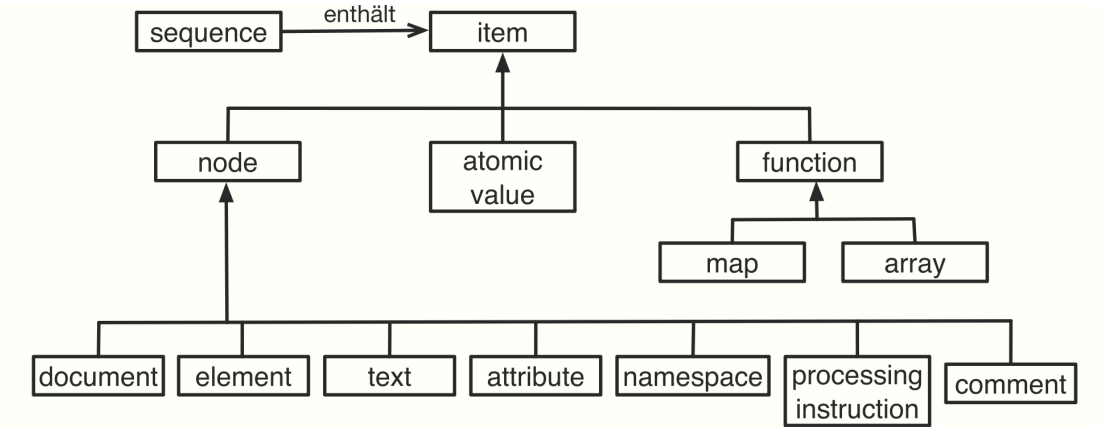

Abbildung 15 verschiedene Knotenarten XPath (Becher, 2022b)

Die Anordnung im XML-Baum wird verwendet, um den Inhalt zu strukturieren und Formatierungsoptionen einzuschränken. Daher ist die Reihenfolge der Knoten, zumindest auf kleiner Ebene, von Bedeutung. Der Begriff „Dokumentreihenfolge“ bezieht sich auf die Reihenfolge, in der Knoten nacheinander angetroffen werden, wenn das Dokument geparst wird (siehe Abbildung 16). Dieses Verhalten wird als geordnetes Baummodell bezeichnet. (Chawathe & Garcia-Molina, 1997b) XML-Parser sind Softwareprogramme, die dazu dienen, XML-Dokumente zu lesen und die darin enthaltenen Markierungen zu identifizieren. Beim Einlesen eines XML-Dokuments überprüfen XML-Parser die Syntax auf Korrektheit und stellen sicher, dass das Dokument den Regeln und Standards von XML entspricht. (Becher, 2022a)

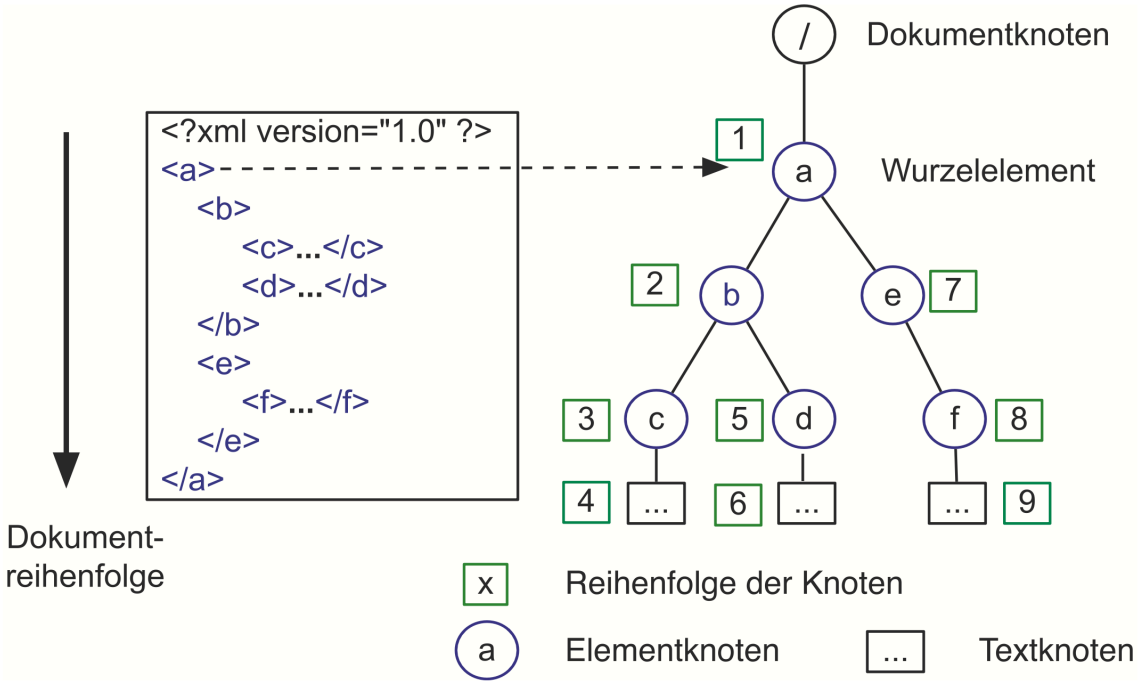

Abbildung 16 Baumdarstellung und Dokumentreihenfolge (Becher, 2022b)

3.2 Inkrementelle Verfahren

Nun soll konkret darauf eingegangen werden, wie Änderungen in XML-Dateien erkannt werden können und warum herkömmliche Textvergleichsmethoden hierfür nicht ausreichend sind. Außerdem wird ein Standard vorgestellt, der ein effizientes Verfahren zur Übertragung von Änderungen in XML-Daten definiert. Anschließend wird erläutert, wie die Modifikationsinformation in der Versionierung von XML-Dateien nutzbar gemacht wird.

3.2.1 Änderungserkennung

Grundsätzlich kann bei der Modifizierung von XML-Dateien zwischen zwei Arten unterschieden werden (Becher, 2022c) :

Semantische Änderung (Wert)

Das Hinzufügen oder Entfernen eines Elements kann Veränderung in der Hierarchie verursachen. Wenn beispielsweise innerhalb eines Issues ein Kommentar hinzugefügt oder entfernt wird, beeinflusst dies die Hierarchie des XML-Dokuments.

Topologische Änderung (Struktur)

Änderungen in den Werten von Attributen können ebenfalls stattfinden. Zum Beispiel könnte man den Wert der Priorität verändern.

Es gibt verschiedene Ansätze zur Änderungserkennung. Eine einfache Methode ist ein Textvergleich. Änderungen werden als eine Abfolge von Zeilen betrachtet, die in einer Datei entfernt, hinzugefügt oder verändert wurden, um eine andere Datei zu erstellen. Beim Ausführen eines Diffs werden zwei Dateien Zeile um Zeile betrachtet. Zurückgegeben werden dabei die abweichenden Zeilen. (MacKenzie et al., 2002)

Ein einfacher Textvergleich ist für die Änderungserkennung zwischen zwei XML-Dateien nicht optimal ausgelegt. Dies liegt an den spezifischen Eigenschaften und der hierarchischen Struktur von XML-Daten. Ein einfacher Textvergleich behandelt den XML-Code als flachen Text, ohne die hierarchische Beziehung der Elemente zu berücksichtigen. Neue Elemente könnten an unterschiedlichen Stellen in der Hierarchie

eingefügt werden. Des Weiteren können XML-Elemente Attribute enthalten und die Reihenfolge der Attribute kann variieren, ohne die Semantik zu ändern. Obwohl sie inhaltlich irrelevant sind, würde ein Textvergleich Unterschiede in der Attributreihenfolge als Änderungen interpretieren. (DeltaXML Ltd, 2014)

Folgende Beispiele sollen dies verdeutlichen. Wir betrachten zwei XML-Dateien, die trotz offensichtlicher Unterschiede im Rahmen von XML dennoch als identisch gelten:

Beispiel 1 – ohne Leerzeichen oder Namensräume

```
<record xmlns="http://www.myco.com/records" id="b123">
  <name>Michael Brown</name>
  <born>1984-03-08</born>
  <sex>M</sex>
</record>
```

Beispiel 2 – mit Leerzeichen und Namensräumen:

```
<staff:record id="b123" xmlns:staff="http://www.myco.com/records">
  <staff:name>Michael Brown</staff:name>
  <staff:born>1984-03-08</staff:born>
  <staff:sex>M</staff:sex>
</staff:record>
```

Formel 2 Beispiel zweier XML-gleicher Dateien (DeltaXML Ltd, 2014)

Es gibt Unterschiede, wie zusätzliche Leerzeichen, Änderungen in der Reihenfolge von Attributen und die Spezifikation des XML-Namensraums mittels eines Präfixes. Trotz dieser Abweichungen hat keine der Änderungen Auswirkungen auf die Informationen, die in den beiden XML-Dateien enthalten sind. Folglich können die beiden Dateien als „XML-gleich“ betrachtet werden. (DeltaXML Ltd, 2014)

Dieses Szenario unterstreicht die Notwendigkeit eines auf XML ausgerichteten Ansatzes zur Änderungserkennung. Um Änderungen zwischen XML-Dateien korrekt zu erkennen, ist eine spezielle Änderungserkennung erforderlich, die die semantischen und strukturellen Aspekte von XML berücksichtigt. Hierzu gehören die bereits beschriebenen Methoden wie der Vergleich von XML-Bäumen, die Beachtung der Hierarchie und die Unterscheidung zwischen unterschiedlichen Arten von Änderungen (z.B. Hinzufügen, Entfernen, Ersetzen von Elementen). Solche Methoden sind in der Lage, präzise und sinnvolle Informationen über die tatsächlichen Änderungen in den XML-Daten zu liefern.

3.2.2 RFC 5261:

Extensible Markup Language (XML) wird allgemein als Container für den Austausch und die Speicherung von Daten in heutigen Systemen verwendet. Traditionell erforderte die Aktualisierung eines XML-Dokuments den Versand einer vollständigen Kopie der neuen Version. RFC 5261 schlägt jedoch einen effizienteren Ansatz vor, bei dem nur die geänderten Teile des Dokuments übermittelt werden. Das Herzstück von RFC 5261 ist das XML-Patch-Framework, das XML Path Language (XPath)-Selektoren nutzt. XPath-Selektoren ermöglichen die genaue Lokalisierung des zu ändernden Teils eines XML-Dokuments. Das Framework führt grundlegende Operationen wie Hinzufügen (<add>), Ersetzen (<replace>) und Entferne (<remove>) ein, die mithilfe von XPath-Selektoren auf das vorhandene XML-Dokument angewendet werden können. Im Rahmen dieses Frameworks sind XPath-Selektorwerte und neue Daten innerhalb von XML-Elementen eingebettet. Die Namen dieser Elemente, wie <add>, <replace> oder <remove>, geben an, welche Art von Modifikation durchgeführt werden soll. Diese Elemente, auch Patch-Operationen genannt, sind durch Schema-Typen mit der W3C-Schema-Sprache definiert. Die XPath-Selektoren zeigen das Ziel für eine Änderung an und werden als Attribute dieser Elemente ausgedrückt. Die Kindknoten der Patch-Operationselemente enthalten die neuen Daten. (Urpalainen, 2008)

3.2.3 Versionierung

Das Versionsmanagement ist ein Konzept der Dokumentenverwaltung. Es ermöglicht die Entwicklung und Änderungen von Dokumenten über die Zeit hinweg zu verfolgen und zu verwalten. Dokumente, die in einem iterativen Prozess entstehen werden als

versionierte Dokumente bezeichnet. In der Versionierung werden Unterschiede zwischen aufeinanderfolgenden Versionen eines Dokuments als Delta dargestellt. Hintergrund ist, dass alle Versionen desselben Dokuments einen gemeinsamen Teil haben und in bestimmten Bereichen Unterschiede aufweisen. Ein Delta macht sich das mit einem Edit Skript (Bearbeitungsskript) zu nutzen, das beschreibt, wie der Unterschied erzeugt wird. So wird der Speicherbedarf minimiert, denn nur die tatsächlichen Änderungen werde gespeichert, statt des ganzen Dokuments einer Version. Darüber hinaus ist ein Delta entweder vorwärts oder rückwärts gerichtet. Die Richtung hängt davon ab, ob es die Differenz zwischen der neuen Version minus der alten Version darstellt oder umgekehrt. So wird beispielsweise eine neue Version durch die Anwendung einer Sequenz von vorwärts gerichteten Deltas auf eine Basis-Version konstruiert. In der Versionierung wird zwischen Revision und Variante unterschieden. Eine Revision bezieht sich auf eine sequenzielle Entwicklung eines Dokuments über die Zeit hinweg. Es ergibt sich ein linearer Verlauf, bei dem jede neue Revision eine Fortsetzung oder Aktualisierung der vorherigen ist. Wohingegen Varianten parallele oder alternative Versionen sind, die gleichzeitig existieren und sich nicht gegenseitig ersetzen. (Wuwongse and Yoshikawa, 2005)

3.3 Integration inkrementeller Verfahren auf BCF-Daten

Gegenstand der weiteren Betrachtung ist lediglich die markup.bcf-Datei im Topic eines entpackten BCF-Zip-Archivs, da es sich bei dieser, wie in Kapitel 2 behandelt, um eine XML-Datei handelt. Zudem ist die markup.bcf die zentrale Komponente eines BCF-Zip-Archivs. Wenn ein Endnutzer ein Issue bearbeitet, beispielsweise indem er einen neuen Kommentar hinzufügt, einen vorhandenen Kommentar löscht, eine Verantwortung zuweist, ein Fälligkeitsdatum einträgt oder die Priorität ändert, werden diese Änderung in der markup.bcf-Datei vorgenommen.

Die praktische Umsetzung - das Erstellen einer Modifikationsinformation zwischen zwei Versionen einer markup.bcf-Datei wird in Python mit der xmldiff-Bibliothek implementiert.

3.3.1 Anforderungen an inkrementelle Verfahren auf BCF-Daten für Versionierung

Speziell im Kontext der Versionierung ergebenen sich Anforderungen an die Anwendung inkrementeller Verfahren auf BCF-Daten. Die Änderungserkennung muss alle Unterschiede zwischen den verschiedenen Versionen der Markup-Datei korrekt erkennen und dokumentieren. Eine Version muss vollständig wiederhergestellt werden können, um sicherzustellen, dass keine Informationen verloren gehen. Außerdem muss ein effektiver Ansatz eine Diff-Operation implementieren, die Unterschiede zwischen zwei Versionen eines Dokuments abbildet. Genauso muss eine Patch-Operation unterstützt werden, die dieses Delta auf ein Dokument anwendet, um eine neue Version zu generieren. Innerhalb eines Deltas ist es wichtig, dass die genaue Position jeder Änderung im Dokument identifiziert wird. Dafür ist ein zuverlässiges Knotenadressierungsschema notwendig.(Rönnau et al., 2005)

3.3.2 XmlDiff:

XmlDiff ist eine Bibliothek und ein Befehlszeilen-Dienstprogramm für Python. Mit den dort definierten Methoden können Unterschiede zwischen XML-Dateien erkannt und dargestellt werden. Dabei wird berücksichtigt, dass sich die Änderungserkennung in hierarchischen Daten deutlich von der Änderungserkennung in flachen Daten unterscheidet. (Python Software Foundation, o. J.)

Für das Ausführen eines Diffs sind drei Funktionen definiert:

Funktion	Eingabe
`xmldiff.main.diff_files()`	Dateipfade / Datei-Streams
`xmldiff.main.diff_texts()`	Unicode-Zeichenfolgen
`xmldiff.main.diff_trees()`	lxml-Bäume

Tabelle 3 xmldiff: Funktionen zum Ausführen eines Diffs gemäß (Python Software Foundation, o. J.)

Lxml-Bäume beziehen sich auf das Modul lxml.etree, zum Parsen und Erstellen von XML-Daten (Behnel, o. J.). Dieses ermöglicht es hierarchische XML-Daten als Bäume darzustellen.

3.3.3 Edit Script

Das Ausführen einer Diff-Funktion gibt standardmäßig ein Edit Script (Änderungsskript) mit der Modifikationsinformation zurück. Es besteht aus einer Liste von Bearbeitungsaktionen. Diese Aktionen geben an, wie der originale Baum in den transformierten Baum umgewandelt werden kann. XPath-Ausdrücke weisen eindeutig darauf hin, welche Knoten oder Elemente von der Aktion angesprochen werden sollen. „xmldiff“ enthält neun verschiedene Bearbeitungsaktionen, welche im Folgenden aufgelistet sind. (Python Software Foundation, o. J.)

InsertNode (target, tag, position)

Damit wird ein neuer untergeordneter Knoten zum in „target“ angegebenen Knoten hinzugefügt. Das „tag“-Attribut gibt das Tag für den neuen Knoten an, während das „position“-Argument definiert, an welcher Stelle im Ziel der neue Knoten eingefügt werden soll. Eine Position von 0 bedeutet, dass der neue Knoten als erstes Kind des „targets“ eingefügt wird. Dies steht im Gegensatz zu XPath, denn dort wird der erste Knoten als 1 gezählt. Aus Gründen der Benutzerfreundlichkeit wurde die Positionierung so gewählt, da Python nullbasiert ist.

DeleteNode (node)

Diese Aktion löscht den in „node“ angegebenen Knoten.

MoveNode (node, target, position)

Der in „node“ angegebene Knoten wird als Kindknoten unter den „target“-Knoten verschoben. Das Argument „position“ legt fest, an welcher Position der neue Knoten eingefügt werden soll. Eine Position von 0 bedeutet, dass der neue Knoten als erstes Kind des Zielknoten eingefügt wird. Wenn der Verschiebevorgang innerhalb desselben Elternelements erfolgt, kann die „position“ mehrdeutig sein. Wir betrachten den Fall, dass ein Kindknoten von der Position 1 auf Position 3 verschoben werden soll. Die „position“ bezieht sich nicht auf den zu verschiebenden Knoten, sondern kennzeichnet die Position, an der der Knoten nach der Ausführung der Funktion landen soll. Bei der Implementierung von „MoveNode()“ ist es daher am einfachsten, den Knoten zuerst aus dem Elternelement zu entfernen und ihn dann an der angegebenen Position einzufügen.

InsertAttrib (node, name, value)

Der in “node” angegebene Knoten erhält ein neues Attribut. „Name“ und „value“ definieren den Namen und den Wert des Attributs.

DeleteAttrib (node,name)

Der in „node“ spezifizierte Knoten wird gelöscht. Durch „name“ wird der konkretisiert um welches Attribut es sich handelt.

RenameAttrib (node, oldname, newname)

Ein Attribut im von „node“ angegebenen Knoten wird umbenannt. „Oldname“ definiert, welches Attribut und „newname“ den neuen Namen.

UpdateAttrib (node, name, value)

Das Attribut des in „node“ angeführten Knoten erhält einen neuen Wert. „Name“ und „value“ spezifizieren welches Attribut und den neuen Wert.

UpdateTextIn (node, text)

Ein Textinhalt, des in „node“ definierten Knotens, erhält einen neuen Wert. „Text“ konkretisiert den neuen Wert des Texts.

UpdateTextAfter (node, text)

Der Text, der auf den in „node“ angegebenen Knoten folgt, erhält einen neuen Wert. Das Argument „text“ gibt den neuen Wert dieses Textes an.

InsertComment (target, position, text)

Die geläufige „InsertNode()“-Aktion ist nicht optimal für Kommentare geeignet, da Kommentare über keine Tags verfügen. Aus diesem Grund existiert eine eigene Aktion für ihr Einfügen. Ähnlich wie bei „InsertNode()“ sind die Eingabeparameter der Zielknoten „target“ und die gleichnamige Position. UpdateTextIN() und DeleteNode() können jedoch wie gewohnt auch bei Kommentaren verwendet werden.

InsertNamespace (prefix, uri)

Dadurch wird dem XML-Dokument ein neuer Namespace hinzugefügt.

DeleteNamespace (prefix)

Der durch „prefix“ definierte Namespace wird gelöscht.

3.3.4 Wichtige Parameter

Wie effektiv und genau die Diff-Funktion Änderungen erkennt, kann durch die Wahl einiger Parameter beeinflusst werden (Regebro, o. J.):

F: Über diesen Parameter wird festgelegt, wie ähnlich sich zwei XML-Knoten sein müssen, um als gleich in beiden Baumstrukturen betrachtet zu werden. „F“ kann einen Wert zwischen 0 und 1 betragen und beeinflusst die Ähnlichkeitsgrenze. Wenn der Wert von „F“ höher ist, bedeutet das, dass die beiden Knoten sich weniger unterscheiden dürfen, um als identisch betrachtet zu werden. Infolgedessen werden bei einem höheren Wert mehr Knoten als gleich angesehen. Das kann dazu führen, dass mehr Knoten eingefügt oder gelöscht werden, anstatt der Aktualisierung vorhandener Knoten. Wenn der Wert von „F“ niedriger ist, dürfen sich die beiden Knoten stärker unterscheiden, um als gleich betrachtet zu werden. Somit werden bei einem niedrigeren Wert mehr Knoten aktualisiert, als dass neue Knoten eingefügt oder vorhandene gelöscht werden.

ratio_mode: Dieser Parameter bestimmt, wie genau die Ähnlichkeit zwischen zwei Knoten berechnet wird. Die Optionen sind „accurate“ (genau), „fast“ (schnell) und „faster“ (schneller). Standardmäßig ist „fast“ definiert. Bei der Verwendung von „faster“ erfolgt die Berechnung der Ähnlichkeit zwischen den Knoten schneller, jedoch kann dies zu weniger präzisen Ergebnissen führen. Das Änderungsskript ist möglicherweise nicht optimal. Es erfordert mehr Aktionen, um das gleiche Ergebnis zu erzielen. Das Auswählen von „accurate“ sorgt für genauere Ergebnisse. Allerdings ist es langsamer, vor allem, wenn die Knoten lange Texte oder viele Attribute enthalten. Dies kann die Bearbeitungszeit erhöhen, aber präzisere Änderungsskripten erzielen.

4 Fallstudie

Ziel ist es, die Übertragbarkeit von Diff- und Patch-Techniken aus dem XML-Bereich auf BCF-Dateien zu demonstrieren.

4.1 Versuchsaufbau

Um die Validierung von Diff- und Patch-Operationen auf BCF-Dateien durchzuführen, werden verschiedene Experimente mit Original- und transformierten Dateien durchgeführt. Zunächst wird ein Diff zwischen den beiden Dateien erstellt, um die genauen Unterschiede in der Struktur und im Inhalt zu identifizieren. Dieser Diff wird dann analysiert, um sicherzustellen, dass er alle erwarteten Änderungen korrekt erfasst.

Anschließend wird der Patch-Vorgang angewendet, um die ursprüngliche Datei mit den im Diff identifizierten Änderungen zu aktualisieren. Die gepatchte Datei wird dann mit der transformierten Datei verglichen. So wird geprüft, ob die Änderung korrekt angewendet wurde und die Struktur und der Inhalt der Dateien konsistent sind.

Um die Validität der Diff- und Patch-Operationen zu überprüfen, werden verschiedenen Szenarien simuliert, darunter:

- Ändern der Knotenreihenfolge
- Hinzufügen neuer Elemente
- Entfernen von Elementen
- Ändern von Attributwerten
- Veränderung der BCF-Version

Diese Fallstudie dient dazu, die Funktionsweise von Diff- und Patch-Techniken bei BCF-Dateien zu veranschaulichen und auch deren praktische Anwendbarkeit zu demonstrieren.

4.1.1 Ablauf

Zunächst wird die originale Datei erstellt, welches als Grundlage für weitere Bearbeitungen dient. Diese wird als original_file bezeichnet. An diesem original_file werden

dann Änderungen vorgenommen, und die modifizierte Version wird als transformed_file gespeichert. Beide Dateien sind Eingangsparameter des Pythoncodes. Der Code führt eine Reihe von Operationen auf die BCF-Dateien durch. Zunächst müssen diese jedoch entpackt werden, damit der Inhalt zugänglich ist, da BCF-Dateien in einem komprimierten Format vorliegen.

Nach dem Entpacken werden die erforderlichen Module importiert, darunter „os“ für Betriebssystemoperationen, „etree“ aus „lxml“ zur Arbeit mit XML-Dateien und „main“ aus „xmldiff“ für Diff- und Patch-Operationen.

Die Funktion „load_xml“ ermöglicht das Laden der Markups der entpackten BCFs. Die Funktion nimmt den Dateipfad als Eingabe und verwendet diesen, um die Datei zu öffnen. Dabei wird die Datei im binären Modus („rb“ für „read binary“) geöffnet. Die Funktion verwendet das Modul „etree“ aus „lxml“, um den Inhalt der XML-Datei zu analysieren und einen entsprechenden XML-Baum zu erstellen. Dies geschieht durch den Aufruf der Methode „parse(file)“ des „etree“-Objekts. „File“ stellt das geöffnete Dateiobjekt dar. Zurückgegeben wird der erstellte XML-Baum, was den anderen Teilen des Codes ermöglicht, mit dem hierarchischen Inhalt der XML-Datei zu arbeiten.

Die Funktion „write_xml“ dient zum Schreiben eines XML-Baums in eine Datei. Der erste Eingangsparameter „xml_tree“ repräsentiert den XML-Baum, der geschrieben werden soll und der zweite Eingangsparameter „file_path“ gibt den Pfad zur Zieldatei an. Durch das Aufrufen von „etree.ElementTree(xml_tree) wird ein ElementTree-Wrapper um den gegebenen XML-Baum erstellt. So kann der XML-Baum in eine Datei geschrieben werden. Die Zieldatei wird im binären Schreibmodus („wb“ für „write binary“) geöffnet. Mit der Methode „tree.write()“ wird der XML-Baum in die geöffnete Datei geschrieben. Dabei werden verschiedene Optionen verwendet, wie „pretty_print=True“, für eine anschauliche Formatierung, „enconding=‘utf-8‘“, zur Codierung der Unicode-Zeichen und „xml_declaration=True“, um eine XML-Deklaration am Anfand der Datei hinzuzufügen. UTF-8 ist ein weit verbreitetes Zeichenkodierungsformat, das eine Vielzahl von Zeichen aus verschiedenen Schriftsystemen unterstützt (Yergeau, 2003). Durch die Verwendung von UTF-8 soll hier sichergestellt werden, dass die enthaltenen Textdaten korrekt und vollständig gespeichert werden können.

Nachdem die XML-Dateien geladen und als XML-Baume repräsentiert wurden, wird ein Diff zwischen den Original- und transformierten XML-Baumen erstellt. Dieser Diff

wird in eine Datei geschrieben, um die identifizierten Unterschiede zu dokumentieren. Der eben beschriebene Vorgang wird in Abbildung 17 verdeutlicht.

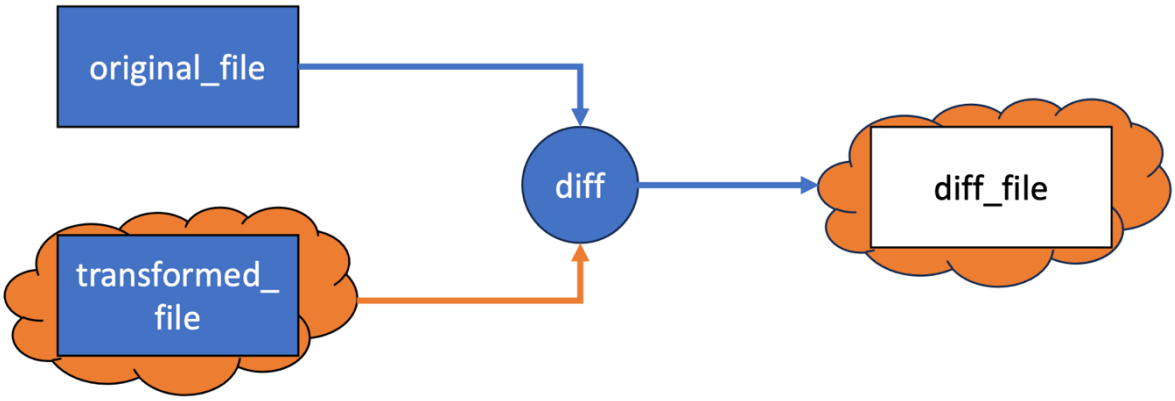

Abbildung 17 Diff der originalen Datei und der transformierten Datei

Wie ein Inhalt eines diff_file aussehen kann, illustriert Abbildung 18. Zu sehen ist das in eine Datei geschriebene diff_file aus dem Beispiel Issue1. Es enthält Bearbeitungsaktionen, wie „MoveNode“ oder „UpdateTextIn“.

Issue1_diff_file.diff

```
MoveNode(node='/Markup/Topic/Comments/Comment[1]', target='/Markup/Topic/Comments[1]',
position=3)
MoveNode(node='/Markup/Topic/Comments/Comment[1]', target='/Markup/Topic/Comments[1]',
position=2)
MoveNode(node='/Markup/Topic/Comments/Comment[1]', target='/Markup/Topic/Comments[1]',
position=1)
UpdateTextIn(node='/Markup/Topic/Comments[1]', text='\n              ')
UpdateTextAfter(node='/Markup/Topic/Comments/Comment[1]', text='\n            ')
UpdateTextAfter(node='/Markup/Topic/Comments/Comment[4]', text='\n        ')
MoveNode(node='/Markup/Topic/Comments/Comment[4]/Author[1]', target='/Markup/Topic/Comments/
Comment[1]', position=1)
MoveNode(node='/Markup/Topic/Comments/Comment[4]/Viewpoint[1]', target='/Markup/Topic/Comments/
Comment[1]', position=4)
MoveNode(node='/Markup/Topic/Comments/Comment[3]/Author[1]', target='/Markup/Topic/Comments/
Comment[2]', position=1)
MoveNode(node='/Markup/Topic/Comments/Comment[3]/Viewpoint[1]', target='/Markup/Topic/Comments/
Comment[2]', position=4)
MoveNode(node='/Markup/Topic/Comments/Comment[2]/Author[2]', target='/Markup/Topic/Comments/
Comment[3]', position=1)
MoveNode(node='/Markup/Topic/Comments/Comment[2]/Viewpoint[2]', target='/Markup/Topic/Comments/
Comment[3]', position=3)
MoveNode(node='/Markup/Topic/Comments/Comment[1]/Author[2]', target='/Markup/Topic/Comments/
Comment[4]', position=1)
MoveNode(node='/Markup/Topic/Comments/Comment[1]/Viewpoint[2]', target='/Markup/Topic/Comments/
Comment[4]', position=3)
```

Abbildung 18 Diff_file mit Modifikationsinformationen

Der Originalbaum wird anschließend anhand des Diffs gepatcht, um die identifizierten Änderungen zu übernehmen (siehe Abbildung 19). Abschließend wird der gepatchte Baum in eine BCF-Datei gespeichert.

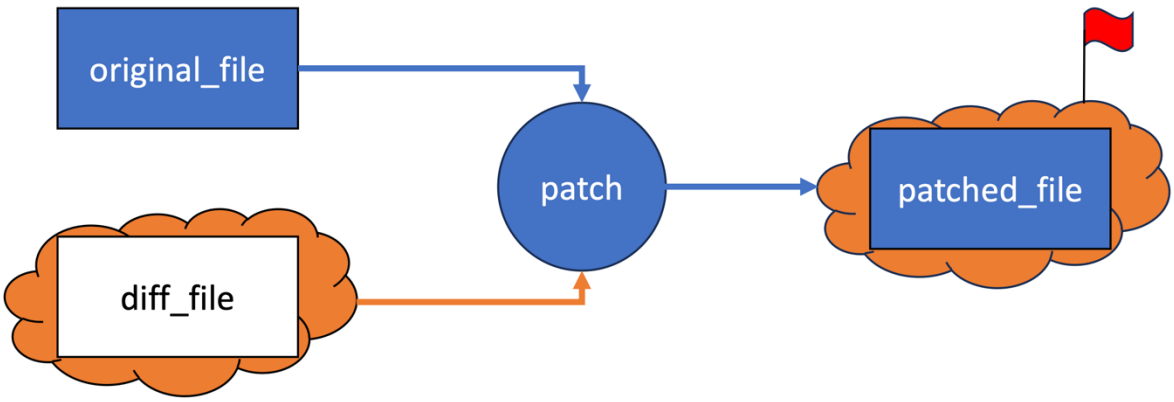

Abbildung 19 Patch der originalen Datei und der Diff-Datei

4.2 Validierung der Ergebnisse

Nachfolgend soll eine theoretische Grundlage für die Validierung von Ergebnissen geschaffen werden. Die Mengenlehre ist entscheidend, um festzulegen, wann zwei Mengen gleich sind. Nur so kann letztendlich validiert werden, ob ein Patch erfolgreich war.

4.2.1 Mengenlehre

Bei der Validierung der Ergebnisse ist die Mengenlehre von zentraler Bedeutung. Sie stellt klare Maßstäbe für die Feststellung der Gleichheit von Mengen bereit. Dies ist besonders relevant, um zu bestimmen, ob zwei XML-Dateien inhaltlich identisch sind.

Nach Cantor wird eine Menge wie folgt definiert:

> „Eine Menge ist eine Zusammenfassung bestimmter wohlunterschiedener Objekte unserer Anschauung oder unseres Denkens zu einem Ganzen. Von jedem dieser Objekte muss eindeutig feststehen, ob es zur Menge gehört oder nicht. Die zur Menge gehörenden Objekte nennt man die Elemente der Menge.“
>
> (Iwanowski and Lang, 2021)

Üblicherweise beschreibt man Mengen mit großen Buchstaben und die Elemente einer Menge mit kleinen Buchstaben. Um anzugeben, dass ein Element a zur Menge M gehört, verwendet man die Notation „ $a \in M$ “. Wenn ein Objekt b nicht zur Menge M gehört, schreibt man „ $b \notin M$ “. (Iwanowski and Lang, 2021)

Das Extensionalitätsaxiom ist ein grundlegendes Konzept in der Mengenlehre, das die Gleichheit von Mengen definiert. Es besagt, dass zwei Mengen genau dann gleich sind, wenn sie exakt die gleichen Elemente enthalten. Diese Definition der Gleichheit von Mengen basiert auf der Idee der Extension oder Ausdehnung einer Menge. Demnach wird eine Menge durch die konkreten Elemente definiert, die sie enthält. Formal ausgedrückt wird die Gleichheit von zwei Mengen A und B durch die Äquivaltenzrelation A=B beschrieben. Dies bedeutet, dass für alle Elemente a gilt: Wenn a ein Element von A ist, dann ist auch a ein Element von B, und umgekehrt. (Cap, 1993)

In symbolischer Schreibweise lautet die Definition der Gleichheit zweier Mengen:

$$(A = B) \Leftrightarrow [\forall x : (x \in A \Leftrightarrow x \in B)]$$

Formel 3 Definition Gleichheit zweier Mengen (Cap, 1993)

Darüber hinaus führt das Extensionalitätsaxiom zu einer klaren Unterscheidung zwischen einer Menge und ihren Elementen. Eine Menge wird nicht durch die Reihenfolge oder die Art und Weise definiert, wie ihre Elemente angeordnet sind, sondern ausschließlich durch die Elemente selbst. (Mildenberger, 2019)

Dem Extensionalitätsaxiom entsprechend werden folgende zwei Bäume T_1 (geordnet) und T_2 (ungeordnet) als gleich betrachtet, da sie die exakt gleichen Elemente enthalten und die Reihenfolge keine Rolle spielt:

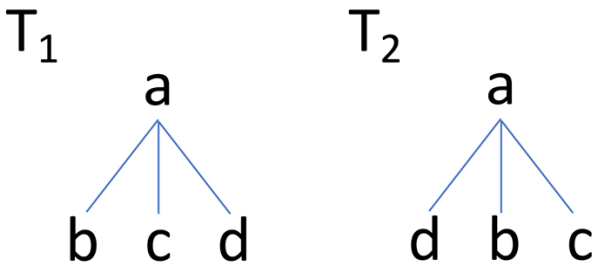

Abbildung 20 geordneter Baum und ungeordneter Baum

Diese Annahmen werden für die folgenden Betrachtungen und Interpretationen von Ergebnissen vorausgesetzt.

4.3 Übersetzung patch-basierter Aktualisierungen auf BCF-Dateien

Im Folgenden sollen nun die Beispiel-Issues präsentiert werden, auf welche Diff- und Patch-Operationen angewendet werden.

4.3.1 Beispiel Issues

Insgesamt wurden fünf Beispiel-Issues auf den Python Code angewendet, welche von 1 bis 5 nummeriert sind. Für jeden Anwendungsfall wurde jeweils ein gleichnamiges Python Skript implementiert.

Issue1 thematisiert eine Änderung der Knotenreihenfolge. Das original_file enthält vier Kommentare, welche im markup.bcf in absteigender Reihenfolge dokumentiert sind (siehe Abbildung 21). Im transformed_file ist die Reihenfolge der Kommentare aufsteigend (siehe Abbildung 22).

```
<Comments>
    <Comment Guid="e62adca2-2e93-4f6c-90d5-4e6b70cb8017">
        <Date>2024-02-06T10:31:46+01:00</Date>
        <Author>carina.sailer@tum.de</Author>
        <Comment>Kommentar 4</Comment>
        <Viewpoint Guid="93effdea-9501-4cc7-8fc2-451d5d363c79"/>
    </Comment>
    <Comment Guid="6f1db81b-2a00-4d52-87be-fb7e1a342327">
        <Date>2024-02-06T10:31:42+01:00</Date>
        <Author>carina.sailer@tum.de</Author>
        <Comment>Kommentar 3</Comment>
        <Viewpoint Guid="93effdea-9501-4cc7-8fc2-451d5d363c79"/>
    </Comment>
    <Comment Guid="53cc01d9-b461-4390-a134-1ed011727a07">
        <Date>2024-02-06T10:31:23+01:00</Date>
        <Author>carina.sailer@tum.de</Author>
        <Comment>Kommentar 2</Comment>
        <Viewpoint Guid="93effdea-9501-4cc7-8fc2-451d5d363c79"/>
    </Comment>
    <Comment Guid="5365526b-9b55-44a4-8f24-016ab15a895c">
        <Date>2024-02-06T10:31:20+01:00</Date>
        <Author>carina.sailer@tum.de</Author>
        <Comment>Kommentar 1</Comment>
        <Viewpoint Guid="93effdea-9501-4cc7-8fc2-451d5d363c79"/>
    </Comment>
```

Abbildung 21 Issue1 original_file Kommentarreihenfolge

```
<Comments>
      <Comment Guid="5365526b-9b55-44a4-8f24-016ab15a895c">
        <Date>2024-02-06T10:31:20+01:00</Date>
        <Author>carina.sailer@tum.de</Author>
        <Comment>Kommentar 1</Comment>
        <Viewpoint Guid="93effdea-9501-4cc7-8fc2-451d5d363c79"/>
    </Comment>
    <Comment Guid="53cc01d9-b461-4390-a134-1ed011727a07">
        <Date>2024-02-06T10:31:23+01:00</Date>
        <Author>carina.sailer@tum.de</Author>
        <Comment>Kommentar 2</Comment>
        <Viewpoint Guid="93effdea-9501-4cc7-8fc2-451d5d363c79"/>
    </Comment>
    <Comment Guid="6f1db81b-2a00-4d52-87be-fb7e1a342327">
        <Date>2024-02-06T10:31:42+01:00</Date>
        <Author>carina.sailer@tum.de</Author>
        <Comment>Kommentar 3</Comment>
        <Viewpoint Guid="93effdea-9501-4cc7-8fc2-451d5d363c79"/>
    </Comment>
    <Comment Guid="e62adca2-2e93-4f6c-90d5-4e6b70cb8017">
        <Date>2024-02-06T10:31:46+01:00</Date>
        <Author>carina.sailer@tum.de</Author>
        <Comment>Kommentar 4</Comment>
        <Viewpoint Guid="93effdea-9501-4cc7-8fc2-451d5d363c79"/>
    </Comment>
</Comments>
```

Abbildung 22 Issue1 transformed_file Kommentarreihenfolge

In Issue2 geht auf das Ändern von Attributwerten ein. Dort ändert sich lediglich der Wert des TopicStatus von „Open“ im original_file zu „Assigned“ im transformed_file. Zudem ergibt sich eine Änderung im „ModifiedDate“, da dadurch dokumentiert wird, wann diese Änderung vorgenommen wurde.

Issue3 beleuchtet das Hinzufügen neuer Elemente. Das transformed_file entsteht, indem zum original_file zwei Kommentare addiert werden.

Issue4 thematisiert einen komplexeren Anwendungsfall. Das original_file enthält 3 Kommentare. Kommentar 2 wird gelöscht und anschließend werden Kommentar 4 und Kommentar 5 hinzugefügt, um das transformed_file zu erhalten (siehe Abbildung 23).

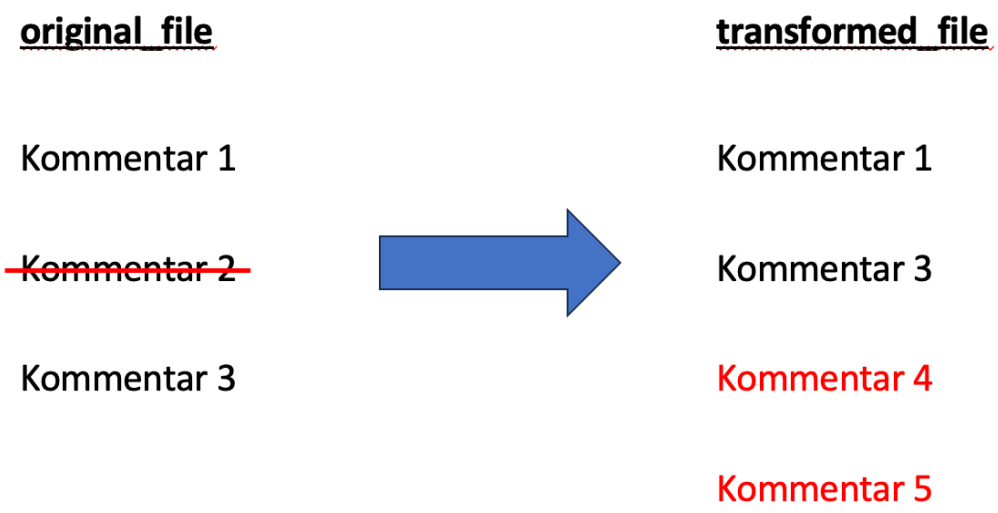

Abbildung 23 Durchgeführte Modifikation in Issue4

Issue5 betrachtet eine Veränderung der BCF-Version, konkret von Version 3.0 auf Version 1.0. In diesem Szenario exportiert beispielsweise ein Fachplaner ein Issue in der Version 3.0 und sendet es an einen anderen Teilnehmer. Dieser öffnet das Issue, bearbeitet es und exportiert es anschließend als Version 1.0. Das original_file enthält 3 Kommentare in der Version 3.0. Das transformed_file beinhaltet 2 zusätzliche Kommentare (Kommentar 4 und Kommentar 5) jedoch in der Version 1.0.

4.4 Ergebnis

In Issue1 wurde die umgekehrte Reihenfolge als Änderung erkannt, was darauf hinweist, dass das System möglicherweise zu sensibel für solche Änderungen ist. Dies steht im Widersprich zu den im Voraus festgelegten Maßstäben für die Feststellung der Gleichheit von Mengen aus der Mengenlehre. Laut dieser sind, wie bereits beschrieben, zwei Mengen genau gleich, wenn sie exakt die gleichen Elemente enthalten, unabhängig von ihrer Reihenfolge. In diesem Zusammenhang sollte die Umsortierung der Reihenfolge der Elemente in der BCF-Datei keine Änderung darstellen, solange die Elemente selbst identisch sind. Dies kann zu unnötigen Patch-Operationen führen. Das Diff-Verfahren sollte dementsprechend eine reine Änderung der Reihenfolge als nicht signifikanten Unterschied erfassen.

Um den Einfluss des F-Parameters zu untersuchen, wurden unterschiedliche Werte getestet. Dieser legt fest, wie ähnlich sich zwei Knoten sein müssen, um als gleich in beiden Baumstrukturen betrachtet zu werden. Dabei ist dieser Parameter standardmäßig mit 0.5 belegt, wohingegen in Issue1.2 und Issue1.3 die Werte 0 und 1 getestet werden. Im Issue1.2 wird F mit dem Wert 0 belegt und in Issue1.3 nimmt der Parameter den Wert 1 an. Bei einem F-Wert von 1 wird ein diff_file erzeugt, welches viele Operationen beinhaltet, darunter das Einfügen neuer Knoten, das Aktualisieren von Attributen und das Verschieben von Knoten. Dahingegen fällt das diff_file bei einem F-Wert von 0 weniger umfangreich aus. Es werden hauptsächlich Verschiebungen von Knoten durchgeführt. Einfüge- oder Aktualisierungsoperationen sind weniger häufig. Trotz dieser Unterschiede erzeugen beide diff_files dasselbe patched_file.

Der Erfolg der Diff- und Patch-Operationen wurde daran gemessen, ob das patched_file mit dem transformed_file übereinstimmt. Ihre Struktur und Inhalt wurden dafür manuell auf Unterschiede geprüft. Wenn beide Inhalte übereinstimmen, wird der Diff- und Patch-Vorgang als erfolgreich betrachtet.

In allen Beispielszenarien wurde beobachtet, dass die Anwendung inkrementeller Verfahren erfolgreich war und die erwarteten Änderungen korrekt angewendet wurden. Dabei wurden verschiedene Szenarien simuliert, darunter beispielsweise das Ändern der Knotenreihenfolge oder komplexere Anwendungsfälle wie das Löschen und Hinzufügen von Knoten. Unabhängig von der Art der durchgeführten Modifikationen konnten die Transformationen korrekt angewendet werden. Damit ist die praktische Anwendbarkeit von Diff- und Patch-Operationen aus dem XML-Bereich auf BCF-Dateien belegt.

5 Zusammenfassung und Fazit

Zusammenfassend wird mit den Ergebnissen der Fallstudie bestätigt, dass Diff- und Patch-Operationen aus dem XML-Bereich auf BCF-Daten angewendet werden können. Die erwarteten Änderungen werden erfolgreich erkannt und angewendet.

Die Implementierung von Diff- und Patch-Operationen eröffnet der Softwareseite neue Perspektiven und Funktionen. Eine Integration der Änderungserkennung für BCFs in bestehende Softwareanwendungen oder die Entwicklung neuer Tools kann effiziente Mechanismen zur Versionierung und Verwaltung von BCF-Dateien bereitstellen. Die Modifikationsinformation zwischen zwei BCFs dient dabei als Grundlage für eine Versionierung. Die durch den Vergleich unterschiedlicher markup-Dateien erfasste Änderung, kann als Stand der Datei gespeichert werden. Damit kann eine strukturierte Historie der Änderungen dokumentiert werden. Darüber hinaus macht die Modifikationsinformationen Änderungen rückverfolgbar.

In Bezug auf die Zukunftsaussichten bietet die Änderungserkennung in BCF-Dateien viele Möglichkeiten für weitere Entwicklungen. Eine Integration von Echtzeitbenachrichtigungen über Änderung in BCF-Dateien in bestehende Projektmanagement- oder Kollaborationstools kann einen konkreten Nutzen für die Kommunikation in Projekten bieten. Beteiligte können sofort über wichtige Entwicklungen in Bauprojekten informiert werden. So können sie schnell auf Änderungen reagieren und sicherstellen, dass alle Teammitglieder auf dem neuesten Stand sind. Beispielsweise könnten durch eine API-Integration Tools wie Microsoft Teams dafür genutzt werden.

Eine wichtige Überlegung in Hinblick auf die Einführung dieses neuen Ansatzes ist die Kompatibilität mit dem bestehenden digitalen Ökosystem. Neue Lösungen müssen nahtlos in ein bestehendes System integriert werden können, um den Nutzen zu maximieren und den Bedarf an zusätzlichen Ressourcen zu minimieren. Ein differenzierter Ausblick erfordert die Berücksichtigung von Kompatibilität, Integration, Komplexität und Nutzerakzeptanz. Nur so kann sichergestellt werden, dass die Anwendung dieses Verfahrens die gewünschten Ergebnisse liefert und den Übergang zu dieser Technologie erfolgreich unterstützt. Hochkomplexe Verfahren können die Akzeptanz beein-

trächtigen und die Effektivität der Anwendung in der Praxis einschränken. Die Benutzererfahrung muss konsistent und intuitiv sein für alle, die die Technologie einsetzen. Mit einem benutzerfreundlichen System können Mitarbeiter ihre Arbeit einfacher und effektiver erledigen. Dies beschleunigt die Implementierung der Informationstechnologie und Unternehmen können schneller Nutzen aus ihrer Investition ziehen. (Hartmann, 2022)

Wenn inkrementelle Verfahren nicht mit dem bestehenden Ökosystem vereinbar sind, könnte dies demnach den Nutzen verringern und eine Implementierung dieser Technologie behindern. Aus den oben genannten Gründen ist es wichtig, dass inkrementelle Verfahren so gestaltet werden, dass sie einfach zu implementieren und zu verwenden sind, ohne die Nutzerfreundlichkeit zu beeinträchtigen.

In Hinblick auf die Entwicklung eines benutzerfreundlichen Systems und die Bedürfnisse des Endnutzers stellt sich die Frage, ob es notwendig ist, in der gepatchten Datei zu kennzeichnen, dass diese mittels eines Patch-Verfahrens anhand von Modifikationsinformationen hergestellt wurde. Für den Endnutzer, der BCFs im Issuemanagement verwendet, ist es nicht relevant zu wissen, ob eine Datei gepatcht wurde oder nicht. Sowohl das transformed_file als auch das patched_file enthalten den gleichen Inhalt, der für den Benutzer signifikant ist. Es ist in diesem Kontext lediglich wichtig, dass die Datei die erforderlichen Änderungen enthält und korrekt funktioniert. Außerdem könnte das Kennzeichnen einer Datei als gepatcht den Endnutzer verwirren, vor allem wenn dieser nicht mit dem Konzept des Patchens vertraut ist. Das könnte zu Fehlinterpretationen führen oder die Benutzererfahrung beeinträchtigen. Daher soll durch das Verbergen technischer Details, wie des Patchvorgangs, das Anwendungserlebnis verbessert werden. Der Benutzer kann sich auf die Funktionalität der Datei konzentrieren, ohne sich um die zugrunde liegenden technischen Grundlagen kümmern zu müssen. Ein Verzicht auf das Markieren einer Datei als gepatcht, vereinfacht auch die Dateiverwaltung. Nutzer müssen nicht zwischen verschiedenen Versionen unterscheiden.

Ausgehend von der Beobachtung, dass die meisten Änderungen an derselben Stelle des Dokuments vorgenommen werden, kann die Berechnungszeit des Deltas verbessert werden. Potenzial für die Optimierung des Diff-Algorithmus liegt beispielsweise in der Indizierung von Unterbäumen. Dadurch wird verhindert, dass der Algorithmus das

gesamte Dokument durchlaufen muss, indem er gezielt nur die betroffenen Unterbäume analysiert. Zudem muss für die Nutzbarkeit des vorgestellten Verfahrens eine Umkehrbarkeit des Deltas implementiert werden. Nur so kann die Modifikationsinformation auch für die Wiederherstellung vorheriger Versionen genutzt werden. Das ist entscheidend für eine flexible und präzise Verwaltung von Versionen in BCFs. (Rönnau et al., 2005)

6 Literaturverzeichnis

Amazon Web Services. (o. J.). *Was ist eine RESTful-API?* 2023. Abgerufen 15. Februar 2024, von https://aws.amazon.com/de/what-is/restful-api/#:~:text=Eine%20RESTful%2DAPI%20ist%20eine,um%20verschiedene%20Aufgaben%20zu%20erfüllen

Bally, A., & Brogini, M. (2020). *Digitale Vernetzung für mehr Marktdominanz* (1. Aufl.). Haufe Lexware.

Becher, M. (2022a). XML-Grundlagen. In *XML: DTD, XML-Schema, XPath, XQuery, XSL-FO, SAX, DOM* (S. 1–22). Springer Fachmedien Wiesbaden. https://doi.org/10.1007/978-3-658-35435-0_1

Becher, M. (2022b). XPath und XQuery. In *XML: DTD, XML-Schema, XPath, XQuery, XSL-FO, SAX, DOM* (S. 183–281). Springer Fachmedien Wiesbaden. https://doi.org/10.1007/978-3-658-35435-0_7

Becher, M. (2022c). XPath und XQuery. In *XML: DTD, XML-Schema, XPath, XQuery, XSL-FO, SAX, DOM* (S. 183–281). Springer Fachmedien Wiesbaden. https://doi.org/10.1007/978-3-658-35435-0_7

Behnel, S. (o. J.). *The lxml.etree Tutorial*. 2024. Abgerufen 15. Februar 2024, von https://lxml.de/tutorial.html

BIMcollab. (2024). *BIMcollab- über BCF*. https://www.bimcollab.com/de/resources/openbim/about-bcf/

BIMpedia. (o. J.). *BCF-BIM Collaboration Format*. 2024. Abgerufen 15. Februar 2024, von https://www.bimpedia.eu/artikel/1004-bcf-bim-collaboration-format

Borrmann André and König, M. and K. C. and B. J. (2021). Die BIM-Methode im Überblick. In M. and K. C. and B. J. Borrmann André and König (Hrsg.), *Building Information Modeling: Technologische Grundlagen und industrielle Praxis* (S. 1–31). Springer Fachmedien Wiesbaden. https://doi.org/10.1007/978-3-658-33361-4_1

buildingSMART International. (o. J.-a). *BIM Collaboration Format (BCF)*. 2024. Abgerufen 15. Februar 2024, von https://technical.buildingsmart.org/standards/bcf/

buildingSMART International. (o. J.-b). *Industry Foundation Classes (IFC)*. 2024. Abgerufen 15. Februar 2024, von https://technical.buildingsmart.org/standards/ifc/

buildingSMART International. (o. J.-c). *Software Implementations*. 2024. Abgerufen 15. Februar 2024, von https://technical.buildingsmart.org/resources/software-implementations/

Cap, C. H. (1993). Mengenlehre. In *Theoretische Grundlagen der Informatik* (S. 17–28). Springer Vienna. https://doi.org/10.1007/978-3-7091-9329-7_2

Chawathe, S. S., & Garcia-Molina, H. (1997a). Meaningful change detection in structured data. *Proceedings of the 1997 ACM SIGMOD International Conference on Management of Data*, 26–37. https://doi.org/10.1145/253260.253266

Chawathe, S. S., & Garcia-Molina, H. (1997b). Meaningful change detection in structured data. *SIGMOD Rec.*, *26*(2), 26–37. https://doi.org/10.1145/253262.253266

Cobena, G., Abiteboul, S., & Marian, A. (2002). Detecting changes in XML documents. *Proceedings 18th International Conference on Data Engineering*, 41–52. https://doi.org/10.1109/ICDE.2002.994696

DeltaXML Ltd. (2014). *Intelligent Diff for XML*.

Dix, A., Finlay, J., Abowd, G., & Beale, R. (2004). *Human-Computer Interaction*.

Esser, S., Vilgertshofer, S., & Borrmann, A. (2022). Graph-based version control for asynchronous BIM collaboration. *Advanced Engineering Informatics*, *53*. https://doi.org/10.1016/j.aei.2022.101664

Fraser, N. (2009). Differential synchronization. *DocEng'09 - Proceedings of the 2009 ACM Symposium on Document Engineering*, 13–20. https://doi.org/10.1145/1600193.1600198

Gardner, J. R., & Gardner, J. R. (2001). *XSLT and XPATH: A Guide to XML Transformations*. Prentice Hall PTR.

Hartmann, U. (2022). *Building Information Modeling - Grundlagen, Standards und Praxis*. Wilhelm Ernst & Sohn Verlag fur Architektur und Technische.

Iwanowski Sebastian and Lang, R. (2021). Mengenlehre. In *Diskrete Mathematik mit Grundlagen: Lehrbuch für Studierende von MINT-Fächern* (S. 47–106). Springer Fachmedien Wiesbaden. https://doi.org/10.1007/978-3-658-32760-6_2

Khorchi, A., & Boton, C. (2024). An OpenBIM-based 4D approach to support coordination meetings in virtual reality environments. *Journal of Building Engineering*, *85*, 108647. https://doi.org/https://doi.org/10.1016/j.jobe.2024.108647

Kulbak, Y. (o. J.). *GitHub Dokumentation - BIM Collaboration Format v3.0 Technical Documentation*. Abgerufen 15. Februar 2024, von https://github.com/buildingSMART/BCF-XML/tree/release_3_0/Documentation

MacKenzie, D., Eggert, P., & Stallman, R. (2002). *Comparing and Merging Files with GNU diff and patch* (B. Gough, Hrsg.). Network Theory Ltd.

Mildenberger, H. (2019). *Axiomatische Mengenlehre*. Albert-Ludwigs-Universität Freiburg.

Preidel, C. (2020). *Automatisierte Konformitätsprüfung digitaler Bauwerksmodelle hinsichtlich geltender Normen und Richtlinien mit Hilfe einer visuellen Programmiersprache (Automated compliance checking of digital building models regarding applicable standards and guidelines using a visual programming language)* [Technical University of Munich, Germany]. https://mediatum.ub.tum.de/1534486

Preidel Cornelius and Borrmann, A. and E. H. and K. M. (2021). Common Data Environment. In M. and K. C. and B. J. Borrmann André and König (Hrsg.), *Building Information Modeling: Technologische Grundlagen und industrielle Praxis* (S.

335–351). Springer Fachmedien Wiesbaden. https://doi.org/10.1007/978-3-658-33361-4_16

Python Software Foundation. (o. J.). *PyPI - xmldiff 2.6.6 - Projekt Beschreibung*. 2023. Abgerufen 15. Februar 2024, von https://pypi.org/project/xmldiff/

Regebro, L. (o. J.). *Dokumentation - xmldiff - Python API*. 2018. Abgerufen 15. Februar 2024, von https://xmldiff.readthedocs.io/en/stable/api.html#main-diffing-api

Rimskaia-Korsakova, O. (2022). *Mittelstand-Digital Zentrum Bau - Common Data Environment (CDE)*. https://www.digitalzentrum-bau.de/kos/WNetz?art=News.show&id=1254

Rönnau, S., & Borghoff, U. M. (2012). XCC: change control of XML documents. *Comput. Sci.*, *27*(2), 95–111. https://doi.org/10.1007/s00450-010-0140-2

Rönnau, S., Scheffczyk, J., & Borghoff, U. M. (2005). Towards XML version control of office documents. *Proceedings of the 2005 ACM Symposium on Document Engineering*, 10–19. https://doi.org/10.1145/1096601.1096606

Schapke Sven-Eric and Beetz, J. and K. M. and K. C. and B. A. (2021). Prinzipien und Techniken der modellgestützten Zusammenarbeit. In M. and K. C. and B. J. Borrmann André and König (Hrsg.), *Building Information Modeling: Technologische Grundlagen und industrielle Praxis* (S. 309–333). Springer Fachmedien Wiesbaden. https://doi.org/10.1007/978-3-658-33361-4_15

Schatz, K. (o. J.). *Bild: Modellbasierter Nachrichtenaustausch mit BCF*. Abgerufen 15. Februar 2024, von https://www.baunetzwissen.de/integrales-planen/fachwissen/standardisierung/modellbasierter-nachrichtenaustausch-mit-bcf-5288163/gallery-1/1

Schulz, O., Oraskari, J., & Beetz, J. (2021, Februar). *bcfOWL: A BIM collaboration ontology*.

Selkow, S. M. (1977). The tree-to-tree editing problem. *Information Processing Letters*, *6*(6), 184–186. https://doi.org/https://doi.org/10.1016/0020-0190(77)90064-3

Steinmann, R. (2021). Zertifizierung von BIM-Software. In M. and K. C. and B. J. Borrmann André and König (Hrsg.), *Building Information Modeling: Technologische*

Grundlagen und industrielle Praxis (S. 193–208). Springer Fachmedien Wiesbaden. https://doi.org/10.1007/978-3-658-33361-4_9

Teufel, S. (1995). *Computerunterstützung für die Gruppenarbeit*. Addison-Wesley.

Urpalainen, J. (2008). RFC 5261 - An Extensible Markup Language (XML) Patch Operations Framework Utilizing XML Path Language (XPath) Selectors. *RFC*.

Word Wide Web Consortium. (o. J.). *W3C- About us*. 2024. Abgerufen 15. Februar 2024, von https://www.w3.org/about/

Wuwongse Vilas and Yoshikawa, M. and A. T. (2005). Temporal Versioning of XML Documents. In H. and M. Q. and F. Y. and F. E. and L. E. Chen Zhaoneng and Chen (Hrsg.), *Digital Libraries: International Collaboration and Cross-Fertilization* (S. 419–428). Springer Berlin Heidelberg.

Yergeau, F. (2003). UTF-8, a transformation format of ISO 10646. *RFC3629*.

7 Abbildungsverzeichnis

8 Tabellenverzeichnis

Anhang A

- Beispiel-Issues als BCF-Dateien als Eingangsparameter für jeweilige Python Skripts
- Python-Skripte jeweils für jedes Beispiel-Issue, welche die Implementierung der Diff- und Patch-Operationen beinhalten
- Alle Produkte, die bei der Ausführung der jeweiligen Python-Skripte entstanden sind
- Verwendetes Solibri-Modell für die Erstellung der BCF-Issues

Erklärung

Hiermit erkläre ich, dass ich die vorliegende Bachelor-Thesis selbstständig angefertigt habe. Es wurden nur die in der Arbeit ausdrücklich benannten Quellen und Hilfsmittel benutzt. Wörtlich oder sinngemäß übernommenes Gedankengut habe ich als solches kenntlich gemacht.

Ich versichere außerdem, dass die vorliegende Arbeit noch nicht einem anderen Prüfungsverfahren zugrunde gelegen hat.

München, 21. März 2024 [redacted]

[redacted]