Diplomarbeit

Entwicklung von Prüfregeln zur (teil-)automatischen Bestimmung und Überprüfung von Brandabschnitten

ausgeführt zum Zwecke der Erlangung des akademischen Grads
Diplom-Ingenieur
eingereicht an der TU Wien, Fakultät für Bau- und Umweltingenieurwesen

---

Diploma Thesis

Development of inspection-rules for the (semi-)automatic determination and verification of fire compartments.

submitted in satisfaction of the requirements for the degree
Diplom-Ingenieur
of the TU Wien, Faculty of Civil and Environmental Engineering

Daniel Pfeiffer, BSc

Matr.Nr.: 11775246

Betreuung: Associate Prof. Dipl.-Ing. Dr.techn **Christian Schranz**, M.Sc.
Assistant Prof. Dipl.-Ing. Dr.techn. **Harald Urban**
Univ.Ass. Dipl.-Ing. **Simon Fischer**
Forschungsbereich Digitaler Bauprozess
Technische Universität Wien
Karlsplatz 13/E235-03, 1040 Wien, Österreich

Wien, im März 2024 _____

Danksagung

Zu Beginn möchte ich mich bei meinen beiden Betreuern Herrn Assoc. Prof. Dipl.-Ing. Dr. techn Christian Schranz und Herrn Univ.Ass. Dipl.-Ing. Dr.techn. Harald Urban bedanken. Beide haben mich umfassend beim Verfassen dieser Arbeit unterstützt und mein Interesse an der Bauinformatik sowie der Digitalisierung des Bauwesens geweckt.

Weiters möchte ich mich für die Unterstützung aus meinem privaten Umfeld bedanken. Ein besonderer Dank gilt dabei meinen Eltern, die es mir ermöglicht haben, diesen Werdegang zu beschreiten und mich jede Sekunde lang unterstützt haben. Abschließend möchte ich mich bei meinen Freunden bedanken, die mich stehts auf meinem Weg begleitet haben und hoffentlich weiterhin werden.

Kurzfassung

Die Digitalisierung im Bauwesen nimmt immer mehr an Fahrt zu. Mittlerweile ist Building Information Modelling weder in der Planung noch in der Ausführung wegzudenken. Diese digitale, modellbasierte Planungsmethode zielt darauf ab, umfassende Gebäudeinformationen in einem oder mehreren dreidimensionalen Modellen abzubilden. Dabei soll allen Beteiligten der Zugriff auf die Daten jederzeit ermöglicht werden. Allerdings fehlt in den Überlegungen oft eine weitere Partei, die maßgeblich von der Digitalisierung profitieren kann: die Baubehörde. Um die vielen Vorteile von Building Information Modelling für die Baubehörde zu nutzen, wurde das Projekt BRISE-Vienna konzipiert. Dieses Projekt befasst sich neben der Optimierung des Behördenverfahrens vor allem mit der automatischen bzw. teilautomatischen Prüfung von BIM-Modellen auf geltende Bauvorschriften. Dies umfasst sowohl baurechtliche als auch bautechnische Gesetze und Richtlinien. Die Automatisierung erfolgt durch vorgefertigte und eigens programmierte Prüfroutinen, welche frei miteinander verschachtelt werden können. Die Hauptziele sind ein möglichst hoher Automatisierungsgrad, die Entlastung der Referenten der Baubehörde und die Beschleunigung und effizientere Gestaltung des Verfahrens. Allerdings benötigt dies genauere Angaben durch die Planenden. Dieser Mehraufwand soll im Zuge der Generierung kompensiert werden.

Ein Themengebiet, welches sowohl durch rechtliche als auch technische Vorgaben definiert ist, ist der Brandschutz von Gebäuden. Die komplexen Bestimmungen reichen von Anforderungen an Materialien hin zu konkreten Ausführungsvorschriften. Diese Masterarbeit fokussiert sich auf die automatische Validierung und Generierung von Brandabschnitten für Aufenthaltsräume. Neben wichtigen Grundlagen zum Verständnis der Arbeit wird der rechtlichen Rahmen genau definiert. Der Hauptteil der Arbeit umfasst die Beschreibung der Algorithmen, sowohl mit Code-Abschnitten als auch durch unterstützende Ablaufdiagramme. Anschließend erfolgt die Bewertung der Validierung und Generierung anhand zweier Modelle.

Die Ergebnisse der Prüfungen zeigen, dass die (teil-)automatische Validierung von Brandabschnitten wesentlich zur Entlastung der Referenten beitragen kann. Trotz zusätzlicher, manueller Prüfungen ergibt sich eine wesentliche Effizienzsteigerung. Die Generierung stellt eine Möglichkeit zur Verifizierung von geplanten brandabschnittsbildenden Elementen dar. Allerdings ergibt sich keine zufriedenstellende Entlastung der Planenden, weswegen eine vollständige Simulierung vorgeschlagen wird.

Abstract

Digitalization in the construction industry is gaining momentum. Building Information Modeling has become an integral part of both planning and execution. This digital, model-based planning method aims to depict comprehensive building information in one or more three-dimensional models. All parties involved should be able to access the data at any time. However, another party that can benefit significantly from digitization is missing: the building authority. The project BRISE-Vienna was designed to exploit the many advantages of Building Information Modeling for the building authorities. In addition to optimizing the official procedure, this project is primarily concerned with the automatic or semi-automatic checking of applicable building regulations in BIM models. This includes both building regulations and building laws and guidelines. The automation is carried out using prefabricated and specially programmed check routines, which can be freely interlinked. In addition to achieving the highest possible degree of automation, the aim is to reduce the workload of the agents and to speed up the process and make it more efficient. However, this requires more precise information from the planners. This additional work is to be compensated for in the course of generation.

One subject area that is defined by both legal and technical requirements is the fire protection of buildings. The complex regulations range from requirements for materials to specific implementation regulations. This master thesis focuses on the automatic validation and generation of fire compartments for common rooms. In addition to important basics for understanding the work, the legal framework is defined in detail. The main part of the thesis comprises the description of the algorithms, both with code sections and with supporting flowcharts. This is followed by an evaluation of the validation and generation using two models.

The results of the tests show that the (partially) automatic validation of fire compartments can make a significant contribution to reducing the workload of the instructors. Despite additional manual checks, there is a significant increase in efficiency. The generation provides a way of verifying planned fire compartment elements. However, it does not quite provide the desired relief for planners, which is why an alternative method of complete simulation is proposed.

Inhaltsverzeichnis

Kapitel 1

Einleitung

Die Digitalisierung im Bauwesen schreitet voran und nimmt eine immer wichtigere Rolle ein. Whyte und Hartmann [48] beschreiben in ihrem Artikel die Digitalisierung des Bauwesens durch Technologien wie BIM (Building Information Modelling), Machine Learning und Smart Infrastructure als wesentliche Stütze zur Verbesserung des Designprozesses. Dabei gehen sie noch einen Schritt weiter und nennen BIM als treibende Kraft zur Veränderung im Bauwesen. Aber nicht nur die Planung profitiert von BIM auch, verschiedene, namhafte Baufirmen berichten über die Vorteile von BIM für ihre Arbeit. Porr [36] beschreibt, dass sie BIM zur Verbesserung der Zusammenarbeit und zur Abwicklung des Bauprozesses heranziehen. Strabag [45] möchte qualitätsvolle und effiziente Bauvorhaben durch transparente und schnelle Wissensübermittlung realisieren und hat dafür die digitale Arbeitsmethode BIM für sich entdeckt. Das zeigt, dass grundlegend Interesse an BIM und dem gesamten Abwicklungsprozess besteht. Aber was ist Building Information Modelling genau? Borrmann et al. [18] beschreibt BIM als digitales Modell eines Bauwerks, dessen Informationsgehalt über die dreidimensionale Darstellung hinausgeht. Dabei besteht die Möglichkeit, geometrische Elemente mit zusätzlichen, alphanummerischen Informationen zu versehen. Damit entstehen neue Optimierungsmöglichkeiten über alle Phasen des Lebenszyklus eines Bauwerks. Neben Planung und Ausführung gibt es eine weitere große Partei, die an der Umsetzung eines Bauwerks beteiligt ist: Die Baubehörde. Um die Vorteile von BIM auch für die Baubehörde zugänglich zu machen, wurde das Projekt BRISE-Vienna [44] ins Leben gerufen. BRISE steht für Building Regulations Information for Submission Envolvement und hat zum Ziel, die Verwaltung zu digitalisieren und effizienter zu gestalten. Mithilfe von Technologien wie Augmented Reality kann das Behördenverfahren [39], [40] aber auch modernisiert werden. Neben diesen Technologien setzt BRISE-Vienna auf BIM zur (teil-) automatischen Prüfung der geltenden Rechtsvorschriften in BIM-Modellen. Dabei sind die Ziele hochgesteckt: Neben gesteigerter Transparenz von Planungs- und Genehmigungsprozess soll die Effizienz bzw. die Schnelligkeit der Genehmigungen um bis zu 50 Prozent gesteigert werden. Weiters soll die automatisierte Prüfung die Referenten[1] entlasten. Zu diesem Zweck sind genormte und detaillierte Angaben in den Modellen notwendig. Zur Kompensation der Mehrarbeit sollen Planende ebenfalls durch die Automatisierung an anderen Stellen entlastet werden. Ein wesentliches und umfangreiches Thema stellt dabei der Brandschutz in Gebäuden dar. Dieser wird in mehreren OIB-Richtlinien [32] sowie in der Bauordnung für Wien [15] definiert. Aufgrund der Komplexität der Thematik existiert eine eigene Kompetenzstelle Brandschutz (KSB) innerhalb der Baubehörde der Stadt Wien.

In dieser Arbeit wird das Thema Brandabschnitte im Kontext der digitalen Baueinreichung behandelt, wobei zwei zentrale Forschungsfragen entstehen. Für die Baubehörde soll geklärt werden, ob eine automatisierte Prüfung bzw. Validierung von vordefinierten Brandabschnitten möglich ist. Dazu werden alle geltenden baurechtlichen und bautechnischen Regulierungen

[1] Der Verfasser legt großen Wert auf Diversität und Gleichbehandlung. Aus Gründen der leichteren Lesbarkeit wurde jedoch oftmals die männliche Form gewählt. Sämtliche Ausführungen gelten natürlich in gleicher Weise für alle Geschlechter.

hinsichtlich der Automatisierung überprüft. Zusätzlich müssen weitere, manuelle Schritte definiert werden, um die gesamte Prüfung von Brandabschnitten zu sichern. Die zweite Forschungsfrage beschäftigt sich mit der Entlastung der Planenenden durch eine automatisierte Generierung von Brandabschnitten. Durch die Angabe von verschiedenen Merkmalen soll der Algorithmus Vorschläge für die Brandabschnittswahl bieten und so den Mehraufwand kompensieren.

Zum Aufbau des notwendigen Grundwissens werden in Kapitel 2 grundlegende Themen rund um BIM und die (teil-)automatisierte Prüfung von Modellen behandelt. Weiters wird die relevante Rechtsmaterie evaluiert und das Konzept zur Entwicklung der Algorithmen besprochen. In Kapitel 3 erfolgt getrennt für Validierung und Generierung eine detaillierte Beschreibung der Umsetzung. Diese Umfasst neben dem Ablauf und relevanten Code-Ausschnitten alle Vereinfachungen sowie die Ergebnisdarstellung. Im Kapitel 4 werden die Tools hinsichtlich ihrer Richtigkeit geprüft sowie an einem eingereichten Modell aus dem Projekt BRISE-Vienna getestet und die Ergebnisse dargestellt. Kapitel 5 fasst die Arbeit zusammen und bewertet beide Algorithmen und definiert Handlungsempfehlungen für zukünftige Weiterentwicklungen.

Kapitel 2

Grundlagen

In diesem Kapitel werden Grundlagen zu BIM erläutert, welche für Prüfung und Genierung von Brandabschnitten relevant sind. Weiters wird die Arbeit im Kontext zum Projekt BRISE-Vienna gesetzt, welches die Rahmenbedingungen für diese Arbeit definiert. Zum Abschluss des Kapitels werden grundlegende Begriffe rund um die Programmiersprache Java genannt sowie die Solibri-API als maßgebende Programmierschnittstelle erläutert.

2.1 Building Information Modeling (BIM)

Building Information Modeling (BIM) steht im Allgemeinen für eine fortschrittliche Methode zur Abwicklung von Projekten im Bereich der Bauindustrie. Es handelt sich um einen digitalen Prozess, der die Erstellung, Verwaltung und Weitergabe umfassender und konsistenter Informationen über ein Gebäude oder ein Infrastrukturbauwerk, während des gesamten Lebenszyklus ermöglichen soll. Für BIM gibt es verschiedene Definitionen:

Eichler et al. [24] – wissenschaftliche Sichtweise
„Die ISO 19650-1 definiert BIM als »Nutzung einer untereinander zur Verfügung gestellten digitalen Repräsentation eines Assets zur Unterstützung von Planungs-, Bau- und Betriebsprozessen als zuverlässige Entscheidungsgrundlage« (zu den baulichen Assets gehören u.a. Gebäude, Brücken, Straßen und Prozessanlagen). Damit weist diese Norm auf die drei wesentlichen Aspekte von BIM: Modell, Technologie und Prozesse. Der Kern von BIM ist das digitale Bauwerksmodell, das die Informationen in Form von Geometrien und Alphanumerik (nicht-geometrische Informationen zur Funktion, Verortung, Material etc.) enthält. Somit liefert BIM eine optimierte, digitale Arbeitsmethode zum Erzeugen, Austauschen und Pflegen von digitalen Bauwerksdaten. BIM fördert die erfolgreiche Kommunikation und Kollaboration zwischen den Beteiligten eines Bauprojekts. Dies unterstützt die Qualitätssicherung entscheidend.“

Austrian Standard Institute [3] – Standardisierungsorganisation
„Unter Building Information Modeling (BIM) oder Gebäudedatenmodellierung versteht man die optimierte Planung und Ausführung von Gebäuden mit Hilfe entsprechender Software. BIM ist ein intelligentes digitales Gebäudemodell, das es allen Projektbeteiligten – vom Architekten und Bauherrn über den Haustechniker bis hin zum Facility Manager – ermöglicht, gemeinsam an diesem integralen Modell zu arbeiten und dieses zu realisieren.“

STRABAG [45] – Ausführende Firma
„Building Information Modelling beschreibt das modellbasierte Planen, Realisieren sowie Betreiben von Bauvorhaben, mit dem Anspruch, den Wissenstransfer, die Ergebnisqualität und die Effizienz aller Beteiligten zu optimieren. Über den gesamten Lebenszyklus des Projekts hinweg werden alle relevanten Daten digital erfasst, kombiniert und vernetzt. Daraus resultiert ein nachvollziehbares, transparentes und belastbares Informationsnetzwerk für alle Beteiligten.“

Stadt Wien [44] – Behörde
„Building Information Modeling (BIM) ist eine vernetzte Planungsmethode für Gebäudemodelle mit aufeinander abgestimmten Informationen von Architektur-, Statik- und Haustechnik-PlanerInnen. Das mit BIM erzeugte Modell lässt sich auch für das Errichten und Betreiben von Gebäuden nutzen.“

Autodesk [11] – Softwarehersteller
„BIM dient zum Erstellen und Verwalten von Daten während des Planungs-, Bau- und Betriebsprozesses. BIM integriert multidisziplinäre Daten für die Erstellung detaillierter digitaler Darstellungen, die in einer offenen Cloud-Plattform verwaltet werden und so eine Zusammenarbeit in Echtzeit ermöglichen. Die Vorteile von BIM bestehen darin, Teams, Arbeitsabläufe und Daten über den gesamten Projektlebenszyklus hinweg – von der Planung über den Bau bis hin zum Betrieb – zu verbinden, um bessere Arbeitsweisen und bessere Ergebnisse zu erzielen.“

Die verschiedenen Definitionen überschneiden sich in wesentlichen Punkten und können damit zu einer allgemeinen Beschreibung zusammengefasst werden. Bei BIM handelt es sich demnach um einen interdisziplinären Prozess, welcher sich über den gesamten Lebenszyklus eines Bauwerkes erstreckt. Ein digitales, dreidimensionales Modell bildet die Basis für gewerkeübergreifende Zusammenarbeit. Neben geometrischen können auch alphanumerische Daten im Modell eingetragen werden.

2.1.1 BIM-Level/BIM-Reifegrad

Gemäß Borrmann et al. [18] kann der Umstieg der Bauindustrie nicht in einem Zug bewältigt werden. Eine stufenweise Einführung der digitalen Technologien und der modellgestützten Planung ist sinnvoll und notwendig. Dafür wird das BIM-Reifegradmodell der britischen BIM Task Group (engl. BIM Maturity Model) genutzt, welches den Einführungsprozess in die Stufen 0 bis 3 unterteilt, welche in Abb. 2.1 dargestellt sind.

- BIM Level 0:
  Dieses Level beschreibt das herkömmliche Arbeiten mit zweidimensionalen CAD-Programmen und den Austausch über ausgedruckte Papierpläne.
- BIM Level 1:
  In Level 1 werden bereits erste dreidimensionale Modelle für schwierige Bereiche erstellt. Diese existieren parallel zu zweidimensionalen Zeichnungen. Der Austausch findet durch Verschicken einzelner Dateien statt, allerdings existiert keine Austauschplattform.
- BIM Level 2:
  In diesem Level erstellen Fachplaner mithilfe von BIM-Software einzelne digitale Gebäudemodelle. Diese werden zur Abstimmung regelmäßig miteinander verglichen. Der Austausch findet über Dateien statt, wobei softwarespezifische Formate genutzt werden.
- BIM Level 3:
  Unter Level 3 ist ein vollständiger, integraler Planungsprozess an einem gemeinschaftlichen digitalem Gebäudemodell zu verstehen. Diese Bauwerksmodelle sollen in Abstimmung aller Beteiligter über den gesamten Lebenszyklus geführt werden. Für den Datenaustausch werden ISO-Standards genutzt.

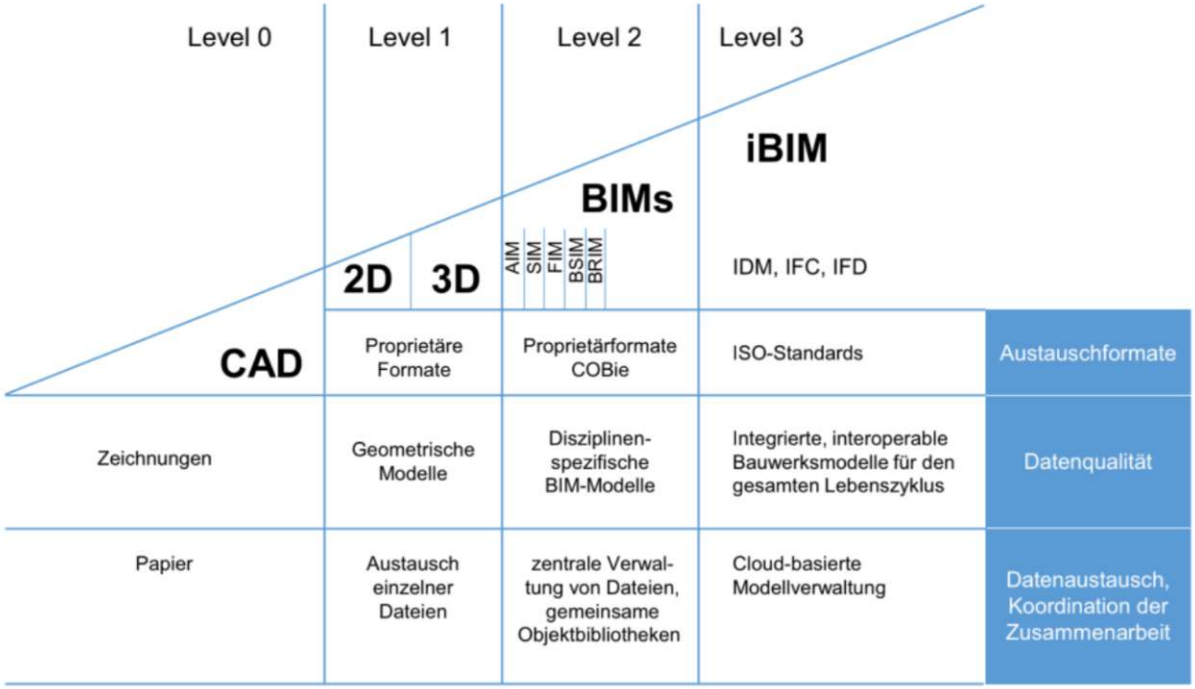


Abb. 2.1: BIM-Reifegrad laut Borrmann et al. [18]

2.1.2 Unterscheidungsmerkmale von BIM

Laut Borrmann et al. [18] sind Änderungen sowohl an unternehmensinternen, als auch an unternehmensübergreifenden Arbeitsprozessen notwendig, um den Übergang von zeichnungsgestützter Planung zur modellbasierter Planung erfolgreich zu implementieren. Gemäß dem BIM-Reifegradmodell ergeben sich dafür verschiedene Technologien pro Stufe. Der erste Schritt ist die Unterscheidung zwischen little BIM und BIG BIM. Little BIM definiert den Einsatz einer spezifischen BIM-Software durch einen einzelnen Planer in dessen Fachdisziplin. Mit Hilfe der BIM-Software wird zwar ein dreidimensionales Gebäudemodell erstellt. Allerdings werden daraus zweidimensionale Pläne abgeleitet. Eine Weiterverwendung des Gebäudemodells für andere Fachdisziplinen oder zur Koordination der Planung erfolgt nicht. Die Methode trägt auch die Bezeichnung Insellösung. Bei der Nutzung von Little BIM gehen trotz erster Effizienzsteigerungen ein Großteil des Potenzials verloren. Im Kontrast dazu beschreibt BIG BIM einen durchgängigen, modellbasierten Austausch aller am Prozess beteiligter Personen über alle Phasen des Lebenszyklus eines Bauwerks hinweg. Der Datenaustausch bei dieser Technologie erfolgt über Internetplattformen und Datenbanken. Der zweite Schritt behandelt die Softwareprodukte. Im Rahmen des Planungsprozesses stellt sich die Frage, ob lediglich jene Softwarelösungen eines Herstellers und damit proprietäre Dateiformate für den Datenaustausch zur Anwendung kommen (closedBIM) oder ob offene, neutrale Austauschformate (openBIM) genutzt werden. Obwohl Softwarehersteller im Bauwesen eine Vielzahl an Softwareprodukten für verschiedenste Anwendungen bieten, bleiben stets Lücken, die durch andere Softwarehersteller gefüllt werden müssen.
In Abb. 2.2 sind die Begriffe little BIM/BIG BIM und closedBIM/openBIM in zwei Dimensionen aufgetragen. Demnach muss, um das volle Potenzial von BIM zu erreichen, sowohl BIG BIM als auch openBIM umgesetzt werden. Daraus entsteht eine neue Bezeichnung – BIG openBIM – welche zusammengefasst folgende Merkmale beschreibt: durch neutrale Dateiformate kommen verschiedene Softwareprodukte für alle Disziplinen zu Erstellung eines dreidimensionalen Modells

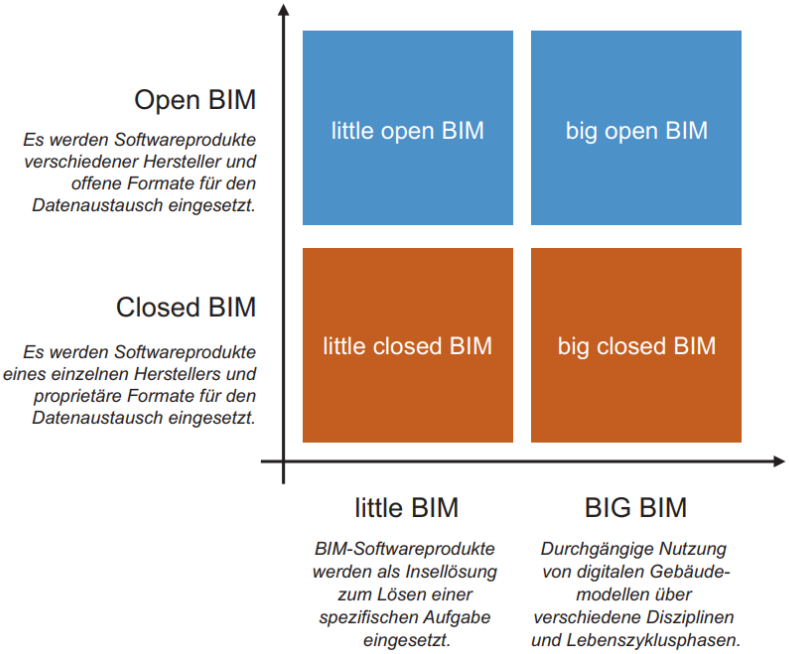


Abb. 2.2: Building Information Modeling-Matrix

zum Einsatz, welches über den gesamten Lebenszyklus genutzt werden kann. Gemäß Eichler et al. [24] ergeben sich aus der openBIM-Zusammenarbeit sowohl für die Umsetzung als auch für die Zusammenarbeit folgende Vorteile:

- Softwareunabhängigkeit bzw. Wahlfreiheit für Softwareprodukte aller Beteiligten
- Selbstständigkeit und Transparenz von software spezifischen Modellinformationen
- Herstellerunabhängige Garantie der Lesbarkeit der Modelldaten

Borrmann et al. [18] merken jedoch an, dass die Verwendung von herstellerneutralen Dateiformaten nicht einwandfrei funktioniert. Vor allem sei es darin begründet, dass sowohl die Schaffung der neutralen Dateiformate als auch die Implementierung der Softwarehersteller äußerst komplex sei. Trotzdem ist sich die Literatur einig [18, 24, 26, 30, 38], dass die Umsetzung von openBIM und die Nutzung offener, herstellerneutraler Dateiformate unabdinglich für die erfolgreiche Nutzung von BIM ist.

Krischmann et al.[29] beschreibt darüber hinaus, dass für die Anwendung innerhalb einer Behörde closedBIM zur Ausschließung Projektbeteiligter führen kann, wenn diese nicht die notwendigen Softwareprodukte besitzen. Diese Benachteiligung einzelner Firmen und Planungsbüros kann seitens der Behörde nicht durchgeführt werden, weshalb openBIM die einzig valide Lösung ist.

2.1.3 LOIN und Detailierungsgrade (LOG, LOI)

In klassischen Planungsprozessen (Level 0 und 1) dienen technische Zeichnungen als grundlegendes Mittel zur Dokumentation und Weitergabe von Planungsleistungen. Der Detaillierungsgrad der Informationen in einer Zeichnung hängt maßgeblich von dem Planmaßstab ab. Historisch bedingt legt der Maßstab eines Plans indirekt fest, wie detailliert die geometrischen Aspekte

des Bauwerks dargestellt werden können. Ein Entwurfsplan im Maßstab 1:200 bietet geringeren Informationsgehalt im Vergleich zu einem Ausführungsplan im Maßstab 1:50. Erfolgt die Planung jedoch mit BIM (Level 2 und 3) ergibt sich folgende Problemstellung: Digitale Bauwerksmodelle kennen keine Maßstäbe [1]. Deswegen bedarf es alternativer Konzepte zur Beschreibung des geforderten Modellgehalts. In der EN 17412-1 [4] wird ein mögliches Konzept, der Level of Information Need (LOIN), normativ geregelt. Dazu müssen folgende begriffe definiert werden:

- LOG – Level of Geometry
  Gemäß Eichler et al. [24] sind geometrische Anforderungen hinsichtlich der Darstellung von Elementen im LOG definiert. Dies beinhaltet die Detaillierung des jeweilige Bauelementes in Abhängigkeit der Projektphase und stellt somit Vorgaben für den Planenden dar.Die Anforderungen werden in LOG-Klassen definiert.

  Die Plattform 4.0 [35] beschreibt die LOG-Klassen wie folgt, wobei die Bezeichnung Level of Detail dem LOG entspricht:

  – LOG 100: Das Modell besteht aus Volumenkörper zum Erkennen und Auswerten von Räumen und Raumgruppen. Die Volumenkörper können zur Ableitung von Kennzahlen herangezogen werden.
  – LOG 200: Das Modell ist in Baugruppen (Wände, Decken, Türen, Räume, ...) mit einzigartiger Bezeichnung und definierter Geometrie dargestellt. Erste nicht-geometrische Informationen sind bereits enthalten.
  – LOG 300: Das Modell beinhaltet detaillierte Elemente, sodass die Koordination und Vergabe der Gewerke möglich sind. Dies bedarf die Modellierung von Bauelementen mit eindeutiger Bezeichnung und weiterer nicht geometrischer Informationen. Elemente der Tragwerksplanung und der Gebäudeausrüstung sind enthalten.
  – LOG 400: Das Modell enthält einzelne Detailelemente, welche eine Werk- und Montageplanung bzw. eine Fertigung ermöglichen. Diese Bauelemente müssen somit mit endgültiger Geometrie dargestellt sein.
  – LOG 500: Alle Elemente des Modells sind wie die Detailelemente in LOG 400 dargestellt. Dabei handelt es sich um die tatsächlich gebauten Geometrien und Informationen für die spätere Nutzung.

- LOI – Level of Information
  Zufolge Eichler et al. [24] definiert der LOI Anforderungen an alphanumerische Informationen von Elementen. In Abhängigkeit von der Projektphase erfolgen Vorgaben zum Informationsgehalt der Elemente für die Planenden. Die Anforderungen werden in LOI-Klassen definiert. Die Klassen korrelieren mit jenen des LOG, jedoch gibt es keine allgemeine Definition. Diese hängen vom jeweiligen Projekt ab und sind eigens zu definieren.

- LOIN – Level of Information Need (Informationsbedarfstiefe)
  Gemäß Eichler et al. [24] werden die Anforderungen, welche der Aufgegebener an das Modell bezüglich Geometrie und alphanumerischen Informationen stellt, im LOIN abgebildet. Beide Informationen sind von projektspezifischen Anwendungsfällen abgeleitet. Aufgrund dessen wird vermieden, dass sowohl der LOG als auch der LOI irrelevante Informationen oder zu wenige enthält. Darüber hinaus fordert der LOIN die Dokumentationen je Anwendungsfall ein, da der LOIN stets von Menschen sowie Maschinen lesbar sein soll.
  Abb. 2.3 zeigt grafisch, wie sich der LOIN ableitet und wie dieser definiert wird. Dabei werden die Detaillierungsgrade LOG und LOI konkret (WAS > WIE + WANN + WER + WOZU) definiert:

- WAS: stellt den Elementbezug zum neutralen Dateiformat her
- WIE: beschreibt die Anforderungen (LOI, LOG)
- WANN: phasenbezogen durch die LOG- und LOI-Klassen
- WER: nimmt Bezug auf die verantwortliche Disziplin
- WOZU: Definiert den Zweck

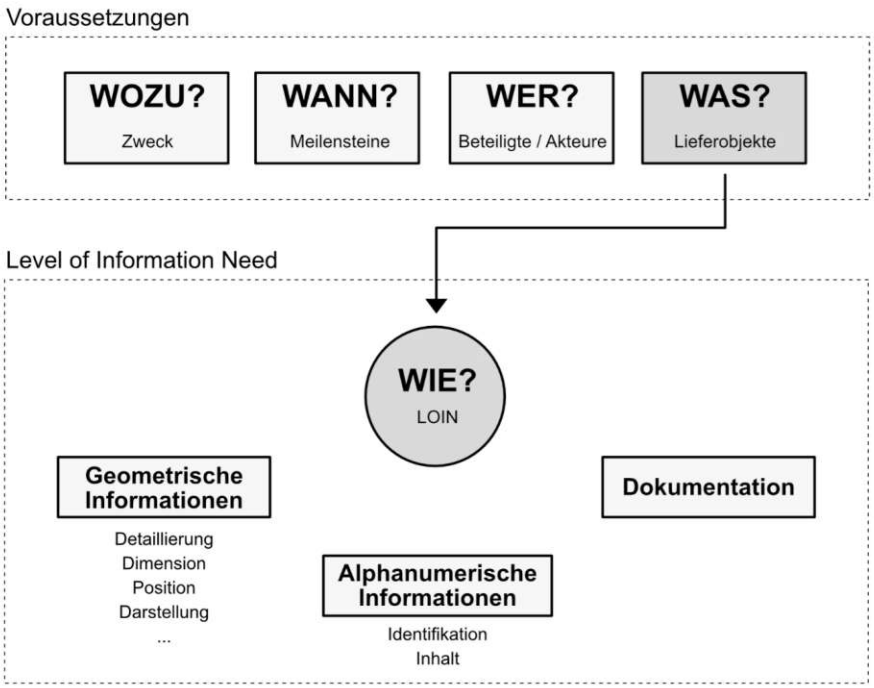


Abb. 2.3: Überblick LOIN [14]

An dieser Stelle sei angemerkt, dass durch die Einführung der EN 17412-1 [4] der LOIN das veraltete Konzept Level of Development (LOD = LOI + LOG) ersetzt.

2.1.4 Sonstige BIM-bezogene Begriffsbestimmungen

- Stakeholder
  Im Bezug auf den Gesamtprozess BIM bezieht sich der Begriff Stakeholder auf alle beteiligten Personen und/oder Gruppen, die direkt und indirekt am Bau- und Planungsprozess oder der Nutzung beteiligt sind. Weiters muss ein finanzielles, technisches und/oder regulatorisches Interesse am Projekt bestehen. Dazu zählen:
  - Auftraggeber/Bauherr*
  - Architekten und Ingenieurkonsulenten*
  - Projektsteuerung*
  - Ausführende
  - Facilitymanagement
  - Behörde*
  - Nutzer

Im Rahmen dieser Arbeit bezeichnet der Begriff Stakeholder nur jene Beteiligte, die an der Planung und Bewilligung beteiligt sind (durch * markiert).

- AIA – Auftraggeber-Informationsanforderungen
  Gemäß buildingSMART Austria [19] beschreibt der AIA das Informationsbedürfnis des Auftraggebers für ein BIM-Projekt. Diese wird für den Auftragnehmer definiert und dient als Basis für die Durchführung im jeweiligen Projekt. Folgende Punkte werden in den AIA dargelegt:
  - Projektinformationen
  - Allgemeine Vorgaben
  - Modellspezifische Vorgaben
  - Projektorganisation
  - Anwendungsfälle
  - Anhänge

  Mit diesen projektspezifischen Anforderungen erstellt die BIM-Projektsteuerung einen vorläufigen BIM-Abwicklungsplan (BAP). Dieser stellt einen expliziten Umsetzungsvorschlag für den gegeben AIA dar.

- Fachmodell(e)
  Gemäß Plattform 4.0 [35] beschreibt ein Fachmodell ein spezialisiertes, digitales Modell. Es wird von einer bestimmten Fachdisziplin erstellt, um spezifische Aspekte eines Bauprojekts zu repräsentieren. Jedes Fachmodell fokussiert sich auf die Bedürfnisse und Anforderungen einer bestimmten Disziplin wie Architektur, Tragwerksplanung und technische Gebäudeausrüstung. Allerdings können Fachmodelle auch abstrakter sein, beispielsweise ein Fluchtwegsmodell [25].

2.2 buildingSMART - IFC

Gemäß ihrer Website ist buildingSMART (bS) [23] eine globale Gemeinschaft von nationalen Verbänden (Chapter), Mitgliedern, Partnern und Sponsoren, die von buildingSMART International (bSI) geleitet werden. Die Gemeinschaft kümmert sich um die Erstellung und Entwicklung digitaler, offener Arbeitsweisen für das Bauwesen. Das österreichische Chapter buildingSMART Austria (bSAT) [20] ist einer der nationalen Verbände von buildingSMART. bSAT ist Verein, dessen Ziel die Entwicklung digitaler Lösungen in allen Sektoren des Bauwesens ist. Sie arbeiten eng mit den Schwesterorganisationen in Deutschland und der Schweiz zusammen, um so die Entwicklungsgeschwindigkeiten im deutschsprachigen Raum zu beschleunigen. Weiters führt die Zusammenarbeit die Verbandswelt von DIN, SIA, ÖNORM etc. zusammen. Laut buildingSMART [22] ist IFC eine standardisierte, digitale Beschreibung der gebauten Umwelt. Es stellt einen internationalen Standard (ISO 16739-1 [10]) dar, der herstellerneutral auf einer vielfältigen Anzahl von Hardwaregeräten und Softwareprodukten als Schnittstelle genutzt werden kann. IFC stellt dabei das wichtigste Ergebnis von buildingSMART International zur Erfüllung ihres Ziels dar, die Förderung von openBIM. Die standardisierte Datenstruktur kodifiziert Identität, Semantik und Merkmale von Objekten und verknüpft diese über Beziehungen miteinander. Der Nutzen von IFC ist vielseitig. So kann ein IFC-Modell zur Neuplanung oder Altbausanierungsplanung dienen. Ein weiterer Anwendungsfall ergibt sich, wenn ein Auftragnehmer ein bestehendes Gebäudemodell durch neue technische Gebäudeausrüstung ergänzt. Weiters kann IFC als Mittel der Archivierung in jeder Phase des Lebenszyklus genutzt werden. Bezüglich des Dateiformates können IFC-Daten

in diversen Formaten wie XML, JSON und STEP übersetzt und über Webdienste übertragen, in teilnehmenden Softwareprodukten importiert oder in zentralen Datenbanken abgelegt werden. Gemäß buildingSMART International [21] stellt das STEP Physical File (SPF) mit der Dateiendung .ifc das am meisten verwendete Format für die IFC-Datenstruktur dar. Es kann sowohl von Maschinen als auch von Menschen gelesen werden. Dabei basiert IFC-SPF auf dem in ISO 10303-21 [5] definierten Standard für „clear text representation of EXPRESS data models".

2.2.1 Aufbau der Datenstruktur

Laut Borrmann et al. [18] wird die IFC-Datenstruktur aufgrund der Komplexität und des Umfangs in mehrere Schichten (Layers) aufgeteilt. Dabei gilt grundsätzlich, dass Elemente der oben liegenden Schichten auf jene weiter unten verweisen dürfen, jedoch nicht umgekehrt. Der Core Layer ist dabei die grundlegendste Schicht des IFC-Datenmodells und enthält die Basisstrukturen, grundlegende Beziehungen und allgemeine Konzepte, die in höheren Schichten wiederverwendet und konkretisiert werden. Das Kernel-Schema stellt den Kern des IFC-Datenmodells dar und besteht aus abstrakten Basisklassen wie IfcRoot, IfcObject, IfcActor, IfcProcess, IfcProduct, IfcProject und IfcRelationship. Darauf bauen direkt die drei Erweiterungsschemata Product Extension, Process Extension und Control Extension auf, die ebenfalls zum Core Layer gehören. Für diese Arbeit besonders wichtig ist das Schema Product Extension, welches zur Beschreibung von physischen Objekten eines Bauwerks dient, die eine Geometrie und Verortung besitzen sowie deren Beziehungen. IfcProdukt beinhaltet Subklassen wie IfcBuilding, IfcSpace, IfcElement und IfcBuiltElement. Eine Stufe über dem Core Layer liegt der Shared Layer, welcher eine Ebene zwischen dem Kern des Datenmodells und den fachspezifischen Schemata darstellt. Klassen im Shared Layer werden für eine Mehrzahl an Anwendungen verwendet und umfassen unter anderem die Bauteilklassen (IfcWall, IfcBeam etc.). Den nächsten Layer bildet der Domain Layer, welcher fachspezifische Definitionen als Klassen enthält. Der letzte Layer, der Resource Layer, liegt wieder auf der unteren Ebene, ist jedoch nicht von IfcRoot abgeleitet. Aufgrund dieser Eigenschaft besitzen Klassen dieser Schicht keine Identität und können so nicht als Objekt instanziiert werden, sondern müssen von einer Instanz einer IfcRoot-Klasse referenziert werden.

Neben der Aufteilung in Schichten ist in der IFC-Datenstruktur als objektorientiertes Datenmodell auch die Vererbungshierarchie von großer Bedeutung. Diese legt generelle Vererbungsregeln fest und damit, welche Merkmale an Subklassen weitergeben werden. Den Startpunkt für die Vererbungshierarchie stellt die Klasse IfcRoot dar. Alle Klassen, die nicht zum Resource Layer gehören, leiten sich direkt oder indirekt aus IfcRoot ab. Dabei stellt die Klasse Grundfunktionen zur Festlegung eines Globaly Unique Identifiers (GUID) zur eindeutigen Identifikation, Herkunft des Objekts und die Modifikationsgeschichte (Ersteller und Bearbeiter, Versionen) zur Verfügung. Darüber hinaus ist es möglich, einen Namen und eine Beschreibung hinzuzufügen. Von IfcRoot leiten sich direkt drei Klassen der nächsten Ebene ab. Die Klasse IfcObjectDefinition vererbt an alle Klassen, die physikalische Objekte, Räume oder konzeptionelle Elemente (wie Prozesse) darstellen. Die zweite Subklasse stellt die Klasse IfcRelationship dar. Diese dient zur Beschreibung von Beziehungen zwischen Objekten. Die letzte Subklasse von IfcRoot ist die Klasse IfcPropertyDefinition. Diese ermöglicht die Festlegung von Merkmalen, die nicht in der IFC-Datenstruktur eingebunden sind. Eine wichtige Subklasse von IfcObjectDefinition stellt die Klasse IfcObject dar. Diese stellt ein eigenes, physisches Objekt dar. IfcObject vererbt an sechs Basisklassen, wobei im Kontext dieser Arbeit eine Subklasse besonders relevant ist:

- IfcProduct: ein physisches Objekt oder Raum-Element. Objekten dieser Klasse kann eine geometrische Form zugeordnet werden.
- Die anderen Subklassen lauten: IfcProcess, IfcControl, IfcResource, IfcActor und IfcGroup.

Da IfcProduct alle Objekte repräsentiert, denen eine geometrische Form zugeordnet werden können, umfasst diese Klasse alle Subklassen, die benötigt werden, um ein Bauwerksmodell virtuell abzubilden. Die weitere Vererbungshierarchie umfasst die Subklasse IfcElement welche Basisklassen wie IfcBuiltElement beinhaltet. Die Klasse IfcBuiltElement vererbt wiederum an wichtige Bauteil-Klassen, wie zum Beispiel IfcWall. Abb. 2.4 stellt die gesamte Vererbungshierachie für IfcBuildingElement und ihre Subklassen dar.

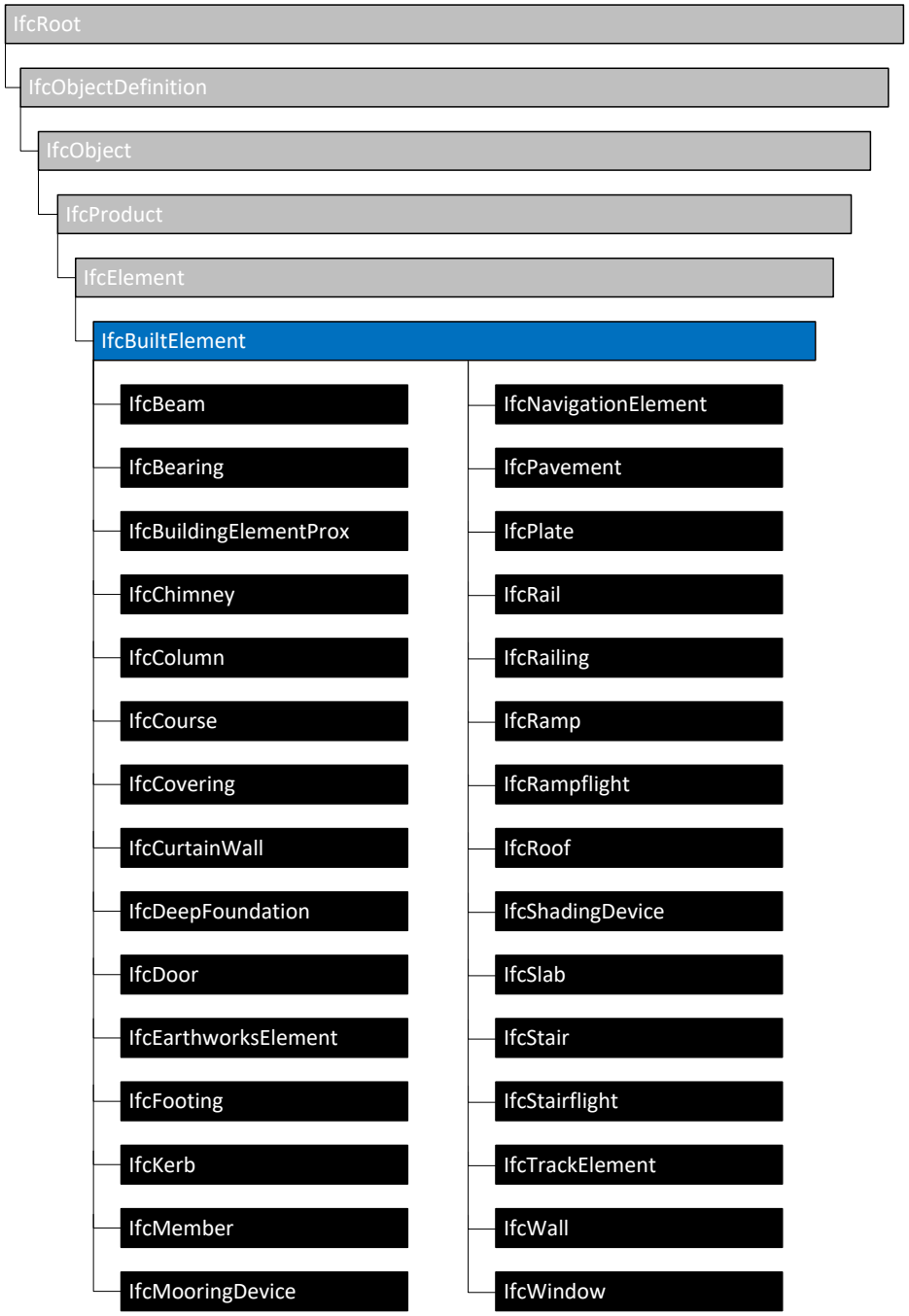


Abb. 2.4: IFC-Vererbungshierachie für IfcBuiltElement

Neben der beschriebenen Struktur erkkärt Borrmann et al. [18] Erweiterungsmechanismen. Obwohl rudimentäre Merkmale von Objekten bereits in der IFC-Datenstruktur durch Attribute in der Klassendefinition enthalten sind, kann die Gesamtheit des Bauwesens nicht abgebildet werden. Um die bereits komplexe Datenstruktur nicht noch komplizierter zu machen, wurde im IFC eine Zweiteilung von Merkmalsdefinitionen eingeführt: bereits im Schema definierte Merkmale, sogenannte Attribute, und die individuell erzeugbaren Merkmale, sogenannte Eigenschaften. Eigenschaften können mithilfe von IfcProperty und ihren Subklassen beliebig definiert werden. Ein Property besteht dabei aus einem Name-Wert-Einheit-Tupel. IfcPropertys werden gruppiert über IfcPropertySet und anschließend einem Objekt zugewiesen. Die Verwaltung der individuellen PropertySets kann mithilfe der bSDD-Plattform erfolgen.

2.2.2 bSDD-Plattform und BCF

Das Arbeiten mit der BIM-Methode erfordert umfangreiche Datenstrukturwerkzeuge und ausreichende Kommunikation. Dazu können die nachfolgenden Tools verwendet werden.

bSDD steht für buildingSMART Data Dictionary. Eichler et al. [24] beschreibt die Plattform als einen webbasierten Service zur Erstellung und Verwaltung von Datenstrukturerweiterungen zur Verfügung. Wichtig ist, dass bSDD keine Norm ist, sondern vollständig im Besitz von buildingSMART liegt. Die Plattform dient als Online-Bibliothek für Objekte und deren Merkmale. Alle abgelegten Inhalte bekommen sprachenabhängig eine Kennzeichnung und werden in ein Klassifizierungssystem abgelegt. Die Anforderungen, die sich aus einem Klassifizierungssystem ergeben, können an Modellelemente über eine eindeutige Kennung (bSDD-GUID) übermittelt werden. Weiters ermöglicht die Plattform, dass individuelle PropertySets, Merkmale und Merkmalwerte erstellt werden können. Die Verantwortung liegt stets bei jener Person/Institution, die für die Erstellung verantwortlich ist.

Zufolge BIMcollab [16] ist das BIM Collaboration Format (BCF), ein Dateiformat zur Kommunikation von Fehlern im BIM-Modell. Dabei kann es vollständig unabhängig des 3D-Modells die Fehler beschreiben bzw. kommunizieren. Die BCF-Datei besteht aus einem Screenshot, dem Winkel der Kamera und einer Liste, welche die betroffenen Modelelemente beinhaltet. Zusammen mit IFC und der eindeutigen Kennung durch den GUID ergibt sich ein umfassendes Kommunikationstool. So können auch Probleme eindeutig kommuniziert werden, wenn mehrere Teams aus unterschiedlichen Unternehmen mit unterschiedlicher Software am gleichen Projekt arbeiten.

2.3 BIM-Applikationen

Eichler et al. [24] beschreiben BIM-Applikationen als Teilgebiet der BIM-Werkzeuge, welche sich mit der Erstellung, Prüfung und Auswertung von Modelldaten befassen. Dabei muss die Applikation den Ansprüchen gemäß der BIM-Methode gerecht werden. Ob die benötigten Bedingungen gegeben sind, kann durch Überprüfung des Status der Applikation in ihrer buildingSMART-Zertifizierung erfolgen. Abb. 2.5 gibt einen Überblick über die verschiedenen Arten von BIM-Applikationen.

Die Grundlage für alle anderen bildet die Autorensoftware. Mit dieser werden die verschiedenen Modelle gemäß der jeweiligen Disziplin und Organisationseinheit erstellt. Mithilfe einer Prüfsoftware können Modellinhalte geprüft, aber nicht verändert werden. Ein Viewer hingegen kann nur Modellinhalte anzeigen, somit weder prüfen noch Inhalte verändern. Alle weiteren Applikationen beziehen bereits freigegebene Modellinhalte und nutzen diese für weitere Berechnungen und Auswertungen. In den nachfolgenden Punkten werden die wichtigsten Anforderungen für die in dieser Arbeit verwendeten BIM-Applikationen beschrieben.

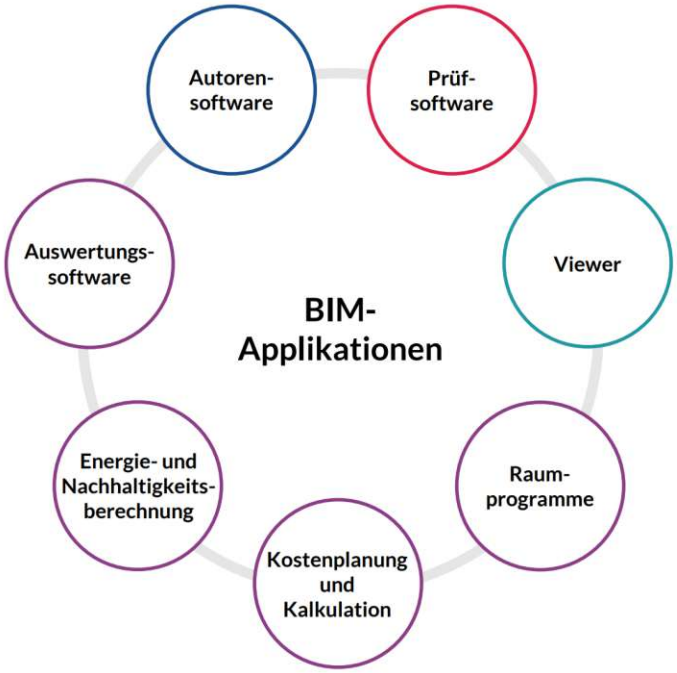


Abb. 2.5: Arten von BIM-Applikationen [24]

Autorensoftware

Gemäß Baunetz Wissen [12] ist eine Autorensoftware eine Softwareanwendung zur Erstellung für 3D-Konstruktionen und digitale Bauwerksmodelle, wobei diese aus parametrischen Modellelementen bestehen. Die Software kann sowohl generalisiert für mehrere Fachdisziplinen dienen, als auch für bestimmte Anwendungsfälle spezialisiert sein. Unabhängig davon, bieten gängige BIM-Autorensoftware nach aktuellem Stand der Technik zumindest folgende Funktionen:

- Möglichkeit, sowohl einfache als auch komplexe Geometrien zu erstellen und visualisieren
- Die Verknüpfung von geometrischen Modelelementen mit alphanumerischen Informationen sowie die Verwaltung von projekt- bzw. organisationsspezifischen Merkmalsets
- Den Im- und Export als IFC-Datei
- Die Möglichkeit des Nachrichtenaustauschs mit BCF oder eine ähnliche Funktion zur modelbasierten Kommunikation

Prüfsoftware

Eine Prüfsoftware ist gemäß Baunetz Wissen [13] eine Software für die Modelanalyse und Koordination. Dabei handelt es sich um Anwendungen, welche durch Autorensoftware erstellte 3D-Modelle zu einem Gesamtmodel zusammenführen, um allumfassende Anylsen/Prüfungen zur Qualitätssicherung durchzuführen. Dafür benötigt die BIM-Prüfsoftware folgende Funktionen:

- Die Möglichkeit, mehrere Fachmodelle zu einem Gesamtmodell zusammenzuführen
- Eine Kollisionsprüfung, um die geometrischen Inhalte auf ihre Lage zu überprüfen und Konfliktlösungen zu verfolgen
- Definition weiterer Regeln für eine regelbasierte Prüfung des Gesamtmodells sowohl hinsichtlich der Geometrie als auch alphanumerische Informationen
- Nachrichtenaustausch mit BCF oder eine ähnliche Methode zur Kommunikation

2.3.1 Solibri

Solibri [43] ist eine BIM-Prüfsoftware, die BIM-Modelle auf Integrität und Qualität prüft. Die Software basiert auf openBIM und kann mittels IFC-Schnittstelle Modelle aus allen gängigen Autorensoftware-Produkten importieren. In diesen können Fehler frühzeitig für den gesamten Lebenszyklus erkannt werden. Zur Sicherstellung der Qualitätsanforderungen bietet Solibri vordefinierte, anpassbare Prüfregeln an. Es gibt Prüfregeln für verschiedenste, qualitative Aspekte eines Bauwerkmodells, wie zum Beispiel Barrierefreiheit oder Parkplatzdimensionen. Besondere Merkmale der Software sind:

Datenaustausch: Solibri bietet einen eigenen „Solibri BCF Live Connector“, um Projektinformationen unter Projektpartner teilen zu können.

Automatische Prüfprozesse: Mithilfe von Solibri-Autorun können Aufgaben automatisiert werden, wie der Start der Überprüfung, die Erstellung des Prüfberichtes oder der Export der gefundenen Probleme als BCF.

Informationen für Berechnungen und Auswertungen: Die Software ermöglicht eine einfache Auswertung hinsichtlich der im Modell hinterlegten geometrischen und alphanumerischen Informationen. So können Massen schnell ermittelt und Elemente geprüft werden.

Modellprüfung ist mehr als nur eine Kollisionsprüfung: Mit zuvor erstellten Prüfregeln und den vordefinierten Prüfregelsets können diverse Prüfungen durchgeführt werden. So können sowohl geometrische Aspekte, wie Abstandsflächen aber auch alphanumerische Informationen, wie die Änderung einzelner Merkmale geprüft werden.

Anpassung an jede BIM-Rolle: Es wird jeder Fachdisziplin eine Bibliothek an vorgefertigten Prüfregelsets geboten. Weiters kann die Solibri-API [42] genutzt werden, um eigene Prüfregeln zu erstellen.

In dieser Arbeit wird die Version Solibri-Office 9.13.5.12 verwendet.

2.4 openBIM-Bewilligungsverfahren – BRISE-Vienna

BRISE-Vienna steht für Building Regulations Information for Submission Envolvement Vienna und ist ein Digitalisierungsprojekt, welches sich umfassend mit dem BIM-Bewilligungsprozess beschäftigt.Ein maßgebender Teil des Projektes ist die automatische bzw. teilautomatische Prüfung von gesetzlichen Bestimmungen an Bauwerke. So wie es für die Bauindustrie gesamt eine stufenweise Einführung von BIM-Technologien (BIM-Reifegradmodel) gibt, kann ein ähnlicher Aufbau für das Behördenverfahren herangezogen werden. Der digitale Reifegrad der Baueinreichung beschreibt laut Eichler et al. [24] das Bewilligungsverfahren in Level 0 bis 3 einzelner Kommunen. Abb. 2.6 zeigt die verschiedenen Level. Dabei steht Level 0 für die Verwendung zweidimensionaler, analoger Planunterlagen, welche in ausgedruckter Form manuell geprüft werden. Um Level 1 zu erreichen, ist eine Soll-Ist-Prozessanalyse notwendig, damit die benötigten technischen und gesetzlichen Entwicklungen bestimmt werden können. Ein wesentlicher Bestandteil dieses Levels ist die Umsetzung einer webbasierten Einreichung. Der nächste Schritt beinhaltet die Einbindung von openBIM-Modellen, um so die modellbasierte Einreichung und (teil-)automatische Prüfung zu ermöglichen. Ist dieser Punkt erfüllt, hat das Behördenverfahren das Level 2 erreicht. Damit Level 3 erreicht wird, müssen Grundlagen für die Planung dreidimensional zur Verfügung stehen. Besondere Bedeutung wird dabei der Bebaubarkeit zugesprochen.

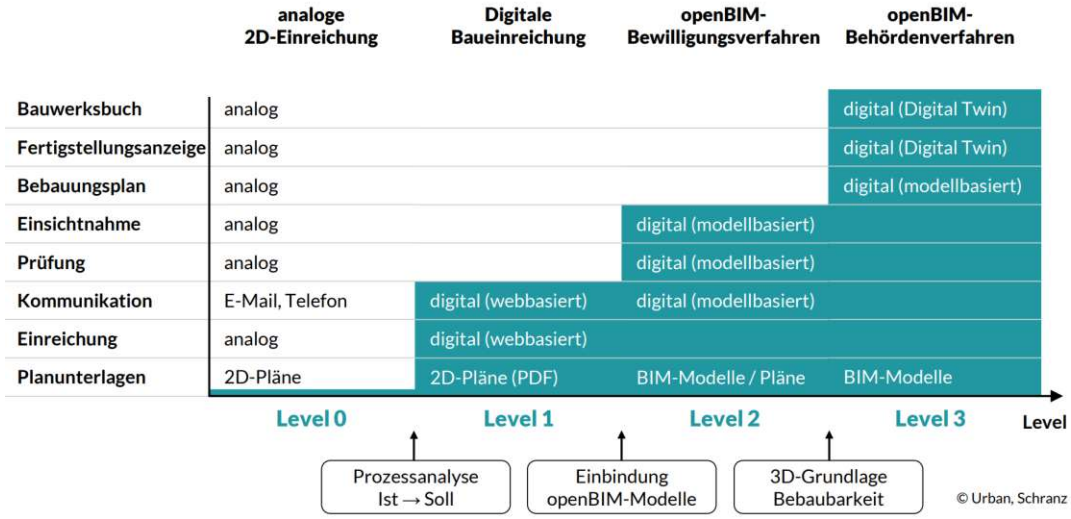


Abb. 2.6: Digitaler Reifegrad der Baueinreichung [24]

2.4.1 Rechtliche Grundlagen

Aktuell erfolgt die Baueinreichung und das Bewilligungsverfahren der Behörde anhand von 2D-Plänen, den sogenannten Einreichplänen. Da immer mehr Planende ihre Prozesse durch BIM unterstützen [37], soll die Behörde ihre Abläufe ebenfalls digitalisieren und so den Einreichprozess verbessern. Laut Urban et al. [46] dienen die BIM-Modelle in Form von Bauantragsmodellen als Grundlage für die behördliche Prüfung. Dabei erfolgt die Prüfung anhand der Wiener Bauordnung, des Wiener Garagengesetzes und den Flächenwidmungs- und Bebauungsplänen. Gemäß Krischmann [29] regelt unter anderem die Wiener Bauordnung Verfahrensabläufe, die Bebaubarkeit und Rechte und Pflichten der beteiligten Personen im Verfahren. Sie stellt auch eine Verbindung zu den Flächenwidmungs- und Bebauungsplänen und den Richtlinien des Österreichischen Instituts für Bautechnik (OIB) her. Die OIB-Richtlinien definieren Vorschriften hinsichtlich der mechanischen Festigkeit und Standsicherheit, des Brandschutzes, der Hygiene und Gesundheit sowie des Umweltschutzes, der Barrierefreiheit, des Schall- und Wärmeschutzes. Der Flächenwidmungs- und Bebauungsplan beinhaltet die baurechtliche Bebaubarkeit von Grundstücken. Neben typischen Merkmalen (z. B. Bauhöhe, Baulinien) kann der Flächenwidmungs- und Bebauungsplan auch besondere textuelle Bestimmungen beinhalten. In Abhängigkeit der Größe eines Bauvorhabens kann von behördlicher Seite in Wien nicht nur die MA 37 (Baubehörde), sondern auch andere Abteilungen am Projekt mitwirken. Diese können weitere Vorgaben (z. B. hinsichtlich des Ortsbildes) stellen, die ebenfalls verpflichtend sind. Im Wesentlichen lassen sich die Anforderungen in baurechtliche Themen und bautechnische Themen unterteilen. In Abb. 2.7 sind die rechtlichen Grundlagen zusammengefasst.

2.4.2 Ablauf des openBIM-Bewilligungsverfahrens

Laut Krischmann [29] werden in Wien jedes Jahr ca. 13 000 Bauvorhaben eingereicht. Die Magistratsabteilung 37 der Stadt Wien (MA 37) fungiert dabei als Baupolizei und ist somit für die Bewilligung bzw. für die Ablehnung der eingereichten Projekte zuständig. Die wichtigsten Bestandteile des Bewilligungsverfahrens im Projekt BRISE-Vienna sind das Bauantragsmodell (BAM), das Referenzmodell (REM), das Antragsinformationsmodell, welches mittlerweile in Service-Informationsmodell umbenannt wurde (SIM), und der 2,5D-Vermessungsplan (VMP). Letzter beinhaltet alle Grundstücke, Bauplätze, Fluchtlinien und Nachbargebäude und wird auf

Rechtsmaterie und Anforderungen	baurechtliche Themen	bautechnische Themen
Bauordnung	X	X
Wiener Garagengesetz	X	X
OIB		X
FWP/BBP	X	
Besondere textuelle Bestimmungen	X	
Behördliche projektbezogene Anforderungen	X	X

Abb. 2.7: Relevante Rechtsmaterie im Projekt BRISE-Vienna [29]

Basis des Flächenwidmungs- und Bebauungsplans erstellt. Außerdem sind öffentliche Einrichtungen (z. B. Straßen, Wege, Hydranten) und Widmungen dargestellt. Mithilfe des VMP sollen Planende unter Berücksichtigung der geltenden gesetzlichen Bestimmungen das BAM erstellen. Dieses Modell stellt das zentrale Einreichdokument dar und entspricht aus aktueller Sicht dem 2D-Genehmigungsplan. Die Grundlage für die Erstellung des BAM stellt in Österreich die ÖNORM A 6241-2 „Digitale Bauwerksdokumentation – Teil 2: Building Information Modeling“ dar. Nach Abschluss der Planungstätigkeiten ist der nächste Schritt, die IFC-Datei aus der Autorensoftware zu generieren. Für den Gesamtprozess ist es entscheidend das BAM im IFC-Format zu übergeben. Nur so können die geometrischen und alphanumerischen Informationen standardisiert ausgelesen und für die Prüfung genutzt werden. Neben den gesetzlichen Bestimmungen muss das BAM auch hinsichtlich LOG und LOI genau definiert sein. Das bedeutet, das Planende gegebenenfalls geforderte Informationen ergänzen müssen. Weiters sind Angaben zur Bebaubarkeit und der Umgebung, z. B. Nachbargebäude, notwendig. Diese Informationen können nicht gesichert dem BAM entnommen werden, da dieses von Planenden erstellt werden. Dies zeigt die Notwendigkeit eines zweiten Modells, welches diese Angaben korrekt abbildet. Das REM wird von der zuständigen Behörde auf Basis des VMP erstellt. Der VMP ist ein sogenannter 2,5D-Plan. Er enthält neben der üblichen zweidimensionalen Darstellung der Fluchtlinien und Gebäuden, Attribute, die eine automatisierte, dreidimensionale Generierung des REM ermöglicht. Das REM kann als Hülle interpretiert werden und dient zur Überprüfung des BAMs auf Überschneidungen mittels Kollisionsprüfung. In Wien können zusätzlich zum VMP textuelle Bestimmungen gesonderte Vorgaben für ein Grundstück definieren. Allgmein sind diese kein Teil des VMP. Allerdings sollen jene textuellen Bestimmungen, die Einfluss auf die Geometrie des Gebäudes haben, im REM ergänzt werden. Weitere textuelle Bestimmungen sind im SIM enthalten. Neben diesen Bestimmungen enthält das SIM auch die Vorgaben anderer am Behördenprozess beteiligter Abteilungen. Viele BIM-Prüfsoftwaren unterstützen lediglich die Prüfung von IFC-Dateien. Deswegen wird das SIM als standardisierte IFC-Datei angelegt und fasst die für die Prüfung relevanten Informationen aus den behördeninternen Datenbanken zusammen.

Sobald die drei Modelle vorliegen, kann die Modellprüfung mithilfe einer BIM-Prüfsoftware beginnen. Im Rahmen des Projekts BRISE-Vienna wird die Software Solibri verwendet. Die Modellprüfung besteht aus vielen einzelnen Prüfregeln, die die geltenden gesetzlichen Bestimmungen überprüfen. Für die bautechnische Prüfung wird dabei nur auf das BAM zugegriffen, während für baurechtliche Prüfungen das BAM gegen das REM und das SIM geprüft wird. Die bautechnische Prüfung, die vor allem auf numerisch erfassbaren Vorgaben basiert, kann vergleichsweise einfach in automatisierte Überprüfungen übersetzt werden. Hingegen lassen sich rechtliche Vorschriften ohne numerische Vorgaben nur schwer ohne menschliche Kontrolle überprüfen. Allerdings ergibt

sich für die Behörde trotzdem ein Mehrwert, da schwierige Sachinhalte durch die dreidimensionale Abbildung im BIM-Modell immer vollständig dargestellt sind.

2.4.3 Informationsanforderung an die Bauantragsmodelle

Zur Sicherstellung der Prüfbarkeit der Rechtsmaterie muss jedes eingereichte BAM hinsichtlich LOI und LOG einheitlich definiert sein. Dafür definierte das Team von BRISE-Vienna die Informationsanforderungen für das Bauantragsmodell. Neben den projektbezogenen Anforderungen für LOI und LOG kann für jede Elementklasse ein IFC-Beispiel bezogen werden. In Abb. 2.8 sind beispielhaft die Anforderungen für die Elementklasse Balken/Unterzug (IfcBeam) dargestellt. In Abb. 2.9 sind alle relevanten Elemente dieser Arbeit samt ihrer Merkmalsets und Merkmale aufgelistet. Weiters wird je Merkmal ein Beispiel gegeben.

2.4.4 Umsetzung der (teilautomatischen) Prüfung

Laut Urban et al. [46] basiert die Entwicklung der Prüfroutinen auf Basis der IFC-Datenstruktur sowie des geforderten Detailierungsgrads hinsichtlich LOI und LOG. Die erste Prüfungsebene ist ein formaler Kriterien-Check. In dieser Ebene wird das BAM auf die Vollständigkeit hinsichtlich der LOI-Anforderungen überprüft. Fällt diese Prüfung negativ aus, können die weiteren Ebenen nicht zufriedenstellend geprüft werden. Es ist auf eine möglichst allgemeine Formulierung der Prüfroutinen zu achten, um Modelle unabhängig ihrer Autorensoftware in gleicher Qualität zu prüfen. In der zweiten Ebene erfolgt die Prüfung des BAM hinsichtlich der Qualitätskriterien. Diese Prüfung beinhaltet zum Beispiel Kollisionsprüfungen. Diese Ebene ist für allgemeine BIM-Projekte von großer Bedeutung. Die dritte und relevanteste Prüfungsebene im Projekt BRISE-Vienna stellen die regulativen Prüfkriterien [24] dar. In dieser Ebene wird die zuvor definierte Rechtsmaterie geprüft. Die für die Prüfung erstellten Prüfroutinen werden dabei in drei Typen unterteilt:

Typ 1 Automatische Prüfroutinen: Die Prüfroutine wird gestartet und nach Fertigstellung der Prüfung liegt automatisiert ein Ergebnis vor. Zusätzliche menschliche bzw. manuelle Überprüfungen eines Referenten sind für diesen Typen nicht notwendig.

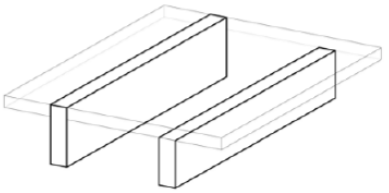
Typ 2 Teilautomatisierte Prüfroutine: Nach Fertigstellung der Prüfung liegt das Ergebnis graphisch vor. Allerdings ist es erforderlich, dass der Referent dieses Ergebnis zusätzlich bewertet.

Typ 3 Unterstützende Prüfroutine: Das Ergebnis der Prüfung steht als graphische Hilfestellung zur Verfügung und ist lediglich als Unterstützung gedacht. Eine manuelle Beurteilung durch die Referenten ist notwendig.

Ein Prüfroutine besteht dabei je nach Paragrafen aus einer oder mehreren Prüfregeln. Diese können verschachtelt zueinander angeordnet werden, was zu Regelkaskaden mit Torwächterregeln führt. Mithilfe der Standardregeln aus Solibri können bereits eine Vielzahl an Paragrafen der relevanten Rechtsmaterie für das Projekt BRISE-Vienna geprüft werden. Um noch mehr Paragrafen durch Typ 1 Prüfroutinen abzudecken, wird auf die Solibri-API-Schnittstelle zurückgegriffen. Zur Rechtfertigung des Entwicklungsaufwands einer neuer API-Prüfregeln wird bei jeder Neuentwicklung neben der Abdeckung des Use-Cases ein weiteres Ziel verfolgt: Bei der Entwicklung neuer Regeln ist eine möglichst breite Anwendbarkeit zu ermöglichen, um so nicht nur einen, sondern so viele Paragrafen wie möglich behandeln zu können.

Unabhängig ob Standardregeln oder API-Regeln verwendet werden, ergibt sich aus diesen Überlegungen nachfolgendes Bearbeitungsschema für die Erstellung einer Prüfroutine. Dieses basiert auf der dem Schema für API-Regeln gemäß Fischer [25].

LOG – Level of Geometry (IfcBeam)

Geometrische Definition:		Beispiel:
Definition: Ein Balken/Unterzug/Träger entspricht geometrisch der Definition gem. seiner IFC-Klasse (IFC = Industry Foundation Classes, ISO 16739:2018). **Modelliervorgabe:** Balken/Träger/Unterzüge sind inkl. ihrer Bekleidungen zu modellieren bis zu den relevanten Schichten ab 1cm.	**Erläuterung Elementklasse:** Ein Balken/Unterzug/Träger ist ein horizontales oder nahezu horizontales Strukturelement, das in der Lage ist, Lasten hauptsächlich durch Biegefestigkeit zu widerstehen. Gemäß ÖNORM A 6241-2 Anhang A ist ein Balken/Unterzug/Träger ein Sekundäres Bauelement.	

LOI – Level of Information (IfcBeam)

BRISE spez.	Merkmal zu übergeben	Merkmal Hilfestellung deutsch	Wertetyp	Werteinheit	Verortung	Auswahl-Set	Anmerkung
(Standard)	**IsExternal**	Außenbauteil	IfcBoolean	Wahrheitswert	Pset_BeamCommon	-	Defaultwert: FALSCH
(Standard)	**Status**	Status	IfcLabel	Text	Pset_BeamCommon	Auswahl-Set	Defaultwert: Bestand
(Standard)	**LoadBearing**	Tragendes Bauteil	IfcBoolean	Wahrheitswert	Pset_BeamCommon	-	Defaultwert: FALSCH
(Standard)	**FireRating**	Feuerwiderstandsklasse	IfcLabel	Text	Pset_BeamCommon	Auswahl-Set	Defaultwert: ND; Beispiel: EI 60
(Standard)	**ElementMainMateriality**	Hauptmaterialität Element	IfcLabel	Text	Pset_BeamSpecific	Auswahl-Set	Defaultwert: ND; Beispiel: Beton
x	**Fassadengestaltung**	Fassadengestaltung	IfcLabel	Text	WienBV_BauwerberObjektdaten	Auswahl-Set	Defaultwert: ND; Beispiel: Rankhilfe
x	**AufbautenNummer**	AufbautenNummer	IfcLabel	Text	WienBV_BeamSpecific	-	Defaultwert: XX-00; Beispiel: AW-01
x	**ObjektArt**	ObjektArt	IfcLabel	Text	WienBV_BauwerberObjektdaten	Auswahl-Set	Defaultwert: ND; Beispiel: Schwimmbecken oder BebauteFlaecheOberidisch

Die Kennzeichnung in der Spalte „BRISE spez." stellt die Unterscheidung zw. Standard-IFC-Merkmalen (aus Standard-BIM-Projekten) und BRISE-spezifischen Merkmalen = X dar.
Eine Beschreibung jedes Merkmals findet sich unter „Merkmal-Beschreibung (Hilfestellung)".

Abb. 2.8: Beispiel AIA für Balken im Projekt BRISE-Vienna

Element	Propertyset	Property	Beispiel
Bauwerksvolumen	WienBV_BauwerberObjektdaten	BauwerksteilArt	BGF unterirdisch
	WienBV_BauwerberObjektdaten	Fluchtniveau	20
Beam	Pset_BeamCommon	IsExternal	true
	Pset_BeamCommon	FireRating	R90
Columns	Pset_ColumnCommon	IsExternal	true
	Pset_ColumnCommon	FireRating	R90
Coverings	Pset_CoveringCommon	IsExternal	true
	Pset_CoveringCommon	FireRating	EI90
Door	Pset_DoorCommon	FireRating	EI90
	Pset_DoorCommon	SelfClosing	
Slab	Pset_Slabcommon	IsExternal	true
	Pset_Slabcommon	FireRating	REI90
	Pset_Slabcommon	LoadBearing	true
Space	Pset_SpaceSpecific	Widmungsbehörde	Verkehrserschließung
	WienBV_SpaceSpecific	Brandabschnitt	Name1
Wall	Pset_WallCommon	IsExternal	true
	Pset_WallCommon	FireRating	REI90
	Pset_WallCommon	LoadBearing	true
Window	Pset_WindowCommon	FireRating	EI90

Abb. 2.9: alphanumerische Anforderungen an Elemente

Analyse der Rechtsmaterie

Der erste Schritt zur Erstellung einer Prüfroutine ist die Analyse der Rechtsmaterie. Prüfroutinen-Entwickler prüfen zusammen mit Referenten der MA 37 Paragrafen der relevanten Rechtsmaterie auf Automatisierbarkeit. Das Ziel ist die Bewertung aller Paragrafen, ob diese als Prüfroutine abgebildet werden können. Gesetztestexte, die keine quantifizierbaren Aussagen (z. B. „ausreichende und geeignete Einrichtungen“) treffen, können nicht direkt automatisiert geprüft werden. Ein Beispiel dafür stellt § 93 (8) der Wiener Bauordnung dar, in welchem für die Brandbekämpfung ausreichende und geeignete Einrichtungen für die erste und erweiterte Löschhilfe gefordert werden. Eine weitere Einschränkung ergibt sich aus den vorhandenen Informationen der Modelle.

Das Ergebnis der Analyse ist die Rechtsinformationsmatrix (RIM). Die RIM fasst alle Paragrafen der relevanten Rechtsmaterie zusammen und gibt Auskunft darüber, ob und wie ein Paragraf durch eine Prüfroutine dargestellt werden kann. Im Zuge dessen erfolgt ebenfalls die Einteilung hinsichtlich des Prüfroutinen-Typs. Weiters wird unterschieden, ob diese mit Solibri-Standardregeln abgebildet werden kann oder ob eine Neuentwicklung mithilfe der Solibri-API notwendig ist.

Konzeptionsphase

In der Konzeptionsphase werden die Anforderungen an die einzelnen Prüfregeln einer Prüfroutine spezifiziert. Diese Phase wird unter enger Zusammenarbeit aller Projektpartner durchgeführt. Durch die Beteiligung der Behörde wird die Qualität der Prüfroutine sichergestellt. Durch die Beteiligung der Kammer der Ziviltechniker werden die Interessen der Planenden berücksichtigt. Mehraufwand durch neu geforderte Informationen sollen durch Erleichterungen und beschleunigte

Verfahren kompensiert werden. Sind Solibri-API-Regeln für die Umsetzung eines Paragrafen notwendig, erfolgt die Definition der Anforderungen ebenfalls in dieser Phase.

Implementierung
Die Implementierungsphase unterscheidet sich für Solibri-Standardregeln und Solibri-API-Regeln. Während erstere direkt eingesetzt und so die gesamte Prüfroutine aufgebaut werden kann, muss eine Solibri-API-Regel neu entwickelt werden. Die aus der Konzeptionsphase gewonnenen Anforderungen werden gegliedert und so das Userinterface und die notwendigen Funktionen erarbeitet. Die Implementierung variert für jede Solibri-API-Regel stark. Ein Grundkonzept für die Entwicklung wird in Kapitel 3 beschrieben. Unter Umständen deckt die Solibri-API-Regel nicht den gesamten Paragrafen ab. Es ist deshalb möglich, dass diese mit Solibri-Standardregeln oder anderen Solibri-API-Regeln zur einer Prüfroutine verschachtelt wird.

Validierung
Nach abgeschlossener Implementierung der neuen Prüfroutine muss diese validiert werden. Die Validierung erfolgt anhand von zwei Kriterien: Übereinstimmung mit der relevanten Rechtsmaterie und Abnahme der Prüfroutine durch die Stakeholder. Um den ersten Punkt zu überprüfen, wird die Prüfroutine an fiktiven Modellen getestet. Diese Modelle sind nach den Anforderungen hinsichtlich LOI und LOG des Projekts BRISE-Vienna erstellt. Mithilfe dieser ersten Prüfung der Prüfroutine kann die generelle Funktionalität bestimmt und eventuelle Fehler gefunden werden. Die Erfüllung vom zweiten Punkt erfolgt durch die Präsentation der Regel vor den Stakeholdern. Abhängig vom eingeholten Feedback erfolgt gegebenenfalls die Freigabe der Prüfroutine/API-Prüfregeln oder eine Anpassung.

Pilotphase
Um die Prüfroutinen abschließend zu inspizieren, erfolgt neben der Validierung mit fiktiven Modellen die Prüfung an echten, eingereichten Modellen in der Pilotphase. Die Pläne werden als pdf-Dateu webbasiert eingereicht und zusätzlich im Rahmen von BRISE-Vienna als BAM zur Verfügung gestellt. Die Überprüfung der relevanten Rechtsmaterie erfolgt anschließend sowohl mithilfe der 2D-Plänen als auch der BIM-Modelle. So kann die Qualität der Prüfroutinen und ihre Übereinstimmung mit der Rechtsmaterie sichergestellt werden.

Umsetzung
Diese fünf Stufen der Regelentwicklung stehen keinesfalls in einer linearer Abfolge. Es gibt mehrere Feedback-Schleifen, welche zur Wiederholung von verschiedenen Phasen führen kann. Abb. 2.10 zeigt den Zusammenhang und die Abhängigkeit der einzelnen Phasen.

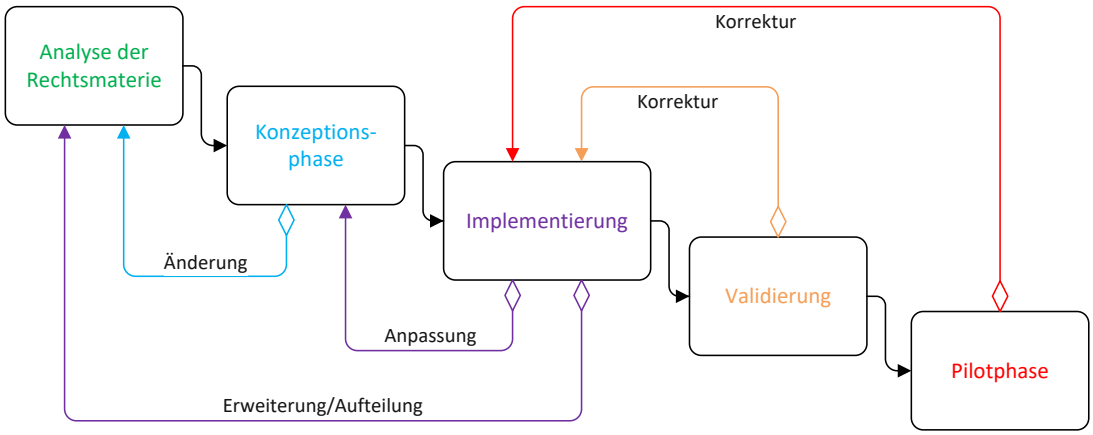


Abb. 2.10: Phasen der Prüfregelentwicklung

Zwischen der Analyse der Rechtsmaterie und der Konzeptionsphase besteht ein ständiger Austausch. Durch im Projekt erlangtes Knowhow können Paragrafen, welche zuerst als nicht automatisierbar eingestuft wurden, nochmals neu bewertet werden. Den zentralen Punkt der Regelentwicklung stellt die Implementierung dar. Neu gewonnene Erkenntnisse während der Umsetzung beeinflussen rückwirkend sowohl die Konzeptionsphase als auch die Analyse der Rechtsmaterie. Fehlerhafte Prüfroutinen, die während der Validierungsphase oder später während der Pilotphase gefunden wurden, müssen erneut in die Implementierungsphase zur Korrektur.

2.5 Relevante Rechtsmaterie für die Brandschutzanforderungen in Wien

Die Anforderungen hinsichtlich des Brandschutzes in Wien sind in verschiedenen rechtsbindenden Dokumenten dargestellt. Gemäß Kolbitsch [28] sind Gesetze Sammlungen von allgemein verbindlichen Rechtsnormen, die vom befugten Staatsorgan im Zuge eines Verfahrens erlassen worden sind. Vereinfacht ausgedrückt handelt es sich bei Gesetzen um die Festlegung von Regeln. Dabei findet in Österreich die Gesetzgebung auf Bundes- und Landesebene statt. Auf europäischer Ebene erfolgt die Gesetzgebung durch die EU-Kommission, welche neue Rechtsvorschriften ausarbeiten und diese an die zuständigen Generaldirektionen weiterleitet. Zum allgemeinen Verständnis sei in Abb. 2.11 das europäische Recht und ihre Stufen dargestellt. Für Österreich steht das Bundes-

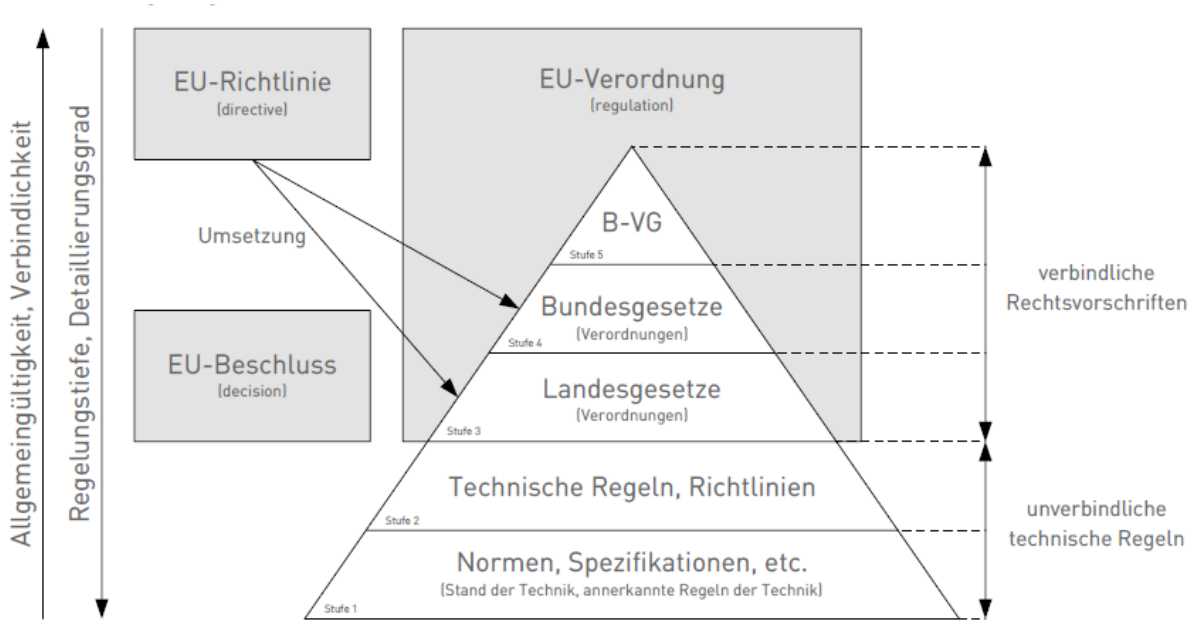


Abb. 2.11: Pyramide der Rechtslage [41]

verfassungsgesetz an oberster Stufe. Darauf folgen die Bundesgesetze und Landesgesetze, welche in ihrer Umsetzung den europäischen Richtlinien nicht widersprechen dürfen. Generell gilt, dass eine niedrigere Stufe durch eine höhere abgedeckt sein muss [47]. Unter den Gesetzen stehen noch unverbindliche Regeln wie Richtlinien, Normen und Spezifikationen. Die Wiener Bauordnung (WBO), das Wiener Garagengesetz und die Wiener Bautechnikverordnung entsprechen dabei der Rechtsstufe 3. Die Richtlinien des Österreichischen Institutes für Bautechnik (OIB) [34] sind in Stufe 4 eingegliedert und damit formal unverbindlich. § 1 der Wiener Bautechnikverordnung [49] lautet:

> *Den im 9. Teil der Bauordnung für Wien festgelegten bautechnischen Vorschriften wird entsprochen, wenn die in den Anlagen enthaltenen Richtlinien des Österreichischen Instituts für Bautechnik, soweit in ihnen bautechnische Anforderungen geregelt werden, eingehalten werden.*

Gemäß diesem sind die bautechnischen Anforderungen erfüllt, sofern die OIB-Richtlinien eingehalten sind. Damit gilt für diese Arbeit, dass die OIB-Richtlinien als zu überprüfendes Mindestmaß gelten. In § 2 der Wiener Bautechnikverordnung [49] wird beschrieben, dass eine Abweichung von den OIB-Richtlinien möglich ist, sofern das gleiche Schutzniveau gilt. Dieser Umstand findet in dieser Arbeit insofern Anwendung, dass die zu erreichenden Werte der OIB-Richtlinien als Mindestmaße gelten. Sollte der Planer stark vom Inhalt abweichen, sind eigene Nachweise zu erbringen, die nicht Teil dieser Arbeit sind. Nachfolgend werden die wichtigsten Grundlagen und Begriffe der gültigen Rechtslage dargelegt.

Bauwerk Gemäß OIB-Richtlinien [32] ist ein Bauwerk wie folgt definiert: *„Eine Anlage, die mit dem Boden in Verbindung steht und zu deren fachgerechter Herstellung bautechnische Kenntnisse erforderlich sind.“*

Gebäude sind überdachte, allseits- bzw. großteils umschlossene Bauwerke, die in Abhängigkeit ihrer Bestimmung von Personen betreten werden können [32].

Oberirdisches Geschoß Laut OIB-Richtlinien [32] ist ein oberirdisches Geschoß ein *„Geschoß, dessen äußere Begrenzungsflächen in Summe zu mehr als der Hälfte über dem anschließenden Gelände nach Fertigstellung liegen. Nicht zu den oberirdischen Geschoßen zählen solche, in denen sich keine Wohnungen, Betriebseinheiten oder Teile von solchen befinden (z. B. nicht ausgebaute Dachräume, Triebwerksräume, Räume für haustechnische Anlagen).“*

Unterirdisches Geschoß Laut OIB-Richtlinien [32] ist ein unterirdisches Geschoß ein *„Geschoß, dessen äußere Begrenzungsflächen in Summe zu nicht mehr als der Hälfte über dem anschließenden Gelände nach Fertigstellung liegen.“*

Fluchtniveau Das Fluchtniveau beschreibt die Höhendifferenz zwischen der Fußbodenoberkante des höchstgelegenen oberirdischen Geschoßes und der angrenzenden Geländeoberfläche nach Fertigstellung im Mittel [32].

Raumfläche In der ÖNORM B 1800 [7] bzw. dessen Beiblatt [6] wird die Bestimmung der verschiedenen Flächen definiert. Abb. 2.12 zeigt den Unterschied zwischen Brutto-Grundfläche, Netto-Grundfläche (NGF) und der Konstruktionsfläche (KGF).

Gebäudeklassen Gebäude werden in Abhängigkeit verschiedenster Faktoren in sogenannte Gebäudeklassen eingeteilt [32]. Diese spezifizieren Anforderungen hinsichtlich des Brandschutzes. Folgende Einteilung wird in den OIB-Richtlinien verwendet:

Gebäudeklasse 1 (GK1) Dabei handelt es sich um freistehende Gebäude mit nicht mehr als drei oberirdischen Geschoßen. Die Brandbekämpfung kann von mindestens drei Seiten erfolgen. Das Fluchtniveau ist geringer als 7 m und die Brutto-Grundfläche der oberirdischen Geschoße liegt unter 400 m^2. Die Geschoße bestehen entweder aus maximal zwei Wohnungen oder einer Betriebseinheit.

Gebäudeklasse 2 (GK2) Die Bedingungen unterteilen sich in drei Kategorien:

- Gebäude mit maximal drei oberirdischen Geschoßen, einem Fluchtniveau von weniger als 7 m und einer oberirdischen Brutto-Grundfläche von $\leq$ 400 m^2.

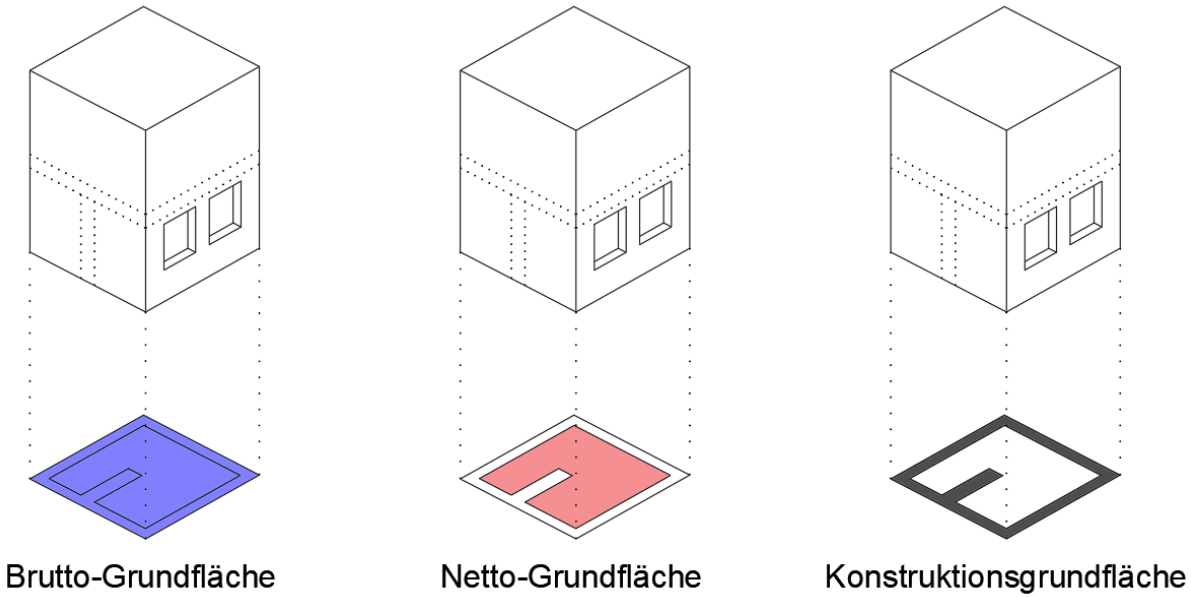


Abb. 2.12: Raumflächen (Eigendarstellung gemäß ÖNORM B 1800 [7])

- Reihenhäuser mit maximal drei oberirdischen Geschoßen, einem Fluchtniveau von weniger als 7 m und einer oberirdischen Brutto-Grundfläche von kleiner-gleich 400 m^2, wobei sich diese aus Wohnungen bzw. Betriebseinheiten zusammensetzen dürfen.
- freistehende Gebäude mit nicht mehr als drei oberirdischen Geschoßen. Die Brandbekämpfung kann von mindestens drei Seiten erfolgen. Das Fluchtniveau ist geringer als 7 m und die Brutto-Grundfläche der oberirdischen Geschoße liegt unter 800 m^2. Nur reine Wohnnutzung ist zulässig.

Gebäudeklasse 3 (GK3) Gebäude mit einem Fluchtniveau unter 7 m und maximal drei oberirdischen Geschoßen, die nicht in GK1 oder GK2 fallen.

Gebäudeklasse 4 (GK4) Dabei handelt es sich entweder um:

- Gebäude mit maximal 4 oberirdischen Geschoßen und einem Fluchtniveau von maximal 11 m. Das Gebäude besteht aus mehreren Wohnungen oder Betriebseinheiten, welche jeweils maximal 400 m^2 Nutzfläche haben dürfen, oder
- Gebäude mit maximal 4 oberirdischen Geschoßen und einem Fluchtniveau von 11 m bestehend aus einer Wohnung/Betriebseinheit. Hier besteht keine Beschränkung der Bruttogrundfläche.

Gebäudeklasse 5 (GK5) Alle Gebäude, die nicht in GK1 bis GK4 fallen und ein Fluchtniveau unter 22 m haben.

Brandabschnitt Sowohl die OIB-Richtlinien [32] als auch § 87 (5) der Wiener Bauordnung [15] definieren Brandabschnitte als Bereiche, die durch brandabschnittsbildende Bauteile von anderen Teilen des Gebäudes getrennt sind.

Trennwand/Trenndecke In den OIB-Richtlinien [32] liegen Trennwände und -decken zwischen Wohnungen bzw. Betriebseinheiten selbst sowie zu anderen Gebäudeteilen.

2.5.1 Klassifizierungen

Sowohl Baustoffe als auch Bauteile werden hinsichtlich ihres Brandverhaltens in Kategorien eingeteilt. Die Kriterien und Einteilung von Baustoffen erfolgt in der Normenreihe ÖNORM EN 13501. Abb. 2.13 enthält die Klassen A1 bis F und eine kurze Beschreibung, entsprechend der ÖNORM EN 13501-1 [8].

Klassen	Kriterien
A1	Bauprodukte dieser Klasse sind nicht brennbar. Damit haben sie keine Auskwirkung auf das Brandgeschehen.
A2	Bauprodukte dieser Klasse sind nicht brennbar. Sie entsprechen jener Bauprodukte der Klasse B mit der zusätzlichen Bedingung, keinen Beitrag zum Brandgeschen zu haben.
B	Bauprodukte dieser Klasse sind sehr schwer entflammbar und entsprechen mindestens der Klasse C mit der zusätzlichen, erhöhten Anforderungen
C	Bauprodukte dieser Klasse sind schwer entflammbar und stellen zusätzliche Anforderungen an die seitliche Ausbreitung
D	Bauprodukte dieser Klasse sind normal entflammbar
E	Bauprodukte dieser Klasse sind normal entflammbar, jedoch hindern sie den Brand an der Ausbreitung über einen kürzeren Zeitraum als die Klasse D
F	Bauprodukte dieser Klasse sind leicht entflammbar - An Produkte dieser klasse werden keine Anforderungen gestellt und somit wird auch kein Prüfverfahren durchgeführt

Abb. 2.13: Baustoffklassen gemäß ÖNORM EN 13501-1 [10]

Bauteile werden in der ÖNORM EN 13501-2 [9] hinsichtlich des Feuerwiderstandes R, E und I sowie den zugehörigen Widerstandszeiten eingeteilt. Diese und weitere Merkmale sind in Abb. 2.14 zusammengefasst.

Klassen	Eigenschaft
R	Tragfähikeit
E	Raumabschluss
I	Wärmedämmung
W	Wärmestrahlung
M	Widerstand gegen mechanische Beanspruchung
C	Selbstschließend (Türen und Fenster)
S	Rauchdichtheit
G	Widerstand gegen Rußbrand
K	Brandschutzfunktion für Wand- oder Deckenbekleidungen
tt/ttt	Klassifizierungszeit in Minuten während alle der genannten Kriterien erfüllt sind

Abb. 2.14: Bauteilklassen gemäß ÖNORM EN 13501-2 [9]

2.5.2 Bestimmungen gemäß Wiener Bauordnung

Die Wiener Bauordnung regelt baurechtliche und bautechnische Themen und stellt damit Forderungen an die Planung bzw. Ausführung und gibt einen Rahmen für die Umsetzung vor. Im Folgenden werden die relevanten Paragrafen genannt und kurz beschrieben.

Allgemein wird in Abschnitt 3 der WBO [15] das Thema Brandschutz beschrieben. Bauwerke müssen so geplant und ausgeführt sein, das Leben und Gesundheit geschützt sowie die Brandausbreitung wirksam eingeschränkt wird. Weiters fordert die WBO, dass die Tragfähigkeit über einen gewissen Zeitraum erhalten bleibt, sodass Flucht und Rettung der Benutzer des Bauwerks möglich sind. Dabei ist die Größe und der Verwendungszweck des Bauwerks zu berücksichtigen. Zur Verringerung der Ausbreitung von Feuer und Rauch innerhalb des Bauwerks müssen Bauteile die Nutzungseinheiten begrenzen, einen Feuerwiderstand aufweisen, die die unmittelbare Gefährdung von Personen ausschließt und die Brandausbreitung einschränkt. Des Weiteren wird in § 93 (3) der WBO folgendes gefordert:

> *„Bauwerke sind in Brandabschnitte zu unterteilen, wenn es auf Grund des Verwendungszweckes oder der Größe des Bauwerkes zur Sicherung der Fluchtwege und einer wirksamen Brandbekämpfung erforderlich ist. Insbesondere ist eine zweckentsprechende Größe und Anordnung der Brandabschnitte erforderlich. Die den einzelnen Brandabschnitt begrenzenden Bauteile müssen die Brandausbreitung wirksam einschränken.“*

Dabei sind eigene Brandabschnitte jedenfalls für Räume mit erhöhter Brandgefahr und für Räume mit sicherheitsrelevanten Einrichtungen anzulegen. Außerdem werden Anforderungen an

Fassaden einschließlich der Dämmstoffe, Unterkonstruktion, Verankerungen sowie an Hohlräume in Bauteilen gestellt. Zur Bekämpfung der Ausbreitung des Brands müssen entsprechende und ausreichende Einrichtungen für die erste und erweiterte Löschhilfe vorhanden sein. Zur Vorbeugung der Brandübertragung auf andere Bauwerke werden Bedingungen in Abhängigkeit der Distanz zum Nachbargebäude gestellt. Außenwände sind so auszuführen, dass das Übergreifen auf andere Bauwerke verhindert wird. Durch entsprechende Ausbildung der Dachdeckungen, Dachaufbauten und sonstige Elemente von Dächern ist die Brandentstehung durch Flugfeuer zu vermeiden. In Abb. 2.15 sind die relevanten Paragrafen aufgelistet. Es erfolgt eine Kurzbeschreibung und anschließend die Einteilung, ob diese im Rahmen dieser Arbeit in der Überprüfung umgesetzt, teilweise umgesetzt oder nicht umgesetzt wurden.

Paragraf	Punkt	Kurzbeschreibung	Umsetzung		
			Ja	teilw.	nein
§91		Allgemeine Anforderungen		x	
§92	(1)	Tragfähigkeit muss über gewissen Zeimtraum erhalten bleiben	x		
	(2)	Schäden durch Einsturz am Nachbargebäude müssen vermieden werden			x
§93	(1)	Ausbreitung von Feuer und Rauch innerhalb des Bauwerkes		x	
	(2)	Bauteile zur Abgrenzung von Nutzungseinheiten müssen einen gewissen Feuerwiderstand aufweisen			x
	(3)	Bauwerke sind in Brandabschnitte zu Unterteilen	x		
	(4)	Brandabschnitte für besondere Räume		x	
	(5)	Anforderungen an Fassaden			x
	(6)	Anforderungen an Hohlräume in Bauteilen	x		
	(7)	Anforderungen an Feuerungsanlage			x
	(8)	Vorhanden sein von Löscheinrichtungen			x
§94	(1)	Ausbreitung von Feuer auf andere Bauwerke		x	
	(2)	Anforderungen an Außenwände		x	
	(3)	Anforderungen an Dächer			x
	(4)	Öffnungen in Feuermauern			x
§95		Fluchtwege			x
§96		Rettung und Löscharbeiten			x

Abb. 2.15: Relevante Paragrafen gemäß Wiener Bauordnung

2.5.3 Bestimmungen gemäß Wiener Garagengesetz

Das Wiener Garagengesetz [50] beinhaltet Bestimmungen über Anlagen zum Einstellen von Kraftfahrzeugen, kraftbetriebene Parkeinrichtungen und Tankstellen. Im Wesentlichen beschreibt

das Gesetz, dass kraftbetriebene Parkeinrichtungen und Tankstellen nach dem Stand der Technik geplant und ausgeführt werden müssen, sodass die notwendigen Erfordernisse hinsichtlich des Brandschutzes gewährleistet sind. Des Weiteren werden bei Tankstellen Anforderungen hinsichtlich der oberirdischen Lagerung von brennbaren Flüssigkeiten außerhalb von Gebäuden gestellt. An Lagerräume, Füllstellen und Zapfsäulen gibt es darüber hinaus konkrete Anforderungen, die brandabschnittsbildenden Elemente betreffen.

Diese Arbeit zielt auf die Überprüfung der Brandabschnitte mit Wohnnutzung ab, weshalb sowohl Garagen und Stellplätze als auch Tankstellen und Lagerräume für brennbare Flüssigkeiten nicht näher betrachtet werden.

2.5.4 Bestimmungen gemäß OIB-Richtlinien

Die Anforderungen an den Brandschutz in den OIB-Richtlinien unterteilt sich in vier Dokumente:

- OIB-Richtlinie 2: Brandschutz
- OIB-Richtlinie 2.1: Brandschutz bei Betriebsbauten
- OIB-Richtlinie 2.2: Brandschutz bei Garagen, überdachten Stellplätzen und Parkdecks
- OIB-Richtlinie 2.3: Brandschutz bei Gebäuden mit einem Fluchtniveau von mehr als 22 m

Der Fokus dieser Arbeit ist die Überprüfung von Brandabschnitten mit Wohnnutzung. In weiterer Folge werden deswegen lediglich die OIB-Richtlinien 2 und 2.3 behandelt. Die OIB-Richtlinie 2 regelt allgemein die bautechnischen Anforderungen hinsichtlich des Brandschutzes. So definiert diese Bedingungen sowohl für Baustoffe (Tabelle 1a) als auch für Bauteile (Tabelle 1b), sofern nicht abweichende Anforderungen definiert sind. Für die Ausbreitung des Brands innerhalb des Gebäudes unterteilt diese Richtlinie die Anforderungen in mehrere Bereiche. Brandabschnitte sind abhängig von der Lage der Geschoße (oberirdisch/unterirdisch) hinsichtlich ihrer Netto-Grundfläche, Längsausdehnung und Geschoßanzahl zu beschränken. Die Abgrenzung untereinander muss durch brandabschnittsbildende Elemente gemäß den Tabellen 2a, 2b und 3 dieser OIB-Richtlinie erfolgen, welche weitere Kriterien hinsichtlich ihrer Dimensionen einzuhalten haben. An Öffnungen in diesen Elementen werden ebenfalls konkrete Anforderungen gestellt. Für Trenndecken und -wände gelten Anforderungen gemäß Tabelle 1b, wobei besonders für Türen in Trennwänden gesonderte Anforderungen gelten. Weiters definiert die Richtlinie Anforderungen an deckenübergreifende Außenwandstreifen, Schächte und Leitungen, Fassenden, Aufzüge, Feuerstätten und Verbindungsstücke, Abgasanlagen, Räume mit erhöhter Brandgefahr und die Löschhilfe. Hinsichtlich der Brandausbreitung auf andere Bauwerke definiert die OIB-Richtlinie 2 Abstände beziehungswiese Maßnahmen bei Nichteinhaltung dieser. Die OIB-Richtlinie 2.3 regelt besondere bautechnischen Anforderungen für Gebäude mit einem hohen Fluchtniveau. Die Bestimmungen zielen vor allem auf Fluchtwege und Sicherheitseinrichtungen wie Brandmeldeanlagen und Löscheinrichtungen ab. Hinsichtlich der brandabschnittsbildenden Elemente wird allgemein eine Widerstandszeit von 90 min festgelegt. Weiters werden strengere Grenzen für die Netto-Grundfläche in Abhängigkeit zu Geschoßzahl und Lage definiert.

In Abb. 2.16 sind die wesentlichen Punkte aufgelistet. Da die beiden Richtlinien viele Themen hinsichtlich des Brandschutzes behandeln, werden nur jene Punkt angeführt, welche im direkten Zusammenhang mit Brandabschnitten stehen. Alle Punkte, die nicht aufgelistet sind, gelten somit als nicht umgesetzt. Nach einer Kurzbeschreibung wird anschließend angeführt, ob diese im Rahmen der Überprüfung umgesetzt, teilweise umgesetzt oder nicht umgesetzt wurden.

Punkt	Absatz		Kurzbeschreibung	Umsetzung Ja	teilw.	nein
			OIB-Richtlinie 2			
3	1	1	Anforderungen an oberirdische Brandabschnitte	x		
		2	Anforderungen an unterirdische Brandabschnitte	x		
		3	Brandabschnitte sind durch brandabschnittsbildende Bauteile zu trennen	x		
		4	mechanische Beaspruchung			x
		5	brandabschnittsbildende Wände müssen über das Dach			x
		6	Anforderungen an Öffnungen in brandabschnittsbildenden Elemente	x		
		7	deckenübergreifender Außenwandstreifen			x
		8	Türen, Tore, Fenster und sonstige Öffnungen in Außenwänden			x
		9	Abstand von Dachöffnungen zu brandabschnittsbildenden Wänden			x
		10	Abstände von Dachöffnungen zu höhere lingenden			x
			OIB-Richtlinie 2.3			
2	2	1	Anforderungen an Sicherheitstreppenhäuser			x
		2	Allgemeine Anforderungen an Bauteile		x	
		3	Anforderungen an nichttragende Trennwände			x
2	4	1	Strengere Anforderungen an Brandabschnitte	x		
		2	deckenübergreifender Außenwandstreifen			x

Abb. 2.16: Relevante Paragrafen gemäß OIB-Richtlinien

2.6 Solibri-API

Zur Erreichung der Zielsetzungen des Projekts BRISE-Vienna müssen die Standard-Prüfregeln von Solibri durch die Nutzung der Solibri-API [42] erweitert werden. API steht für Application Programming Interface und bezeichnet einen Satz von Befehlen, Funktionen und Objekten, die Programmierer verwenden können, um mit einem bestehenden System zu interagieren. Im Fall von Solibri sind das Methoden in der Programmiersprache Java [33], welche die Erweiterung der bestehenden Prüfregelsets ermöglichen. Im Zusammenhang mit Solibri und der Solibri API wird von *Nutzer* gesprochen. Ein Nutzer wendet Solibri Prüfregeln bzw API-Prüfregeln zur Prüfung von BIM-Modellen an. Das kann z. B. die Baubehörde der Stadt Wien sein. Planende, die Modelle für die Einreichung planen, liefern die zu prüfenden Modelle. Wendet ein Planender Prüfregeln zur Kontrolle seines Modells an, nimmt eine Person beiden Rollen an.

2.6.1 Begriffsbestimmungen Java und Vererbungshierarchie

Bonacina [17] beschreibt in seinem Buch Java als objektorientierte Programmiersprache. Objektorientiert steht für ein Programmierungsparadigma, das auf dem Konzept von Objekten basiert.

Ähnlich wie in der realen Welt, in der Objekte bestimmte Eigenschaften besitzen, können in Java Objekte mit eigens definierten Eigeschaften erstellt werden. Ziel ist es, den komplexen Aufbau von Softwareprodukten durch dieses System abzubilden und besser verständlich zu gestalten. Die nachfolgenden Begriffe bilden die Basis für objektorientiertes Programmieren:

Variablen dienen dazu, bestimmte Werte zwischenzuspeichern. Auf Basis von Variablen können mithilfe von Operatoren diese geändert, in anderen Variablen gespeichert oder am Bildschirm ausgegeben werden. Um sie verwenden zu können, müssen diese deklariert werden. Mit der Deklaration einer Variable wird für diese entsprechender Speicherplatz im Computersystem reserviert. In Java ist es außerdem notwendig, der Variable einen Typ zu zuordnen. Variablen können primitive Datentypen wie Integer oder Float annehmen, oder Klassen entsprechen. Für diese Arbeit werden die typschen Java-Variablentypen Boolean, Integer, Double, String, Array, ArrayList und Hashmap verwendet.

Klassen sind in der objektorientierten Programmierung eine abstrakte Vorlage oder Blaupause, die die gemeinsamen Eigenschaften und das Verhalten einer Reihe von Objekten beschreibt. Es handelt sich um eine Art Blaupause für die Erstellung von Objekten, die Instanzen dieser Klasse sind. In Klassen können Attribute und Methoden definiert werden. Variablen, die für die gesamte Klasse deklariert werden, nennt man Klassenvariablen. Häufig werden diese ganz am Anfang definiert, sodass sie für den gesamten nachfolgenden Code zugänglich sind. Klassen können von einer anderen Klassen Eigenschaften erben.

Objekte sind konkrete Instanzen von Klassen und werden in Variablen abgespeichert. Eine Instanziierung der Klasse ist notwendig, um mit den vorgegebenen Strukturen arbeiten zu können. Jedes Objekt teilt sich zwar die gegebene Vorlage der Klasse, kann jedoch die konkreten Werte gesondert befüllen und fungiert somit eigenständig.

Methoden sind in einer Klasse definierter Codeblöcke, die bestimmte Aufgaben oder Operationen ausführen. Sie werden benötigt, um die Funktionalität der Klasse zu organisieren und zu strukturieren. Beim Deklarieren eines Objekts können die in der zugehörigen Klasse definierten Methoden aufgerufen werden. Zwei besonders wichtige Methoden sind Getter- und Setter-Methoden. Diese bieten die Möglichkeit, Variablen, die in einer Klassen definiert wurden, für ein bestimmtes Objekt zu befüllen oder abzurufen.

Interfaces (Schnittstellen) stellen sicher, dass bestimmte Eigenschaften oder Methoden in einer implementierenden Klasse vorhanden sind. Sie stellen ein wesentliches Element für die Vererbungshierarchie dar. Da eine Klasse nur von einer anderen Klasse, jedoch von mehreren Interfaces erben kann, können so komplexe Hierarchien programmiert werden.

Vererbung: Durch die Vererbung kann eine hierarchische Reihenfolge für Klassen festlegt und so die Menge an redundantem Code reduziert werden. Das bedeutet, dass die Eigenschaften und Methoden einer Klasse vererbt werden können, ohne sie erneut programmieren zu müssen. Ein weiterer Vorteil besteht darin, dass durch die Vererbung mittels Interface sichergestellt ist, dass gewisse Attribute immer vorhanden sind. So kann im Kontext einer globalen Software darauf vertraut werden, dass benötigte Ergebnisse sicher bereitgestellt werden.

2.6.2 Aufbau einer Solibri-API-Prüfregel

Im Rahmen des Projekts BRISE-Vienna definierten die Mitarbeiter des Forschungsbereichs Digitaler Bauprozess der TU Wien einen Standardaufbau für Solibri-API-Prüfregel. Dieser setzt

sich im Wesentlichen aus den benötigten Imports von Java-Bibliotheken und der gewählten Solibri-Regelklasse zusammen. Die Regelklasse unterteilt sich in drei Abschnitte: Zu Beginn werden die Klassenvariablen definiert, danach folgt die Prüfung und am Ende jeder Regelklasse ist die Definition des Userinterfaces angeordnet. Mit diesem Standardaufbau erleichtert sich die Zusammenarbeit und Wartung der einzelnen Prüfregeln. Kernelement der Solibri-API ist das Rule-Interface. Dieses definiert grundlegende Methoden und regelt den Ablauf und die Integration in die Hauptsoftware Solibri. Die Klassen OneByOneRule und ConcurrentRule erben direkt vom Rule-Interface. Bei der Implementierung einer API-Prüfregel wird üblicherweise eine der beiden Klassen erweitert. An dieser Stelle soll angemerkt werden, dass in einer eigenen Prüfregel auch direkt das Rule-Interface implementiert werden kann. Da aber die beiden Klassen nützliche Funktionen für die weitere Bearbeitung bieten, konzentriert sich die weitere Beschreibung auf diese. Zu den Klassenvariablen zählen die sogenannten Ruleparameter. Diese definieren, welche Elemente später im Userinterface angezeigt werden können. Abgerufene Elemente können direkte Eingaben der Nutzer selbst sein (z. B. Zahlenwerte) oder Bezug auf Modellinformationen nehmen, beispielsweise die Merkmale einer Wand. Der wichtigste Ruleparameter ist der DefaultFilterParameter. Dieser ermöglicht die Auswahl der zu überprüfenden Elemente und stellt damit die Basis einer API-Prüfregel dar. Dabei werden nur dem DefaultFilterparameter innerhalb einer Wasserfallanordnung Komponenten übergeben. Deswegen ist die Unterscheidung und klare Abgrenzung zu weiteren FilterParameter entscheidend und muss auch im Userinterface deutlich gekennzeichnet sein. Bis auf den DefaultFilterParameter benötigen alle anderen Userinterface-Elemente eine Instance des Interfaces RuleParameter. Abgeschlossen werden die Standardklassenvariablen mit einem Bezug zu den Regelressourcen. Über diesen erhält die Regel Zugriff auf Bilddateien und die Property-Datei, in welcher Anzeigenamen von Userinterface-Elementen oder Ergebnissen in verschiedenen Sprachen definiert werden können. Der zweite Teil der Klasse (Prüfung) besteht aus drei Methoden: *preCheck()* und *check()* der Klassen und *postCheck()* direkt aus dem Rule-Interface. Mithilfe der *preCheck()*-Methode kann zu Beginn der Prüfung frühzeitig entschieden werden, ob die weitere Prüfung mit dem gegeben Input aus dem Userinterface möglich bzw. sinnvoll ist. Aber auch andere Bedingungen, wie zum Beispiel die Modellinhalte oder ähnliches, können bereits in dieser Methode überprüft werden. Als Rückgabewert der Methode können zwei Ergebnistypen definiert werden:

- *preCheckResult.createRelevant()*: Wird im *preCheck()* die weitere Prüfung für sinnvoll und durchführbar erachtet, erklärt dieser Typ die Überprüfung als relevant.
- *preCheckResult.createIrrelevant()*: Sollten im *preCheck()* vordefinierte Bedingungen nicht eingehalten werden, wird die weitere Überprüfung als irrelevant erklärt und die Prüfung an dieser Stelle abgebrochen.

Unabhängig vom Ergebnistyp kann eine String-Nachricht zur Erklärung weitergeben werden, um so den Nutzer auf mögliche Eingabefehler hinzuweisen. Eine weitere wichtige Eigenschaft der Methode ist, dass sie immer nur einmal ausgeführt wird. Sollte die *preCheck()*-Methode der erweiterten Klasse nicht überschrieben werden oder liefert diese den Ergebnistyp relevant, folgt darauf die *check()*-Methode. Hier unterscheiden sich die OneByOneRule und die ConcurrentRule. Wie der Name OneByOneRule schon vermuten lässt, wird die *check()*-Methode für jedes übergebene Element des DefaultFilterParameters durchgeführt. Im Gegensatz dazu wird bei der ConcurrentRule für jedes Element ein eigener Task gestartet, die je nach Rechenleistung und Anzahl der Elemente parallel bzw. gleichzeitig (concurrent) ausgeführt werden. Bei Wahl der ConcurrentRule kann sich aufgrund der gleichzeitigen Bearbeitung die Rechenzeit verringern. Dabei gilt: Die Überprüfung darf sich stets nur auf ein Element beziehen. Werden für die Überprüfung allgemeine Informationen benötigt und die einzelnen Tasks greifen gleichzeitig darauf zu,

kann das zum Abbruch oder sogar zum Absturz der API-Prüfregel führen. Für beide Klassen gilt, dass nur für Elemente aus dem DefaultFilterParameter ein Ergebnis erstellt werden kann. Diese Ergebnis wird an eine Sammlung von Ergebnissen innerhalb der *check()*-Methode übergeben. Da die programmierten API-Prüfregeln dieser Arbeit stets auf mehrere Merkmale verschiedenster Elemente zugreifen, wird die Klasse OneByOneRule gewählt. Zum Schluss der Überprüfung kann die *postCheck()*-Methode überschrieben werden. Wie bei der *preCheck()*-Methode wird der *postCheck()* nur einmal ausgeführt und immer erst dann, wenn die *check()*-Methode vollständig abgeschlossen ist. Diese Methode eignet sich, um Ergebnisse in externen Dateiformaten zu speichern oder erzeugte Klassenvariablen zurückzusetzen. Der letzte Teil der Klasse bezieht sich auf die Anordnung der UserInterface-Elemente der Klassenvariablen. Das Rule-Interface definiert prinzipiell einen Standardaufbau, weshalb die Methode *getParametersUIDefnition()* überschrieben werden muss. Die Struktur der OneByOneRule ist im Code 2.1 dargestellt:

```
1  public final class ExampleRule extends OneByOneRule {
2  // Definition RuleParameters
3  private final RuleParameters params = RuleParameters.of(this);
4  // FilterParameter
5  private final FilterParameter rpComponentFilter = this.getDefaultFilterParameter
     ();
6  // sonstige UI Elemente
7  private final StringParameter stringParameter = params.createString("
     MyStringParameter");
8  // um auf die Resources zugreifen zu koennen
9  private final RuleResources resources = RuleResources.of(this);
10
11 @Override
12 public PreCheckResult preCheck() {
13   return PreCheckResult.createRelevant();
14 }
15 @Override
16 public Collection<Result> check(Component component, ResultFactory resultFactory)
     {
17   String s = stringParameter.getValue();
18   Result r = resultFactory.create(s, "Description");
19   return Collections.singleton(r);
20 }
21 @Override
22 public void postCheck() {
23   // Ergebnisse Extern Speichern.
24   super.postCheck();
25 }
26 public UIContainer getParametersUIDefinition() {
27   UIContainer uiContainer = UIContainerVertical.create();
28   return uiContainer;
29 }
30 }
```

Code 2.1: *generateUsage()*-Methode

Der Aufbau und Ablauf einer API-Prüfregel sind in Abb. 2.17 grafisch dargestellt. Diese zeigt, dass nach dem Start der Regel die *preCheck()*-Methode ausgeführt wird. In Abhängigkeit davon, ob zuvor definierte Bedingungen nicht eingehalten oder Fehler gefunden werden, wird der Ergebnistyp Irrelevant oder Relevant zurückgeben. Beim Ergebnistyp Relevant folgt darauf die *check()*-Methode, die sich für OneByOneRule und ConcurrentRule darin unterscheidet, ob der Check nacheinander für jedes Element des DefaultFilterParameters ausgeführt wird oder ob dieser gleichzeitig stattfindet. Sobald die API-Prüfregel alle Elemente behandelt und alle Ergebnisse erstellt hat, wird die *postCheck()*- Methode ausgeführt. Dabei hat jede der drei Methoden der Prüfung Zugriff auf den Nutzer-Input.

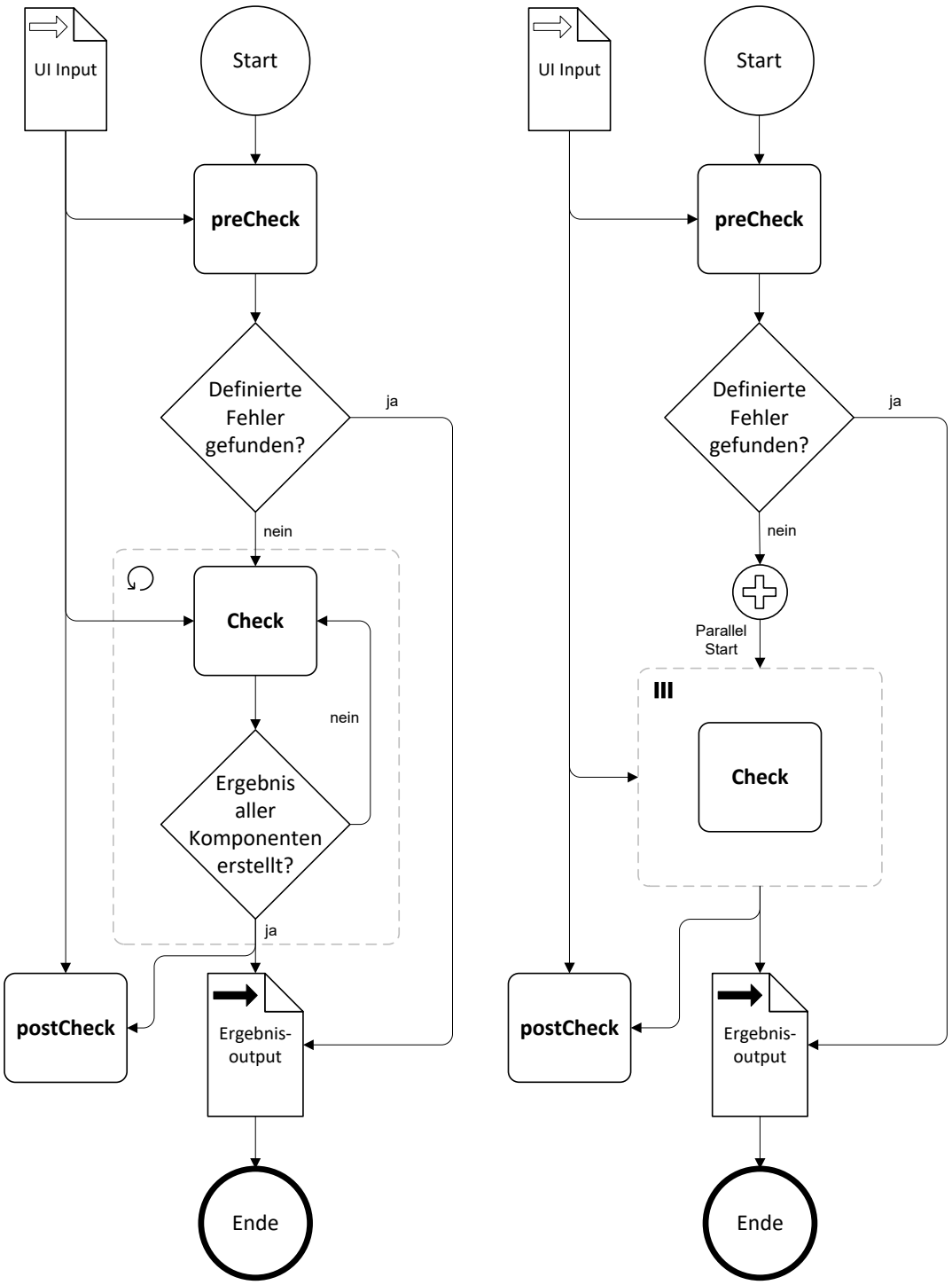


Abb. 2.17: Mögliche Abläufe einer API-Prüfregel - OneByOneRule (li.) und ConcurrentRule (re.)

2.6.3 Ausgewählte Elemente des Userinterfaces

Die Solibri-API [42] definiert eine Vielzahl von verschiedenen Möglichkeiten, das UserInterface an die spezifischen Anforderungen anpassen zu können. In den nachfolgenden Punkten werden jene Elemente genannt, die essenziell für diese Arbeit sind:

Container: Container stellen die Grundstruktur des UserInterfaces dar. Dabei gibt es einen Container, der darin enthaltene Objekte vertikal und einen, der die Objekte horizontal anordnet. Weitere Container können miteinander verschachtelt werden, um so ein komplexes UserInterface zu erstellen.

RuleParameter: Die Berücksichtigung von Nutzereingaben in Solibri erfolgt über RuleParameter. Namen und Beschreibungen können RuleParametern mithilfe der Property-Datei hinzugefügt werden.

- **FilterParameter:** ermöglicht die Filterung von Elementen des IFC-Modells. Als Rückgabe erhält die API-Prüfregel stets nur das Ergebnis der Filterung. Einzelne Eingaben in den Filter können nicht ausgelesen werden.
- **DoubleParameter:** für die Eingabe von Zahlenwerten
- **StringParameter:** zur Eingabe von Text
- **BooleanParameter:** für Logikwertabfragen
- **PropertyReferenceParameter:** Properties aus dem IFC-Modell wobei die Auswahl durch die vorhandenen Properties im aktuellen Modell eingeschränkt ist
- **TableParameter:** komplexer Parameter der eine Tabelle für die Eingabe erzeugt. Mögliche Eingabewerte sind Zahlen und Text

2.6.4 Relevante Elemente der Solibri-API

Zur effizienten Verarbeitung von Informationen eines Modells bietet die Solibri-API diverse Klassen und Methoden. Die nachfolgende Auflistung stellt einen Ausschnitt der wichtigsten Begriffe im Kontext dieser Arbeit dar. Für eine vollständige Auflistung aller Inhalte der Solibri-API steht die Onlinedokumentation zur Verfügung [42].

Geometrie: Zur Erfassung von geometrischen Objekten zur Verarbeitung ihrer Dimensionen finden unter anderen Vektoren, Segmente, Polygone und Flächen Anwendung. Vektoren sind das kleinste Element zur Abbildung von Geometrie. Diese können sowohl zweidimensional als auch dreidimensional erstellt werden und benötigen die Angaben der gewünschten Koordinaten im Bezug auf den Koordinatenursprung. Um eine Linie im Raum zu erstellen, benötigt der Anwender die nächste Stufe: ein Segment. Dieses besteht aus einem Start- und einem Endvektor. Der Zusammenschluss mehrerer Segmente zu einem geschlossenen Objekt ergibt ein Polygon. Dieses kann ebenfalls eben oder beliebig im Raum liegen. Eine Fläche ist hingegen fest als zweidimensionales Objekt in der XY-Ebene definiert. Um von einem beliebigen Element auf die gegebenen Attribute zu schließen, steht die Klasse Footprint zur Verfügung, welche mit der Methode *.getFootprint()* instanziert werden kann. Wie der Name vermuten lässt, erstellt die Methode eine zweidimensionale Projektion des Objekts in der XY-Ebene. Mit diesem Objekt kann die Fläche und das umschließende Polygon bestimmt werden.

Visualisierung: Mithilfe der Visualisierungsoption können Elemente, Segmente, Flächen, Winkel und Ähnliches bei einem Prüfergebnis als zusätzliche Information farblich hervorgeho-

ben werden. Durch die graphische Darstellung steht eine Vielzahl von Möglichkeiten zur Verfügung, Ergebnisse besser zu kommunizieren.

Relation: Gemäß der IFC-Datenstruktur bestehen zwischen verschiedenen Elementen des Modells gewisse Beziehungen (Relations). Diese können mithilfe der Solibri-API abgerufen werden. Dafür ist die Angabe des Ausgangselements, die Beziehungen und die Bestimmungsrichtung notwendig. Typische Beziehungen sind etwa Nearest_Spaces, welche die nächstgelegenen Räume eines Elements bestimmt, sowie Fills, welches von einem Objekt alle Öffnungen zeigt. Die Bestimmungsrichtung gibt dabei vor, von welchem Element ausgegangen wird. Sollten beispielsweise die Öffnungen in einer Wand gesucht werden, kann die Beziehung Voids mit der Richtung Backward verwendet werden. Wird hingegen zu einer Öffnung die zugehörige Wand gesucht, kann ebenfalls Voids jedoch jetzt mit der Richtung Forward verwendet werden.

Ergebniskategorien: Die Solibri-API ermöglicht zur übersichtlicheren Darstellung der Prüfergebnisse die Gruppierung in Ergebniskategorien. Diese können beliebig erstellt werden und beinhalten stets zwei Angaben: Erstens den Anzeigename und zweitens die Beschreibung der Kategorie.

Kapitel 3

Entwicklung von Prüfregeln mithilfe von BIM-Methoden

Aufbauend auf dem Grundlagenkapitel beschreibt dieses die tatsächliche Umsetzung beziehungsweise Programmierung von zwei Solibri-API-Prüfregeln zur Validierung und Generierung von Brandabschnitten mithilfe von BIM-Methoden. Die Validierung überprüft von Planenden festgelegte Brandabschnitte auf die Einhaltung von zuvor definierten Bestimmungen (z. B. relevante Rechtsmaterie gemäß Abb. 2.7). Bei der Generierung werden auf Basis von gegebenen Merkmalen in den Bauteilen Brandabschnittskonfigurationen erzeugt, welche Planende bei der Festlegung von Brandabschnitten unterstützen sollen. Neben den verfolgten Zielen wird auf die Anforderungen bzw. deren Vereinfachungen eingegangen. Anschließend wird die Methodik der Entwicklung aufgezeigt. Darauf aufbauend folgt die Beschreibung der einzelnen API-Prüfregeln, wobei konkret auf den jeweiligen Aufbau und Ablauf eingegangen wird.

3.1 Ziele

Durch die Entwicklung werden zwei wesentliche Ziele verfolgt. Das erste Ziel ist die Entlastung der Baupolizei der Stadt Wien (MA37). Die Überprüfung von Brandabschnitten stellt laut MA37 eine komplizierte und aufwendige Prüfung im aktuellen Verfahren dar. Zur Vereinfachung und gleichzeitig zur Beschleunigung dieses Prozesses soll durch die Automatisierung der Großteil geprüft und so dem Referenten mehr Zeit gegeben werden, wesentliche Details kontrollieren zu können. Dabei sind die einfache Handhabung sowie eine möglichst individuelle Angabe von Anforderungen von großer Bedeutung. Nur so können die Referenten selbst Änderungen in der relevanten Rechtsmaterie in die Prüfung übernehmen. Ein weiterer Punkt ist die Zuverlässigkeit und Überprüfbarkeit der Ergebnisse. Vor allem in der Pilotphase muss sofort ersichtlich sein, weshalb die Prüfung ein Ergebnis zurückgibt, um Fehler oder falsche Annahmen zu erkennen. Das zweite Ziel betrifft die Planenden des BIM-Modells. Damit ein hoher Automatisierungsgrad der Prüfung erreicht wird, sind genaue Angaben zu den einzelnen Elementen des BIM-Modells erforderlich. Zur Kompensation des Zusatzaufwands für die Planenden soll mithilfe der API-Prüfregel zur Generierung von Brandabschnitten die Festlegung neuer Brandabschnitte vereinfacht und der Gesamtprozess beschleunigt werden.

3.2 Abweichungen und Vereinfachungen

Die relevante Rechtsmaterie bzw. die umgesetzten Paragrafen können Abschnitt 2.5 entnommen werden. Im Zuge der Entwicklung wurden gemeinsam mit den Referenten Abgrenzungen und Klarstellungen in der Rechtsmaterie getroffen.

Bestimmung der Längsausdehnung

In den Bestimmungen für einen Brandabschnitt definiert die OIB-Richtlinie 2 [31], Absatz 3.1.1 eine nutzungsabhängige, maximale Längsausdehnung. Allerdings ist nicht eindeutig definiert, wie diese zu messen ist bzw. von welchen Parametern sie abhängt. Während des Projekts BRISE-Vienna wurde in Rücksprache mit den Referenten der Baupolizei für die automatisierte Prüfung festgelegt, dass die Längsausdehnung auf der sicheren Seite liegen muss. So können Brandabschnitte, die die Prüfung bestehen, sicher als richtig gewertet werden. Schließt die API-Prüfregel einen Abschnitt aufgrund der Längsausdehnung aus, soll in einem zweiten Schritt der Referent manuell die Längsausdehnung prüfen und anschließend eine Entscheidung treffen. Zur Einhaltung dieser Bedingungen ergeben sich zwei zusätzliche Anforderungen:

- Die Längsausdehnung eines Brandabschnitts wird durch die zwei am weitesten entferten Punkte des Umschließungspolygons bestimmt. Diese Bedingung ist in Abb. 3.1 für einen rechteckigen Raum, einen Raum in L-Form und einen beliebigen Raum dargestellt.

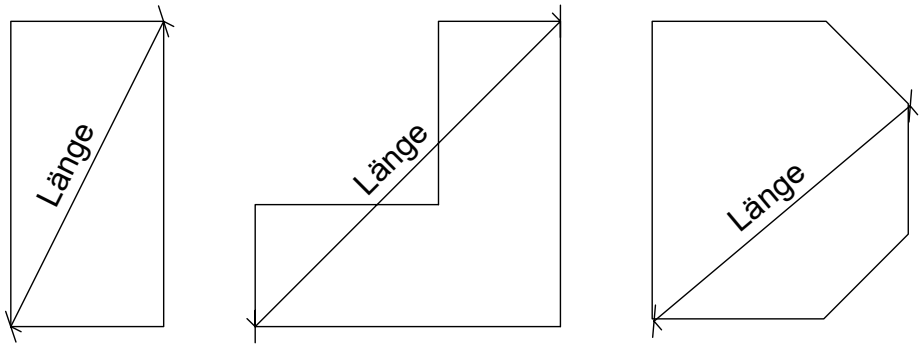


Abb. 3.1: Bestimmung der Längsausdehnung für verschiedene Brandabschnitte

- Bei Überschreitung der maximal zulässigen Längsausdehnung müssen die zur Bestimmung herangezogenen Punkte dem Referenten in einer eigenen Ergebnisausgabe ersichtlich gemacht werden. So kann dieser rasch eine Nachprüfung und Abänderung des Ergebnisses durchführen.

Unterscheidung unter- und oberirdischer Geschoße

Gemäß OIB-Richtlinie 2 [31] ergeben sich Anforderungen für Brandabschnitte in oberirdischen und unterirdischen Geschoßen. Es ist allerdings nicht klar ersichtlich, ob sich ein Brandabschnitt über unter- und oberidische Geschoße erstrecken darf. Zur Abdeckung dieses Falles wird in dieser Arbeit nicht mehr von unter-/oberirdischen Geschoßen sondern von Brandabschnitten gesprochen. Ein Brandabschnitt ist unterirdisch, wenn sich zumindest ein Raum mit einem unterirdischen Volumenelement verschneidet. Ein Beispiel für ein unterirdisches Volumenelement stellt der Körper zur Bestimmung der BGF-unterirdisch dar, welcher im Projekt BRISE-Vienna explizit gefordert wird.

Elemente der Gebäudehülle

Gemäß der Definition von brandabschnittsbildenden Elementen liegen diese zwischen zwei Brandabschnitten. Damit müssen Elemente der Gebäudehülle nicht zwingendermaßen brandabschnittsbildend sein. Zur Vereinfachung des komplexen Userinterfaces wird auf eine weitere Eingabemöglichkeit verzichtet und stattdessen die strengeren Anforderungen der brandabschnittsbildenden Elemente auf jene der Gebäudehülle übertragen.

Bauteilklasse und Baustoffklassen

Die relevante Rechtsmaterie stellt sowohl Anforderungen an die Bauteilklasse als auch an Baustoffklassen. Beide API-Prüfregeln konzentrieren sich darauf, die Bauteilklasse richtig zu bestimmen, wobei zum Zeitpunkt der Fertigstellung dieser Arbeit nur die Klassen R, E, I und die Klassifizierungszeit geprüft werden.

3.3 Methodik der Implementierung

Die Methodik der Implementierung bezieht sich auf die Implementierungsphase des fünfstufigen Bearbeitungsschemas für die Erstellung einer Prüfroutine. Das bedeutet, dass die Analyse der Rechtsmaterie und die Konzeptionsphase bereits im Projekt BRISE-Vienna durch die teilnehmenden Projektpartner durchgeführt wurde und die Implementierung auf den Ergebnissen aufbaut. Die Implementierung lässt sich in drei Schritten gliedern, wobei der gesamte Prozess iterativ ist:

1. Untersuchung der Problemstellung – Auswertung der Konzeptionsphase
   In der Konzeptionsphase wurden bereits Anforderungen an einzelne Prüfroutinen spezifiziert, welche in diesem Schritt kontrolliert werden. Nach erfolgter Kontrolle können erste Überlegungen zur Umsetzung erfolgen. Dazu sollen folgende Fragestellungen geklärt werden:
   - Auf welche Komponten des Modells bezieht sich die Prüfung und welche Eigenschaften sind direkt abrufbar?
   - Welche Eingabewerte sind notwendig, um die geforderten Anforderungen der Rechtsmaterie zu prüfen?
   - Definiert die Rechtsmaterie Sonderfälle, die andere Herangehensweisen benötigen?
   - Können Benutzereingaben zum Absturz führen und können diese durch eine Vorprüfung in der *preCheck()*-Methode abgefangen werden?

   Sind diese Fragestellungen geklärt wird, ein Konzept für den Ablauf der API-Prüfregel entwickelt. Nach Fertigstellung folgt der zweite Schritt.

2. Erweiterung auf weitere Paragrafen
   Der hohe Zeitaufwand für die Entwicklung neuer API-Prüfregeln erfordert stets die Prüfung, ob die Abdeckung anderer Paragraphen mit der selben Regel möglich ist. Auf Basis des Ablaufkonzepts sollen noch offene Paragraphen auf Kompatibilität untersucht werden. Falls durch geringen Mehraufwand weitere Punkte abgedeckt werden können, ist dies in der API-Prüfregel zu berücksichtigen. Gegebenfalls muss dadurch der erste Schritt wiederholt werden. Ergeben sich keine zusätzlichen Fragestellungen, folgt der dritte Schritt.

3. Umsetzung
   Sobald alle Fragestellungen rund um die Problemstellung geklärt sind, beginnt die Entwicklung. Dazu empfiehlt es sich, zuerst das Userinterface anzulegen, damit bereits alle möglichen Eingabewerte in der weiteren Bearbeitung zur Verfügung stehen. Danach kann Schritt für Schritt das überlegte Konzept in einzelne Methoden aufgeteilt und umgesetzt werden. Während der Entwicklung können weitere Fragestellungen auftauchen, die ein erneutes Durchlaufen der drei Schritte erfordert.

3.4 API-Prüfregel Validierung

Mithilfe der API-Prüfregel Validierung sollen Brandabschnitte auf Einhaltung der geforderten Kennwerte gemäß der relevanten Rechtsmaterie im Projekt BRISE-Vienna geprüft werden. Im Modell muss der Planende dabei zusammengehörige Räume, welche einen Brandabschnitt bilden, durch ein Merkmal angeben. Der Nutzer der API-Prüfregel muss schließlich Anforderungen an Wände, Decken, Stützen, Balken und Beläge sowie Türen und Fenster definieren. Weiters fordert die Überprüfung die Angabe der maximal erlaubten Fläche, Längserstreckung und der Geschoßzahl. Zum Abschluss muss der Nutzer noch die zu unterscheidenden Nutzungskategorien angeben. Nach erfolgter Überprüfung werden die Brandabschnitte in verschiedene Ergebniskategorien eingeteilt.

Nachfolgende Auflistung gibt einen Überblick über die erforderlichen Funktionen gemäß Schritt 1 und 2 zur Bearbeitung der relevanten Rechtsmaterien in Wien. Gemäß dem Grundsatz, die API-Regeln möglichst viele Problemstellungen zu behandeln, werden ebenfalls die zusätzlichen Funktionen aufgelistet.

- Notwendige Funktionen
  - Auswahl der Komponenten (Räume)
  - Zusammenführen von Räumen zu einem Brandabschnitt über ein Space-Property
  - Nutzungskategorieabhängige (z. B. Wohnen und Aufenthalt) Überprüfung der Brandabschnitte
  - Nutzungskategorienaufteilung ab einem gewissen Schwellenwert und anschließende Überprüfung, ob diese gemischt in einem Brandabschnitt auftreten
  - automatische Erkennung, ob es sich bei dem aktuellen Brandabschnitt um einen ober- oder unterirdischen handelt
  - Überprüfung unterschiedlicher Anforderungen an ober- und unterirdische Brandabschnitte
  - Angabe und Überprüfung zusätzlicher Bedingungen für Gebäude mit einem Fluchtniveau über 22,0 m
  - Überprüfung der maximalen
    - Größe des Brandabschnitts
    - Längsausdehnung des Brandabschnitts
    - Geschossanzahl des Brandabschnitts
  - Prüfung der brandrechtlichen Anforderungen von
    - tragenden und nichttragenden Wänden
    - tragenden und nichttragenden Decken
    - Stützen
    - Balken
    - Belägen
    - Türen und Fenster
- Zusätzliche Funktionen
  - Ausschluss von nicht betrachteten Räumen ist möglich.

– Es können die gleichen Anforderungen an unterirdische Brandabschnitte gestellt werden.
– Berücksichtigung der Nutzungskategorienaufteilung ist optional.
– Die Flächenberechnungsmethode kann individuell angepasst werden.
– Der Export der Ergebnisse ist auch in eine Excel-Datei möglich.

3.4.1 Userinterface

Das Userinterface ist in Abb. 3.2 bis Abb. 3.8 dargestellt und wird nachfolgend erklärt. Es bietet die Möglichkeit, die Parameter einzugeben.

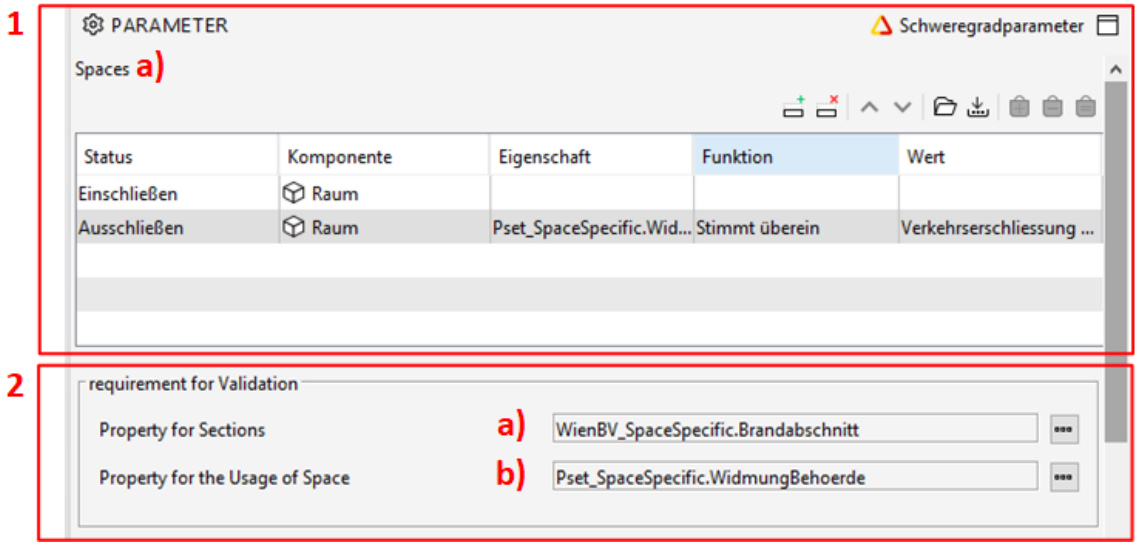


Abb. 3.2: Fire Compartment Validation Rule – Userinterface – Teil 1/7

1. **Spaces:** Diese Eingabemöglichkeit stellt den DefaultFilterParamerter der API-Prüfregel dar.
   a) In diesem Komponentenfilter sollen nur Räume hinzugefügt werden. Gemäß den zuvor definierten Rahmenbedingungen werden Verkehrsflächen (Garagen, Stellplätze) nicht geprüft. Darum erfolgt der Ausschluss an dieser Stelle.
2. **requirement for Validation:** Dies umfasst zwei wichtige PropertyReferenceParameter, die Merkmale von Räumen abfragen.
   a) Hier muss der Nutzer jenes Merkmal angeben, mit der die einzelnen Brandabschnitte unterschieden werden können. Im Projekt BRISE-Vienna entspricht dies dem Merkmal Brandabschnitt „a“ aus dem PropertySet WienBV_SpaceSpecific. Der Wert des Merkmals kann individuell eingegeben werden, wobei Räume im gleichen Brandabschnitt den selben Merkmalwert aufweisen müssen.
   b) Zur Erfüllung der Anforderungen gemäß OIB-Richtlinie soll der Nutzer hier das Merkmal angeben, welches die Widmung bzw. die Nutzungskategorie eines Raums definiert. Für BRISE-Vienna handelt es sich dabei um das Merkmal WidmungBehoerde aus dem PropertySet Pset_SpaceSpecific.

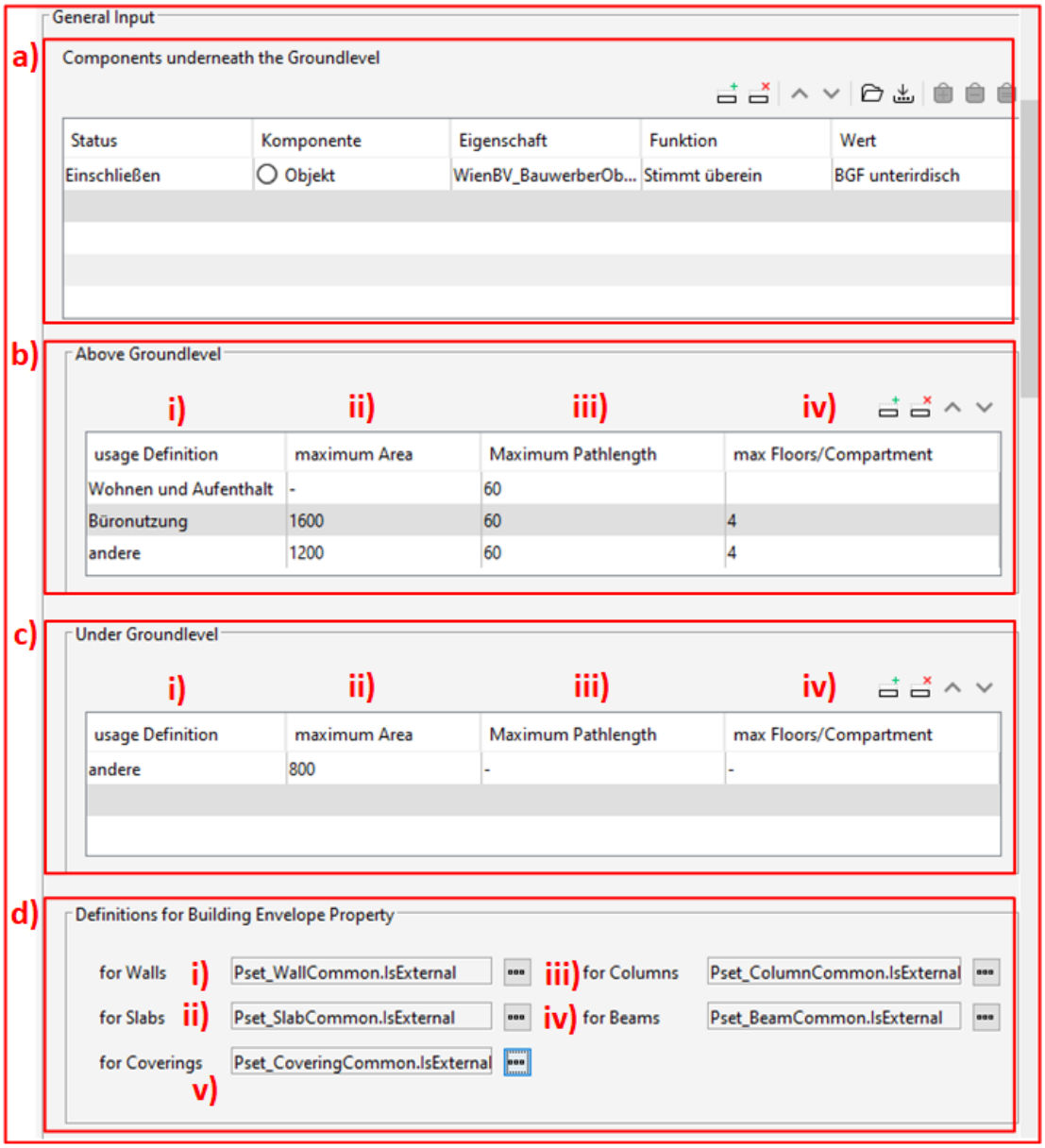


Abb. 3.3: Fire Compartment Validation Rule – Userinterface – Teil 2/7

3. **General Input:** Dieser beinhaltet alle Angaben, die für jeden einzelen Brandabschnitt gesamt zu erfüllen sind. Gemäß den Projektanforderungen erfolgt dabei eine Unterteilung in oberirdische und unterirdische Brandabschnitte. An dieser Stelle sei vermerkt, dass im Rahmen dieser Arbeit ein unterirdischer Brandabschnitt vorliegt, sobald ein Raum unterirdisch ist.

   a) In diesem Filter müssen Objekte ausgewählt werden, die eine eindeutige Zuordnung der Räume zu unterirdisch und oberirdisch ermöglichen. Im Rahmen des Projekts BRISE-Vienna soll das Volumenelement BGF-unterirdisch gewählt werden. Mit diesem Objekt bestimmt die Regel, ob ein Brandabschnitt unterirdisch oder oberirdisch liegt.

   b) Oberirdische Brandabschnitte: Für die korrekte Definition der Ausmaße eines oberirdischen Brandabschnitts bedarf es der folgenden vier Angaben je Nutzungskategorie,

weshalb für die Eingabe ein Tableparameter gewählt wurde. Werden bei bestimmten Kategorien keine Anforderungen gestellt, kann der Nutzer das in der Eingabe durch „-“ berücksichtigen.

i. Angabe der Nutzungskategorie
ii. Angabe der maximalen Fläche je Nutzungskategorie in m²
iii. Angabe der maximalen Längsausdehnung je Nutzungskategorie in m
iv. Angabe der maximalen Geschossanzahl je Nutzungskategorie

c) Unterirdische Brandabschnitte: Für unterirdische Brandabschnitte gilt lediglich eine Beschränkung hinsichtlich der Fläche. Allerdings wird die Eingabe für zukünftige Anpassungen bereits erweitert. Auch hier kann durch die Eingabe von „-“ keine Anforderung berücksichtigt werden.

i. Angabe der Nutzungskategorie
ii. Angabe der maximalen Fläche je Nutzungskategorie in m²
iii. Angabe der maximalen Längsausdehnung je Nutzungskategorie in m
iv. Angabe der maximalen Geschossanzahl je Nutzungskategorie

d) Definitionen der Gebäudehülle: Aufgrund dessen, wie der Algorithmus die brandabschnittsbildenen Elemente registriert, ist die Definition der Gebäudehülle notwendig um auch nach außen richtig abzuschließen. Dafür benötigt es die Angabe der Properties für die Gebäudehülle für Wände, Decken, Stützen, Balken und Beläge.

i. Eigenschaft für Wände, die die Gebäudehülle bilden
ii. Eigenschaft für Decken, die die Gebäudehülle bilden
iii. Eigenschaft für Stützen, die die Gebäudehülle bilden
iv. Eigenschaft für Balken, die die Gebäudehülle bilden
v. Eigenschaft für Beläge, die die Gebäudehülle bilden

e) Optionale Einstellungen zu Nutzungskategorien: Werden diese berücksichtigt, erfolgt vor der Brandabschnittsprüfung eine Überprüfung der Flächen der eingetragenen Nutzungen. Bei Überschreitung dürfen in einem Brandabschnitt nicht mehrere Nutzungen vorhanden sein. Sollte bei Unterschreitung in einem Brandabschnitt mehrere Nutzungskategorien auftreten, werden stets die strengeren Anforderungen herangezogen.

i. Booleanparameter: entscheidet, ob die Überprüfung der Nutzungskategorien angewendet werden soll.
ii. TableParameter: Bezeichnung der Nutzungskategorie – sofern die optionale Prüfung gewünscht ist, muss eine Bestimmung für „andere“ oder „alle“ Nutzungskategorien angeben sein, sonst wird eine Fehlermeldung zurückgegeben.
iii. TableParameter: maximale Fläche der einzelnen Nutzungskategorie in m²

f) Optionale Einstellungen zu zusätzlichen Anforderungen: Werden diese berücksichtigt, gelten ab den angegeben Grenzstockwerken neue Anforderungen bezüglich der maximalen Fläche. Liegt zumindest ein Raum in oder oberhalb der Grenzstockwerke werden die neuen Bedigungen gültig.

i. Booleanparameter: entscheidet, ob die zusätzlichen Anforderungen für hohe Gebäude berücksichtigt werden

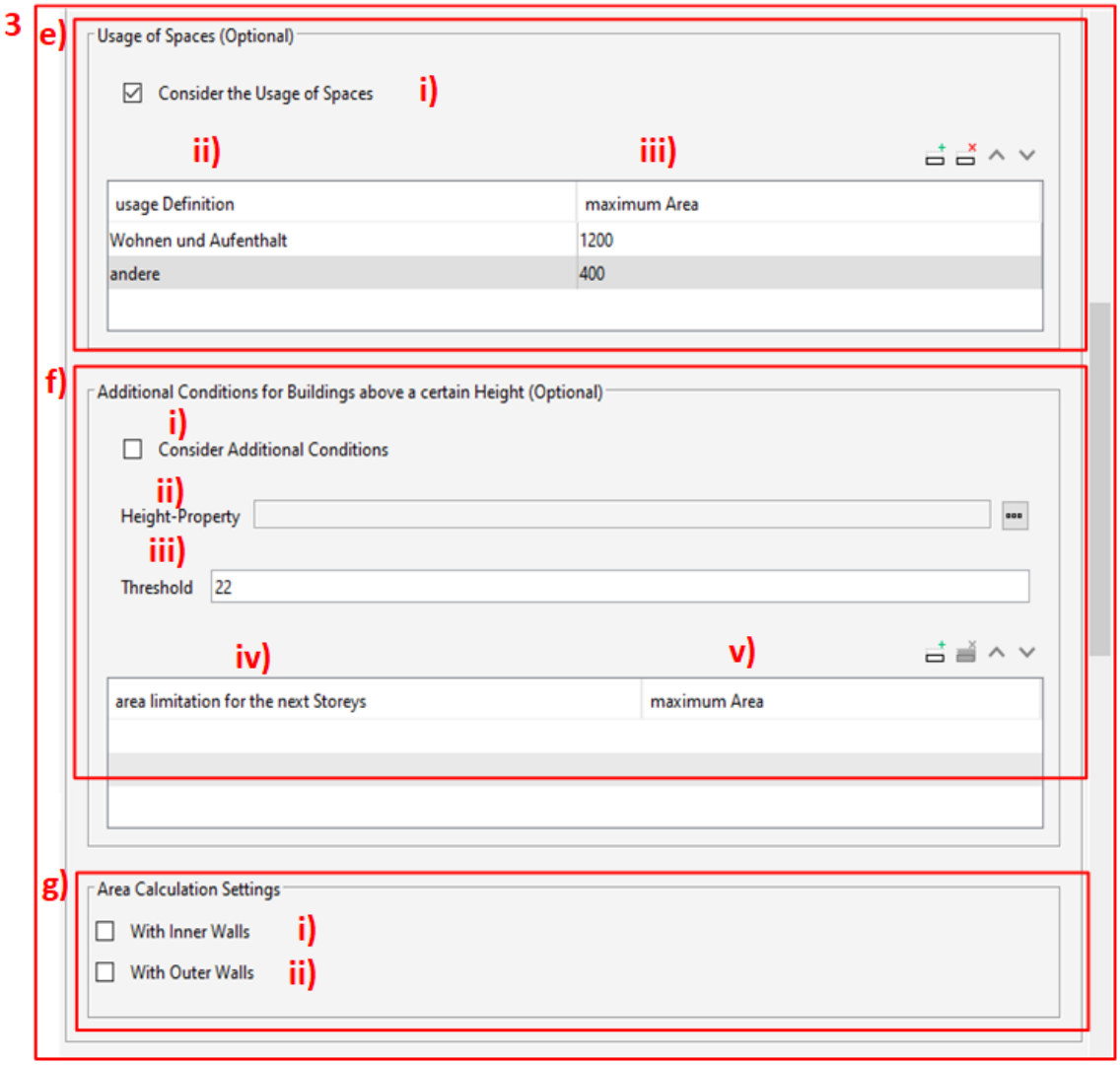


Abb. 3.4: Fire Compartment Validation Rule – Userinterface – Teil 3/7

ii. Angabe der Höhen-Eigenschaft – gemäß OIB-Richtlinie ist die Bezugshöhe das Fluchtniveau.

iii. Vergleichswert für die Höheneigenschaft, bei Überschreitung werden die zusätzlichen Bestimmungen aktiv. Die Angabe erfolgt in m.

iv. TableParameter: Grenzstockwerk, ab welchem die zusätzlichen Anforderungen gelten.

v. TableParameter: maximale Fläche für Brandabschnitte die auf oder oberhalb des Grenzstockwerkes liegen in m²

g) Einstellungen zur Berechnung der Fläche eines Brandabschnitts: Für die Ermittlung der Fläche eines Brandabschnitts werden die Raumflächen herangezogen. Hier hat der Nutzer die Möglichkeit, zusätzlich die Flächen der brandabschnittsbildenden Wände und jene Wände, die innerhalb des Brandabschnitts liegen, heranzuziehen.

i. BooleanParameter: mit Innenwänden (Wände, innerhalb des Brandabschnitts)

ii. BooleanParameter: mit brandabschnittsbildenden Wänden

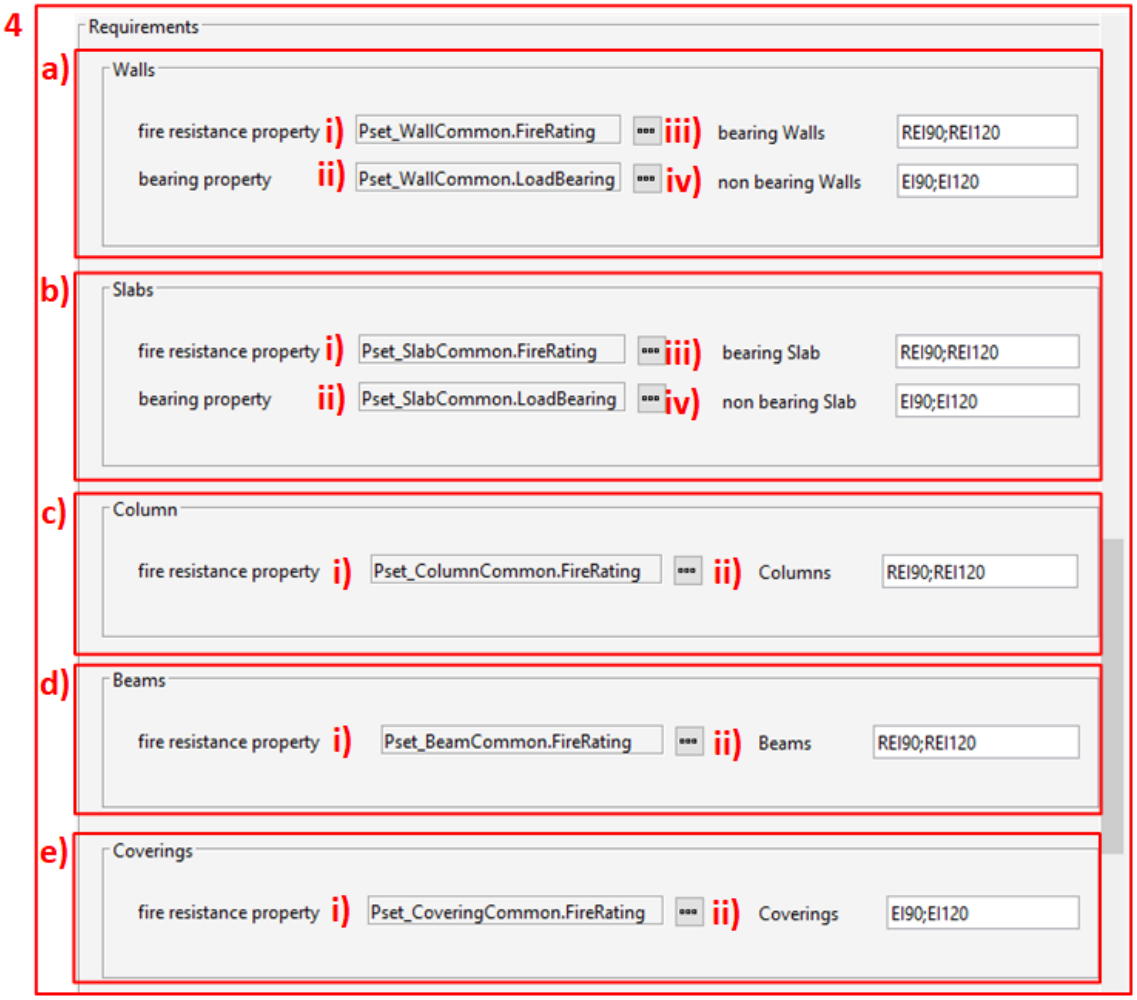


Abb. 3.5: Fire Compartment Validation Rule – Userinterface – Teil 4/7

4. **Requirements:** Dies beinhaltet alle Anforderungen, die an einzelne, brandabschnittsbildene Elemente gestellt werden. Dabei erfolgt eine Unterscheidung zwischen nichttragenden und tragenden für Wände und Decken. Im Sinne der Tragwerksplanung wird in dieser Arbeit davon ausgegangen, dass es keine nichttragenden Balken und Stützen gibt, weshalb die Unterscheidung entfällt. Für Beläge wird im Gegenzug die Annahme getroffen, dass diese immer nichttragend sind. Aufgrund der Anforderungen der maßgebenden Rechtsmaterie erfolgt eine weitere Unterteilung für Abdeckungen von Öffnungen (geringe Anforderungen und strenge Anforderungen):
   a) Anforderung an Wände
      i. Angabe der Brandwiderstandseigenschaft von Wänden
      ii. Eigenschaft, ob eine Wand tragend oder nichttragend ist
      iii. zugelassene Vergleichswerte für tragende Wände – Vergleichswerte werden durch ";"getrennt – Beispiel: REI 90
      iv. zugelassene Vergleichswerte für nicht tragende Wände – Vergleichswerte werden durch ";"getrennt – Beispiel: EI 90
   b) Anforderung an Decken

i. Angabe der Brandwiderstandseigenschaft von Decken

ii. Eigenschaft, ob eine Decke tragend oder nichttragend ist

iii. zugelassene Vergleichswerte für tragende Decken – Vergleichswerte werden durch ";"getrennt – Beispiel: REI 90

iv. zugelassene Vergleichswerte für nicht tragende Decken – Vergleichswerte werden durch ";"getrennt – Beispiel: EI 90

c) Anforderung an Stützen

i. Angabe der Brandwiderstandseigenschaft von Stützen

ii. zugelassene Vergleichswerte für tragende Stützen – Vergleichswerte werden durch ";"getrennt – Beispiel: REI 90

d) Anforderung an Balken

i. Angabe der Brandwiderstandseigenschaft von Balken

ii. zugelassene Vergleichswerte für tragende Balken – Vergleichswerte werden durch ";"getrennt – Beispiel: REI 90

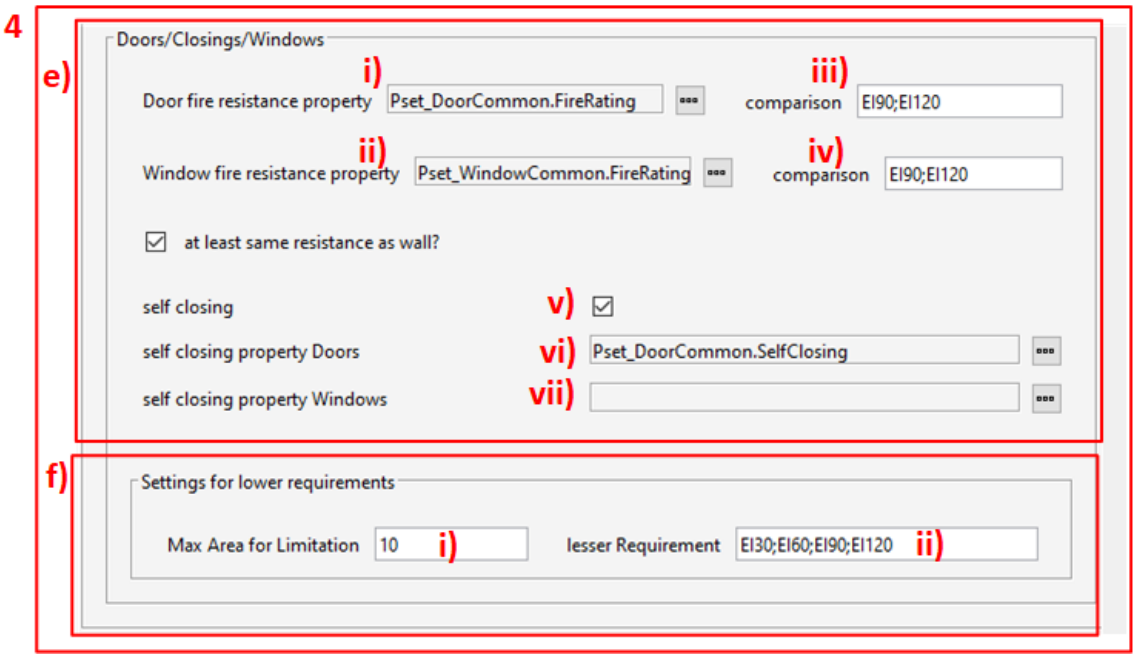


Abb. 3.6: Fire Compartment Validation Rule – Userinterface – Teil 5/7

e) Anforderung an Beläge

i. Angabe der Brandwiderstandseigenschaft von Belägen

ii. zugelassene Vergleichswerte für Beläge – Vergleichswerte werden durch ";"getrennt – Beispiel: REI 90

f) Anforderung an Türen/Abdeckungen/Fenster

i. Angabe der Brandwiderstandseigenschaft von Türen/Abdeckungen

ii. Angabe der Brandwiderstandseigenschaft von Fenster

iii. zugelassene Vergleichswerte für Türen/Abdeckungen – Vergleichswerte werden durch ";"getrennt

iv. zugelassene Vergleichswerte für Fenster - Vergleichswerte werden durch ";"getrennt

v. BooleanParameter: soll die Tür/Abdeckung mindestens die gleiche Eigenschaft wie die Wand/Decke haben?
Wenn diese Checkbox angewählt wird, müssen bei den Vergleichswerten für Wände, Decken und Türen gleich viele Werte eingetragen sein

vi. BooleanParameter: soll überprüft werden, ob Türen/Abdeckungen/Fenster selbstschließend sind

vii. Eigenschaft von Türen/Abdeckungen, ob diese selbstschließend sind

viii. Eigenschaft von Fenster, ob diese selbstschließend sind

g) Angaben für geringe Anforderungen an Türen/Abdeckungen/Fenster

i. Begrenzung der Summe der einzelnen Öffnungen je Wand und Decke für geringere Anforderungen

ii. zugelassene Vergleichswerte für niedrigere Anforderung – Vergleichswerte werden durch ";"getrennt – Beispiel: EI 90

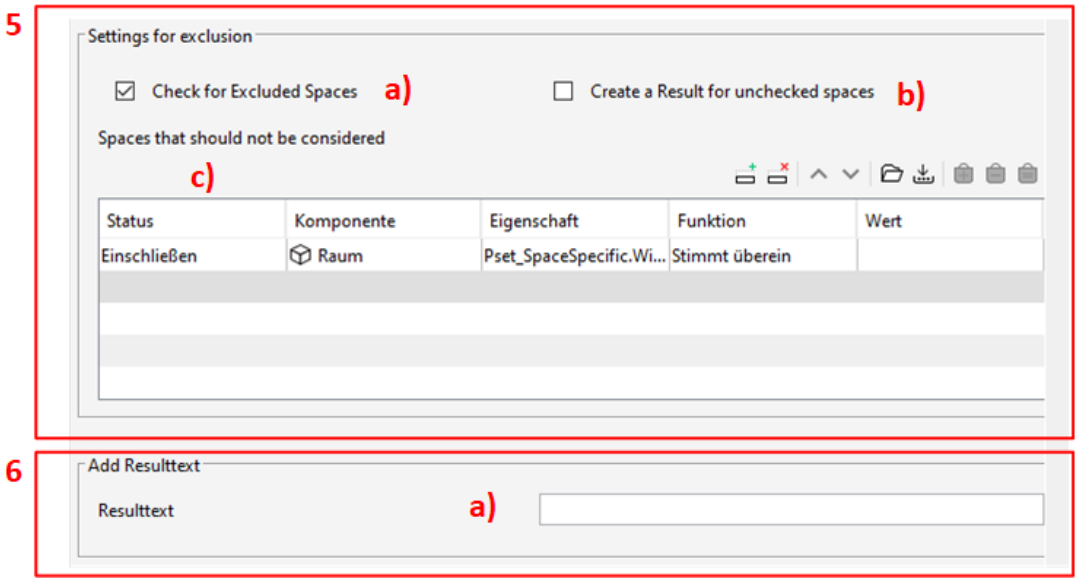


Abb. 3.7: Fire Compartment Validation Rule – Userinterface – Teil 6/7

5. **Settings for exclusion:** Hier können Räume und ganze Abschnitte aus der Überprüfung ausgeschlossen werden. Allerdings besteht ein Unterschied zum Ausschluss aus dem DefaultFilterParameter: Mithilfe dieses Filters können Brandabschnitte, die aus dem DefaultFilterParameter bestimmt werden auf diese Räume untersucht und bei Bedarf markiert werden.

a) BooleanParameter: Sollen die Brandabschnitte auf ausgeschlossene Räume überprüft werden?

b) BooleanParameter: Soll eine eigene Ergebniskategorie für ausgeschlossene Räume erstellt werden?

c) In diesem Filter definiert der Nutzer Räume, die nicht zu berücksichtigen sind. So kann die Identifizierung von fehlerhaft definierten Brandabschnitten erfolgen.

6. **Add Resulttext:** Mit diesem Eingabefeld wird dem Nutzer ermöglicht, die Ergebnisausgabe individuell zu gestalten.
   a) In diesem Feld kann zusätzlicher Text für Ergebnisse angegeben werden. Dieser wird immer oberhalb des voreingestellten Textes angezeigt.

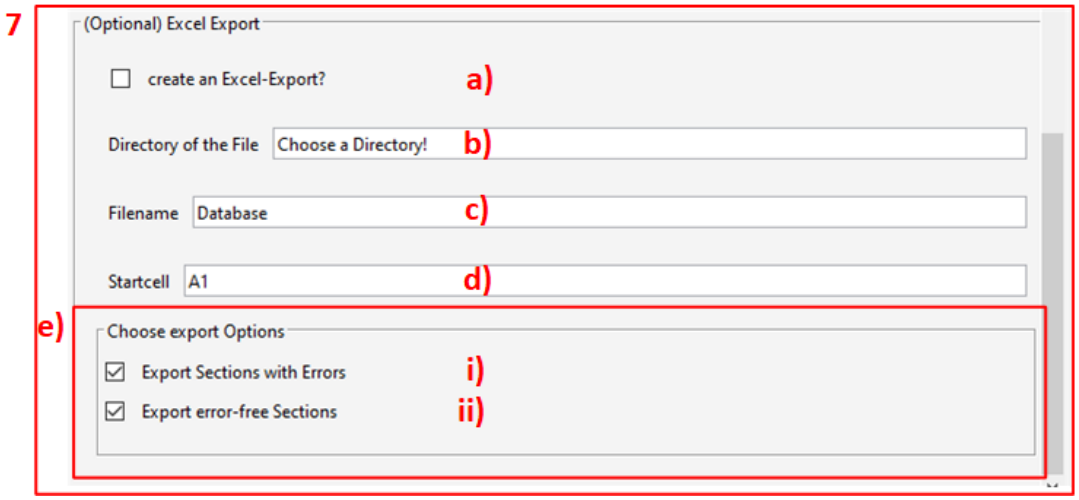


Abb. 3.8: Fire Compartment Validation Rule – Userinterface – Teil 7/7

7. **(Optional) Excel Export:** Das Projekt BRISE-Vienna hat gezeigt, dass das Abspeichern von Ergebnissen in externe Dateiformate viele Vorteile bringt. Basierend darauf wird auch für diese API-Prüfregel ein Excel Export entworfen.
   a) BooleanParameter: der Excel Export ist optional, weswegen der Nutzer hier bestätigen muss, ob dieser stattfinden soll.
   b) Hier muss der Dateipfad angegeben werden. Zur Erleichterung der Eingabe für den Nutzer wurde der Stringparameter so modifiziert, dass einfach nur der Dateipfad aus dem Windows-Explorer kopiert und eingefügt werden muss.
   c) In diesem Eingabefeld soll der Dateiname eingegeben werden, wobei der Standardname auf „Database“ voreingestellt ist.
   d) Mit der Startzelle kann der Nutzer entscheiden, ab welcher Stelle im gegebenen Tabellenblatt der Export stattfindet. So können bereits bestehende Listen ergänzt werden.
   e) In den Export-Optionen kann der Nutzer festlegen, welche Ergebnisse exportiert werden sollen. Dabei hat er die Wahl aus:
      i. fehlerhaften Brandabschnitten und/oder
      ii. fehlerfreien Brandabschnitten

3.4.2 Funktionsweise

Die Validierung setzt sich aus drei Java-Klassen zusammen:

- FCVR_Section
- FireCompartmentValidationRule
- FCVR_FurtherFunctions.

FCVR_Section

Dies ist eine eigens entwickelte Java-Klasse, die zur Abbildung von Brandabschnitten dient. Für jeden Brandabschnitt wird dabei ein neues Objekt der Klasse instanziiert. Es stehen Variablen für die Fläche, die Längsausdehnung und der Geschoßanzahl zur Verfügung. Des Weiteren werden die Nutzungskategorie sowie die Lage (ober- oder unterirdisch) gespeichert. Alle brandabschnittsbildenden Elemente sowie die Räume sind ebenfalls je Objekt hinterlegt. Außerdem werden Fehler je Komponente und die Nutzungskategorien sowie die zugehörigen Räume gesammelt. Um weitere notwendige Informationen zu erhalten bzw. die Variablen zu befüllen sind neben den Getter- und Setter- folgende Methoden definiert:

- *generateComponentsOfSection()*
  Diese Methode fügt in Abhängigkeit des Userinterfaces die Komponenten aus den anderen Variablen zu einer großen Sammlung zusammen. Aus dem Userinterface wird bestimmt, ob die Innenwände und brandabschnittsbildende Wände ebenfalls addiert werden sollen.

- *calculateAreaOfSection()*
  Diese Methode befüllt die Variable für die Fläche des Brandabschnitts. Sie bezieht sich auf das Ergebnis der *generateComponentsOfSection()*-Methode, um die Fläche zu bestimmen.

- *generateDescriptionMap()*
  Diese Methode erstellt die Variable zur Ergebnisbeschreibung eines Brandabschnitts. Die Map umfasst den Brandabschnittsnamen, die Fläche, die Anzahl der Räume, die Lage, die Längsausdehnung und die Fehler.

- *sectionDescriptionOutput()*
  Die Methode erzeugt aus der erstellten Variable der *generateDescriptionMap()*-Methode einen String. Dieser kann einfacher als Ergebnistext in Solibri angezeigt werden.

- *generateErrorMap()*
  Diese Methode erstellt die Variable zur Speicherung von Fehlern innerhalb eines Brandabschnitts.

FireCompartmentValidationRule und FCVR_FurtherFunctions

Die beiden Klassen umfassen alle Variablen und Methoden, um die Validierung durchzuführen. Die FireCompartmentValidationRule beinhaltet die *preCheck()*-, *check()*- und *postCheck()*-Methode und definiert das Userinterface. Die Klasse erbt von der OneByOne-Rule und kann damit als Hauptklasse im Sinne der Solibri-API betrachtet werden. Die FCVR_FurtherFunctions-Klasse hat den Zweck, die FireCompartmentValidationRule-Klasse verständlich und übersichtlich zu halten. Deswegen werden einige Funktionen in die FCVR_FurtherFunctions-Klasse ausgelagert.

Die FireCompartmentValidationRule stellt einen komplexen Prüfprozess dar, welcher sich aus mehreren Teilprozessen zusammensetzt. Abb. 3.9 zeigt eine grobe Darstellung des Ablaufs und soll die nachfolgende Erläuterung unterstützen. Die Implementierung unterteilt sich gemäß Kapitel 2.6.2 in den *preCheck()*, den *check()* und *postCheck()*.

preCheck

Die *preCheck()*-Methode bestimmt allgemein alle Informationen zur Definition der Brandabschnitte. Dabei wird in der **Vorbereitung** aus dem defaultFilterParameter die zu betrachtenden Räume entnommen und in einer Variable abgespeichert. Der gleiche Prozess wird für die auszuschließenden Räume aus dem Userinterface und für die Komponenten zur Bestimmung der Lage ebenfalls durchgeführt. Anschließend werden die Anforderungen an die brandabschnittsbildenden Elemente, also Wände, Decken, Stützen, Balken und Beläge, in jeweils einer Variable abgelegt. Da es mehrere zulässige Werte je Element gibt und die Eingabe mittels Trennzeichen („;“)

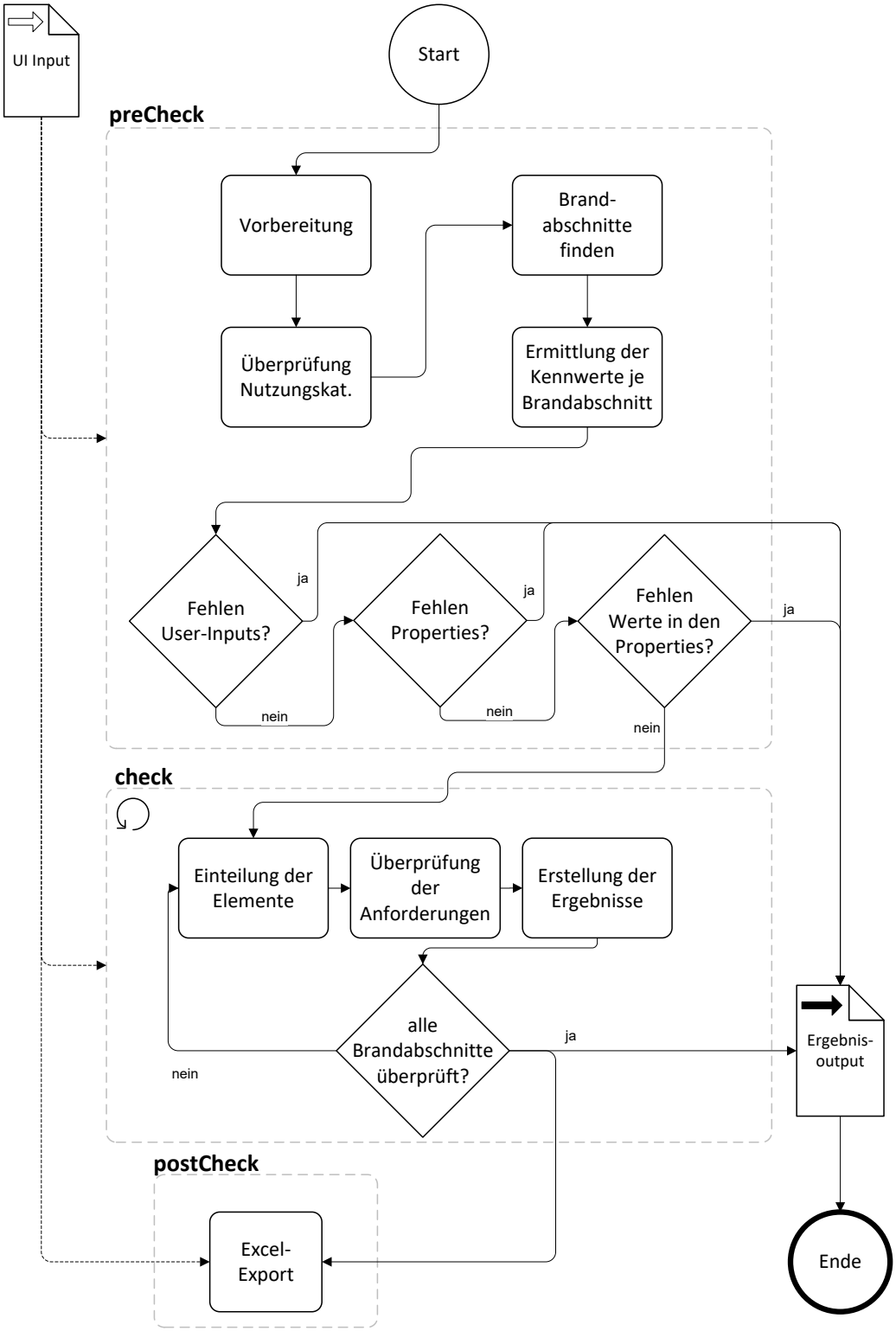


Abb. 3.9: Funktionsweise der Validierung – Gesamtüberblick

erfolgt, wird die Eingabe an dieser Stelle aufgesplittet und in einer Liste je Element abgelegt. Dabei ist darauf zu achten, dass es für Wände und Decken in Abhängigkeit der Tragfähigkeit zwei Listen gibt. Bei Türen gibt es anforderungsabhängig zwei Listen: vereinfachte Anforderungen und strenge Anforderungen. Diese Unterteilung bezieht sich auf die Vereinfachung gemäß OIB-Richtlinie 2, Punkt 3.1.6. Anschließend erfolgt die Abspeicherung der Anforderungen je Nutzungskategorie hinsichtlich der maximalen Fläche, der maximalen Längsausdehnung und der maximalen Geschoßanzahl. Diese Abspeicherung wird mit der *generateUsage()*-Methode in Code 3.1 dargestellt. Diese entnimmt dem jeweiligen Tableparameter die Einträge und legt sie in einer Variablen ab. Zur eindeutigen Kennzeichnung wird als Schlüssel (Key) die jeweilige Nutzungskategorie verwendet. Da der Tableparameter die Einträge als String zurückgibt, wandelt die *createDoubleValue()*-Methode die gebenen Zahlenwerte um. Damit der Nutzer einen größtmöglichen Spielraum hat, können Angaben im Tableparameter entfallen, zum Beispiel wenn für diese Nutzungskategorie keine Anforderungen an die Geschoßzahl gestellt werden.

```
1  public static Map<String, Map<String, Double>> generateUsage(ParameterTable table
     ) {
2
3    Map<String, Map<String, Double>> returnValue = new HashMap<>();
4
5    int rowCount = table.getRowCount();
6
7    for (int i = 0; i < rowCount; i++) {
8
9      Map<String, Double> props = new HashMap<>();
10
11     String usage = table.getValueAt(i, 0);
12     Double maxArea = createDoubleValue(table.getValueAt(i, 1));
13     Double maxPathlength = createDoubleValue(table.getValueAt(i, 2));
14     Double maxFloors = createDoubleValue(table.getValueAt(i, 3));
15
16     props.put("maxArea", maxArea);
17     props.put("maxPathlength", maxPathlength);
18     props.put("maxFloors", maxFloors);
19
20     returnValue.put(usage, props);
21   }
22   return returnValue;
23 }
```

Code 3.1: *generateUsage()*-Methode

Im nächsten Schritt kann die optionale **Überprüfung der Nutzungskategorienaufteilung** erfolgen. Sollte der Nutzer der API-Prüfregel die Aufteilung fordern, so werden alle Räume des defaultFilterParameters auf die angegebene Eigenschaft im Userinterface durchsucht. Anschließend werden die gefundenen Merkmalwerte und die zugehörigen Räume in einer Variable gespeichert. Sollte ein Raum des defaultFilterParameters das gesuchte Merkmal nicht aufweisen, so wird dieser unter „No Usage“ abgelegt und im Sinne der OIB-Richtlinie 2, Punkt 3.1.1 als „andere Nutzung“ identifiziert. Anschließend wird die Fläche je Nutzungskategorie ermittelt und mit den Maximalwerten aus dem Userinterface für einzelne Nutzungskategorien verglichen. Bei Überschreitung wird eine Boolean-Variable auf wahr gesetzt. Das hat zur Folge, dass in der anschließenden Überprüfung der Brandabschnitte Nutzungskategorien nicht gemischt werden dürfen. Wichtig bei der Eingabe im Userinterface ist die Angabe der Nutzungskategorie „andere“. Nur so können nicht definierte Nutzungen gesammelt und in Bezug zu den Anforderungen gesetzt werden. Sollte ein Flächenlimit für alle Nutzungskategorien bestehen, kann der Nutzer diese durch „alle“ kennzeichnen. Damit darf „andere“ entfallen.

Anschließend erfolgt das **Zusammenführen der einzelnen Brandabschnitte**. Um Inputfehler zu vermeiden, werden an dieser Stelle nur Komponenten des defaultFilterParameters des Typs Raum verwendet. Anschließend werden alle Räume, welche den gleichen Eigenschaftswert besitzen, zusammen in einer Variablen abgespeichert. Sollte bei einem Raum das Merkmal nicht existieren oder nicht befüllt sein, wird dieser in einer gesonderten Liste abgelegt. Als zusätzliche Sicherheitstufe werden anschließend alle Räume des Projekts auf die bekannten Merkmalwerte der Brandabschnitte geprüft. Mithilfe dieser Überprüfung können auch Räume mit fehlerhaftem Merkmalwert zur Bestimmung der Widmungsbehörde den korrekten Abschnitten zugeordnet werden. Code 3.2 zeigt die Aufteilung der Räume in Brandabschnitte.

```
// Brandabschnitte zusammen führen
// Section-Map befuellen ueber Eigenschaft als KEY
for (Component component : targetComponents) {
  if (component instanceof Space) {
    guidOfComponents.add(component.getGUID());
    try {
      String sectionProperty = String.valueOf(component
      .getPropertyValue(rpValidationProperty1PropertyReferenceParameter.getValue
  ())
      .get());

      if (sectionProperty.isEmpty() || sectionProperty.equals("")) {
        componentsWithoutSection.add(component);
      } else {
        if (predefinedSections.keySet().contains(sectionProperty)) {
          predefinedSections.get(sectionProperty).add(component);
        } else {
          ArrayList<Component> sectionList = new ArrayList<>();
          sectionList.add(component);
          predefinedSections.put(sectionProperty, sectionList);
        }
      }
    } catch (Exception e) {
      componentsWithoutSection.add(component);
    }
  }
}

// Heranziehen aller Raeume, ob ein weiterer Raum die Eigenschaft des
// Brandabschnittes hat
Set<Component> spacesOfModel = SMC.getModel()
.getComponents(ComponentFilter.componentTypeIs(ComponentType.SPACE));

for (Component otherSpaces : spacesOfModel) {
  try {
    String sectionProperty = String.valueOf(otherSpaces
    .getPropertyValue(rpValidationProperty1PropertyReferenceParameter.getValue())
  .get());
    if (predefinedSections.keySet().contains(sectionProperty)) {
      predefinedSections.get(sectionProperty).add(otherSpaces);
    }
  } catch (Exception e) {
    // Bei anderen Raeumen muss die Eigenschaft nicht vorhanden sein
  }
}
```

Code 3.2: Zusammenführen der einzelnen Brandabschnitte

Mit den so gefundenen Brandabschnitten können nun die **Kennwerte je Brandabschnitt** ermittelt werden. Diese werden durch mehrere Subprozesse bestimmt. Der Ablauf ist in Abb. 3.10 dargestellt.

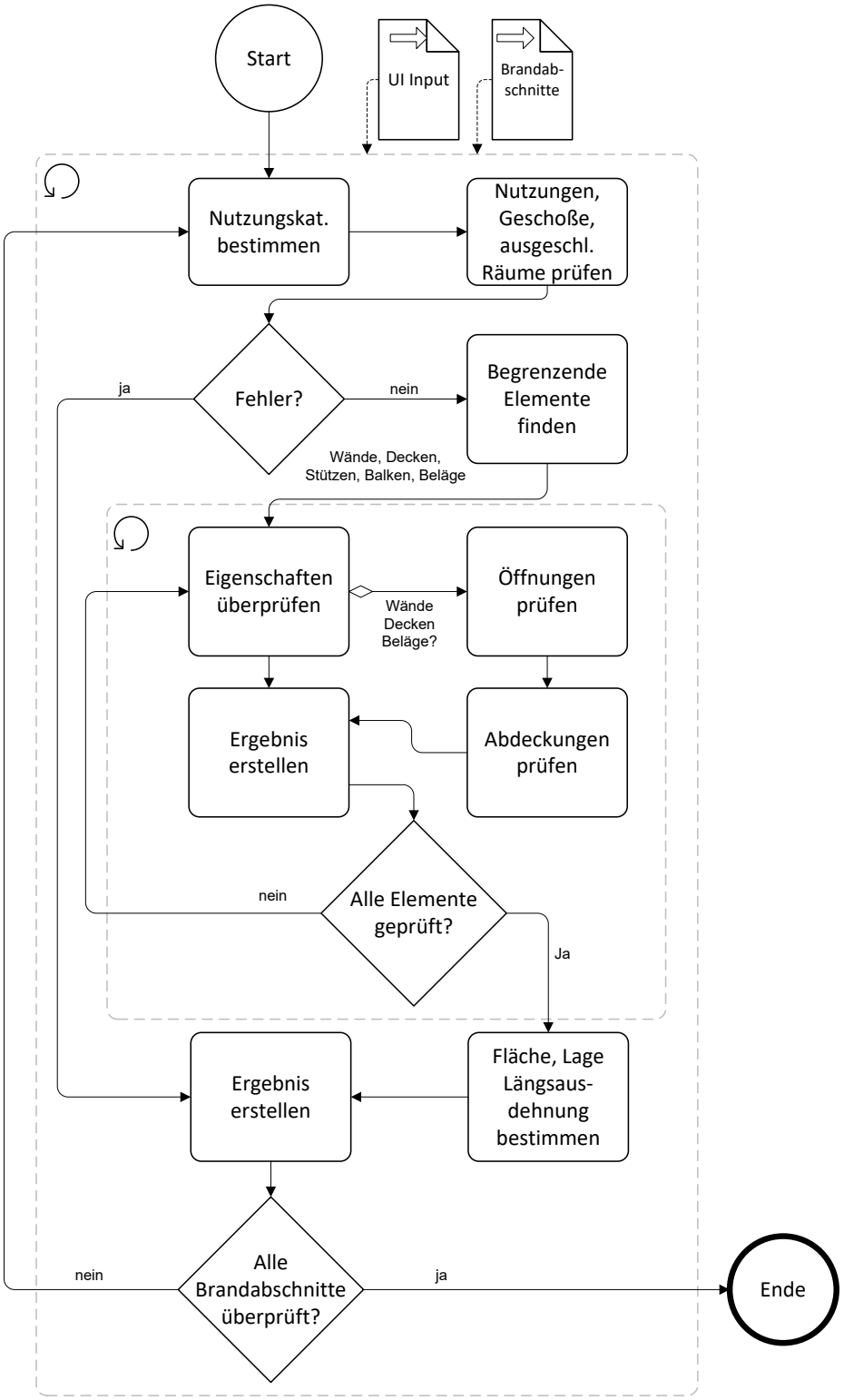


Abb. 3.10: Überblick über den Subprozess zur Bestimmung der Kennwerte

Zu Beginn bestimmt der Algorithmus die Nutzungskategorien. Dafür werden die Nutzungen aller Räume betrachtet und gespeichert. Darauf folgt die Einteilung der Nutzungen gemäß jener aus dem Userinterface. In Abhängigkeit der optionalen Überprüfung der Nutzungskategorienaufteilung

wird validiert, ob alle Räume die gleiche Nutzungskategorie aufweisen. Darauffolgend überprüft die API-Prüfregel, ob keine ausgeschlossenen Räume innerhalb des Brandabschnitts existieren. Sollte eine der beiden Bedingungen nicht erfüllt sein, so stellt dies ein K.O.-Kriterium dar und für den Brandabschnitt wird ein fehlerhaftes Ergebnis erstellt. In der gleichen Methode wird unabhängig vom K.O.-Kriterium noch die Anzahl der Geschoße für den Brandabschnitt ermittelt und je Geschoß alle Räume des Brandabschnitts gespeichert. Sollte ein Raum keinem Geschoß zugewiesen sein, erfolgt eine gesonderte Abspeicherung in einer eigenen Liste. Nach den allgemeinen Anforderungen beginnt die Prüfregel die brandabschnittsbildenden Elemente des Brandabschnitts zu bestimmen. Für jeden Raum des Brandabschnitts werden zwei Methoden ausgeführt. Im ersten Schritt erfolgt die Aufteilung der Elemente in Wände, Decken, Stützen, Balken, Beläge und Spaceboundaries. Spaceboundaries sind Raumbegrenzungen, die keinem Bauelement zugeordnet werden können, z. B. andere Räume. Die genannten Bauelemente und Spaceboundaries werden gesammelt als Raumbegrenzungen bezeichnet. Die Aufteilung erfolgt mit der Methode *seperateSpaceBoundaries()*. Code 3.3 zeigt, dass die Methode vom Typ void als Übergabeparameter die aktuelle Raumkomponente (currentSpace) und die Listen für Wände, Decken, Stützen, Balken, Beläge und Raumbegrenzungen benötigt.

```
1  public static void seperateSpaceBoundaries(Space currentSpace, ArrayList<
     Component> walls,
2  ArrayList<Component> slabs, ArrayList<Component> columns, ArrayList<Component>
     beams,  ArrayList<Component> coverings,
3  ArrayList<Component> spaceBoundaries) {
4    Collection<SpaceBoundary> allSpaceBoundaries = currentSpace.getSpaceBoundaries
     ();
5    for (SpaceBoundary spaceBoundary : allSpaceBoundaries) {
6      Component spaceBoundaryComponent = null;
7      try {
8        spaceBoundaryComponent = spaceBoundary.getRelatedBuildingElement().get();
9
10       if (spaceBoundaryComponent instanceof Wall) {
11         walls.add(spaceBoundaryComponent);
12       } else if (spaceBoundaryComponent.getIfcType().get().toString().contains("
     Slab")) {
13         slabs.add(spaceBoundaryComponent);
14       } else if (spaceBoundaryComponent.getIfcType().get().toString().contains("
     Column")) {
15         columns.add(spaceBoundaryComponent);
16       } else if (spaceBoundaryComponent instanceof Beam) {
17         beams.add(spaceBoundaryComponent);
18       } else if (spaceBoundaryComponent.getIfcType().get().toString().contains("
     Covering")) {
19         coverings.add(spaceBoundaryComponent);
20       }
21     } catch (Exception e) {
22       try {
23         spaceBoundaries.add((Component) spaceBoundary);
24         String s = "";
25       } catch (Exception f) {
26         String s = "";
27       }
28     }
29   }
30 }
```

Code 3.3: *seperateSpaceBoundaries()*-Methode

Anschließend werden mithilfe einer Methode der space-Klasse alle Raumbegrenzungen gefunden und diese an ein zugehöriges Bauelement über *getRelatedBuildingElement().get()* geknüpft. Sollte kein Bauelement gefunden werden, entsteht eine Exception, welche im zweiten Teil der Metho-

de abgefangen wird. Hier werden die restlichen Raumbegrenzungen der Liste hinzugefügt. Im Anschluss erfolgt die Einteilung der Bauelemente in abschnittsbildend und innenliegend. Dies erfolgt mithilfe der Methode *defineOuterComponentsOfSection()*. Diese Methode ist ebenfalls vom Typ void und benötigt als Eingangsparameter die zu betrachtenden Elemente, welche aus der vorherigen Methode als Liste zu Verfügung stehen, die aktuelle Raumkomponente, die restlichen Räume des Brandabschnitts, die Listen der abschnittsbildenen Elemente und Innenelemente des Brandabschnitts, den zugehörigen PropertyReferenceParameter zur Eigenschaft der Gebäudeaußenhaut und eine Boolean-Variable. In der Methode wird für jedes zu betrachtende Element eine Beziehung (Relation) definiert, womit die in Beziehung stehenden Räume gefunden werden können. Für jeden Raum, der in Beziehung mit dem aktuellen Element steht wird anschließend überprüft, ob

1. dieser der aktuellen Raumkomponente entspricht. Tritt der Fall ein, wird überprüft, ob es sich um ein Element der Gebäudehülle handelt, denn dann gilt dieses Element ebenfalls als brandabschnittsbildend. Ist das nicht der Fall, wird geprüft, ob

2. der Raum zum aktuellen Brandabschnitt gehört. Trifft das ebenfalls nicht zu, muss das aktuelle Element brandabschnittsbildend sein. Sollte der Raum zum Brandabschnitt gehören wird erneut überprüft, ob

3. es sich bei dem Element um eines der Gebäudehülle handelt. Gilt diese Bedingung erfüllt, so ist das Element ebenfalls brandabschnittsbildend.

Die verschiedenen Fälle sind in Abb. 3.11 in unterschiedlichen Farben dargestellt.

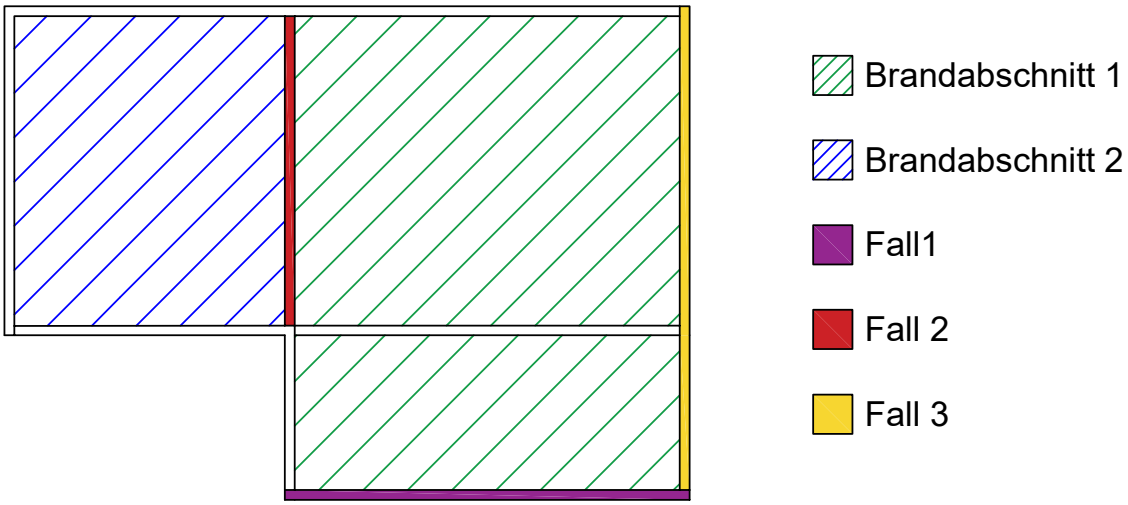


Abb. 3.11: Verschiedene Konfigurationen für abschnittsbildende Elemente am Beispiel Wand

In diesen drei Fällen wird das Element der Liste für brandabschnittsbildende Elemente hinzugefügt. In allen anderen Fällen wird das Element als Innenelement gewertet und in der Liste für Innenelemente abgespeichert. In Code 3.4 ist die *defineOuterComponentsOfSection()*-Methode dargestellt.

```
1 public static void defineOuterComponentsOfSection(ArrayList<Component> components,
      Space currentSpace,
2 ArrayList<Component> spacesOfSection, ArrayList<Component> outerElements,
```

```
3 ArrayList<Component> innerElements, PropertyReferenceParameter prp, boolean
      spaceNotNextToEachOther) {
4   for (Component c : components) {
5
6     // Von Elemnt auf Nebenraum/raume
7     // Relation definieren
8     Relation relation = Relation.of(Relation.Type.NEAREST_SPACES, Relation.
      Direction.BOTH);
9     // Related Spaces
10    Collection<Component> relatedComponents = c.getRelated(relation);
11
12    for (Component relatedComponent : relatedComponents) {
13      if (!relatedComponent.getGUID().equals(currentSpace.getGUID())) {
14        // Aussenwand der Sektion hinzufuegen
15        if (!spacesOfSection.contains(relatedComponent)) {
16          outerElements.add(c);
17          innerElements.remove(c);
18          break;
19        } else {
20          // Ueberpruefungg ob Wand eine Gebauedeaussenenwand ist
21          Boolean buildingEnvelopeBool = Boolean
22          .valueOf(String.valueOf(c.getPropertyValue(prp.getValue()).get()));
23
24          if (buildingEnvelopeBool == true) {
25            outerElements.add(c);
26            innerElements.remove(c);
27            break;
28          } else {
29            // Innenwand der Sektion hinzufuegen
30            innerElements.add(c);
31            spaceNotNextToEachOther = false;
32          }
33        }
34      } else {
35        // Ueberpruefungg ob Wand eine Gebauedeaussenenwand ist
36        Boolean buildingEnvelopeBool = Boolean
37        .valueOf(String.valueOf(c.getPropertyValue(prp.getValue()).get()));
38
39        if (buildingEnvelopeBool == true) {
40          outerElements.add(c);
41          innerElements.remove(c);
42          break;
43        }
44      }
45    }
46  }
47 }
```

Code 3.4: *defineOuterComponentsOfSection()*-Methode

Die Methode wird für alle Bauelementlisten ausgeführt, also Wände, Decken, Stützen, Balken und Beläge. Für Spaceboundaries wird davon ausgegangen, dass es sich dabei um Räume handelt, die direkt aneinandergrenzen. Damit benötigt es lediglich eine Prüfung, ob diese Räume im selben Brandabschnitt liegen. Ist diese Bedingung nicht erfüllt, kann der Abschnitt als fehlerhaft markiert werden. Im Zuge der Bestimmung der brandabschnittsbildenden Elemente wird außerdem der niedrigste Raum in Bezug auf die globale Höhenkoordinate bestimmt. Für jeden Raum wird dabei die Eigenschaft GlobalBottomElevation betrachtet und verglichen. Jener mit dem niedrigsten Wert wird abgespeichert. Während der Bestimmung der brandabschnittsbildenden Elemente wird auch der übergebene Boolean-Wert der *defineOuterComponentsOfSection()*-Methode befüllt. Sollte ein Raum kein Begrenzungselement besitzen, welches zumindest an einen anderen Raum des Brandabschnitts grenzt, und die Raumanzahl des Brandabschnitts mehr als eins betragen, gilt der Brandabschnitt als fehlerhaft und der Raum wird in einer Liste abgespeichert. Der nächste Schritt

ist die Prüfung der Eigenschaften der Elemente (Wände, Decken, Stützen, Balken und Beläge). Die *checkOuterElements()*-Methode vom Typ void übernimmt den Abgleich der Elementmerkmale mit den Anforderungen gemäß Userinterface. Die Methode startet eine Überprüfung für jedes übergebene brandabschnittsbildende Element und bestimmt in Abhängigkeit eines Boolean-Wertes und der Elementeigenschaft, ob es sich um ein tragendes Element handelt. Abhängig davon entscheidet die If-Abfrage, ob die Bestimmungen für tragende oder nichttragende Elemente betrachtet werden. Anschließend erfolgt der Abgleich über die *checkConditionForValidation()*-Methode, wobei diese lediglich den Merkmalwert des Elements und die Anforderungen ohne Leerzeichen vergleicht. Sie liefert als Rückgabewert einen Boolean, wobei dieser Null sein kann. Ist dies der Fall, wird eine Fehlermeldung zurückgeben und der Nutzer über eine String-Nachricht über die fehlerhafte Eingabe informiert. Andernfalls wird der Boolean-Wert in einer Liste abgespeichert. Sollte keine der Anforderungen aus den Eingaben im Userinterface erfüllt werden, wird das Element mit zugehörigem Fehlertext in einer Liste abgespeichert. In Code 3.5 ist die *checkOuterElements()*-Methode samt Übergabeparameter dargestellt.

```
1  public static void checkOuterElements(ArrayList<Component> outerElements, Map<
     Component, String> sectionErrors,
2  ArrayList<ArrayList<Boolean>> boolLists, ArrayList<String> elementRules,
3  ArrayList<String> nonBearingElementRules,
4  PropertyReferenceParameter rpElementProperty1PropertyReferenceParameter,
5  PropertyReferenceParameter rpElementProperty2PropertyReferenceParameter,
     ArrayList<String> errorInUserInput,
6  Boolean differentiatonInBearing) {
7
8    for (Component c : outerElements) {
9      ArrayList<Boolean> boolList = new ArrayList<>();
10     Boolean bearingBool;
11     if (differentiatonInBearing) {
12       try {
13         bearingBool = (Boolean) c.getPropertyValue(
   rpElementProperty2PropertyReferenceParameter.getValue())
14         .get();
15
16       } catch (Exception propertyMissing) {
17         bearingBool = false;
18       }
19     } else {
20       bearingBool = true;
21     }
22     if (bearingBool == true) {
23       for (String rule : elementRules) {
24         Boolean boolElement = FCVR_FurtherFunctions.checkConditionForValidation(c
   ,
25         rpElementProperty1PropertyReferenceParameter, rule);
26
27         if (boolElement == null) {
28           errorInUserInput.add("The Type of the choosen Property and the
   Comparisonvalue do not match");
29         }
30         boolList.add(boolElement.booleanValue());
31       }
32     } else {
33       for (String rule : nonBearingElementRules) {
34         Boolean boolElement = FCVR_FurtherFunctions.checkConditionForValidation(c
   ,
35         rpElementProperty1PropertyReferenceParameter, rule);
36
37         if (boolElement == null) {
38           errorInUserInput.add("The Type of the choosen Property and the
   Comparisonvalue do not match");
39         }
40
41         boolList.add(boolElement.booleanValue());
```

```
42          }
43        }
44        if (!boolList.contains(true)) {
45          sectionErrors.put(c, "Element does not meet requirements");
46        }
47        boolLists.add(boolList);
48      }
49    }
```

Code 3.5: *checkOuterElements()-Methode*

Liefert die *CheckOuterElements()*-Methode für alle Bauteilelemente eines bestimmten Typs, dass diese fehlerfrei sind, kann die optionale Überprüfung der Öffnungen und Abdeckungen beginnen. An dieser Stelle bedeutet optional, dass im Programmcode Öffnungen und Abdeckungen nur für Wände, Decken und Beläge überprüft werden. Die *checkOpeningsOfElements()*-Methode ist eine sehr komplexe Methode vom Typ void. In Abb. 3.12 ist der Verlauf dargestellt. Die Übergabeparameter lassen sich in drei Gruppen teilen: die aktuellen brandabschnittsbildenden Elemente, die Anforderungen sowohl an diese Elemente selbst als auch an die Abschließungselemente (Türen/Fenster) und die PropertyReferenceParameter zur Bestimmung der Merkmale je Element. Zu Beginn der Methode steht eine Schleife über alle übergebenen brandabschnittsbildenden Elemente. Anschließend werden für jedes Element gemäß Abb. 3.12 Öffnungen gesucht. Findet der Algorithmus keine, endet die Methode und ein Ergebnis wird erstellt. Findet der Algorithmus Öffnungen, beginnt die Vorsortierung. Mithilfe dieser können nicht-relevante Öffnungen im Sinne des Brandschutzes gefunden und aussortiert werden. Sie entstehen aufgrund von Modellierungsvereinfachungen. Bei der Überprüfung werden drei Arten von Öffnungen in brandabschnittsbildenden Elementen unterschieden:

- Öffnung, welche einen Raum des aktuellen Brandabschnitts mit einem beliebigen Raum verbindet. Der zweite Raum ist nicht Teil des aktuellen Brandabschnitts. Diese Öffnung ist relevant für die Brandschutzbetrachtung.
- Öffnung, welche Räume des aktuellen Brandabschnitts verbindet. Diese Öffnung ist nicht relevant, da diese innerhalb des Brandabschnitts liegt.
- Öffnung, die Räume verbindet, welche beide nicht Teil des aktuellen Brandabschnitts sind. Diese Öffnung ist ebenfalls nicht Prüfungsrelevant.

Zum besseren Verständnis sind die drei Arten von Öffnungen in Abb. 3.13 beispielhaft für Wände dargestellt. In allen drei Bildern ist in grün der aktuelle Brandabschnitt dargestellt. Für die letzten beiden Öffnungen ist die Voraussetzung, dass ein brandabschnittsbildendes Element über die Außengrenzen des Brandabschnitts hinweg modelliert ist. Die Einteilung erfolgt mithilfe der *checkIfSpacesAreNotInTheSameCompartment()*-Methode, welche in Code 3.6 dargestellt ist. Dieser zeigt, dass die Räume allein durch ihren Wert im Merkmal Brandabschnitt unterschieden werden.

Nach dem Aussortieren der nicht-relevanten Öffnungen wird die Gesamtfläche aller Öffnungen je brandabschnittsbildendem Element bestimmt. Dies erfolgt durch Iterieren über alle relevanten Öffnungen und Addieren ihrer Fläche (Quantity). Anschließend beginnt die Überprüfung. Diese wird als Schleife für jede Öffnung ausgeführt. Dabei steht an erster Stelle ein K.O.-Kriterium: Sollte eine Öffnung keine Abdeckung (Tür, Fenster) besitzen, ist diese als fehlerhaft zu betrachten und es wird direkt ein Ergebnis erstellt. Findet der Algorithmus eine Abdeckung, wird die zuvor berechnete Gesamtfläche der Öffnungen herangezogen, um die nachfolgenden Nachweise zu bestimmen. Im Userinterface kann der Nutzer einen Grenzwert festlegen bis zu dem geringere

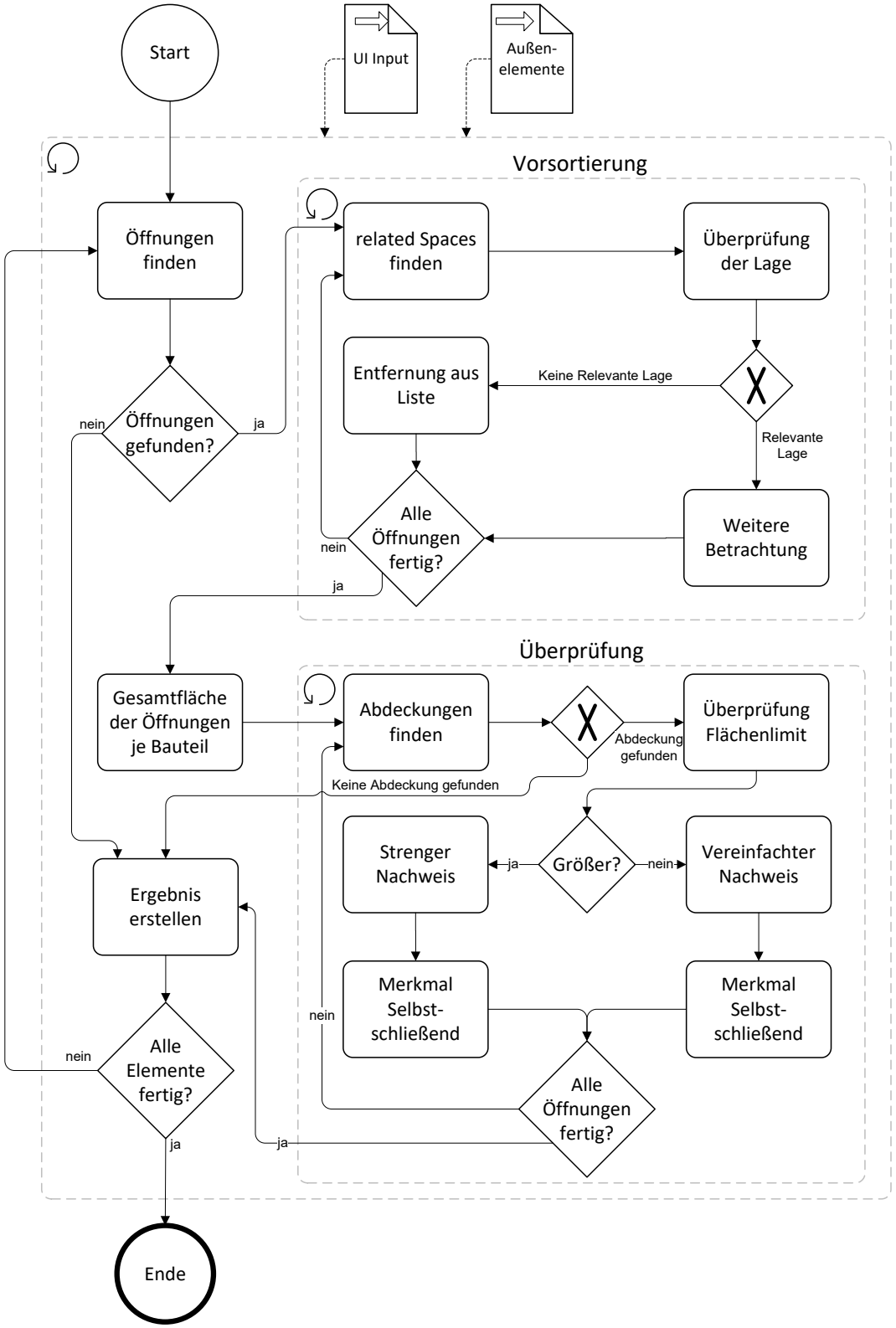


Abb. 3.12: Ablauf der *checkOpeningsOfElements()*-Methode

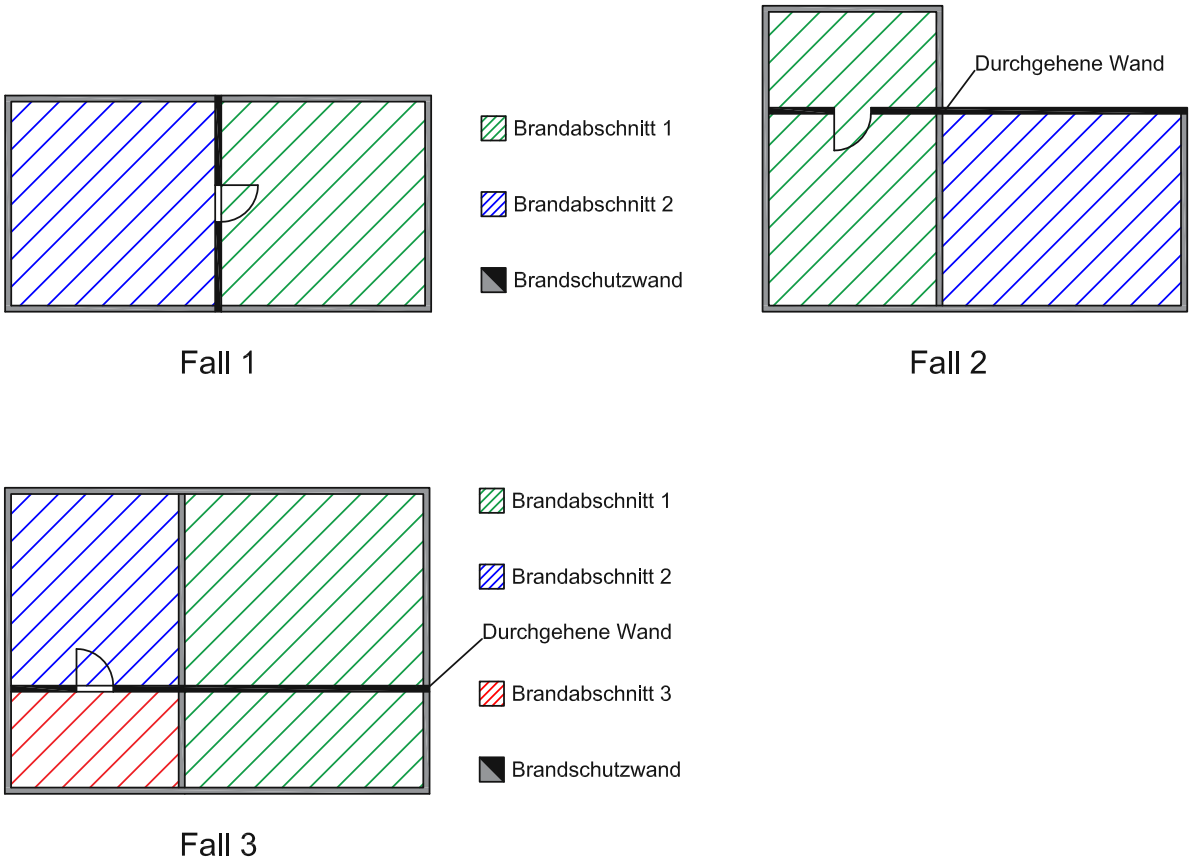


Abb. 3.13: Öffnungsarten im Kontext der Brandschutzprüfung

```
1  public static Boolean checkIfSpacesAreNotInTheSameCompartment(Component space1,
     PropertyReferenceParameter pr1,
2  String currentCompartment) {
3    Boolean bool = false;
4    String sp1;
5    try {
6      sp1 = String.valueOf(space1.getPropertyValue(pr1.getValue()).get());
7    } catch (Exception e) {
8      sp1 = "";
9    }
10   if (sp1.equals(currentCompartment)) {
11     // Fall das einer der beiden nicht im brandabschnitt liegt
12     bool = true;
13   }
14   else {
15     // Fall das beide nicht im Brandabschnitt liegen
16     bool = false;
17   }
18   return bool;
19 }
```

Code 3.6: *checkIfSpacesAreNotInTheSameCompartment()*-Methode

Anforderungen gelten. Sollte der Wert überschritten werden, müssen die strengeren Anforderungen eingehalten werden. Für beide Nachweise wird im Fall, dass die Anforderungen erfüllt sind, die Abdeckung auf das Merkmal Selbstschließend geprüft. Diese Prüfung wird durch die

checkIfCoveringIsSelfClosing( )-Methode durchgeführt. Sind alle Öffnungen überprüft, wird ein Ergebnis für das betrachtete Element erstellt. Dieser Vorgang wird für alle brandabschnittsbildenden Elemente wiederholt. Gemäß Abb. 3.10 folgt nach der Prüfung der brandabschnittsbildenden Elemente die **Bestimmung der Fläche, Längsausdehnung und der Lage**. Für die Flächenberechnung wird aus dem Userinterface zuerst entnommen, ob und welche Wände herangezogen werden sollen. Anschließend erfolgt die Berechnung der mithilfe *calculateAreaOfSection()*-Methode des Typs void. Code 3.7 zeigt, dass die *calculateAreaOfSection()*-Methode den Footprint aller Raumkomponenten und Wandelemente heranzieht, um so die Area-Komponente zu erhalten. Die Area-Klasse ermöglicht direkt den Zugriff auf die Flächengröße. Dann kann die Gesamfläche durch Aufsummieren ermittelt werden.

```
1 public void calculateAreaOfSection() {
2   this.area = 0.0;
3
4   for (Component c : this.componentsOfSection) {
5     Area areaOfOneComponent = c.getFootprint().getArea();
6     this.area += areaOfOneComponent.getSize();
7   }
8 }
```

Code 3.7: *calculateAreaOfSection()*-Methode

Die Berechnung der Längsausdehnung erfolgt entsprechend der Vereinfachung gemäß Abschnitt 3.2. Sie erfolgt geschoßweise, weshalb auf die abgespeicherten Räume je Geschoß aus den Kennwerten je Brandabschnitt zurückgegriffen wird. Anschließend werden die Räume über ihren Footprint zu einer großen Flächen zusammengefügt, wobei Löcher entfernt werden. Durch die Bestimmung der Eckpunkte kann mithilfe einer doppelten Schleife die maximale Distanz gefunden werden. Diese wird je Geschoß betrachtet und der größte Wert für den Brandabschnitt abgespeichert. Code 3.8 zeigt die soeben beschriebene Ermittlung der Längsausdehnung.

```
1   for (String keyForFloors : differentFloors.keySet()) {
2     // Fläche erstellen
3     MArea areaComponent = MArea.create();
4     for (Component comp : differentFloors.get(keyForFloors)) {
5       // Flächen zusammen addieren
6       areaComponent.add(comp.getFootprint().getArea());
7     }
8     areaComponent.removeHoles();
9     // Eckpunkte holen, distanz, startpunkt, endpunkt initialisieren
10    ArrayList<MVector2d> vertices = new ArrayList<MVector2d>(areaComponent.
      getVertices());
11    Double maxDistance = 0.0;
12    Vector3d startPoint = null;
13    Vector3d endPoint = null;
14
15    for (int i = 0; i < vertices.size(); i++) {
16      for (int j = i + 1; j < vertices.size(); j++) {
17        if (maxDistance < vertices.get(i).distance(vertices.get(j))) {
18          maxDistance = vertices.get(i).distance(vertices.get(j));
19          Vector3d newStartPoint = Vector3d.create(vertices.get(i).getX(),
20          vertices.get(i).getY(),
21          differentFloors.get(keyForFloors).get(0).getGlobalBottomElevation());
22          Vector3d newEndPoint = Vector3d.create(vertices.get(j).getX(),
23          vertices.get(j).getY(),
24          differentFloors.get(keyForFloors).get(0).getGlobalBottomElevation());
25
26          startPoint = newStartPoint;
27          endPoint = newEndPoint;
28        }
29      }
```

```
30      }
31      // nur wenn die Distanz eines neuen Geschosses größer ist als jenes davor
32      if (section.getLongitudinalDimension() == null
33      || (section.getLongitudinalDimension() < maxDistance)) {
34        section.setLongitudinalDimension(maxDistance);
35        if (startPoint != null && endPoint != null) {
36          Segment3d seg = Segment3d.create(startPoint, endPoint);
37          section.setLongestDistanceSegment(seg);
38        }
39      }
40    }
```

Code 3.8: Bestimmung der Längsausdehnung

Zum Abschluss der *preCheck()*-Methode erfolgt die Überprüfung, ob ein Brandabschnitt gemäß der Vereinfachung im Abschnitt 3.2 als unterirdischer Brandabschnitt zählt. Dafür wird der zuvor bestimmte, niedrigste Raum mit einem garantiert unterirdischen Element, welches im Userinterface angegeben sein muss, verschnitten. Im Projekt BRISE-Vienna wird der Volumenkörper BGF-unterirdisch genutzt. Ergibt sich ein Verschnittkörper, so gilt die Bedingung als erfüllt.

Treten während der *preCheck()*-Methode Fehler auf, die entweder auf fehlerhafte Angabe des Nutzer oder auf fehlende Merkmalangaben zurückzuführen sind, erstellt der *preCheck()* ein preCheckResult, welches den Nutzer auf dieses Umstand hinweißt. Dieses Ergebnis kann zum Beispiel auftreten, wenn der Nutzer einen Text mit einem Merkmal vergleichen möchte, dass einen Zahlwert besitzt.

Check

In der *check()*-Methode teilt der Algorithmus die zuvor bestimmten Kennwerte und Fehler der Brandabschnitte in verschiedene Ergebniskategorien ein. Eine nähere Beschreibung der einzelnen Kategorien erfolgt unter Abschnitt 3.4.3. Die *check()*-Methode wird für jedes Element des defaultFilterParameters durchlaufen. Ein Ergebnis würde für jeden Raum des Brandabschnitts erstellt werden. Fehlerhafte Räume eines Brandabschnitts sollen gesammelt als Ergebnis zurückgegeben werden, weshalb alle Elemente in einer Klassenvariablen gespeichert sind. Gemäß Abb. 3.9 beginnt die Methode während der **Einteilung der Elemente** die Überprüfung damit, ob das aktuelle Element noch in der Klassenvariable liegt. Trifft diese Bedingung zu und ist das Element nicht als ausgeschlossen deklariert, durchsucht der Algorithmus die bestehenden Brandabschnitte auf das aktuelle Element. Sobald dieser gefunden wurde, werden alle Räume aus der Klassenvariable entfernt. So kann ein Ergebnis pro Brandabschnitt erstellt werden. Nachfolgend beginnt die **Überprüfung der Anforderungen**. In dieser Phase erfolgt die Einteilung in die verschiedenen Ergebniskategorien. Zuerst werden die Fehler behandelt, welche die allgemeine Definition von Brandabschnitten betrifft. Das betrifft die Existenz eines Werts im Merkmal für den Brandabschnitt sowie das Vorhandensein von ausgeschlossenen Räumen im Brandabschnitt. Werden keine dieser Fehler gefunden, ermittelt der Algorithmus die Anforderungsgrenzen hinsichtlich der Fläche, der Längsausdehnung und der maximalen Geschoßanzahl. Da diese optional von der Nutzungskategorie und immer von der Lage hinsichtlich ober- bzw. unterirdisch abhängen, wird wie in Code 3.9 eine Variable erstellt, die durch If-Abfragen die jeweiligen Bedingungen je Brandabschnitt berücksichtigt.

```
Double areaLimitation = 0.0;
Double lengthLimitation = 0.0;
Double floorLimitation = 0.0;

Double areaOfSection = section.getArea();

Map<String, Map<String, Double>> resultMap = new HashMap<String, Map<String,
  Double>>();
if (section.getAboveGroundLevel() == true) {
  resultMap.putAll(aboveGroundFromUI);
} else {
  resultMap.putAll(underNeaveGroundFromUI);
}
if (!resultMap.keySet().contains("alle")) {
  for (String usage : section.getUsages().keySet()) {
    if (resultMap.keySet().contains(usage)) {
      Double area = resultMap.get(usage).get("maxArea");
      Double length = resultMap.get(usage).get("maxPathlength");
      Double floors = resultMap.get(usage).get("maxFloors");

      if (areaLimitation < area) {
        areaLimitation = area;
      }
      if (lengthLimitation < length) {
        lengthLimitation = length;
      }
      if (floorLimitation < floors) {
        floorLimitation = floors;
      }
    }
  }
} else {
  areaLimitation = resultMap.get("alle").get("maxArea");
  lengthLimitation = resultMap.get("alle").get("maxPathlength");
  floorLimitation = resultMap.get("alle").get("maxFloors");
}
```

Code 3.9: Ermittlung der Bedingungen je Brandabschnitt

Dabei sind zwei Bedingungen gegeben: Sollte die Nutzungskategoriegrenze nicht überschritten werden, dürfen unterschiedliche Nutzungen innerhalb eines Brandabschnitts vorkommen. Deswegen werden stets die strengeren Anforderungen unabhängig der Kategorie herangezogen. Fordert zum Beispiel der Nutzer bei Nutzung1: „Wohnen und Aufenthalt“ eine maximale Fläche von 500 m^2 und eine maximale Geschoßanzahl von vier, jedoch bei Nutzung2: „Büroarbeit“ 800 m^2 und zwei Geschoße, ergibt das für den gesamten Brandabschnitt eine maximale Fläche von 500 m^2 und maximal zwei Geschoße. Die zweite Bedingung ist die Angabe der Nutzung „alle“. Sofern der Nutzer diese Nutzungskategorie angibt, werden alle anderen Einträge ignoriert und rein diese Anforderungen herangezogen. Nachdem alle Anforderungen bekannt sind, werden die Kennwerte je Brandabschnitt geprüft und ein **Ergebnis erstellt**. Diese drei Schritte werden solange wiederholt bis keine Elemente in der Klassenvariable mehr vorhanden sind bzw. alle Brandabschnitte behandelt wurden. Anschließend wird das Gesamtergebnis als Liste mit Ergebnissen an Solibri übergeben.

postCheck

In der *postCheck()*-Methode wird optional der Excel-Export der Ergebnisse ausgeführt. Abhängig von den Einstellungen im Userinterface erfolgt die Ausgabe von fehlerhaften und/oder fehlerfreien Brandabschnitten. Zur Erstellung einer Excel-Datei wird die Java-Bibliothek Apache POI [2] Version 5.1 verwendet. Zu Beginn der *postCheck()*-Methode generiert die API-Prüfregel den Dateipfad aus den Angaben im Userinterface. Anschließend wird eine neue Datei mit einem

Tabellenblatt „Brandabschnitte“ erstellt. Im nächsten Schritt wird das Tabellenblatt geöffnet und mit den Brandabschnitten je Ergebniskategorie beschrieben. Sobald alle Brandabschnitte eingetragen sind, wird die Datei geschlossen und der Export ist fertig gestellt.

3.4.3 Ergebnis in Solibri

Brandabschnitte werden durch den Algorithmus in verschiedene Ergebniskategorien eingeteilt. Abhängig davon, in welche Ergebniskategorie der Brandabschnitt durch die API-Prüfregel eingeteilt wird, erfolgen andere Ergebnisausgaben. Dabei gibt es drei Ausgaben, die mehrmals verwendet werden:

- Allgemeine Angaben als Beschreibung
  Für Brandabschnitte, die keine ausgeschlossen Räume enthalten sowie die Anforderungen an die Nutzungskategoriegrenzen eingehalten sind, werden allgemeine Kennwerte als Beschreibungstext im Ergebnis dargestellt. Diese Ausgabe, dargestellt in Abb. 3.14, umfasst den Namen, die Fläche in m², die Nutzunskategorie(n), die Anzahl der Räume, die Lage, die Längsausdehnung in m und ob Fehler enthalten sind.

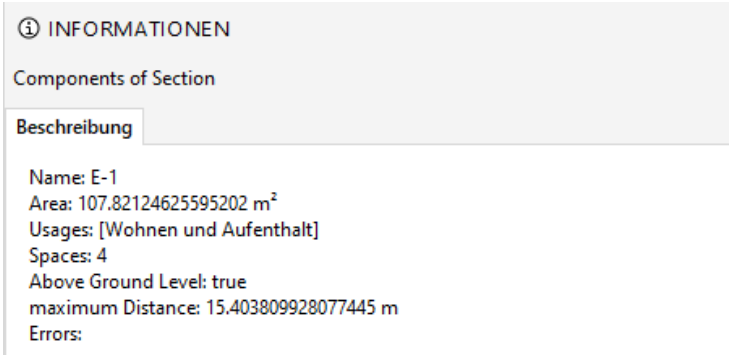


Abb. 3.14: Beschreibung eines fehlerfreien Brandabschnitts

- Components of Section
  In dieser Ausgabe sind alle Räume des Brandabschnitts zusammengefasst und können so grafisch erfasst werden. Die Räume eines typischen Brandabschnitts sind in Abb. 3.15 dargestellt.

- Visualization of Distance bzw. Distance exceeds Limit
  Die unterschiedlichen Bezeichnungen für diese Ausgabe ergeben sich je nach dem, ob die Längsausdehnung die Anforderungen überschreitet oder nicht. Die Darstellung ist für beide Fälle mit einer Bemaßungslinie gleich. Abb. 3.16 zeigt die Bemaßungslinie für einen Brandabschnitt in L-Form.

Diese drei Ausgaben bilden die Basis für sechs Ergebniskategorien. Nachfolgend werden die einzelnen Kategorien erläutert und die einzelen Ergebnisausgaben dargestellt. Weiters erfolgt eine kurze Beschreibung über Umfang bzw. Inhalt der einzelnen Ergebniskategorien.

All requirements are fullfilled

Diese Ergebniskategorie beinhalteten jene Brandabschnitte, die alle Anforderungen gemäß Userinterface erfüllen. Dabei werden die drei Basis-Ausgaben je Brandabschnitt ausgegeben. Somit umfasst ein Ergebnis in dieser Kategorie die folgenden Ausgaben:

- allgemeine Angaben als Beschreibung

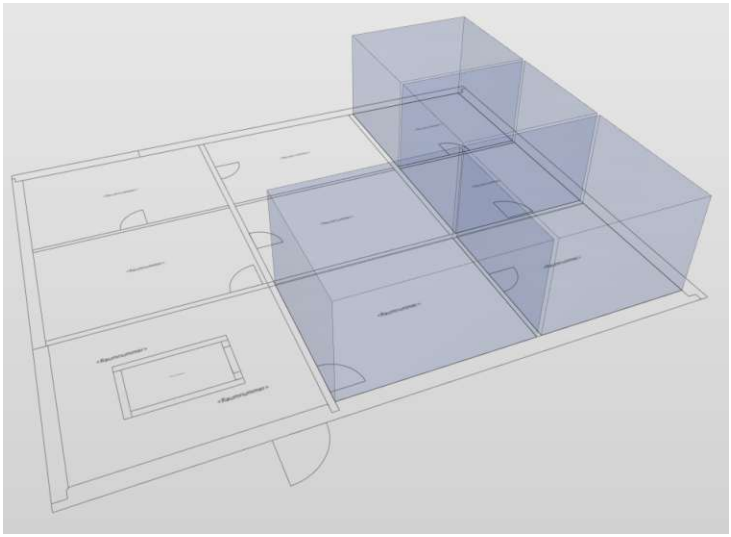


Abb. 3.15: Komponenten eines Brandabschnitts

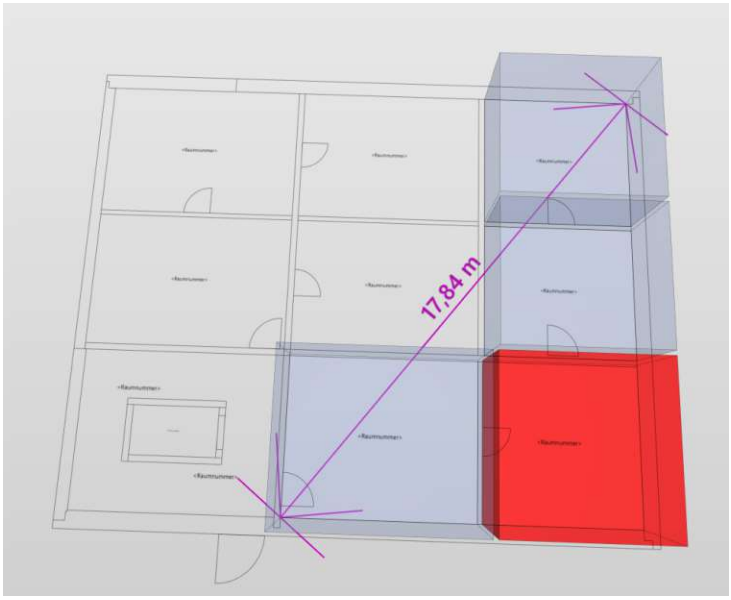


Abb. 3.16: Längsausdehnung eines Brandabschnitts

- Components of Section
- Visualization of Distance

One or more errors have been found

Sollte der Brandabschnitt nicht alle Anforderungen erfüllen, wird dieser unter der Kategorie „one or more errors have been found“ abgelegt. Neben den Basis-Ausgaben können in Abhängigkeit der Fehler weitere Ausgaben entstehen. Überschreitet die Fläche des Brandabschnitts die Bedingungen des Userinterfaces, wird die Ausgabe Area exceeds Limit erstellt. Das Gleiche gilt für die maximale Geschoßanzahl und der entsprechenden Ausgabe „More Floors than allowed“. Komponenten, welche die geforderten, brandabschnittsbildenden Eigenschaften nicht erfüllen, stellt der Algorithmus mit der Ausgabe „Element does not meet requirements“ dar. Für den Abschluss von Öffnungen (Türen/Fenster) gibt es weiters die Ausgaben „No Covering“ und

„not Selfclosing“. Sollten Räume unterschiedlicher Brandabschnitte direkt aneinander grenzen, erfolgt die Ausgabe mit „Spaces directly next to each Other“. Einen falsch platzierten Raum zeigt die API-Prüfregel durch die Ausgabe „Space is not within the Section“. Zusammengefasst können Ergebnisse dieser Kategorie folgende Ausgaben haben:

- allgemeine Angaben als Beschreibung
- Components of Section
- Distance exceeds Limit
- Area exceeds Limit
- More Floors than allowed
- Element does not meet requirements
- No Covering
- Not Selfclosing
- Spaces directly next to each Other
- Space is not within the Section

Contains excluded spaces

Beinhaltet ein ansonsten korrekt definierter Brandabschnitt ausgeschlossene Räume, z. B. durch direkt aneinandergrenzende Räume oder durch falsche Definition der Brandabschnittseigenschaft, wird dieser mit der Ausgabe „Contains excluded Spaces“ in die gleichnamige Ergebniskategorie eingeteilt. Da an dieser Stelle die Angabe der Basisangaben sowie der Längsausdehnung sinnbefreit ist, werden lediglich folgende Ausgaben erstellt:

- Components of Section
- Contains excluded Spaces

Wrong usage

Diese Ergebniskategorie ist optional und wird nur unter zwei Bedingungen aktiv: Erstens muss der Nutzer die optionale Überprüfung der Nutzung im Userinterface anwählen und zweitens müssen im Modell die angegeben Nutzungskategoriegrenzen überschritten sein. Treffen beide Bedingungen zu, wird der fehlerhafte Brandabschnitt in diese Ergebniskategorie gelegt. Ähnlich zur Kategorie „Contains excluded Spaces“ werden dabei lediglich folgende Ausgaben erstellt:

- Components of Section
- Contains excluded Spaces

Without section definition

Sollten geprüfte Räume die Eigenschaft zur Bestimmung des Brandabschnitts nicht besitzen, werden sie in dieser Kategorie abgelegt. Durch die separate Einteilung kann der Referent rasch fehlende Definitionen erkennen und diese mit den Planenden kommunizieren. Für Elemente in dieser Kategorie werden keine gesonderten Ausgaben erstellt, sondern direkt die Komponenten abgelegt.

Excluded spaces
Sollten sich unter den Elementen des defaultFilterparameters ausgeschlossene Räume befinden, werden diese gesondert in einer Kategorie abgelegt. Dies dient vor allem dem Zweck zur Überprüfung falscher Einstellungen im Userinterface, da ausgeschlossene Räume per Definition nicht unter den zu prüfenden Elementen sein sollten. Wie für Räume ohne Eigenschaft zur Definition des Brandabschnitts werden die Komponenten direkt ausgegeben.

3.4.4 Zusätzliche Prüfungen

Die API-Prüfregel deckt nicht das gesamte Spektrum der Anforderungen an brandabschnittsbildende Elemente ab. Deshalb sind zusätzliche, manuelle Prüfungen durch die Baubehörde notwendig. Solibri bietet die Möglichkeit, erstellte Ergebnisse einer Prüfregel manuell zu korrigieren. Folgende manuelle Prüfungen sind notwendig:

Räume mit erhöhter Brandgefahr: Sollte ein Planer fälschlicherweise einen Raum mit erhöter Brandgefahr anders kategorisieren und diesen mit anderen Räumen zu einem Brandabschnitt gruppieren, könnte der Fall entstehen, dass die API-Prüfregel den Brandabschnitt als korrekt markiert. Deswegen muss der Nutzer der API-Prüfregel die Räume auf ihre korrekte Nutzungskategorie überprüfen.

Elemente zu Stiegenhäusern: Die OIB-Richtlinie stellt an brandabschnittsbildende Elemente, welche an Stiegenhäuser grenzen die zum Flüchten dienen, strengere Anforderungen. Da dieser Fall in der aktuellen API-Prüfregel nicht berücksichtigt wird, muss der Nutzer mithilfe der grafische Ausgabe diese Elemente suchen und manuell überprüfen.

Schächte durch Brandabschnitte: Schächte können sich durch mehrere Brandabschnitte hindurch erstrecken. Gemäß OIB-Richtlinie müssen diese gesonderte Anforderungen hinsichtlich des Brandschutzes erfüllen. Diese Anforderung wird in der aktuellen API-Prüfregeln nicht geprüft und erfordert eine manuelle Prüfung.

Überprüfung der Vereinfachungen: Die Vereinfachungen gemäß Abschnitt 3.2 ergeben ein Ergebnis auf der sicheren Seite. Allerdings müssen fehlerhafte Brandabschnitte, welche die entsprechende Fehlerkategorie beinhalten, zusätzlich manuell geprüft werden.

Neben den manuellen Prüfungen ist zuvor die Gebäudeklasse zu bestimmen. Diese Prüfroutine ist nicht Teil dieser Arbeit. Zusätzlich ist die Modellqualität vor der Validerung der Brandabschnitte zu prüfen. Falls ein Planer einen Raum so konstruiert, dass dieser nicht durch Elemente begrenzt wird, werden die umliegenden Elemente auch nicht zur Prüfung herangezogen.

3.5 API-Prüfregel Generierung

Die Grundidee dieser API-Prüfregel war die Zusammenführung von Raumflächen basierend auf den Eigenschaften der Elemente rund um den Ausgangsraum. So können zum Beispiel Wohnungseinheiten über ihren Namen automatisch zusammengeführt werden, ohne dass eine weitere Eigenschaft vom Planer benötigt wird. Aufbauend auf dieser Grundfunktion wurde eine API-Prüfregel zur automatisierten Generierung von Brandabschnitten erstellt. Der Nutzer der API-Prüfregeln gibt die Anforderungen an einen Abschnitt und somit an Wände, Decken, Stützen, Balken und Beläge sowie Türen und Fenster vor. Eine Besonderheit der Regel stellt die Erweiterung zur Betrachtung von verschiedenen Sektionsanforderungen dar. Dabei können vom Nutzer sogenannte SectionRules erstellt werden, welche als eine Art Regelsystem betrachtet werden können. Im Gegensatz zur API-Prüfregel für die Validierung benötigt diese API-Prüfregel den Wert

im Merkmal Brandabschnitte nicht. Stattdessen werden die SectionRules durchlaufen, beginnend mit der ersten und so die Räume gesammelt. Anschließend erfolgt die Prüfung der Anforderungen. Sind diese nicht erfüllt, beginnt der Algorithmus von Neuem, allerdings mit der zweiten Regel. Dieser Ablauf wird so lange wiederholt, bis die generierte Fläche eine Regel erfüllt oder keine Regeln mehr zur Verfügung stehen. Mithilfe dieses Systems können die Flächen individuell gruppiert werden. Die Anforderungen pro Regel betreffen die Fläche, die Längsausdehnung und die Geschoßzahl. Um Brandabschnitte möglichst konform zur relevanten Rechtsmaterie zu generieren, kann ebenfalls eine Unterscheidung der Nutzungskategorie erfolgen. Nachfolgend ist ein Überblick über die erforderlichen Funktionen zur Generierung von Brandabschnitten gegeben. Weiters wird im Sinne der allgemeinen Anwendbarkeit der API-Regel auch die zusätzlichen Funktionen genannt.

- Notwendige Funktionen zur Bestimmung von Brandabschnitten
  - Auswahl der Räume (Räume)
  - Angabe und Überprüfung der Anforderungen an die abschnittsbildenden Elemente
  - Nutzungskategorienaufteilung ab einem gewissen Schwellenwert und anschließende Überprüfung. Ausgabe eines Fehlers bei fehlerhafter Gruppierung.
  - Automatische Erkennung, ob es sich bei dem aktuellen Brandabschnitt um einen ober- oder unterirdischen handelt
  - Angabe von unterschiedlichen Eigenschaften für ober- und unterirdische Brandabschnitte
  - Ermittlung der Kennwerte zur Brandabschnittsbildung für
    - Maximale Fläche
    - Längsausdehnung
    - Geschoßanzahl
- Zusätzliche Funktionen zur Bestimmung von Brandabschnitten
  - Räume können basierend auf ihren Eigenschaften ausgeschlossen werden. Sollten Eigenschaften von abschnittsbildenden Elemente die Eigenschaften nicht erfüllen und deswegen ein ausgeschlossener Raum mit gruppiert werden, erfolgt eine Fehlerausgabe.
  - Berücksichtigung der Nutzungskategorienaufteilung ist optional.
  - Flächenberechnungsmethode kann individuell angepasst werden.
  - Export der Ergebnisse ist auch in eine Excel-Datei möglich.
  - Es können sogenannte SectionRules erstellt werden. Mit diesen kann eine regelabhängige Prüfung von Abschnitten erfolgen, wobei die Unterteilung individuell ist.
- Weitere Funktionen für allgemeine Zwecke
  - Der Nutzer kann Anforderungen an benachbarte Räume stellen. So können zum Beispiel Wohnungseinheiten gruppiert werden.
  - Zur Überprüfung der Anforderungen kann ein beliebiger Textausschnitt der Merkmale verwendet werden.

3.5.1 Userinterface

Das Userinterface ist in Abb. 3.17 bis Abb. 3.22 dargestellt und wird nachfolgend genauer erklärt. Durch die umfassenden Eingabemöglichkeiten können Planer ihrere Brandabschnittsgenerierung individuell anpassen.

1. **Spaces:** Dieser Filterparameter stellt den DefaultFilterParamerter der API-Prüfregel dar.
   a) In diesem Komponentenfilter können nur Räume hinzugefügt werden. Da diese Regel generell Flächen gruppieren kann, besteht keine Notwendigkeit, bestimmte Räume auszuschließen. Allerdings werden im Kontext dieser Arbeit Verkehrsflächen erneut ausgeschlossen. Desweiteren bietet sich die Möglichkeit, direkt nur Räume mit Wohnnuztung einzustellen.

2. **General Input:** beinhaltet allgemeine Anforderungen an die Gruppierung sowie an die begrenzenden Elemente der Gebäudehülle. Die Berechnung der Fläche kann definiert werden sowie eine optionale Restriktion basierend auf der Nutzungskategorieunterscheidung erfolgen.
   a) Definitionen der Gebäudehülle: Da der Algorithmus stets Räume auf allen Seiten der Elemente sucht, ist für den korrekten Abschluss der Brandabschnittsbildung die Definition der Gebäudehülle notwendig. Dafür sind die Properties für die Gebäudehülle für Wände, Decken, Stützen, Balken und Beläge anzugeben.
      i. Eigenschaft für Wände, die die Gebäudehülle bilden
      ii. Eigenschaft für Decken, die die Gebäudehülle bilden
      iii. Eigenschaft für Stützen, die die Gebäudehülle bilden
      iv. Eigenschaft für Balken, die die Gebäudehülle bilden
      v. Eigenschaft für Beläge, die die Gebäudehülle bilden
   b) Einstellungen zur Berechnung der Fläche eines Brandabschnitts
      i. BooleanParameter: mit Innenwänden (Wände die innerhalb der Gesamtfläche liegen)
      ii. BooleanParameter: mit brandabschnittsbildenden Wänden
   c) optionale Einstellungen zu Nutzungskategorien: Werden diese berücksichtigt, erfolgt vor der Gruppierung eine Überprüfung der Flächen der eingetragenen Nutzungen. Werden diese überschritten, dürfen in einem Brandabschnitt nicht mehrere Nutzungskategorien auftreten.
      i. Booleanparameter: definiert, ob Nutzungskategorien berücksichtigt beziehungsweise unterschieden werden sollen
      ii. TableParameter: Bezeichnung der Nutzungskategorie – sofern die optionale Prüfung gewünscht ist, muss eine Bestimmung für „andere“ Nutzungskategorien angeben sein, ansonsten wird eine Fehlermeldung zurückgegeben.
      iii. TableParameter: maximale Fläche der einzelnen Nutzungskategorie in m^2
      iv. Zur Bestimmung der Nutzungskategorie ist die Angabe eines Merkmals notwendig, welches eine eindeutige Unterscheidung ermöglicht. Im Projekt BRISE-Vienna kann dafür das Merkmal WidmungBehoerde aus dem PropertySet Pset_SpaceSpecific herangezogen werden.

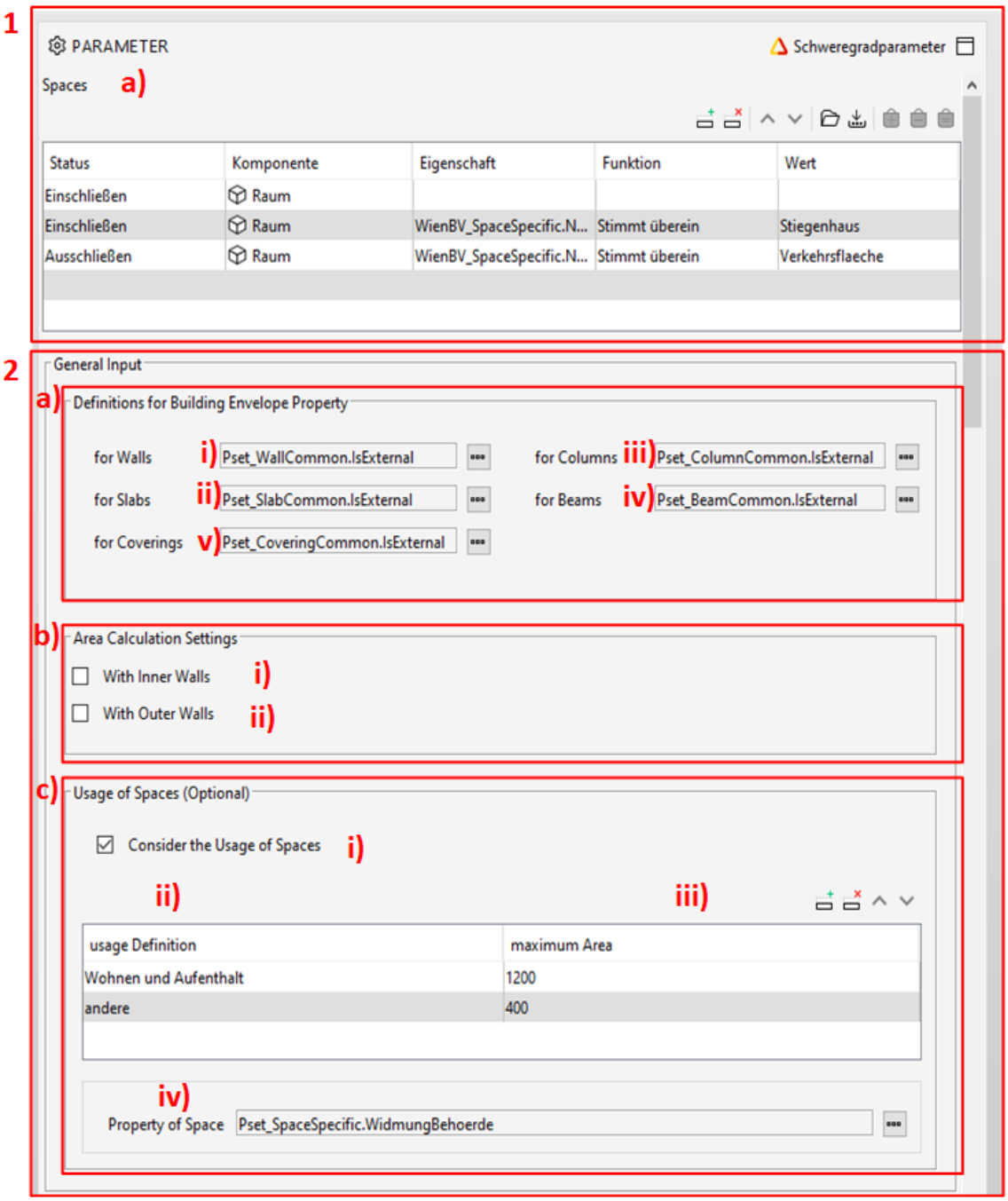


Abb. 3.17: Add Up Area Rule – Userinterface – Teil 1/6

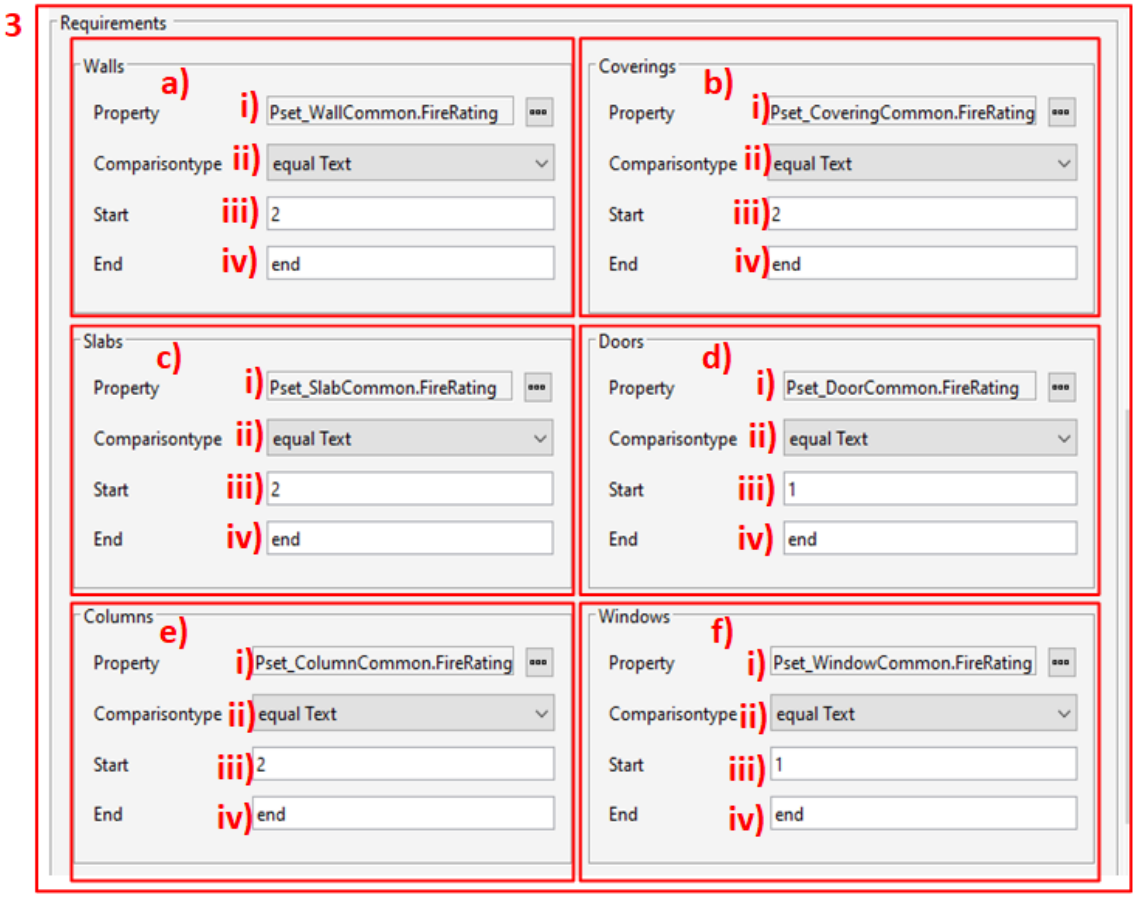


Abb. 3.18: Add Up Area Rule – Userinterface – Teil 2/6

3. **Requirements:** Beinhaltet allgemeine Vergleichswerte zur Bestimmung von zusammengehörigen Flächen. Die geforderten Bestimmungen unterteilen sich dabei auf die begrenzenden Elemente bzw. angrenzenden Räume. Dabei kann ein Vergleichstyp individuell für jedes Element gewählt werden. Weiters ist es möglich, nur Textausschnitte eines gewählten Merkmals heranzuziehen. Gemäß den Projektanforderungen BRISE-Vienna benötigt es zur Zeit für begrenzende Elemente nur den Vergleichstyp „equal Text“. Im Sinne der allgemeinen Anwendbarkeit wurde trotzdem bereits ein Drop-Down-Menü angelegt, um weitere Typen schnell und einfach hinzufügen zu können. Für die Vergleichswerte steht ein TableParameter zur Verfügung. In diesen können die Bestimmungen für die Brandabschnittsbildung stufenweise eingetragen werden. Dabei wird ein bestimmtes Prinzip verfolgt: Für einzelne Elemente, z. B. Wände oder Räume, werden die Vergleichswerte ab der aktuellen Stufe bis zur letzten Zeile des Tableparameters als korrekt betrachtet, hingegen für die Anforderungen an die gesamte Gruppierung, z. B. Fläche, gilt immer der Wert der aktuellen Stufe. In Abschnitt 3.5.2 ist eine detaillierte Beschreibung der Funktionalität gegeben.
   a) Anforderung an Wände
      i. Angabe der Eigenschaft von Wänden
      ii. Vergleichstyp - Drop-Down-Menü: gleicher Text
      iii. Startposition in der Eigenschaft

iv. Endposition in der Eigenschaft (so kann nur ein Teil der eingetragenen Eigenschaft herangezogen werden; um bis zum Ende der Eigenschaft zu gelangen, kann end eingetragen werden)

b) Anforderung an Beläge
   i. Angabe der Eigenschaft von Belägen
   ii. Vergleichstyp – Drop-Down-Menü: gleicher Text
   iii. Startposition in der Eigenschaft
   iv. Endposition in der Eigenschaft (so kann nur ein Teil der eingetragenen Eigenschaft herangezogen werden; um bis zum Ende der Eigenschaft zu gelangen, kann end eingetragen werden)

c) Anforderung an Decken
   i. Angabe der Eigenschaft von Decken
   ii. Vergleichstyp – Drop-Down-Menü: gleicher Text
   iii. Startposition in der Eigenschaft
   iv. Endposition in der Eigenschaft (so kann nur ein Teil der eingetragenen Eigenschaft herangezogen werden; um bis zum Ende der Eigenschaft zu gelangen, kann end eingetragen werden)

d) Anforderung an Türen/Abdeckungen
   i. Angabe der Eigenschaft von Türen/Abdeckungen
   ii. Vergleichstyp – Drop-Down-Menü: gleicher Text
   iii. Startposition in der Eigenschaft
   iv. Endposition in der Eigenschaft (so kann nur ein Teil der eingetragenen Eigenschaft herangezogen werden; um bis zum Ende der Eigenschaft zu gelangen, kann end eingetragen werden)

e) Anforderung an Stützen
   i. Angabe der Eigenschaft von Stützen
   ii. Vergleichstyp – Drop-Down-Menü: gleicher Text
   iii. Startposition in der Eigenschaft
   iv. Endposition in der Eigenschaft (so kann nur ein Teil der eingetragenen Eigenschaft herangezogen werden; um bis zum Ende der Eigenschaft zu gelangen, kann end eingetragen werden)

f) Anforderung an Fenster
   i. Angabe der Eigenschaft von Fenster
   ii. Vergleichstyp – Drop-Down-Menü: gleicher Text
   iii. Startposition in der Eigenschaft
   iv. Endposition in der Eigenschaft (so kann nur ein Teil der eingetragenen Eigenschaft herangezogen werden; um bis zum Ende der Eigenschaft zu gelangen, kann end eingetragen werden)

g) Anforderung an Balken
   i. Angabe der Eigenschaft von Balken

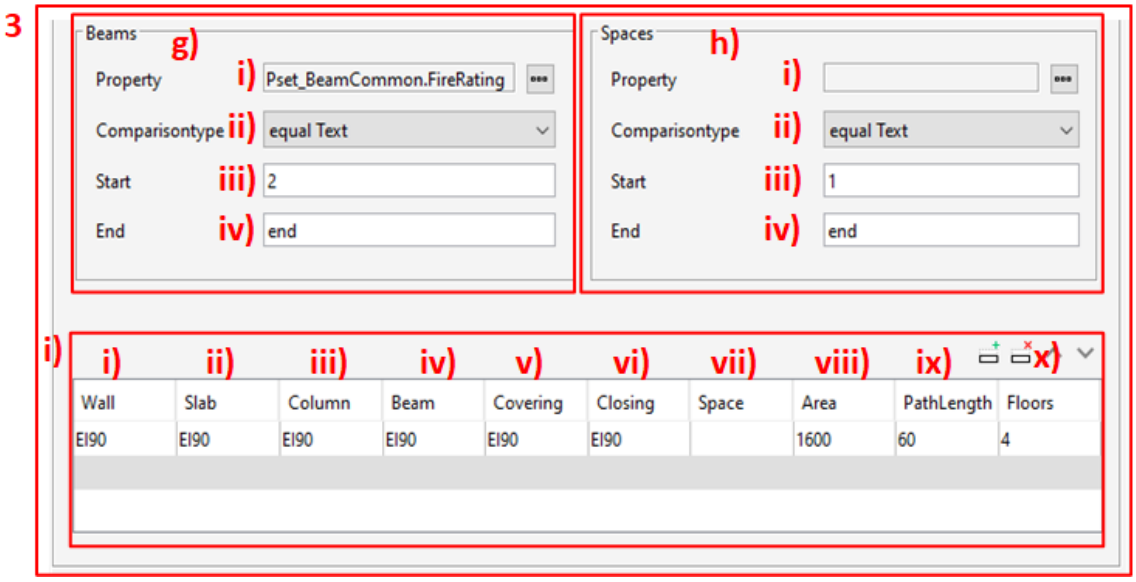


Abb. 3.19: Add Up Area Rule – Userinterface – Teil 3/6

ii. Vergleichstyp – Drop-Down-Menü: gleicher Text

iii. Startposition in der Eigenschaft

iv. Endposition in der Eigenschaft (so kann nur ein Teil der eingetragenen Eigenschaft herangezogen werden; um bis zum Ende der Eigenschaft zu gelangen, kann end eingetragen werden)

h) Anforderung an Räume

i. Angabe der Eigenschaft von Räumen

ii. Vergleichstyp: Für Räume existieren bereits zwei Wahlmöglichkeiten: gleicher Text oder stimmt mit anderen überein

iii. Startposition in der Eigenschaft

iv. Endposition in der Eigenschaft (so kann nur ein Teil der Eigenschaft herangezogen werden; mit „end“ wird der gesamte Text übernommen)

i) TableParameter – Anforderungsklassen: Eine Zeile des TableParameters steht für eine Anforderungsklasse.

i. Vergleichswert Wände

ii. Vergleichswert Decken

iii. Vergleichswert Stützen

iv. Vergleichswert Balken

v. Vergleichswert Belägen

vi. Vergleichswert Türen/Fenster/Abdeckungen

vii. Vergleichswert Räume

viii. Maximale Fläche je Anforderungsklasse in m^2

ix. Maximale Längsausdehnung je Anforderungsklasse in m

x. Maximale Geschossanzahl je Anforderungsklasse

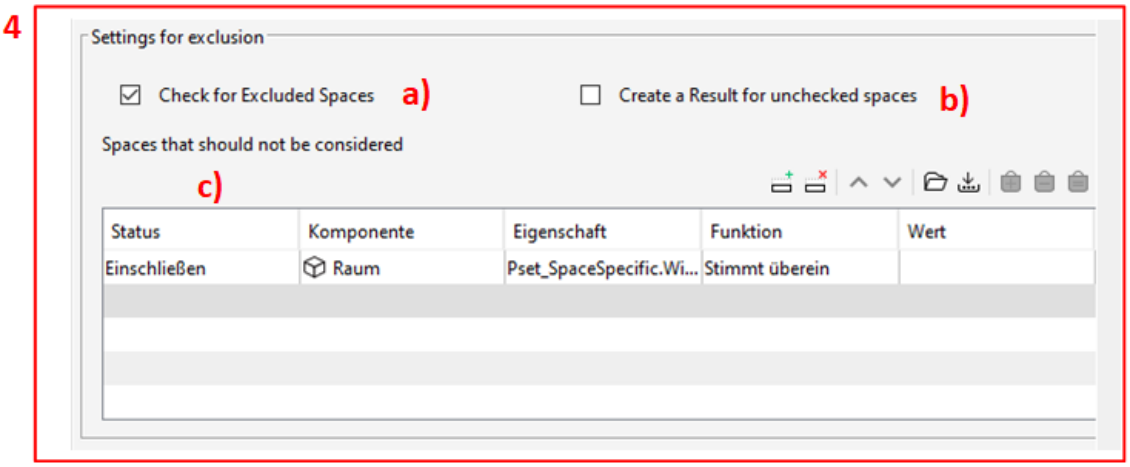


Abb. 3.20: Add Up Area Rule – Userinterface – Teil 4/6

4. **Settings for exclusion:** Hier können Räume und ganze Nutzungskategorien aus der Generierung ausgeschlossen werden. Der Unterschied zum Ausschluss im DefaultFilter-Parameter besteht darin, dass mithilfe dieser Einstellungsmehtode die Brandabschnitte auf die ausgeschlossenen Räume untersucht werden können. Damit sind diese eindeutig identifizierbar.
   a) BooleanParameter: Sollen die Abschnitte auf ausgeschlossene Räume überprüft werden?
   b) BooleanParameter: Soll eine eigene Ergebniskategorie für ausgeschlossene Räume erstellt werden?
   c) In diesem Filter definiert der Nutzer Räume, die nicht zu berücksichtigen sind. So können Brandabschnitte mit ausgeschlossenen Räumen identifiziert werden.

5. **(Optional) Weitere Bedingungen zur Anwendung für Brandabschnitte:** Da diese Regel grundlegend allgemeine Flächen gruppieren soll, ist eine Erweiterung zur Bestimmung von Brandabschnitten gemäß der relevanten Rechtsgrundlagen notwendig. Die notwendigen, weiteren Bedingungen sind in den nachfoldenden Userinterface-Abschnitten dargestellt.
   a) Unterschiedliche Anforderungen für Räume die unter- bzw. oberirdisch liegen. Gemäß OIB-Richtlinie sind die Anforderungen hinsichtlich der Lage der Brand- abschnitte zu unterscheiden. Mithilfe dieses Unterpunktes können die Anforderungen überprüft werden. Wird eine andere Prüfung mithilfe dieser Regel durchgeführt, in der die Anforderungen unabhängig der Lage sind, ist dieser Unterpunkt auszuschließen.
      i. BooleanParameter: Unterscheiden sich die Anforderungen bezüglich der Lage?
      ii. In diesem Filter müssen Objekte ausgewählt werden, die eine eindeutige Zuordnung der Räume hinsichtlich der Lage ermöglichen.
      iii. TableParameter – Anforderungsklassen: Vergleichswert Wände
      iv. TableParameter – Anforderungsklassen: Vergleichswert Decken
      v. TableParameter – Anforderungsklassen: Vergleichswert Stützen
      vi. TableParameter – Anforderungsklassen: Vergleichswert Balken
      vii. TableParameter – Anforderungsklassen: Vergleichswert Belägen

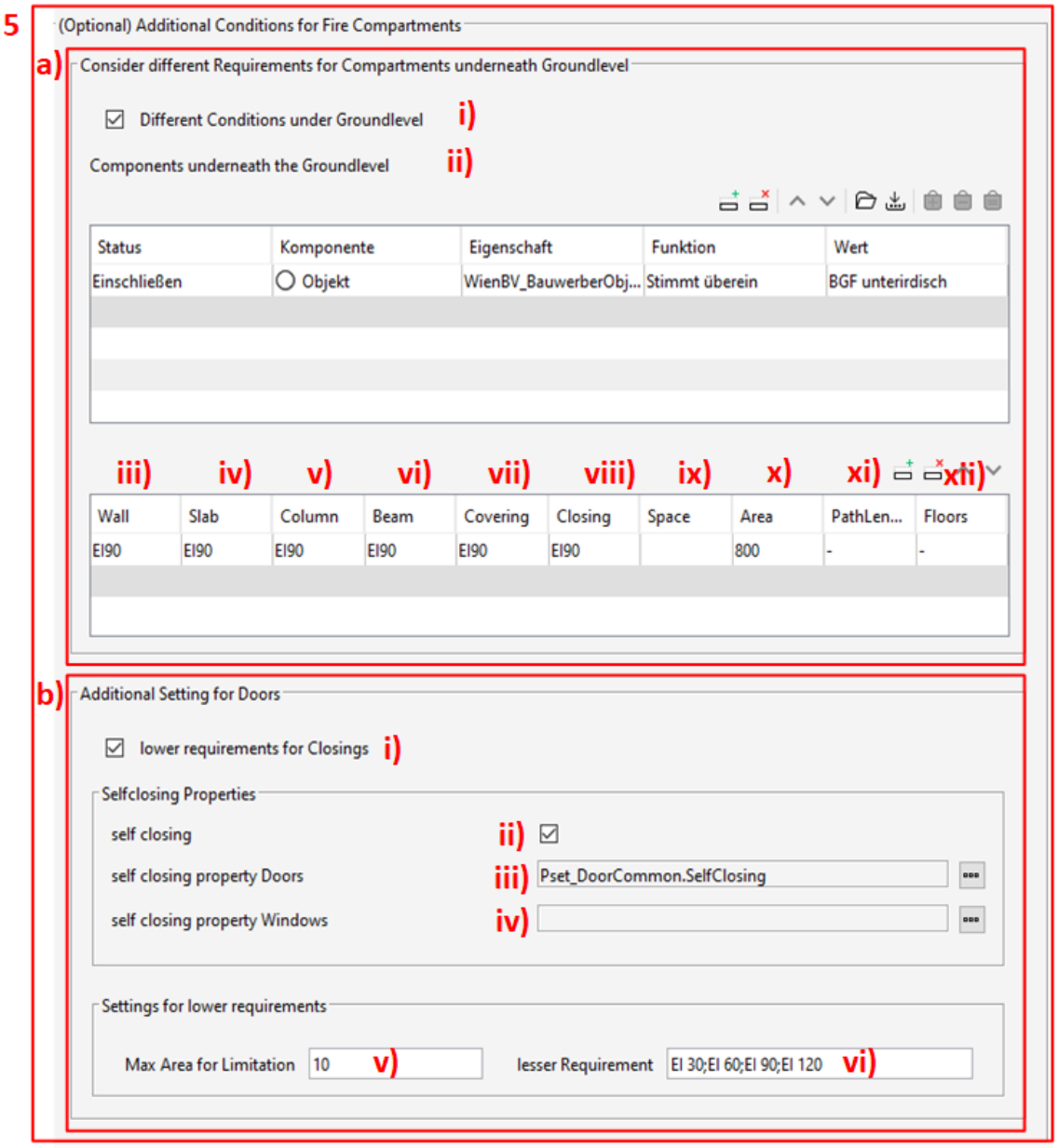


Abb. 3.21: Add Up Area Rule – Userinterface – Teil 5/6

viii. TableParameter – Anforderungsklassen: Vergleichswert Türen/Fenster/Abdeckungen

ix. TableParameter – Anforderungsklassen: Vergleichswert Räume

x. TableParameter – Anforderungsklassen: Maximale Fläche je Anforderungsklasse in m^2

xi. TableParameter – Anforderungsklassen: Maximale Längsausdehnung je Anforderungsklasse in m

xii. TableParameter – Anforderungsklassen: Maximale Geschossanzahl je Anforderungsklasse

b) Zusätzliche Anforderungen an Türen/Fenster/Abdeckungen: Abdeckungen von Öffnungen müssen spezielle Anforderungen für Brandabschnitte erfüllen, wobei in Abhängigkeit der Größe aller Öffnungen eines Elements andere Anforderungen gelten.

i. BooleanParameter: sollen die zusätzlichen Anforderungen für Türen berücksichtigt werden?

ii. BooleanParameter: soll überprüft werden, ob Türen/Abdeckungen/Fenster selbstschließend ausgeführt sind?

iii. Eigenschaft von Türen/Abdeckungen, ob diese selbstschließend sind

iv. Eigenschaft von Fenster, ob diese selbstschließend sind

v. Begrenzung der Summe der einzelnen Öffnungen je Wand und Decke für geringere Anforderungen

vi. zugelassene Vergleichswerte für niedrigere Anforderung – Vergleichswerte werden durch ";"getrennt – Beispiel: EI 90

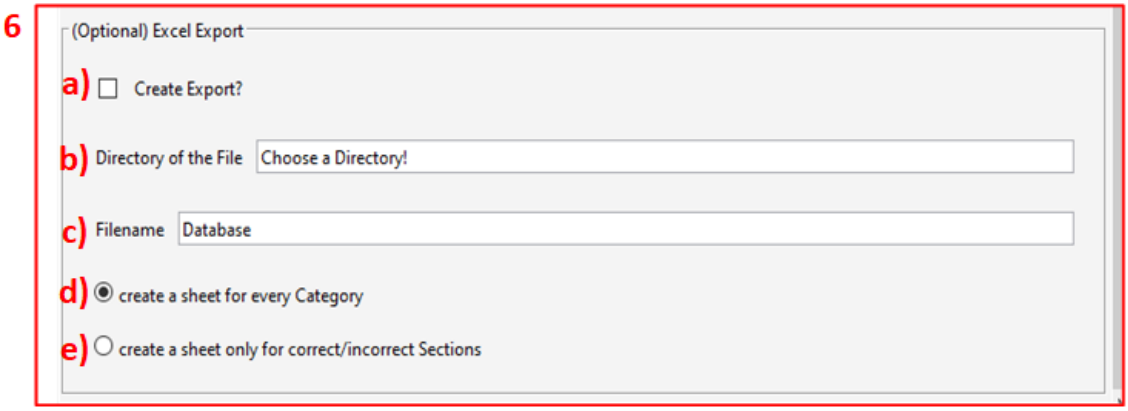


Abb. 3.22: Add Up Area Rule – Userinterface – Teil 6/6

6. **(Optional) Excel Export:** Gemäß den Anforderungen des Projekts BRISE-Vienna kann die API-Prüfregel die Ergebnisse in eine Excel-Datei exportieren.

a) BooleanParameter: der Excel-Export ist optional, weswegen der Nutzer hier bestätigen muss, ob dieser stattfinden soll.

b) Hier muss der Dateipfad angegeben werden. Um dem Nutzer die Eingabe zu erleichtern, wurde der Stringparameter so modifiziert, dass einfach nur der Dateipfad aus dem Windows-Explorer kopiert und eingefügt werden muss.

c) In diesem Eingabefeld soll der Dateiname eingegeben werden, wobei der Standardname auf „Database“ voreingestellt ist.

d) Soll ein Tabellenblatt für jede Ergebniskategorie

e) oder soll es nur für korrekte und nicht korrekte Abschnitte erstellt werden?

3.5.2 Funktionsweise

Die API-Prüfregel zur Generierung besteht aus vier Java-Klassen:

- AAR_Section

- AAR_SectionRule
- AddUpAreaRule
- AAR_Furtherfunction.

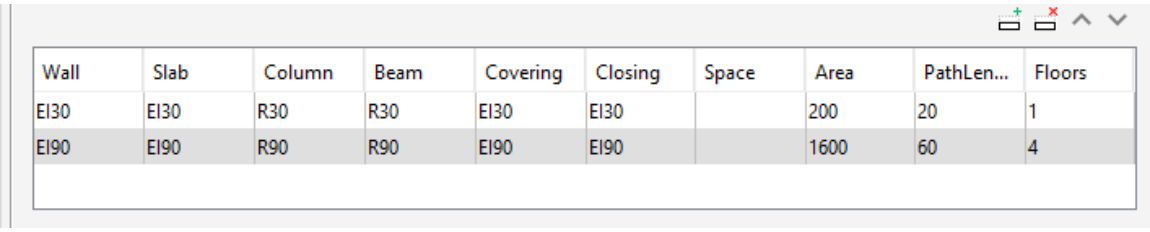
AAR_Section

Die Java-Klasse bildet die verschiedenen generierten Brandabschnitte ab. Dabei wird zu Beginn für jeden Startraum eine Instanz dieser Klasse erstellt. Durch den nachfolgenden Algorithmus werden gemäß dem Userinterface zugehörige Räume hinzugefügt. Es stehen Variablen zur Speicherung der Fläche, Längsausdehnung und Geschoßanzahl zur Verfügung. Weiters wird die Nutzungskategorie des ersten Raums sowie die Bestimmung der Lage des Abschnitts (ober- oder unterirdisch) abgespeichert. Die abschnittsbildenden Elemente sowie die Elemente, die zwischen Räumen liegen werden jeweils gesondert in einer Liste abgelegt. Des Weiteren sind Variablen zur Abspeicherung von Fehlern je Element bereitgestellt. Zur Befüllung der Variablen sowie zu Erstellung zusätzlicher Werte stehen neben den Getter- und Setter- folgende Methoden zur Verfügung:

- *generateComponentsOfSection()*
  Diese Methode stellt in Abhängigkeit der Nutzereinstellungen die Komponenten in einer Listenvariable zusammen. Ob Innenwände und brandabschnittsbildende Wände hinzugezählt werden, kann durch das Userinterface bestimmt werden.
- *calculateAreaOfSection()*
  Basierend auf dem Ergebnis der generateComponentsOfSection()-Methode bestimmt und befüllt diese Methode die Flächenvariable.
- *generateDescriptionMap()*
  Diese Methode erstellt die Variable zur Ergebnisbeschreibung eines Brandabschnitts. Die Variable umfasst die Fläche, die Anzahl der Räume, die Lage, die Längsausdehnung und die Fehler.
- *sectionDescriptionOutput()*
  Die Methode erzeugt aus der erstellten Variable der generateDescriptionMap()-Methode einen String. Dieser kann einfacher als Ergebnistext in Solibri angezeigt werden.
- *generateErrorMap()*
  Diese Methode erstellt die Variable zur Speicherung von Fehlern innerhalb einer generierten Fläche.

AAR_SectionRule

Diese Java-Klasse wurde zum Abspeichern der Anforderungen je SectionRule erstellt. Wie Abb. 3.23 zeigt, umfasst die Definition einer SectionRule minimal Anforderungen an folgende Elemente: Wände, Decken, Stützen, Balken, Beläge, Abdeckungen (Türen/Fenster) und Räume. Weiters können für die so gefundenen Brandabschnitte Anforderungen an die Fläche, die Längsausdehnung und die maximale Geschoßanzahl gestellt werden. Gibt es mehr als eine SectionRule wurde folgende Bestimmungen definiert: Die Anforderungen, die an einzelne Elemente gestellt werden, dürfen auch durch darunterliegende SectionRules erfüllt werden. So kann SectionRule 1 (erste Zeile in Abb. 3.23) auch durch eine Wand mit EI90 erfüllt werden, sofern die anderen Anforderungen eingehalten sind.



Wall	Slab	Column	Beam	Covering	Closing	Space	Area	PathLen...	Floors
EI30	EI30	R30	R30	EI30	EI30		200	20	1
EI90	EI90	R90	R90	EI90	EI90		1600	60	4

Abb. 3.23: Beispiel SectionRule

AddUpAreaRule und AAR_FutherFunctions

Diese Klassen führen die Generierung selbst durch. Die AddUpAreaRule umfasst die *peCheck()*-Methode, *check()*-Methode sowie *postCheck()*-Methode und definiert das Userinterface. Sie erbt von der OneByOne-Rule und stellt somit für diese Regel die Hauptklasse im Sinne der Solibri-API dar. In die AAR_FutherFunctions-Klasse sind wiederkehrende Methoden gelegt, um die Übersichtlichkeit der Hauptklasse zu verbessern. Die Generierung stellt einen komplexen Prüfprozess aus mehreren Subprozessen dar. Abb. 3.24 bildet den Prüfablauf grob ab und soll zum besseren Verständnis beitragen. Die Implementierung unterteilt sich gemäß Abschnitt 2.6.2 in den *preCheck()*, den *check()* und *postCheck()*.

preCheck

In dieser Methode erfolgt die Generierung der Brandabschnitte. Während der **Vorbereitung** werden die entsprechenden Räume aus dem defaultFilterParameter entnommen und für die spätere Bearbeitung in einer Variable abgelegt. Für die gemäß dem Userinterface ausgeschlossenen Räume sowie für die Komponenten zur Bestimmung der Lage wird dieser Prozess wiederholt. Anschließend wird die optionale Überprüfung der Nutzungskategorien durchgeführt. Diese Überprüfung ist ident zu jener der API-Prüfregel zur Validierung. Im Wesentlichen werden die Flächen je angegebener Nutzungskategorie aufsummiert und mit den eingegebenen Parametern im Userinterface verglichen. Bei Überschreitung des Schwellenwerts wird eine Booleanvariable auf wahr gesetzt, um für den späteren Prüfverlauf die Nutzungskategorie zu berücksichtigen. Im nächsten Schritt werden die **SectionRules** aus dem Tableparameter durch die *generateSectionRules()*-Methode erstellt und abgespeichert. Als Übergabeparameter benötigt die Methode den Tableparameter. Danach erfolgt mithilfe einer Schleife die Betrachtung zeilenweise, bis keine Zeile mehr vorhanden ist. Dabei wird für jede Zeile ein Objekt der Klasse AAR_SectionRule instanziiert und in einer Listenvariablen abgelegt. Dieser Prozess wird sowohl für die Anforderungen an oberirdische als auch für unterirdische Abschnitte durchgeführt und ist in Code 3.10 dargestellt.

Zur Abdeckung von Brandabschnitten mit dieser API-Prüfregel kann der Nutzer zusätzliche Anforderungen an Türen und Fenster gemäß der relevanten Rechtsmaterie stellen. Da diese unabhängig von den SectionRules sind, erfolgt die Abspeicherung in einer eigenen Variable. Auf Basis der erstellten Grundlagen beginnt die API-Prüfregel mit der **Flächengenerierung**. Abb. 3.25 bildet den Prozess mit all seinen Subprozessen ab. Die Generierung besteht aus drei verschachtelten Schleifen, wobei die erste Schleife eine for-Schleife darstellt, die für alle Räume des defaultFilterParameters durchlaufen wird. Der erste Schritt ist die Überprüfung, ob der aktuelle Raum bereits zu einem anderen Abschnitt gehört. Trifft diese Bedingung zu, überspringt der Algorithmus den Raum und zieht den nächsten heran. Wurde der Raum noch keinem Abschnitt hinzugefügt, wird einen neues Objekt der Klasse AAR_Section instanziiert und der Raum als Startraum für diesen Abschnitt festgelegt. Anschließend beginnt die zweite Schleife (While-Schleife) mit der Überprüfung für die weitere Prüfung. Diese besteht darin zu bestimmen, ob ein Ergebnis zur Verfügung steht. Erfüllt der Brandabschnitt bereits die Anforderungen der ersten

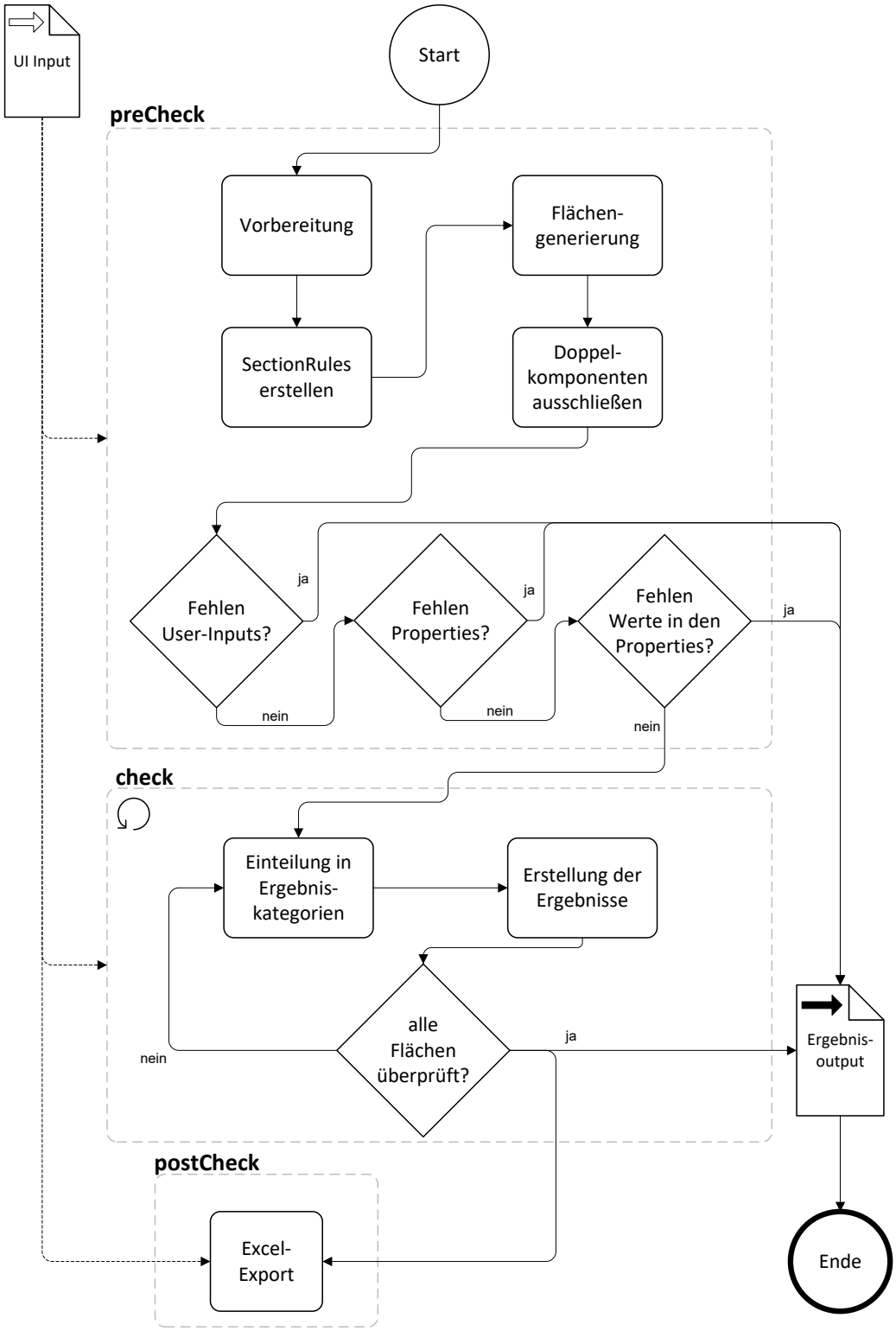


Abb. 3.24: Funktionsweise der Generierung – Gesamtüberblick

```
1  public static List<AAR_SectionRule> generateSectionRules(ParameterTable table) {
2
3    List<AAR_SectionRule> returnValue = new ArrayList<>();
4    int rowCount = table.getRowCount();
5
6    for (int i = 0; i < rowCount; i++) {
7      String name = "Row" + i;
8      // Zeilweise Anforderungen entnehmen
9      String wallRequirement = returnTableValue(table, i, 0);
10     String slabRequirement = returnTableValue(table, i, 1);
11     String columnRequirement = returnTableValue(table, i, 2);
12     String beamRequirement = returnTableValue(table, i, 3);
13     String coveringRequirement = returnTableValue(table, i, 4);
14     String doorRequirement = returnTableValue(table, i, 5);
15     String windowRequirement = returnTableValue(table, i, 5);
16     String spaceRequirement = returnTableValue(table, i, 6);
17     Double maxArea = returnTableValueDouble(table, i, 7);
18     Double pathLength = returnTableValueDouble(table, i, 8);
19     Double floors = returnTableValueDouble(table, i, 9);
20     // Neues Objekt erstellen
21     AAR_SectionRule sectionRule = new AAR_SectionRule(name, wallRequirement,
   slabRequirement, columnRequirement,
22     beamRequirement, coveringRequirement, doorRequirement, windowRequirement,
   spaceRequirement, maxArea,
23     pathLength, floors);
24     // In Klassenvariable speichern
25     returnValue.add(sectionRule);
26   }
27   return returnValue;
28 }
```

Code 3.10: Erstellung der SectionRules

SectionRule, gilt die Prüfung als bestanden. In allen anderen Fällen ist noch kein Ergebnis vorhanden. Somit erfolgt die zweite Prüfung der Schleife: Die Kontrolle der Anzahl der SectionRules. Diese Prüfung stellt das Hauptkriterium der While-Schleife dar. Um gruppierte Räume zu bewerten, stehen eine endliche Anzahl an SectionRules zur Verfügung. Stehen noch SectionRules zur Verfügung, werden alle Abschnittsvariablen zurückgesetzt und die dritte Schleife beginnt, welche ebenfalls eine while-Schleife ist. Diese wird so lange durchlaufen, bis entweder abschnittsbildende Elemente entsprechend der aktuellen SectionRule den gesamten Abschnitt umschließen oder bis keine Räume mehr vorhanden sind. Zu Beginn wird dabei die Listen-Variable des aktuellen Abschnitts auf einen ungeprüften Raum durchsucht. Findet der Algorithmus keinen, wird die Schleife abgebrochen und die Bedingungen der darüberliegenden Schleife geprüft. Liegt ein ungeprüfter Raum vor, erfolgt die Kontrolle der Lage. Dazu wird der aktuelle Raum mit den Elementen aus dem Userinterface, welche garantiert unterirdisch liegen, verschnitten. Ergibt sich ein Verschnittkörper, gilt der Abschnitt als unterirdisch und die Anforderungen werden entsprechend angepasst. Anschließend folgen die Prozesse: Begrenzende Elemente finden, Eigenschaften überprüfen, Öffnungen prüfen und Abdeckungen prüfen. Die Prozesse laufen fast ident zu den gleichnamigen Prozessen der Validierung ab. Der einzige Unterschied liegt darin, dass für den aktuellen Raum jedes Element als Außenelement des Abschnitts betrachtet wird und deswegen die *defineOuterComponentsOfSection()*-Methode nicht ausgeführt wird. Sollte ein Element die Anforderungen aus dem Userinterface nicht erfüllen, wird der angrenzende Raum der Liste des Abschnitts als ungeprüfter Raum hinzugefügt. Wurden alle abschnittsbildenden Elemente des aktuellen Raumes geprüft, erfolgt die Bestimmung der Anzahl der Räume. Dieser Prozess besteht aus einer einfachen If-Abfrage und überprüft, ob sich seit dem Start die Anzahl der Räume des aktuellen Abschnitts geändert hat. Hat sich die Anzahl geändert, beginnt die Schleife von vorne mit einem neuen ungeprüften Raum. Bleibt die Anzahl allerdings gleich, können die

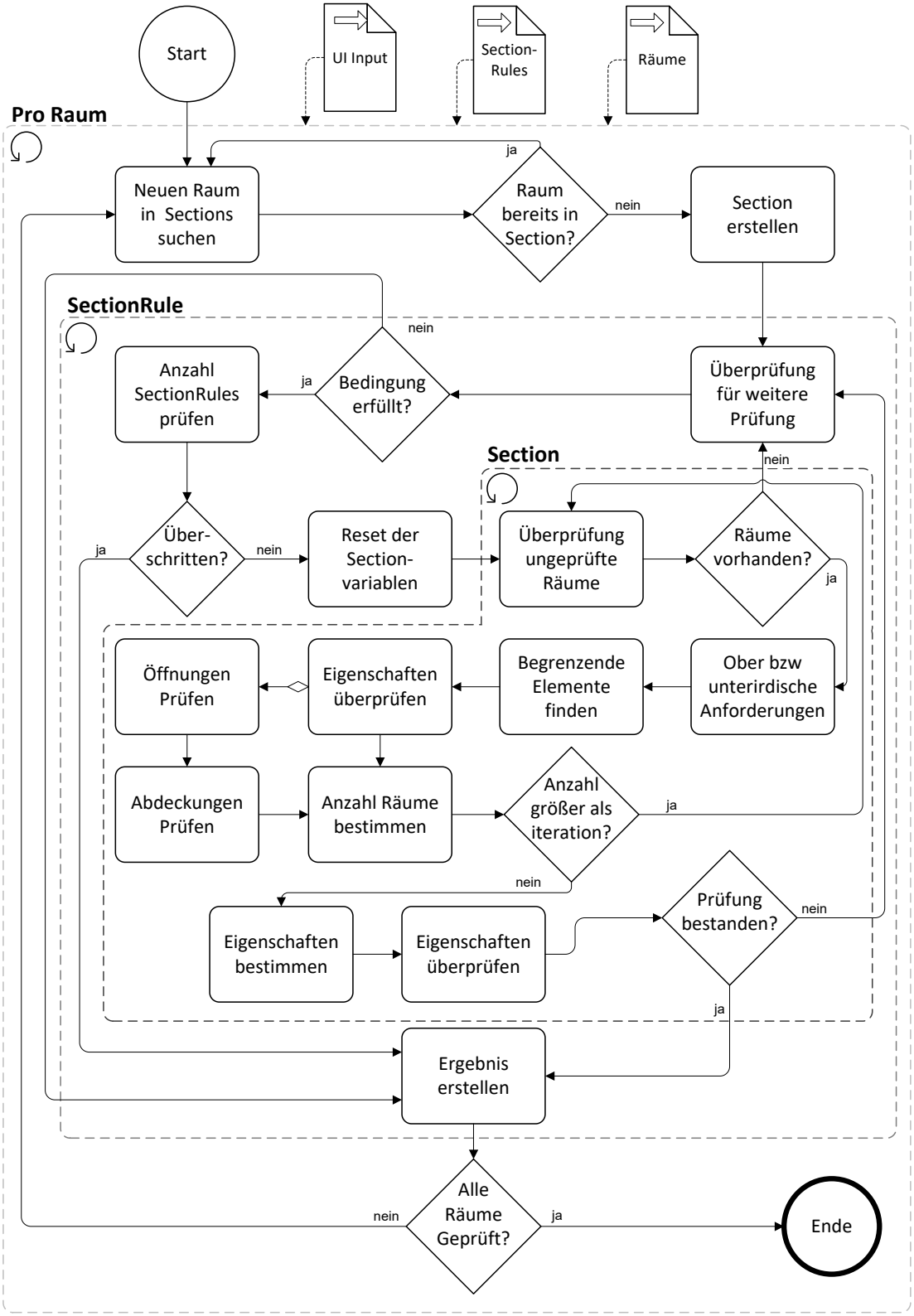


Abb. 3.25: Überblick über den Subprozess zur Flächengenerierung

Abschnittseigenschaften gesamt bestimmt werden. Die Ermittlung erfolgt ident zur API-Prüfregel zur Validierung und kann Code 3.7 und Code 3.8 entnommen werden. Zum Abschluss erfolgt die Prüfung der zuvor bestimmten Abschnittseigenschaften. Erfüllen diese die Bedingungen der aktuellen SectionRule werden, beide Schleifen abgebrochen und ein Ergebnis erstellt. Während der Ergebniserstellung werden auch alle Räume des aktuellen Abschnitts in einer Liste abgelegt. Diese wird benötigt, um effizient die erste Bedingung der ersten Schleife, also ob der Raum bereits einem Abschnitt angehört, zu überprüfen. Kann der Abschnitt die geforderten Anforderungen nicht einhalten, wird die dritte Schleife abgebrochen und die zweite Schleife beginnt von vorne. Sobald alle Räume des defaultFilterParameters behandelt wurden, ist die Flächengenerierung abgeschlossen und der Prozess **Doppelkomponenten auszuschließen** kann begonnen werden. Dafür wird die Liste mit den Abschnitten dupliziert und anschließend miteinander verglichen. Code 3.11 zeigt das Verfahren um Doppelkomponenten auszuschließen.

```
1  sortedComponentList.addAll(noDuplicates);
2  sortedComponentListGlobal.addAll(sortedComponentList);
3
4  Iterator<AAR_Section> it_AAR_Section = sortedComponentList.iterator();
5
6  // Im Fall, dass ein Raum zweimal vorkommt, werden die Sections zusammengefühhrt
7  int j = 0;
8  while (it_AAR_Section.hasNext()) {
9    try {
10     for (int i = j + 1; i < sortedComponentList.size(); i++) {
11       AAR_Section section1 = sortedComponentList.get(j);
12       AAR_Section section2 = sortedComponentList.get(i);
13
14       if (section1.getComponentsOfSection().containsAll(section2.
   getComponentsOfSection())) {
15         sortedComponentListGlobal.remove(section2);
16
17       } else if (section2.getComponentsOfSection()
18       .containsAll(section1.getComponentsOfSection())) {
19         sortedComponentListGlobal.remove(section1);
20       }
21     }
22     j++;
23   } catch (Exception e) {
24     break;
25   }
```

Code 3.11: Außschließen von Doppelkomponenten

Zur Steigerung der Effizienz der verschachtelten Schleifen werden nicht immer alle Abschnitte mit allen verglichen, sondern nur mehr übrige Kombinationen. Beispielhaft können drei Abschnitte A,B und C betrachtet werden. Anstatt der Kombinationen AB, BA, AC, CA, BC und CB erfolgt die Kontrolle von AB, AC und BC. Dieser Umstand wird in der For-Schleife durch i = j+1 beschrieben.

Treten während der Ausführung der *preCheck()*-Methode Fehler auf, die entweder auf eine fehlerhafte Nutzereingabe oder fehlende Merkmalangaben zurückzuführen sind, generiert die *preCheck()*-Methode ein preCheckResult. Dieses Ergebnis informiert den Nutzer über den festgestellten Umstand. Ein solcher Fall kann beispielsweise auftreten, wenn der Nutzer versucht, einen Text mit einem Merkmal zu vergleichen, das einen numerischen Wert besitzt.

Check

Die *check()*-Methode dient dazu, die zuvor bestimmten Abschnitte entsprechend ihrem Ergebnis in Ergebniskategorien einzuteilen. Dafür durchläuft die Methode die Räume des defaultFilterParameters. Um zu verhindern, dass für jeden Raum jedes Abschnitts ein Ergebnis erstellt wird,

liegen alle Elemente in einer gemeinsame Klassenvariable. Wie Abb. 3.24 zeigt, beginnt die API-Prüfregel bei der **Einteilung in Ergebniskategorien** damit zu bestimmen, ob der aktuelle Raum noch in der Klassenvariable liegt. Trifft dies nicht zu, wird das Element übersprungen. Ansonsten werden die gespeicherten Abschnitte auf den aktuellen Raum durchsucht. Sobald der korrekte Abschnitt gefunden wurde, werden alle zugehörigen Räume aus der Klassenvariable entfernt. Anschließend werden die Nutzungen entsprechend der optionalen Einstellung im Userinterface geprüft. Gleichzeitig erfolgt die Überprüfung, ob ausgeschlossene Räume im Abschnitt liegen. Werden beide Kriterien erfüllt, prüft der Algorithmus, ob Fehler vorhanden sind. Entsprechend werden die Abschnitte eingeteilt und anschließend ein **Ergebnis erstellt**. Diese beiden Schritte werden so lange wiederholt, bis alle Räume behandelt wurden. Anschließend werden die Ergebnisse gesammelt in einer Liste an Solibri übergeben.

postCheck

In der *postCheck()*-Methode besteht die Möglichkeit, optional einen Excel-Export der Ergebnisse durchzuführen. Die Ausgabe der fehlerhaften und/oder fehlerfreien Brandabschnitte erfolgt abhängig von den Einstellungen im Userinterface. Zur Erstellung der Excel-Datei wird die Java-Bibliothek Apache POI [2] Version 5.1 verwendet. Diese ermöglicht den Zugriff auf Elemente der Excel-Datei als Java-Klassen. Die *postCheck()*-Methode startet zunächst, indem die API-Prüfregel den Dateipfad aus den Angaben im Userinterface generiert. Danach wird eine neue Datei erstellt, die je nach Angaben im Userinterface Tabellenblätter für die verschiedenen Ergebniskategorien besitzt. Im nächsten Schritt wird das Tabellenblatt geöffnet und mit den Abschnitten für jede Ergebniskategorie beschrieben. Sobald alle relevanten Informationen eingetragen sind, erfolgt das Schließen der Datei und der Exportvorgang ist abgeschlossen.

3.5.3 Ergebnis in Solibri

Die Darstellung der Ergebnisse hängt davon ab, ob der Abschnitt die Anforderungen erfüllt oder nicht. Für jedes Ergebnis werden ähnlich zur Validierung allgemeine Angaben zu den Abschnitten als Beschreibung in Solibri hinzugefügt. Abb. 3.26 zeigt ein Ergebnis der API-Regel zur Generierung. Die Darstellung ist unabhängig der Ergebniskategorie für alle gleich, wobei die Längsausdehnung explizit nur für die Kategorien „Limits have been exceeded“ und „all Requirements are Fullfilled“ dargestellt wird.

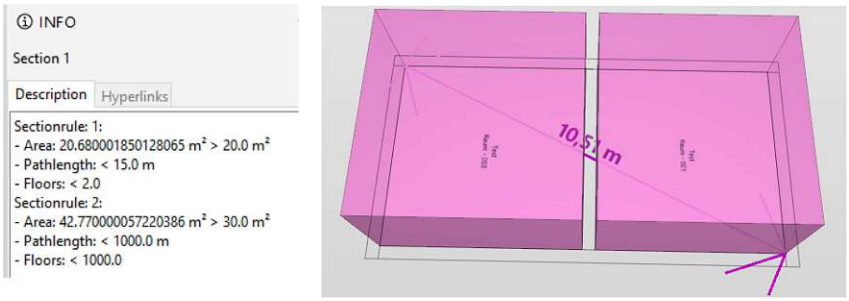


Abb. 3.26: Add Up Area Rule – Ergebniskategorie – nicht fehlerfrei

Limits have been exceeded

Sollte der Abschnitt nicht alle Anforderungen erfüllen, erfolgt die Eintelung in diese Ergebniskategorie. Für jeden Abschnitt werden die zugehörigen Komponenten und die Längsausdehnung des Brandabschnitts dargestellt. Dabei ist es unerheblich, welche der Anforderungen nicht erfüllt

wurden. Darüber hinaus wird in der Ergebnisbeschreibung in Solibri ein detailliertes Ergebnis zu den einzelnen Durchläufen der Anforderungsklassen aufgelistet.

Building envelope is missing the property
Sollte ein Abschnitt Elemente beinhalten, die zur Gebäudehülle gehören und diese die Eigenschaften nicht erfüllen, wird der Abschnitt in dieser Kategorie abgelegt.

Usages are mixed
Sollte der Nutzer im Userinterface die optionale Überprüfung von Nutzungskategorien aktivieren werden Abschnitte mit unterschiedlichen Nutzungskategorien in dieser Ergebniskategorie abgelegt.

Contains excluded spaces
Falls im Userinterface Räume aus der Generierung ausgeschlossen werden, erfolgt die Einteilung von Abschnitten, die ausgeschlossene Räume beinhalten, in diese Ergebniskategorie.

All requirements are fullfilled
Kann eine generierte Fläche alle Anforderungen einer SectionRule erfüllen, werden die zugehörigen Räume als Ergebnis in dieser Kategorie abgelegt. Neben der allgmeinen Beschreibung erfolgt auch die Darstellung der Längausdehnung.

Kapitel 4

Validierung der API-Prüfregeln

In diesem Kapitel erfolgt gemäß Abschnitt 2.4.4 die Validierung der API-Prüfregeln. Die Prüfung erfolgt in zwei Schritten. Zuerst wird an einem eigens für die API-Prüfregeln erstellten Modell die Umsetzung verifiziert. Dazu wird das Modell in zwei Varianten betrachtet. Variante 1 soll den Anforderungen gemäß relevanter Rechtsmaterie entsprechen. Hingegen sind in Variante 2 absichtlich Fehler eingebaut, um auch diesen Fall zu testen. Liefert das vereinfachte Modell korrekte Ergebnisse kann ein umfangreicheres Modell überprüft werden. Dafür wird ein Gebäudemodell herangezogen, welches im Rahmen vom Projekt BRISE-Vienna erstellt und getestet wurde.

4.1 Testmodell

Zur Überprüfung der Ergebnisse der API-Prüfregeln zur Validierung und Generierung wurde im Rahmen dieser Arbeit ein kleines Modell erstellt. Dieses beinhaltet alle Angaben hinsichtlich der AIA des Projekts BRISE-Vienna. Bezüglich der Gebäudeklasse werden Anforderungen gemäß Gebäudeklasse 2 gewählt. Abb. 4.1 zeigt eine 3D-Ansicht des Modells.

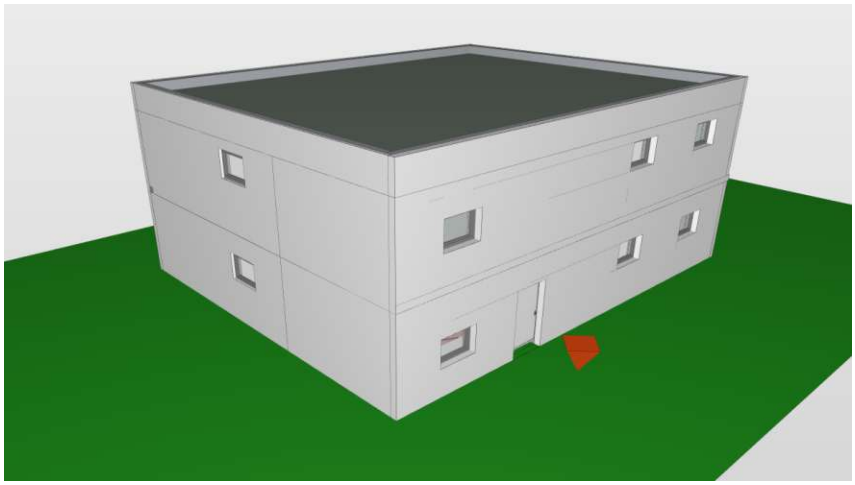


Abb. 4.1: 3D-Ansicht des Testmodells

4.1.1 Beschreibung

Das fiktive Bauwerk umfasst zwei oberirdische Geschoße sowie ein unterirdisches Geschoß, die über ein brandtechnisch entkoppeltes Stiegenhaus erreichbar sind. Die beiden oberirdischen Geschoße sind bezüglich der Fläche und der Aufteilung ident. Beide besitzen zwei Brandabschnitte, welche über vier Räume verfügen und der Nutzungskategorie Wohnen und Aufenthalt angehören. Das unterirdische Geschoß besteht aus einem großen Raum, in welchem sich mehrere Balken und Stützen befinden. Dieser soll eine Art Hobbyraum darstellen, weswegen der Raum ebenfalls

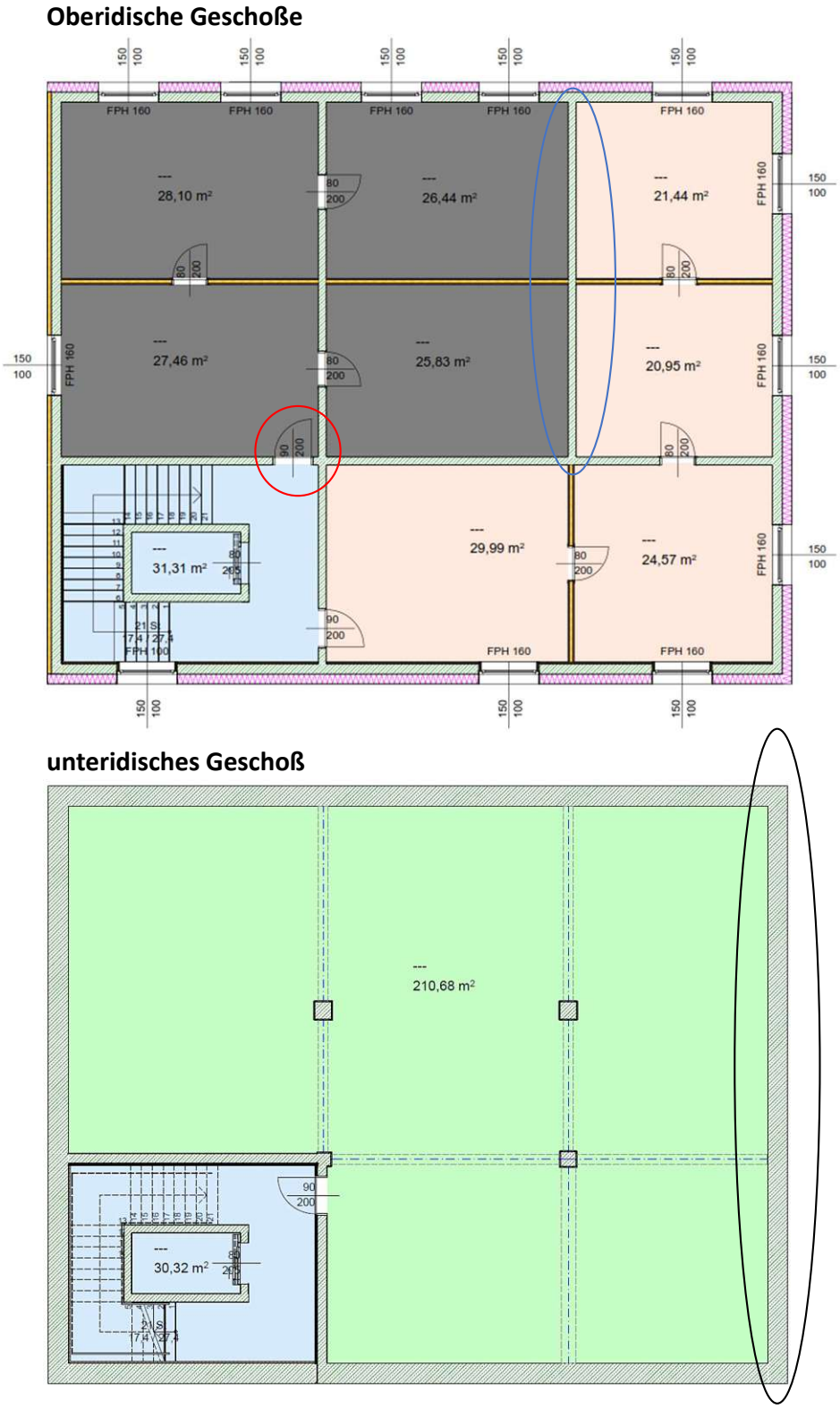


Abb. 4.2: Darstellung der Geschoße und ihrer Brandabschnitte

die Nutzungskategorie Wohnen und Aufenthalt besitzt. Um die Funktionen der API-Prüfregeln entsprechend zu testen, wird das Modell in zwei Varianten betrachtet:

- Variante 1: Alle Einstellungen entsprechen den geforderten Angaben der relevanten Rechtsmaterie. In dieser Konfiguration wird für die Validierung ein korrektes Ergebnis für alle Brandabschnitte erwartet. Das Ergebis der Generierung muss mit den geplanten Brandabschnitten übereinstimmem.
- Variante 2: Es werden gezielt fehlerhafte Elemente geplant. Für diese Modifikation des Modells wird erwartet, dass die Validierung die fehlerhaften Elemente findet. Die Generierung soll hingegen alternativ mögliche Brandabschnitte im Rahmen der Anforderungen finden.

Als Elementparameter werden für alle tragenden Elemente REI90 bzw. R90 angenommen, für alle nichttragenden Elemente EI90. Alle Türen, die in einer brandabschnittsbildenden Wand liegen, entsprechen der Widerstände EI30 und alle Fenster EI90. Abb. 4.2 zeigt die verschiedenen Geschoße sowie ihre Brandabschnitte. Weiters sind für Variante 2 die fehlerhaften Elemente markiert. In Rot wird das Merkmal der Tür aus dem Brandabschnitt im Erdgeschoß entfernt. In Blau wird die brandabschnittsbildende Wand im ersten Oberschoß mit einem zu geringen Brandwiderstand von REI30 versehen. Im Kellergeschoß ist in Schwarz eine Außenwand dargestellt, bei der das Merkmal zum Brandwiderstand entfernt wird.

4.1.2 Ergebnisse

Für Variante 1 ergeben sowohl die Validierung als auch die Generierung die erwarteten Ergebnisse. Alle Brandabschnitte sind anforderungskonform und werden auch so erkannt. Abb. 4.3 zeigt die Ergebnisse in Solibri sowie die Beschreibung und das grapische Ergebnis eines Brandabschnitts im Erdgeschoß. Für Variante 2 variieren die Ergebnisse stark von einander. Die API-Prüfregel zur Validierung erkennt die vorsetzlichen Fehler und ordnet diese entsprechend in ihre Ergebniskategorie ein. Die API-Prüfregel zur Generierung liefert neue mögliche Brandabschnitte, da die Längsausdehnungs- und Flächenbegrenzungen bei weitem nicht ausgereizt sind. Allerdings führt Fehler Rot in Abb. 4.2 dazu, dass die Genierung ausgeschlossene Räume zum Abschnitt hinzuzählt. Dieser Umstand wird ebenfalls erkannt und als Fehler ausgegeben. Für die fehlerhafte Wand im Kellergeschoß kann ebenfalls keine korrekte Gruppierung stattfinden, weshalb diese in die entsprechende Ergebniskategorie für fehlerhafte Elemente der Gebäudehülle eingeteilt wird. Die gesammelten Ergebnisse aus Solibri sind in Abb. 4.4 dargestellt

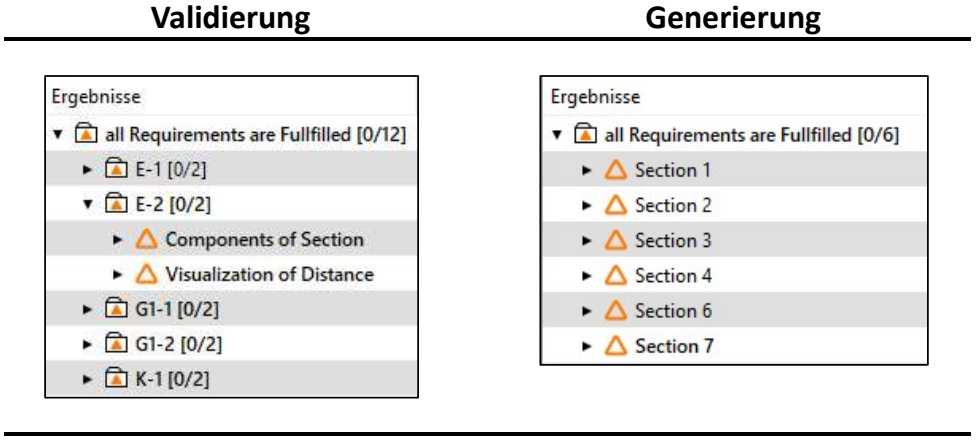


Abb. 4.3: Ergebnisdarstellung Variante 1

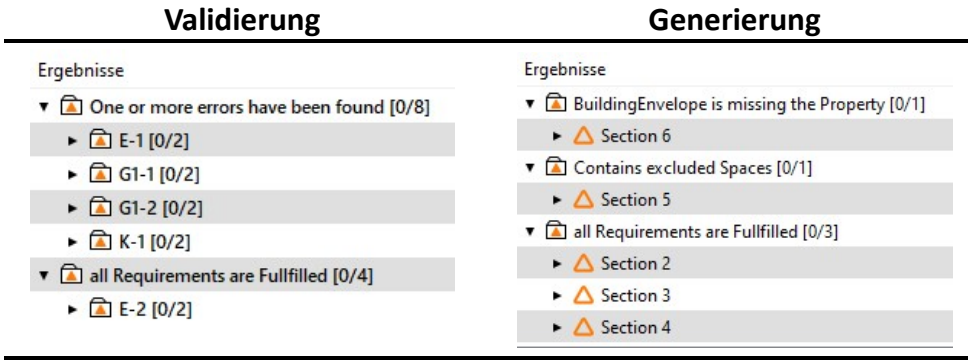


Abb. 4.4: Ergebnisdarstellung Variante 2

4.2 Studentenmodell im Rahmen von BRISE-Vienna

Um die Funktionsfähigkeit der beiden Regeln an einem realitätsnahem Bauwerk zu testen, wurde eines der Prüfmodelle des Projekts BRISE-Vienna herangezogen. Das Modell wurde mithilfe von Archicad (Version 25) von Gabriel Pelikan (TU Wien) erstellt und beinhaltet den vollständigen BRISE-Vienna-AIA. Es unterteilt sich in einen Wohn- und einen Gewerbebereich. Da sich die Anforderungen für den Bauteilwiderstand laut OIB-Richtlinie 2, Tabelle 1B [31] für verschiedene Gebäudeklassen nicht unterscheiden, können diese einfach übernommen werden. Abb. 4.5 zeigt eine 3D-Ansicht des Modells.

Abb. 4.5: 3D-Ansicht des Einreichmodells in BRISE-Vienna

4.2.1 Beschreibung

Das Bauwerk umfasst insgesamt neun Stockwerke, wobei das unterste Stockwerk unterirdisch liegt. Dieses besteht aus einer Garage sowie einem Technikraum, welcher die technischen Anlagen für das gesamte Gebäude beinhaltet. Im Erdgeschoss sind der Müllraum, Räumlichkeiten für Gewerbe und ein Fahrradabstellraum situiert. Der Zugang zum Treppenhaus ist für Bewohner und Gewerbe getrennt. Ab dem ersten Obergeschoss beginnen die Regelgeschosse. Diese beinhalten

jeweils eine barrierefreie Wohnung und vier weitere Wohnungen, welche alle samt Zugang zu einem Balkon, einer Loggia oder einer Terrasse haben.

4.2.2 Ergebnisse

Das Ergebnis der Validierung zeigt, dass das modellierte Gebäude nicht den Brandschutzbestimmungen gemäß OIB-Richtlinie entspricht. Bei der Überprüfung des Modells durch die API-Prüfregel zur Validierung wurden folgende Fehler erkannt:

- Falsche Raumzuordnung in den Brandabschnitten: Der Planer hat in vielen Brandabschnitten Schächte mit in den Branabschnitt aufgenommen. Da die Schächte Räume je Geschoß haben und diese jeweils zu anderen Brandabschnitten gehören, ergibt sich dieser Fehler. (Spaces directly Next to Each other) Weiters befinden sich Öffnungen in den Decken für besagte Schächte, weswegen eine zweite Fehlerkategorie (No Covering) als häufiger Fehler auftritt.
- Türen sind nicht selbstschließend: Viele der Türen haben zwar die bezeichnung EI2 30-C-Sm, aber in der zu überprüfenden Eigenschaft der Tür ist diese nicht auf wahr gestellt.

Da es sich um lediglich falsche Eigenschaftszuweisung handelt und alle Anforderungen hinsichtlich der Längsausdehnung, der Fläche und der Geschoßanzahl erfüllt sind, kann der Planer unter Rücksprache mit dem Referent die Fehler schnell korrigieren.

Die Generierung zeigt, dass die Anforderungen an den Brandschutz vom Planer selbst nicht ausgereizt wurden. Trotz der zuvor genannten Fehler konnten bis auf einen Aufenthaltsraum dennoch korrekte Brandabschnitte generiert werden. Der einzige nicht korrekte Abschnitt scheitert daran, dass eine nicht-selbstschließende Tür direkt in ein Stiegenhaus führt, weshalb ausgeschlossene Räume (Nutzungskategorie: Verkehrserschliessung und -sicherung) darin liegen. Diese alternative Lösung könnte der Planer nutzen und eventuell Aufbauten von den gewählten brandabschnittsbildenden Elementen wirtschaftlicher zu gestalten. In Abb. 4.6 ist ein Ausschnitt der Ergebnisse aus den beiden API-Prüfregeln zu sehen.

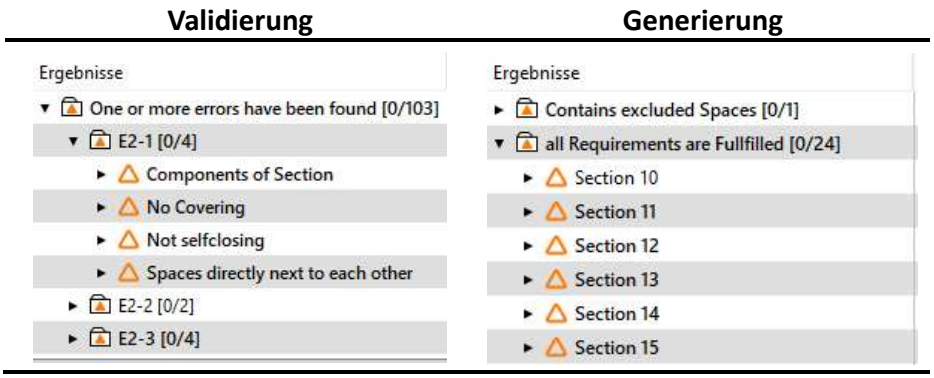


Abb. 4.6: Ergebnisdarstellung Einreichmodells in BRISE-Vienna

Kapitel 5

Diskussion und Ausblick

Immer mehr Planer und Ausführende benutzen BIM zur Abwicklung ihrer Projekte. Das dadurch gewonnene Potential soll auch für die Baubehörde transparentere und schnellere Verfahren ermöglichen. Durch das Projekt BRISE-Vienna wurde der Rahmen geschaffen, die Baueinreichung zu digitalisieren und zu beschleunigen. Durch die Automatisierung können Referent wiederkehrende Arbeiten an die Prüfsoftware abgeben und sich dafür gezielter auf Detailprüfungen konzentrieren. Komplexe bautechnische und baurechtliche Themen benötigen im aktuellen Verfahren ebenfalls zeitaufwendige Prüfungen und bieten somit besonders die Möglichkeit zur Effizienzsteigerung. Entsprechend groß ist der Entwicklungsaufwand. Kapitel 3 zeigt, dass die Umsetzung eines komplexen Themengebietes wie der Brandschutz von Wohngebäuden, entsprechenden Programmier- und Validierungsaufwand mit sich bringt. Zur Beantwortung der beiden gestellten Forschungsfragen bezüglich der Validierung von vordefinierten Brandabschnitten sowie der Generierung von Brandabschnitten werden die Ergebnisse aus Kapitel 4 herangezogen.

Gemäß Abb. 2.15 und Abb. 2.16 automatisiert die API-Prüfregel diverse Paragrafen der Bauordnung für Wien sowie Punkte der OIB-Richtline 2. Die Anwendung auf das Testmodell sowie das Studentenmodell zeigt das enorme Potential, die Behörde für wiederkehrende Überprüfungsaufgaben effizient zu unterstützen. Durch die unterschiedlichen Ergebniskategorien können Referenten schnell und einfach die Fehler nachvollziehen sowie die Prüfung stichprobenartig kontrollieren. Da die zusätzlichen, manuellen Prüfungen im Vergleich zum aktuellen Verfahren einen wesentlich geringeren Aufwand darstellen, kann die Prüfung zur Validierung als Erfolg verzeichnet werden. Allerdings ergibt sich aufgrund der Einschränkung auf Brandabschnitte mit der Nutzungskategorie Wohnen und Aufenthalt, dass Referenten Brandabschnitte von Garagen, Parkdecks und von Räumen mit erhöhter Brandgefahr weiterhin manuell prüfen müssen. Eine Erweiterung hinsichtlich dieser Gebäudeteile sollte in einem nachfolgenden Forschungsprojekt aufbauend auf dieser Arbeit umgesetzt werden.

Für den Planer stellt sich die API-Prüfregel zur Generierung von Brandabschnitten als nützliche Unterstützung dar – vor allem, weil diese vor der Einreichung beliebig oft wiederholt werden kann um verschiedene Varianten der Brandschnitte zu finden. Jedoch wird der Aufwand bei der Eingabe von Informationen nicht reduziert. Für fehlenden Angaben werden zwar mögliche Lösungen präsentiert, aber nur dann, wenn bereits entsprechende Merkmale in allen brandabschnittsbildenden Elementen eingetragen wurden. Somit muss der Planer im Vorhinein trotzdem bereits ein grundlegendes Konzept festlegen und die Merkmale für alle Elemente definieren. Die API-Prüfregel zu Generierung stellt vielmehr eine Überprüfung der Ansätze des Planers dar. Ein wesentlicher Vorteil liegt allerdings darin, dass die Räume nicht manuell zusammengefügt werden müssen, sollte der Planer mit den automatisch erstellen Brandabschnitten zufrieden sein. Aufbauend auf diesem Ergebnis können sich zukünftige Arbeiten mit der Generierung von Brandabschnittsmodellen beschäftigen.

Um die Planer aktiv zu entlasten, sollte aufbauend auf der Generierung zur vollständigen Simulation übergegangen werden. So soll unabhängig von den eingetragenen Merkmalen der

brandabschnittsbildenden Elementen automatisch verschiedene Varianten von Brandabschnitten erstellt und anschließend dem Planer die notwendigen Merkmale vorgeschlagen werden.

Diese und andere Arbeiten [25, 27] zeigen, dass auch komplexe Themen mithilfe der BIM-Methode zumindest teilautomatisch geprüft werden können. Damit kann die behördliche Prüfung nicht nur transparenter und effizienter werden, sondern auch zur Entlastung der Referenten beitragen sowie das Merkmal auf andere Details wie z. B. Umweltschutz oder Energieeffizienz zu legen.

Literatur

[1] J. Abualdenien, A. Borrmann und M. König. „Ausarbeitungsgrade von BIM-Modellen“. In: *Building Information Modeling: Technologische Grundlagen und industrielle Praxis*. Hrsg. von A. Borrmann, M. König, C. Koch und J. Beetz. VDI-Buch. Wiesbaden: Springer Fachmedien, 2021, S. 165–191.

[2] *Apache POI – the Java API for Microsoft Documents*. URL: `https://poi.apache.org/` (Zugriff am 17. 01. 2024).

[3] Austrian Standards International. *Building Information Modeling (BIM)*. Dez. 2023. URL: `https://www.austrian-standards.at/de/themengebiete/bau-immobilien/building-information-modeling/alles-zu-bim` (Zugriff am 22. 12. 2023).

[4] Austrian Standards International. *EN 17412-1: 2020 11 – Bauwerksinformationsmodellierung – Informationsbedarfstiefe – Teil 1: Konzepte und Grundsätze*. 2020.

[5] Austrian Standards International. *ISO 10303-21:2016: 2016 02 18 – Industrial automation systems and integration – Product data representation and exchange – Part 21: Implementation methods: Clear text encoding of the exchange structure*. 2016.

[6] Austrian Standards International. *ÖNORM B 1800 Beiblatt 1: 2014 01 01 – Ermittlung von Flächen und Rauminhalten von Bauwerken und zugehörigen Außenanlagen – Beiblatt 1: Anwendungsbeispiele*. 2014.

[7] Austrian Standards International. *ÖNORM B 1800: 2013 08 01 – Ermittlung von Flächen und Rauminhalten von Bauwerken und zugehörigen Außenanlagen*. 2013.

[8] Austrian Standards International. *ÖNORM EN 13501-1: 2020 01 15 – Klassifizierung von Bauprodukten und Bauarten zu ihrem Brandverhalten – Teil 1: Klassifizierung mit den Ergebnissen aus den Prüfungen zum Brandverhalten von Bauprodukten*. 2020.

[9] Austrian Standards International. *ÖNORM EN 13501-2: 2023 09 01 – Klassifizierung von Bauprodukten und Bauarten zu ihrem Brandverhalten – Teil 2: Klassifizierung mit Ergebnissen aus Feuerwiderstandsprüfungen und/oder Rauchschutzprüfungen, mit Ausnahme von Lüftungsanlagen*. 2023.

[10] Austrian Standards International. *ÖNORM EN ISO 16739-1: 2020 11 01 – Industry Foundation Classes (IFC) für den Datenaustausch in der Bauwirtschaft und im Anlagenmanagement – Teil 1: Datenschema (ISO 16739-1:2018, nur HTML-Format)*. 2020.

[11] Autodesk. *BIM-Vorteile | Gute Gründe für BIM | Autodesk*. URL: `https://www.autodesk.de/solutions/bim/benefits-of-bim` (Zugriff am 22. 12. 2023).

[12] BauNetz. *Software für 3D-Konstruktion und Modellierung | Integrales Planen | Software | Baunetz_Wissen*. URL: `https://www.baunetzwissen.de/integrales-planen/fachwissen/software/software-fuer-3d-konstruktion-und-modellierung-5289496` (Zugriff am 08. 01. 2024).

[13] BauNetz. *Software für die Modellanalyse und Koordination | Integrales Planen | Software | Baunetz_Wissen*. URL: `https://www.baunetzwissen.de/integrales-planen/fachwissen/software/software-fuer-die-modellanalyse-und-koordination-5289530` (Zugriff am 08. 01. 2024).

[14] BauNetz. *Was bedeutet LOIN? | Integrales Planen | Modellinhalte | Baunetz_Wissen*. URL: https://www.baunetzwissen.de/integrales-planen/fachwissen/modellinhalte/was-bedeutet-loin-8062492 (Zugriff am 27. 12. 2023).

[15] *Bauordnung für Wien – Landesrecht konsolidiert Wien, Fassung vom 30.12.2023*. URL: https://www.ris.bka.gv.at/GeltendeFassung.wxe?Abfrage=LrW&Gesetzesnummer=20000006 (Zugriff am 30. 12. 2023).

[16] BIMcollab. *Was ist BCF*. URL: https://www.bimcollab.com/de/resources/openbim/about-bcf/ (Zugriff am 08. 01. 2024).

[17] M. Bonacina. *Java Programmieren für Einsteiger*. 2. Aufl. Landshut, Deutschland: BMU Media GmbH, Juli 2018.

[18] A. Borrmann, M. König, C. Koch und J. Beetz, Hrsg. *Building Information Modeling: Technologische Grundlagen und industrielle Praxis*. VDI-Buch. Wiesbaden: Springer Fachmedien Wiesbaden, 2021. DOI: 10.1007/978-3-658-33361-4. (Zugriff am 17. 03. 2024).

[19] buildingSMART Austria. *BIM Regelwerk (AIA+BAP)*. URL: https://www.buildingsmart.co.at/muster-aia-und-muster-bap/ (Zugriff am 27. 12. 2023).

[20] buildingSMART Austria. *Über uns*. Jan. 2018. URL: https://www.buildingsmart.co.at/ueber-uns/ (Zugriff am 27. 12. 2023).

[21] buildingSMART International. *IFC Formats*. URL: https://technical.buildingsmart.org/standards/ifc/ifc-formats/ (Zugriff am 27. 12. 2023).

[22] buildingSMART International. *Industry Foundation Classes (IFC)*. URL: https://technical.buildingsmart.org/standards/ifc/ (Zugriff am 27. 12. 2023).

[23] buildingSMART International. *Who We Are*. Jan. 2019. URL: https://www.buildingsmart.org/about/who-we-are/ (Zugriff am 27. 12. 2023).

[24] C. C. Eichler, C. Schranz, T. Krischmann, H. Urban, M. Hopferwieser und S. Fischer. *BIMcert Handbuch: Grundlagenwissen openBIM*. Mironde-Verlag, Jan. 2024. DOI: 10.34726/5384. (Zugriff am 17. 03. 2024).

[25] S. Fischer, C. Schranz, H. Urban und D. Pfeiffer. „Automation of escape route analysis for BIM-based building code checking“. In: *Automation in Construction* 156 (Dez. 2023). DOI: 10.1016/j.autcon.2023.105092.

[26] G. Goger, M. Piskernik und H. Urban. *Studie: Potenziale der Digitalisierung im Bauwesen*. Hrsg. von WKO, BMVIT. 2018.

[27] M. Haselberger. „Realisierung von Fluchtwegsdaten im IFC-Format mithilfe von IfcOpenShell“. Diplomarbeit. Technische Universität Wien, 2023. URL: https://repositum.tuwien.at/handle/20.500.12708/190553 (Zugriff am 01. 02. 2024).

[28] A. Kolbitsch. *Tragwerksentwurf*. Vorlesungsskriptum, TU Wien, 2019.

[29] T. Krischmann, H. Urban und C. Schranz. „Entwicklung eines openBIM-Bewilligungsverfahrens“. In: *Bauingenieur* 95 (Sep. 2020), S. 335–344. DOI: 10.34726/2421.

[30] Nemetschek Group. *OPEN BIM | Erklärung & Projektbeispiele*. URL: https://www.nemetschek.com/de/topics/open-bim (Zugriff am 27. 12. 2023).

[31] *OIB-Richtlinie 2 | OIB*. URL: https://www.oib.or.at/de/oib-richtlinien/richtlinien/2023/oib-richtlinie-2 (Zugriff am 04. 02. 2024).

[32] *OIB-Richtlinien, Begriffsbestimmungen | OIB*. URL: https://www.oib.or.at/de/oib-richtlinien/richtlinien/2023/oib-richtlinien-begriffsbestimmungen (Zugriff am 30. 12. 2023).

[33] Oracle. *Java*. URL: https://www.java.com/de/ (Zugriff am 28. 12. 2023).

[34] Österreichisches Institut für Bautechnik. *Über uns*. URL: https://www.oib.or.at/de/ueber-uns (Zugriff am 30. 12. 2023).

[35] Plattform 4.0. *Begriffe zu BIM und Digitalisierung Terminology for BIM and Digitalisation*. Dez. 2017. URL: https://www.oiav.at/wp-content/uploads/2021/01/Plattform4.0_Schrift08.pdf (Zugriff am 27. 12. 2023).

[36] PORR AG. *BIM – Digitalisierung von Planung & Bau – PORR Group – PORR AG*. URL: https://porr-group.com/leistungen/design-engineering/bim-building-information-modeling/?gclid=CjOKCQjwn_OlBhDhARIsAG2y6zOD_-YmPDZL38zWgRN4iUCurGcZnUZUEHTpRVem1vcG3BC5OIqNIY0aAkYnEALw_wcB (Zugriff am 24. 07. 2023).

[37] R. Sacks, C. Eastman, G. Lee und P. Teicholz. *BIM Handbook: A Guide to Building Information Modeling for Owners, Designers, Engineers, Contractors, and Facility Managers*. en. 3. Aufl. John Wiley & Sons, Aug. 2018. ISBN: 978-1-119-28753-7.

[38] A. Sanchez, K. Hampson und S. Vaux. *Delivering Value with BIM: A Whole-of-Life Approach*. Routledge, London, 2016.

[39] C. Schranz, A. Gerger und H. Urban. „Augmented Reality im Bauwesen: Teil 1 – Anwendungs- und Anforderungsanalyse“. In: *Bauingenieur* 95.10 (2020), S. 379–388.

[40] C. Schranz, H. Urban und A. Gerger. „Potentials of Augmented Reality in a BIM based building submission process“. In: *Journal of Information Technology in Construction (ITcon)* 26.24 (Juli 2021), S. 441–457.

[41] M. Skolud. „Brandschutzrechtliche Rahmenbedingungen holzbasierter Gebäude in Österreich“. Diplomarbeit. Technische Universität Wien, 2021. DOI: 10.34726/hss.2022.96682.

[42] *Solibri API*. URL: https://solibri.github.io/Developer-Platform/ (Zugriff am 28. 12. 2023).

[43] *Solibri Office*. Nov. 2020. URL: https://www.solibri.com/de/solibri-office (Zugriff am 28. 12. 2023).

[44] Stadt Wien. *BRISE-Vienna – Smarte und innovative Verwaltung*. de-DE. URL: https://digitales.wien.gv.at/projekt/brisevienna/ (Zugriff am 21. 08. 2023).

[45] Strabag SE. *BIM 5D®*. URL: https://bim5d.strabag.com/ (Zugriff am 22. 12. 2023).

[46] H. Urban, C. Schranz, T. Krischmann, H. Asmera und B. Pinter. „Einsatz von openBIM und KI im Bewilligungsverfahren der Stadt Wien“. In: *OIAZ* 166 (Sep. 2021), S. 1–9.

[47] Vereinigung der österreichischen Richterinnen und Richter. *Stufenbau der Rechtsordnung*. URL: https://richtervereinigung.at/ueber-uns/ (Zugriff am 30. 12. 2023).

[48] J. K. Whyte und T. Hartmann. „How digitizing building information transforms the built environment“. In: *Building Research & Information* 45.6 (Aug. 2017), S. 591–595. DOI: 10.1080/09613218.2017.1324726.

[49] *Wiener Bautechnikverordnung 2020 – WBTV 2020 – Landesrecht konsolidiert Wien, Fassung vom 30.12.2023*. URL: https://www.ris.bka.gv.at/GeltendeFassung.wxe?Abfrage=LrW&Gesetzesnummer=20000593 (Zugriff am 30. 12. 2023).

[50] *Wiener Garagengesetz 2008 – Landesrecht konsolidiert Wien, Fassung vom 30.12.2023*. URL: https://www.ris.bka.gv.at/GeltendeFassung.wxe?Abfrage=LrW&Gesetzesnummer=20000052 (Zugriff am 30. 12. 2023).