Master's Thesis in der Vertiefungsrichtung Holzbau:

Computergestützte Tragwerksplanung im Holzbau

Lehrstuhl:

Technische Universität München
Lehrstuhl für Holzbau und Baukonstruktion
Univ.-Prof. Dr.-Ing. Stefan Winter
Univ.-Prof. Dr.-Ing. Mike Sieder

Betreuer:

Dipl.-Ing. Stefan Loebus
M.Sc. Fabian Ritter
Dipl.-Ing. (FH) Simon Daum

Verfasser:

Beatrice Kutterer

eingereicht am:

25. Juni 2014

Kurzfassung

Thema der Masterarbeit ist die computergestützte Tragwerksplanung innerhalb des Holzbaus. Im ersten Teil der Arbeit werden eine Marktübersicht und eine Bewertung der verfügbaren Bemessungsprogramme durchgeführt. Die verfügbaren Programme können einerseits in allgemeine Stabwerks- und FEM-Berechnungsprogramme mit zusätzlichem Holzbemessungsmodul und andererseits in speziell für den Holzbau entwickelte Programme unterteilt werden. Als Fazit der Untersuchung wird festgestellt, dass auch wenn der Umfang der verfügbaren Programme noch wesentlich geringer ist als für andere Bauarten, auch für den Holzbau schon einige brauchbare Programme zur Verfügung stehen. Diese Programme stecken jedoch teilweise noch in der Entwicklungsphase und weisen Einschränkungen auf. Aus diesem Grund wird in der vorliegenden Arbeit die Idee einer modularen, offenen Programmplattform vorgestellt. Eine zentrale Fragestellung hierbei ist, inwieweit ein Programm, das ständig von verschiedenen Entwicklern durch weitere Elemente ergänzt werden kann, eine Verbesserung zu vorhandenen kommerziellen Bemessungsprogrammen darstellt. Als Ergebnis des ersten Teils der Arbeit zeigt sich, dass das dabei entstehende multifunktionale Bemessungsprogramm eine gute, kostenfreie und vor allem universell anpassbare und anwendbare Alternative zu den vorhandenen Programmen darstellt.

Im zweiten Teil der Arbeit wird die Struktur eines solchen Programmes entworfen und beispielhaft durch eigene Programmierung umgesetzt. Hierzu wird vorerst die Nachweisführung nach dem Eurocode 5 bezüglich ihrer Systematik und Struktur genauer untersucht, da diese als Vorlage für die Programmentwicklung dient. Als Ergebnis der Untersuchung wird eine erste, objektorientierte Programmarchitektur entworfen. Auf Basis dieses Entwurfs werden die objektorientierte .NET-Sprache C# als Programmiersprache und als Programmieroberfläche Visual Studio von Microsoft für die Umsetzung des Programms ausgewählt.

Im nächsten Schritt wird anhand der beispielhaften Umsetzung der ersten Programmversion, eine prinzipielle Programmstruktur für ein multifunktionales Berechnungsprogramm entwickelt. Nach der Entwicklung des ersten Programms wurde deutlich, dass eine gute, grundlegende Bauteil- und Nachweisstruktur umgesetzt werden konnte, die Erweiterung jedoch nur umständlich und nicht ohne Zugriff auf den bestehenden Programmcode möglich ist. Um die Erweiterungsmöglichkeit zu optimieren, wurde ein zweites Programm entwickelt, welches das Managed Extensibility Framework, das MVVM-Pattern und WPF verwendet. Hier wird eine dynamische Erweiterung des Programms durch weitere Module ermöglicht und durch die Trennung von Logik und Darstellung innerhalb des Codes, zusätzlich eine klare und nachvollziehbare Struktur geschaffen.

Abstract

This Master's Thesis deals with the topic of computer-aided structural design of timber structures. Issue of the first section is a market review and an assessment of the available design programs. The disposable programs can be divided into frame- and FEM-calculation programs with an additional timber design module and programs, especially developed to design wooden structures. The result of the study shows, that even if the extent of the programs available is less than for other construction types, there are already some useful programs on hand. However, these programs are partly still in their development phase and show limitations. Out of this reason the idea of a modular and open program platform is presented. It is discussed in which way a program, which can be constantly extended by further elements, evolved by different developers, outlines an improvement to the present design programs. The conclusion of the first part of the thesis is, that the resulting multifunctional design program would be a good, free and most of all universally customizable and usable alternative to the available programs.

In the second part of this Master's Thesis the structure of such a program is designed and exemplarily implemented by own programming. Therefore the designing process according to the Eurocode 5 is examined, regarding its classification and structure. As a result of the examination a first, object-oriented program architecture is created. Based on this design, the object-oriented .Net-language C# and the programming environment Visual Studio by Microsoft are selected to implement the program.

Next a basically program structure for a multifunctional design program is developed by the exemplary implementation of the first version of the program. The outcome of the development of the first program is that a good, basic pattern for structural elements and verifications could be implemented. The extension by further elements however is inconvenient and not possible without accessing the existing code. For this reason a second program was developed using the Managed Extensibility Framework, the MVVM-pattern and WPF. Thus a dynamic extension of the second program by further modules is enabled and especially by the separation of logic and design in the code, a plain and comprehensible structure is created.

Inhaltsverzeichnis

1 Einleitung

In den letzten Jahrzehnten hat die computergestützte Planung im Bauingenieurwesen stetig an Bedeutung gewonnen. Sei es beim Design, in der Entwurfsplanung und Bemessung oder zur ganzheitlichen Lebenszyklusplanung, inzwischen ist die Verwendung von Software im Bauwesen nicht mehr wegzudenken. Im Rahmen dieser Masterarbeit wird die Bedeutung der computergestützten Tragwerksplanung im Bereich des Holzbaus thematisiert. Während in anderen Bauarten wie dem Stahl- oder Betonbau die Planungs- und Berechnungssoftware schon fest in den Entwurfs- und Bemessungsprozess integriert ist, ist die computergestützte Planung im Holzbau weniger präsent. Aus diesem Grund wird im ersten Teil dieser Arbeit eine Marktübersicht über die verfügbaren Bemessungsprogramme erstellt. Während der Untersuchung wird ein Vergleich der verschiedenen Programme durchgeführt und mögliche Schwächen werden aufgezeigt.

Den vorhandenen Programmen zur Holzbaubemessung wird die Idee einer modularen, offenen Programmplattform gegenüber gestellt. Eine solche Plattform kann durch einzelne Bauelemente von verschiedenen Entwicklern beliebig erweitert werden. Somit entsteht ein Berechnungsprogramm, das durch die Vielzahl an Entwicklern und den modularen Charakter multifunktional angewendet und ständig erweitert werden kann. In diesem Bereich wurden vom Lehrstuhl für Holzbau und Baukonstruktion bereits mehrere Untersuchungen anhand von Excel durchgeführt.

Im zweiten Teil der Arbeit wird ein solches Bemessungsprogramm entworfen und somit dessen Umsetzbarkeit untersucht. Zunächst wird hierzu das Anwendungsgebiet eines multifunktionalen Programms erörtert. Da sich das Programm an der Struktur des Eurocodes 5 orientieren soll, wird vorerst die Nachweisführung anhand dessen Systematik genauer betrachtet. Neben der Überprüfung der Datenbanktauglichkeit des Eurocodes, werden anhand der Eurocodestruktur erste Entwurfsideen für eine Programmarchitektur entwickelt. Nach der Auswahl einer passenden Programmiersprache und Entwicklungsumgebung wird der Entwurf beispielhaft durch eigene Programmierung umgesetzt. Im Mittelpunkt der Programmentwicklung stehen neben dem modularen Charakter eine benutzerfreundliche Oberfläche und eine nachvollziehbare und klare Struktur zur Sicherstellung der Erweiterbarkeit durch andere Anwender.

Das Ziel der Arbeit ist neben der bewertenden Marktübersicht die Machbarkeit und Anwendbarkeit eines modularen und multifunktionalen Programmes für den Holzbau, anhand der eigenen Programmierung zu untersuchen.

2 Marktübersicht und Bewertung verfügbarer Bemessungsprogramme

Der erste Teil der Arbeit umfasst eine Übersicht und Bewertung zu den bestehenden Berechnungs- und Bemessungsprogrammen. Hierzu wurde eine ausführliche Recherche zu den verfügbaren, kommerziellen Programmen durchgeführt. Es zeigte sich, dass bereits einige Programme auf dem Markt sind, die sich auch für die Bemessung im Holzbau eignen. In welchem Umfang diese tatsächlich angewendet werden können, zeigte sich bei der genaueren Untersuchung. Da die Berechnungsprogramme sehr teuer sind, wurden meist kostenlose Testversionen zur Untersuchung herangezogen. Während einige Hersteller wie zum Beispiel Dlubal und Nemetschek kostenlose Studentenlizenzen für ihre Programme zur Verfügung stellen, bietet Bentley keinen Zugriff auf sein Programm STAAD.Pro, ohne dass sich die jeweilige Hochschule für einen Zugang zu deren STUDENTserver registriert. Zur repräsentativen Untersuchung der Programme wurde jeweils dasselbe Beispiel, ein Brettschichtholzträger als Einfeldträger, bemessen. Im Folgenden wird kurz auf die einzelnen Programme der verschiedenen Firmen eingegangen, um diese anschließend auf Basis der genauen Untersuchungen zu vergleichen und zu bewerten.

2.1 Nemetschek Frilo

Nemetschek bietet mit seinem Berechnungsprogramm Frilo eine Möglichkeit einfach und schnell Holztragwerke und Dächer sowie deren Verbindungen zu bemessen. Nemetschek Frilo ist in mehrere, nach Themen geordnete Programmmodule aufgeteilt. Der Nutzer kann unter verschiedenen Bauarten und anschließend aus vordefinierten Tragwerkstypen wählen. Neben den Modulen „Holzbau" und „Hausdächer" stehen für eine Bemessung im Holzbau auch die Tools „Allgemein" und „Stabwerke" zur Verfügung, bei denen Holz als Material gewählt werden kann.

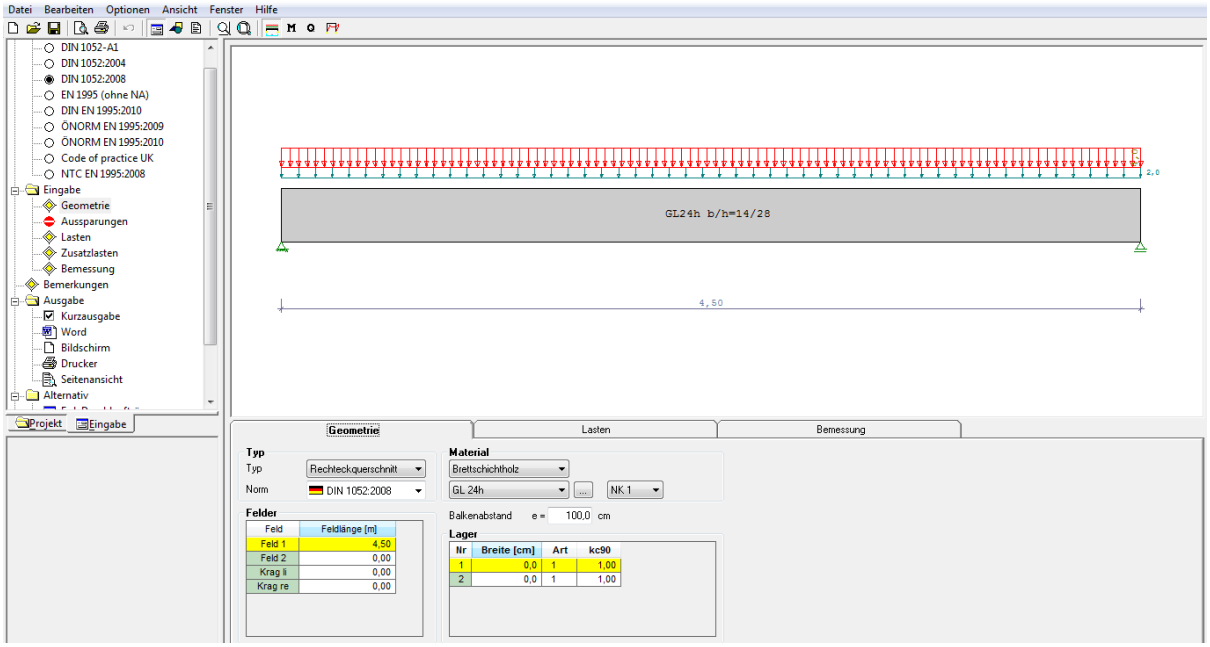


Abbildung 1: Eingabemaske in Frilo

Die Eingabe erfolgt Schritt für Schritt in einem zentralen Eingabefenster (siehe Abbildung 1). Frilo stellt eine sehr übersichtliche und benutzerfreundliche, aber dennoch umfassende Anwendung für die

Bemessung im Holzbau dar. Das Eingabefenster ist übersichtlich aufgebaut und leitet mit Hilfe eines Baummenüs nahezu automatisch durch die Berechnung. Auch wenn das Tragwerk und die Belastung für das Beispiel lediglich in einer zweidimensionalen Grafik dargestellt werden, ist die Darstellung klar und anschaulich. Als Kritik bleibt zu erwähnen, dass einige Kenngrößen nicht näher benannt oder beschrieben, oder wie in anderen Programmen durch ein Beispiel veranschaulicht werden. Es wird vom Nutzer erwartet, dass er entweder mit der Materie vertraut ist, oder mit den vorhandenen Handbüchern arbeitet. Neben den genannten Vorteilen ist auch die Kompatibilität zu anderen Modulen und Programmen ein sehr positiver Aspekt des Programms. So werden dem Nutzer im Zweig „Lastweiterleitung" mögliche Schnittstellen zur Weiterleitung der ermittelten Lasten zur Verfügung gestellt. Neben der Übernahme von Systemen und Ergebnissen in andere Module, hat Frilo auch eine ASCII-Schnittstelle zu den Stabwerks- und Dachmodulen. Zudem können Daten in den folgenden Formaten beziehungsweise von den folgenden Programmen importiert und exportiert werden:

- GLASER –isb cad-
- ALLPLAN
- Step dtH (nur Import)
- DXF
- DTH
- DSTV
- EMF
- RTF (Word)(nur Export)
- XML (nur Export)
- Enhanced Metafile (nur Export)

2.2 Nemetschek Scia Engineer

Das Modellierungs- und Berechnungsprogramm Scia Engineer ist ebenfalls von der Firma Nemetschek und ermöglicht im Gegensatz zu Frilo die Bemessung selbst modellierter Systeme aus unterschiedlichen Materialien, darunter auch Holz. Die Bemessung erfolgt nach Eurocode 5 im Grenzzustand der Tragfähigkeit und der Gebrauchstauglichkeit. Scia Engineer stellt mit seinen zahlreichen Schnittstellen zu CAD- und CAM-Programmen, sowie zu anderen Stabwerks- und Statikprogrammen und seiner hohen Datenkompatibilität ein sehr umfangreiches und leistungsfähiges Modellierungs-, Berechnungs- und Planungstool dar. Dank dem zugehörigen Modul für Nachweise gemäß Eurocode 5 kann es auch als umfangreiches Bemessungsprogramm für anspruchsvolle Holztragwerke verwendet werden. Das Programm bietet zahlreiche Bibliotheken für Materialien, Querschnitte und Lasten, in denen nicht nur aus einer großen Menge an vorhandenen Einträgen gewählt werden, sondern auch eigene, benutzerdefinierte Varianten erstellt werden können. Des Weiteren bietet es in allen Bereichen eine sehr vielfältige und spezifische Auswahl. Dadurch erscheint Scia Engineer in der Anwendung für einfache Systeme schnell nicht mehr intuitiv und sehr komplex. In diesem Fall ist Frilo sicherlich vorzuziehen, während Scia Engineer sich mehr für die Eingabe und Bemessung von anspruchsvollen 2D- und 3D-Systemen eignet.

2.3 RX-HOLZ

RX-HOLZ ist ein Statik-Berechnungstool der Firma Dlubal für den Bereich Holzbau, das Module für verschiedene Tragsysteme zur Verfügung stellt. Mit diesem Programm können Nachweise der

Tragfähigkeit, der Gebrauchstauglichkeit sowie des Brandschutzes nach DIN 1052 und Eurocode 5 durchgeführt werden. Die RX-HOLZ-Programmmodule von Dlubal bilden sowohl in der Systemeingabe als auch in der Ergebnisauswertung ein sehr umfangreiches und trotzdem benutzerfreundliches Programm.

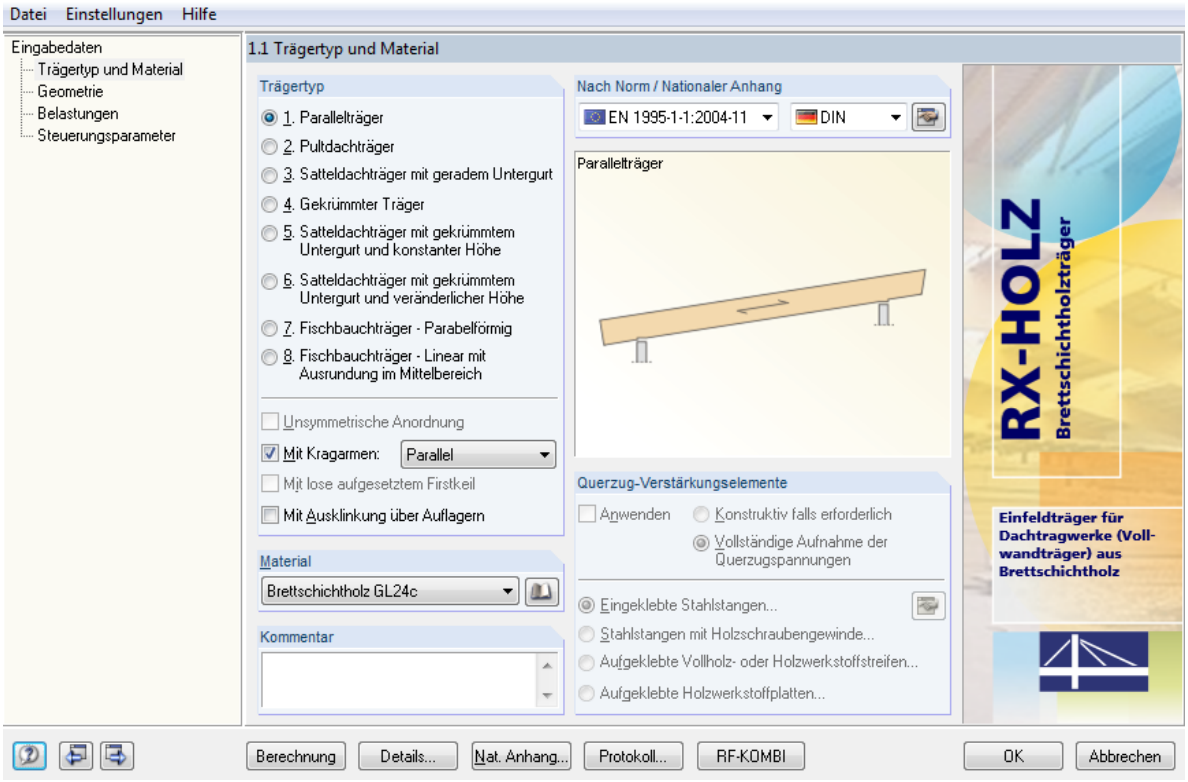


Abbildung 2: Eingabeoberfläche in RX-HOLZ

Sämtliche Eingabewerte und Parameter werden durch Grafiken und Beispiele verdeutlicht und sind daher klar verständlich. Auch die Ergebnisausgabe ist dank der verschiedenen Sortierkriterien, der Einfärbung der Ergebnisse und der grafischen Darstellung sehr ansprechend und einfach lesbar. Dabei können nach Wunsch auch einzelne Werte an einer bestimmten Stelle ausgelesen werden. Obwohl das Programm sehr komplex ist, was durch die zahlreichen veränderbaren Kenngrößen und Parameter verdeutlicht wird, bleibt es gleichzeitig übersichtlich. Ermöglicht wird dies vorallem durch die vorgegebenen, variierbaren Standardeinstellungen. Viele Lasten wie die Schneelast, die Windlast oder das Gewicht aus dem Dachaufbau werden bei RX-HOLZ automatisch berechnet und die Eingabe der benötigten Kenngrößen wird durch Hilfsmittel, wie die Schneelastzonenkarte und verschiedene Bibliotheken vereinfacht. Sämtliche, an ein Bemessungsprogramm zu stellende Anforderungen, wie beispielsweise die Querschnittsoptimierung werden für den einfachen Gebrauch sowie für höhere Ansprüche erfüllt. Mit dem Projektmanager ist zudem die Möglichkeit geboten einen einfachen, schnellen Zugriff auf das gewünschte Modul zu erhalten und bereits bestehende Systeme abzuändern und wiederzuverwenden.

2.4 RFEM/ RSTAB-Zusatzmodule: RF-/ HOLZ Pro

Neben dem eigenständigen Programm RX-HOLZ bietet die Firma Dlubal ebenfalls die Zusatzmodule HOLZ Pro und RF-HOLZ Pro für ihre Programme RSTAB und RFEM an. Die Zusatzmodule für das Stabwerksprogramm RSTAB und das 3D-FEM-Programm RFEM bieten die Möglichkeit Holzstäbe beziehungsweise ein Holzstabwerk, welches bereits in RSTAB oder RFEM eingegeben oder dorthin importiert wurde nach aktuellen Normen zu bemessen (siehe Abbildung 3).

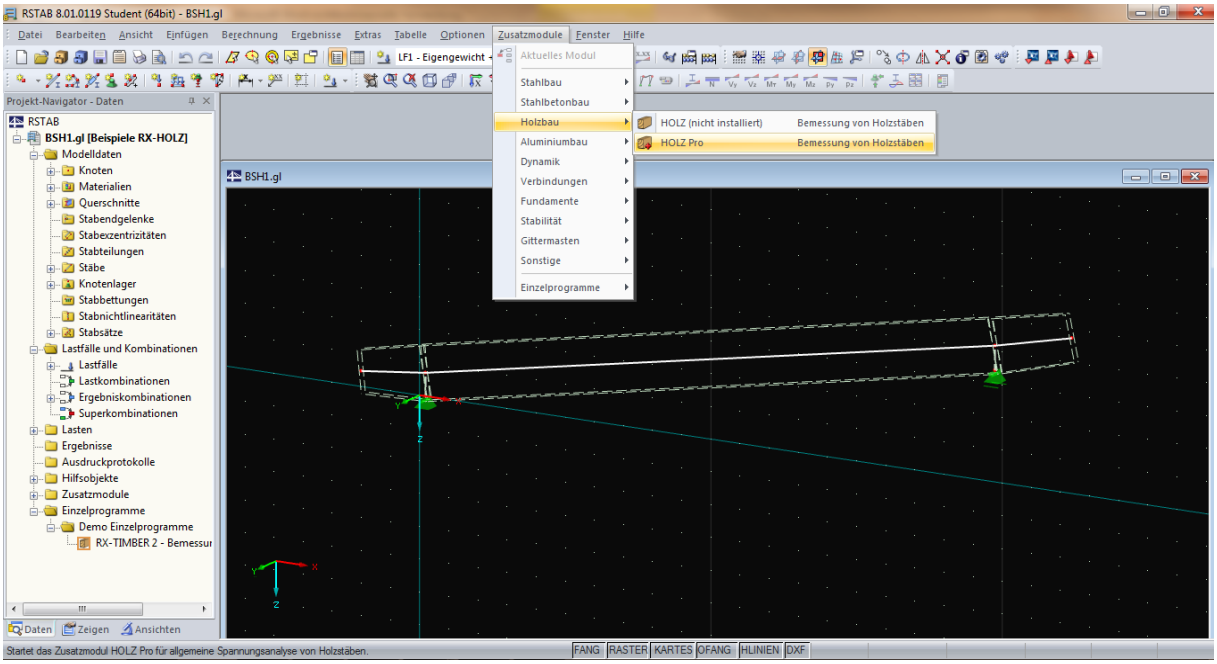


Abbildung 3: Start des Bemessungsmoduls in RSTAB

RF-HOLZ Pro und HOLZ Pro stellen eine sehr gute Ergänzung zum eigenständigen Programm RX-HOLZ dar. Mit diesen Zusatzmodulen ist es möglich beliebige Stabwerksmodelle zu entwerfen und zu bemessen. Zusammen mit ihren zahlreichen Schnittstellen zu anderen CAD-, CAM- und Statikprogrammen, bieten die Module eine umfassende Planung vom Entwurf bis hin zur Bemessung. Die Benutzeroberfläche in RSTAB und RFEM ist verständlicherweise wesentlich anspruchsvoller und umfangreicher als die in HOLZ Pro, bietet dem Anwender jedoch damit die Möglichkeit verschiedenste Holzkonstruktionen zu planen. Da die eigentliche Eingabe des Systems und mögliche Änderungen in RSTAB vorgenommen werden müssen, ist das Komplettpaket für den geschulten Nutzer konzipiert. Die Anwendung der Module, in welchen die eigentliche Bemessung stattfindet, ähnelt RX-HOLZ und ist durch zahlreiche grafische Hilfestellungen und Zusatzinformationen sehr benutzerfreundlich gestaltet. Hier werden die gewohnten Möglichkeiten wie beispielsweise die Querschnittsoptimierung und der Datenexport in Tabellenkalkulationsprogramme angeboten. Zusätzlich kann jedoch auch die Spannungsverteilung über den Stabquerschnitt grafisch dargestellt werden. Neben einer programmierbaren COM-Schnittstelle besitzen RFEM und RSTAB eine direkte Importmöglichkeit von Tekla Structures und Autodesk AutoCAD und unterstützen das *.stp-Format für Stabwerke und weitere Formate zahlreicher CAD- und Statikprogramme.

2.5 HoB.Ex

HoB.Ex ist ein einfach aufgebautes Berechnungsprogramm speziell für den Holzbau und wurde von dem INGENIEURBÜRO HOLZBAU in Karlsruhe entwickelt. Es besteht aus einer Eingabeoberfläche und etwa 250 Excel-Dateien, in denen die eigentliche Nachweisrechnung stattfindet. Neben einfacheren Trägern und Pfetten- sowie Sparrendächern eignet sich das Programm auch für die Bemessung zahlreicher Verbindungen und Anschlüsse. Zusammen mit der Software–Firma Dietrich's wurde als Ablösung für HoB.Ex das im Anschluss beschriebene Programm DC-Statik entwickelt. Trotz dieser Ablösung wurde das Programm HoB.Ex genauer betrachtet, weil es einen einfachen Bemessungsvorgang mithilfe von Excel darstellt und Excel für die Umsetzung des eigenen Bemessungsprogramms in Betracht gezogen wird.

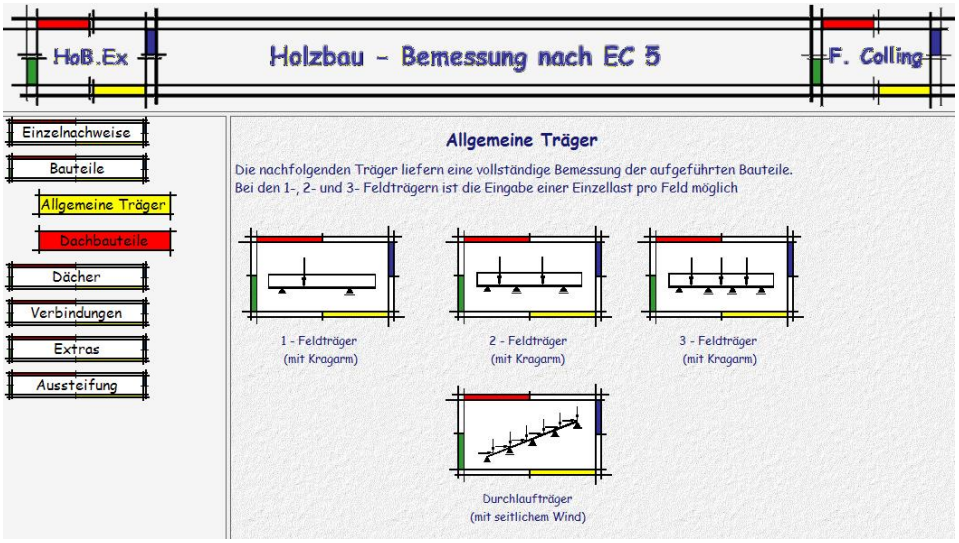


Abbildung 4: Programmoberfläche von HoB.Ex

HoB.Ex ist trotz seiner vergleichsweise einfachen Umsetzung ein Programm, das dank seiner zahlreichen Module vielfältig im Holzbau einsetzbar ist und die wichtigsten Anforderungen erfüllt. Es ist intuitiv und einfach zu bedienen und stellt Erklärungen und Beispiele für Kennwerte zur Verfügung. Durch die ausführliche Darstellung der Nachweise und der verwendeten Rechenwerte ist der Bemessungsvorgang zudem gut nachvollziehbar. Eine Querschnittsoptimierung ist nicht möglich. Es wird jedoch sofort nach Eingabe des Materials und des Querschnitts angegeben, ob alle Nachweise erfüllt sind, sodass diese Größen schnell angepasst werden können. Die Nachteile des Programms bestehen darin, dass schon vor Beginn der Berechnung festgelegt werden muss, um welche Art von System es sich handelt (siehe Abbildung 4) und, dass keine Schnittstelle zu anderen Programmen oder ein kompatibles Dateiformat vorhanden ist. Trotz der Möglichkeit die charakteristischen Schnittgrößen für die Weiterrechnung abzulesen, eignet sich das Programm somit eher für einzelne detaillierte Bemessungen und nicht für eine ganzheitliche Planung.

2.6 DC – Statik 12.04

Das Programm DC-Statik wurde in Zusammenarbeit vom INGENIEURBÜRO HOLZBAU von Dr. F. Colling und dem Softwarehersteller Dietrich's aufbauend auf dem Programm HoB.Ex entwickelt. Es bietet die Möglichkeit verschiedenste Systeme des Holzbaus sowohl nach DIN als auch nach

Eurocode zu bemessen. Es stellt zur Vereinfachung der Eingabe verschiedene Eingabehilfen und Vorlagen sowie eine Materialdatenbank zur Verfügung. Das Programm wurde von der Europäischen Vereinigung des Holzbaus (EVH) und deren Trägerverbänden initiiert und gefördert.

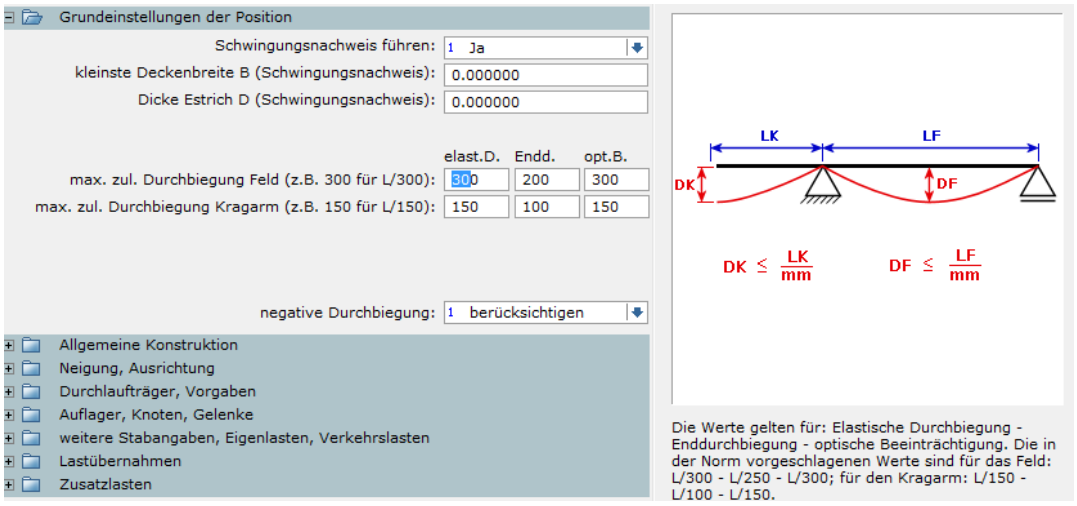


Abbildung 5: Eingabeoberfläche in DC-Statik

Das Bemessungsprogramm ist durch viele Hilfestellungen für die einzugebenden Kenngrößen und die sich parallel aktualisierende Grafik sehr benutzerfreundlich in der Anwendung und durch die Eingabe im Baummenü zudem sehr übersichtlich gestaltet (siehe Abbildung 5). Wesentlich besser als in HoB.Ex ist in DC-Statik nicht nur die Eingabeumgebung sondern beispielsweise auch die umfangreiche Materialauswahl, in der jedoch keine eigenen Materialien definiert werden können. Zudem besitzt DC-Statik auch eine Bibliothek zur Lastermittlung aus Deckenaufbauten und eine automatische Berechnung von Kennwerten, wie der Windlastzone durch Eingabe von Projektdaten. Anders als in vielen Programmen ist hier keine Möglichkeit der automatischen Querschnittsoptimierung vorhanden. Stattdessen werden verschiedene Querschnittsvarianten inklusive ihrer Ausnutzungsgrade dargestellt, sodass der Nutzer selbst entscheiden kann welche Kombination er wählt (siehe Abbildung 6).

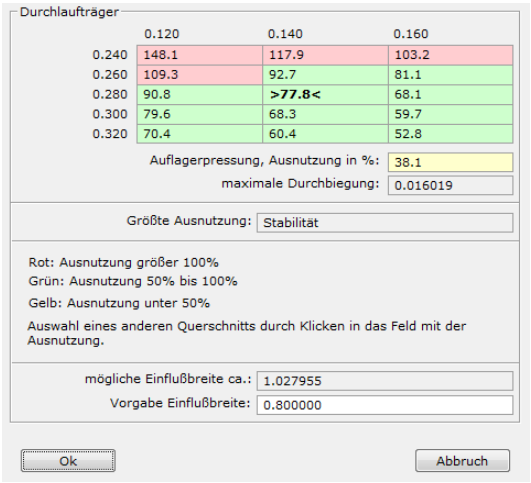


Abbildung 6: Darstellung der Ergebnisse in DC-Statik

DC-Statik bietet zudem die Möglichkeit DXF- und DXW-Dateien zu importieren um daraus Punkte abzugreifen. Neben dem alleinstehenden Modul zur Berechnung von Anschlüssen ist im Stabwerksprogramm bereits die Berechnung der Stabanschlüsse integriert.

2.7 Harzer – Statik

Das Berechnungsprogramm von Harzer – Software kann neben Massivbau, Stahlbau und Grundbau auch für die Berechnung statischer Nachweise im Holzbau verwendet werden. Es stehen dabei unterschiedlichste, vordefinierte Systeme und Anschlüsse zur Verfügung. Die Nachweise nutzen in der aktuellsten Version den Eurocode 5, es stehen jedoch auch ältere Versionen nach DIN 1052 zur Verfügung.

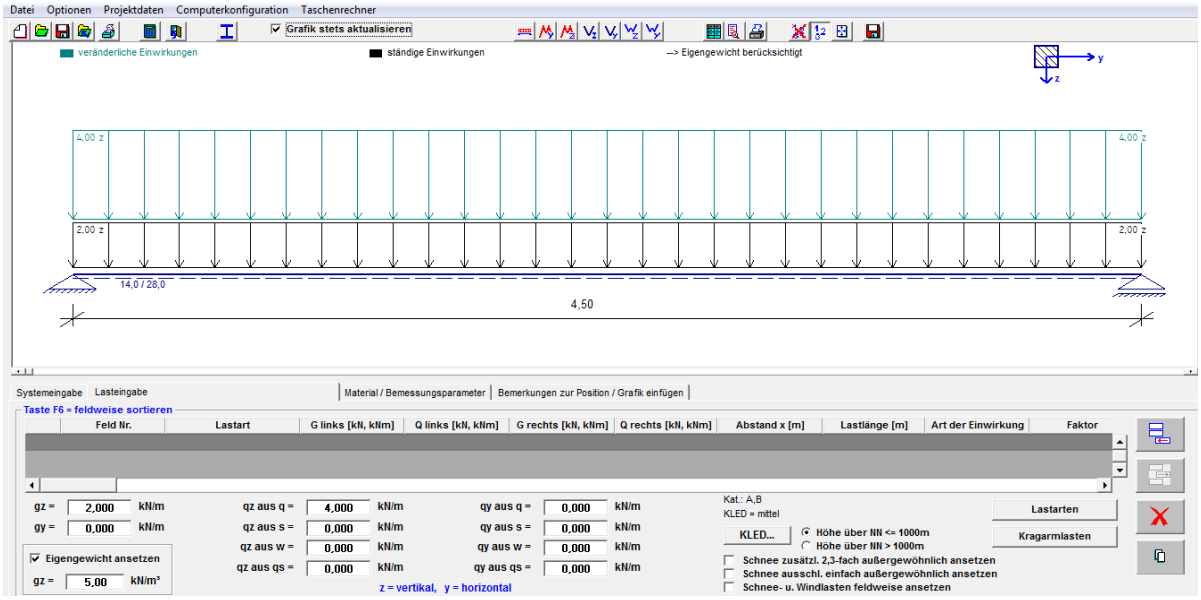


Abbildung 7: Eingabeoberfläche in Harzer-Statik

Das Harzer - Statik-Programm gehört zu den einfacheren Bemessungsprogrammen, stellt jedoch eine große Anzahl an Systemen zur Bemessung zur Verfügung. Der Programmaufbau ist übersichtlich und gut für den Nutzer nachvollziehbar. Kennwerte werden durch zusätzliche Informationsfenster erklärt und mit Grafiken und Beispielen verdeutlicht. Die Ergebnisse der Berechnung können benutzerdefiniert zusammengestellt und als RTF-Datei ausgegeben werden. Die Auswahl, beispielsweise für Lagerungsmöglichkeiten ist jedoch im Vergleich zu anderen Programmen oft eingeschränkter und auch eine Optimierungsmöglichkeit des Querschnitts, oder eine Schnittstelle zu anderen Programmen ist nicht vorhanden. Dennoch eignet sich das Programm durch die leichte Eingabe gut für schnelle Berechnungen von einzelnen Systemen.

2.8 RIBtec – RTholzbau und RTbsholz

Die beiden Nachweispakete RTholzbau und RTbsholz der Firma RIB stellen ein umfangreiches Tool zur Berechnung und Tragwerksplanung im Holzbau dar. Die Bemessung kann dabei nach DIN1052 sowie nach Eurocode 5 erfolgen. Zur Bemessung stehen dabei zahlreiche, vordefinierte Tragsysteme zur Auswahl.

Die Eingabe erfolgt über thematisch, nach dem Berechnungsablauf geordnete Tabs im unteren Teil des Fensters (siehe Abbildung 8). Auch in diesem Programm wird parallel eine Grafik des Systems angezeigt.

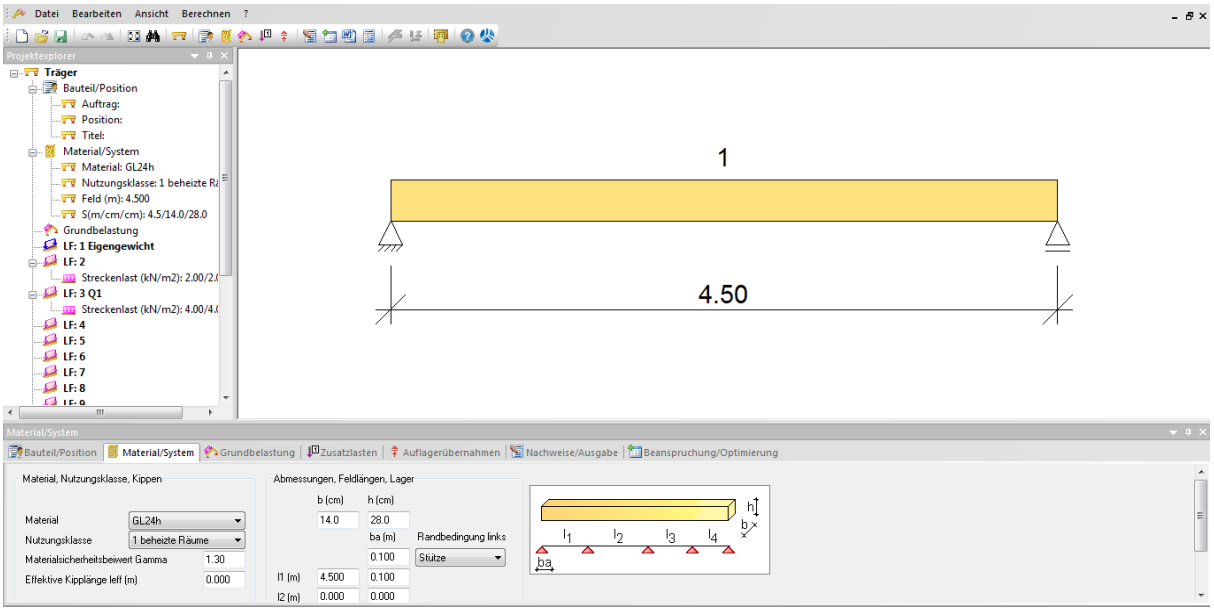


Abbildung 8: Eingabefenster in RTholzbau

Während Skizzen bei der Eingabe von Parametern zur Verdeutlichung zur Verfügung stehen, sind einige Kenngrößen gar nicht erklärt. Auch in der aufrufbaren Hilfe wird dazu teilweise nur eine ungenaue Erklärung oder ein Verweis auf die zugehörige Norm genannt. Das Programm liefert zwar keine Schnittstellen zu anderen Programmen, oder die Möglichkeit des Exports oder Imports von Daten, die Lastübernahme aus anderen, bereits erstellten Positionen ist jedoch möglich. Zusätzlich zur übersichtlichen und intuitiv zu benutzenden Eingabeoberfläche ist positiv zu erwähnen, dass sowohl die vorhandene Beanspruchung berechnet wie auch eine Querschnittsoptimierung der verschiedenen Querschnittsgrößen durchgeführt werden kann. Details können entweder für die ermittelten Schnittgrößen automatisch mit bemessen oder mit dem eigenen Anschlussmodul nach DIN 1052 berechnet werden.

2.9 CS–STATIK Holzbau

CS-STATIK der Firma GRAITEC besteht aus mehreren Programmen für verschiedene Bauarten. Die beiden Programmpakete Holzbau und Stabwerke ermöglichen die Bemessung für zahlreiche Anwendungsgebiete des Holzbaus nach DIN 1052 und Eurocode. Zusammengefasst bieten die beiden Programmpakete, trotz einfacher Handhabung eine sehr umfassende und vielfältige Möglichkeit zur Bemessung von Holztragwerken.

Neben der benutzerfreundlichen, klaren und übersichtlichen Darstellung werden trotzdem auch komplexere Ansprüche, wie beispielsweise die Definition eigener Materialien und die Weiterverwendung von Lasten aus anderen Positionen oder die Einstellung der Federwerte bei den Auflagerdefinitionen erfüllt. Positiv ist auch die Möglichkeit zwischen dem Nachweis- und Bemessungsmodus zu wählen. Im Nachweismodus wird die Ausnutzung eines festgelegten

Querschnitts ermittelt. Im Bemessungsmodus wird einer Querschnittsgröße kein Wert zugewiesen. Diese Querschnittsgröße wird automatisch optimiert und als Ergebnis der Berechnung ausgegeben. Als Kritik zu CS-STATIK bleibt lediglich zu nennen, dass im Vergleich zu anderen Programmen keine direkte Erklärung oder gar Beispiele der Kennwerte zum besseren Verständnis für den Nutzer zur Verfügung stehen. Obwohl eine eigene Programmhilfe zur Verfügung steht, ist diese vergleichsweise sparsam. CS-Statik bietet zudem die Möglichkeit DWG-Dateien aus dem CAD-Programm CS-CADI zur Weiterverarbeitung ins Stabwerksprogramm zu importieren.

2.10 D.I.E. – CAD- und Statik-Software

Die Firma D.I.E. stellt CAD- und Statik-Software für verschiedenste Anwendungen zur Verfügung. Die Programme sind in die Themen FEM, Hochbau, Grundbau und CAD gegliedert und weiter nach System und Material unterteilt. Speziell für den Holzbau gibt es kein Programm. Angeboten werden Programme für verschiedene Tragwerkstypen, die für die Bemessung unterschiedliche Materialien, darunter auch Holz, zur Verfügung stellen. Folgende Anwendungsgebiete im Holzbau können mittels der D.I.E.-Programme bemessen werden:

- Hochbau
  - Stützen
  - Durchlaufträger
  - Sparren-, Pfetten-, und Kehlbalkendächer
- FEM
  - Faltwerk
  - Platten
  - Scheiben
  - Stabwerke (räumlich und eben)
  - Trägerrost

Die Baustatik-Programme ermöglichen mittels Schnelleingabefenster eine schnelle und unkomplizierte Berechnung für Holzbauanwendungen. Im Schnelleingabefenster können Material, Nutzungsklasse, Querschnitt, Trägerabmessungen, Auflagerbreite und Lasten eingegeben werden. Zur Auswahl des Materials steht eine Datenbank zur Verfügung, in welcher die wichtigsten Materialparameter eingesehen werden können. Anhand dieser Eingaben wird eine vorläufige Berechnung durchgeführt, deren Ergebnisse danach im eigentlichen Eingabefenster angezeigt werden. Anschließend hat der Nutzer die Möglichkeit im eigentlichen Eingabefenster anspruchsvollere Einstellungen vorzunehmen. So sind beispielsweise bei der Lagerung des Trägers verschiedene Steifigkeiten einstellbar und bei den Lasten verschiedene Lastangriffspunkte möglich. Es kann zudem eine automatische Querschnittsoptimierung durchgeführt werden und mithilfe von Ergebnispunkten können Ergebnisse an mehreren Stellen überwacht werden. Der Brandschutznachweis hingegen kann hier nur für Stützen geführt werden und auch die Berechnung von Anschlüssen ist nur bei Dächern und nach DIN 1052 möglich. Das Programm bietet die Möglichkeit des Imports von DWG-, ASF-, VEC- und GEO-Dateien in FEM-Projekte. Zusätzlich können Makros erstellt und verwendet werden.

2.11 mb WorkSuite 2013

mbAEC bietet mit mb WorkSuite 2013 und den darin enthaltenen Programmen BauStatik, MicroFe, PlaTo, EuroSta.stahl/ EuroSta.holz, ProfilMaker, ViCaDo und CoStruc ein umfassendes Planungstool an. Dank dem enthaltenen Projektmanager können Projekte zentral verwaltet und mit den verschiedenen Programmen bearbeitet werden. Für die Holzbaubemessung stehen die Berechnungsprogramme BauStatik, MicroFe und EuroSta.holz zur Verfügung.

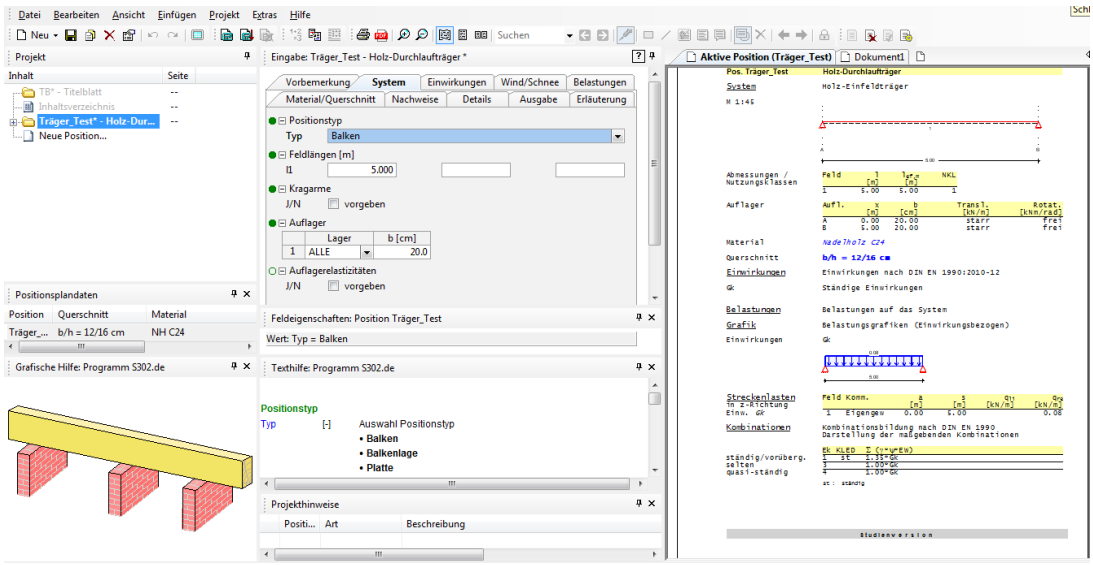


Abbildung 9: Programmoberfläche in BauStatik

Die mb WorkSuite 2013 ermöglicht eine umfassende Tragwerksplanung von der Modellierung bis hin zur Bemessung. Der Projektmanager ermöglicht die Verwaltung und Bearbeitung der Projekte mit den in der WorkSuite enthaltenen Programmen. Dabei können IFC-Dateien in das Modellierungsprogramm ViCADo importiert werden. Die erstellten DWG- und DXF-Dateien können dann in die Berechnungsprogramme übernommen werden. Zusätzlich können Daten wie Auflagerlasten und Einwirkungen einer Position in andere Positionen übernommen werden. Eine Materialbibliothek in welcher die Materialparameter eingesehen und eventuell sogar verändert werden können, wie in den anderen Programmen, steht nicht zur Verfügung. Neben der automatischen Querschnittsoptimierung besteht zudem die Möglichkeit für die Nachweise den maximalen Ausnutzungsgrad benutzerdefiniert festzulegen. Die Eingabeoberfläche, beispielsweise von BauStatik, hebt sich deutlich von den anderen Programmen ab. Neben einer grafischen Hilfe, gibt es anstelle der grafischen Darstellung des Systems eine, sich ständig aktualisierende Voransicht des Ausgabedokuments. Das System wird dabei in Form einer zweidimensionalen Skizze dargestellt (siehe Abbildung 9).

2.12 SOFiSTiK

Die SOFiSTiK-Software besitzt eine modulare Struktur, wobei sämtliche Daten in einer zentralen Datenbank gespeichert werden und so untereinander ausgetauscht werden können. Die Kommunikation zwischen den einzelnen Modulen erfolgt über den SOFiSTiK Structural Desktop (SSD). Die Eingabe der Bauteilgeometrie erfolgt im Modul SOFIPLUS über sogenannte Strukturelemente. Dabei stehen die folgenden drei Elemente zur Auswahl:

- Strukturfläche
- Strukturkante
- Strukturpunkt

SOFiSTiK stellt eine sehr umfassende Software zur Verfügung, die es anhand der einzelnen Module ermöglicht, sehr detailliert und anspruchsvoll zu bemessen. Dadurch erscheinen jedoch die Benutzeroberfläche und der eigentliche Bemessungsvorgang schnell unübersichtlich und nicht intuitiv. Jedoch steht in jedem der Module jederzeit Hilfe und eine zugehörige Anleitung zur Verfügung. Durch die Eingabe der Systeminformationen zu Beginn, können bereits automatische Anpassungen der Berechnungsumgebung und Voreinstellungen vorgenommen werden. Die Detailtiefe des Programms spiegelt sich auch darin wieder, dass beispielsweise Materialien und Querschnitte benutzerdefiniert konfiguriert werden können und in jedem der Bearbeitungsschritte zahlreiche verschiedene Einstellungen möglich sind. Zudem gibt es die Möglichkeit bestehende Materialien und Querschnitte zu importieren. Zur Eingabe des Systems im Modul SOFIPLUS sind AutoCAD-Grundlagen von Vorteil, wenn nicht sogar erforderlich. Durch die Erstellung in diesem Modul ist es jedoch auch möglich bereits bestehende CAD-Modelle zu verwenden. Ein großer Kritikpunkt in Bezug auf die Bemessung ist, dass momentan noch keine Tasks für die Bemessung im Holzbau zur Verfügung stehen. Nach eigenen Angaben von SOFiSTiK befindet sich die Software in Bezug auf die Holzbemessung noch in der Entwicklungsphase. So können zwar durchaus Verformungen und Schnittgrößen anspruchsvoller Holztragwerke berechnet werden, die eigentliche Bemessung anhand der aktuellen Holzbaunorm ist jedoch noch nicht möglich.

2.13 STAAD.Pro

Die Firma Bentley stellt mit ihrem Programm STAAD.Pro ein umfangreiches und anspruchsvolles FEM-Entwurfsprogramm zur Verfügung, das unter anderem auch für den Holzbau verwendet werden kann. Mit dem Programm können beliebige 2D- und 3D-Modelle entworfen werden. Zusätzlich bietet Bentley die Möglichkeit bereits in Bentley Structure entworfene Modelle in STAAD.Pro zu importieren und auch wieder dorthin zu exportieren. Die Grafikeingabe selbst erinnert stark an AutoCAD. Neben der Möglichkeit eigene Modelle zu entwerfen, stehen in einer Bibliothek zudem zahlreiche Tragwerksvorlagen zur Verfügung. Selbst erstellte Modelle können ebenfalls gespeichert und als Vorlage für spätere Projekte verwendet werden. Die Dateneingabe kann über die GUI in den jeweiligen Eigenschaftenblättern erfolgen oder über eine textbasierten Eingabespracheneditor. Die Lastermittlung erfolgt automatisch Insgesamt stehen in STAAD.Pro über 70 internationale Normen zur Verfügung. Welche Normen speziell für den Holzbau zur Verfügung gestellt werden, kann ohne eine Studentenversion des Programms jedoch nicht gesagt werden. Wie einige andere Programme, ermöglicht auch STAAD.Pro, eine eigene Ausgabedatei zusammenzustellen und anschließend zu drucken oder direkt in Microsoft Word zu exportieren. Als Besonderheit unterstützt das Programm VBA-Makros zum Schreiben eigener Routinen und die automatische Verknüpfung der Modellausgaben mit Excel oder MathCAD.

2.14 VCmaster

VCmaster ist ein Berechnungs- und Dokumentationsprogramm von VEIT CHRISTOPH, das den Übergang von manueller zu automatischer Berechnung und Bemessung darstellt. Mit ihm können Berechnungsformeln und beschreibende Texte benutzerdefiniert erfasst werden und somit als Vorlage

für die Bemessung dienen. Hierzu können Daten aus der Berechnungsausgabe eines Statikprogramms dank universeller und produktunabhängiger Schnittstellen übernommen und wiederverwendet werden.

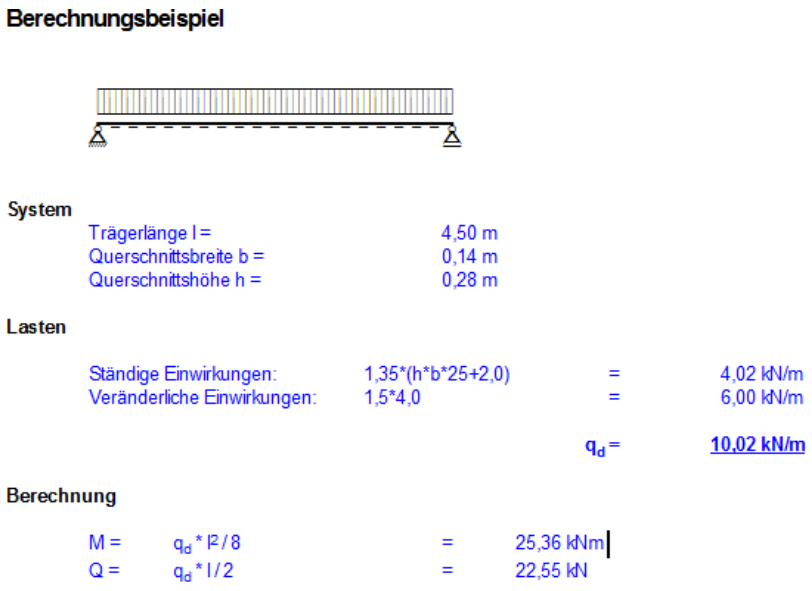


Abbildung 10: Automatische Berechnung mit selbstdefinierten Variablen

Die Berechnung kann dann nach Wunsch des Anwenders zusammengestellt werden und wird direkt in VCmaster durchgeführt (siehe Abbildung 10). Zusätzlich zur Datenübernahme aus anderen Programmen stehen dem Nutzer auch eine Bibliothek mit über 1000 Rechenblättern und zahlreiche Kennwert-Datenbanken zur Verfügung. Diese Vorlagen können erweitert, kombiniert und an die gestellten Anforderungen angepasst werden. Das Programm VCmaster ermöglicht eine individuelle vom Nutzer gesteuerte Berechnung und Dokumentation der Bemessung. Es bildet die Schnittstelle zwischen manueller und automatischer Bemessung. Besonders positiv ist, dass das Programm eine universelle Schnittstelle hat und Daten aus allen Windows-Programmen übernehmen kann. So können bereits erstellte statische Berechnungen übernommen und benutzerdefiniert angepasst werden. Diese statischen Analysen können durch leicht durchführbare Änderungen immer wieder variiert werden, wodurch eine Art Schablone für die Dokumentation und Berechnung entsteht. Neben der Schnittstelle zu anderen Statikprogrammen, stellt das Programm auch zahlreiche Vorlagen für die Bemessung zur Verfügung. Hier assistieren Eingabehilfen und ermöglichen eine leichte Handhabung. Die grafische Oberfläche, die hier anders als in den bereits vorgestellten Bemessungsprogrammen fehlt, wird durch selbst erstellte Skizzen und die Übernahme von bestehenden Grafiken ersetzt. Insgesamt ist VCmaster ein Programm das erwartet, dass der Anwender mit der Materie gut vertraut ist. Ist dies der Fall, ist das Programm eine gute Möglichkeit eine, den eigenen Ansprüchen entsprechende Bemessung und Dokumentation zu automatisieren.

2.15 Hersteller-Software

Neben den genannten Bemessungsprogrammen gibt es auch Software zur Anschlussbemessung die von den Herstellern der Verbindungsmittel zur Verfügung gestellt wird. Die Bemessung mit der herstellereigenen Software ist dabei auf die eigenen Verbindungsmittel und die dazugehörigen Anschlüsse beschränkt. Positiv ist jedoch, dass die Software von den Herstellern kostenlos zur

Verfügung gestellt wird. Als Beispiele für Hersteller-Software sind die Bemessungssoftware von Fischer WOOD FIX (siehe Abbildung 11), HECO-HCS, Würth Technical Software und die Online-Bemessungssoftware von SPAX zu nennen.

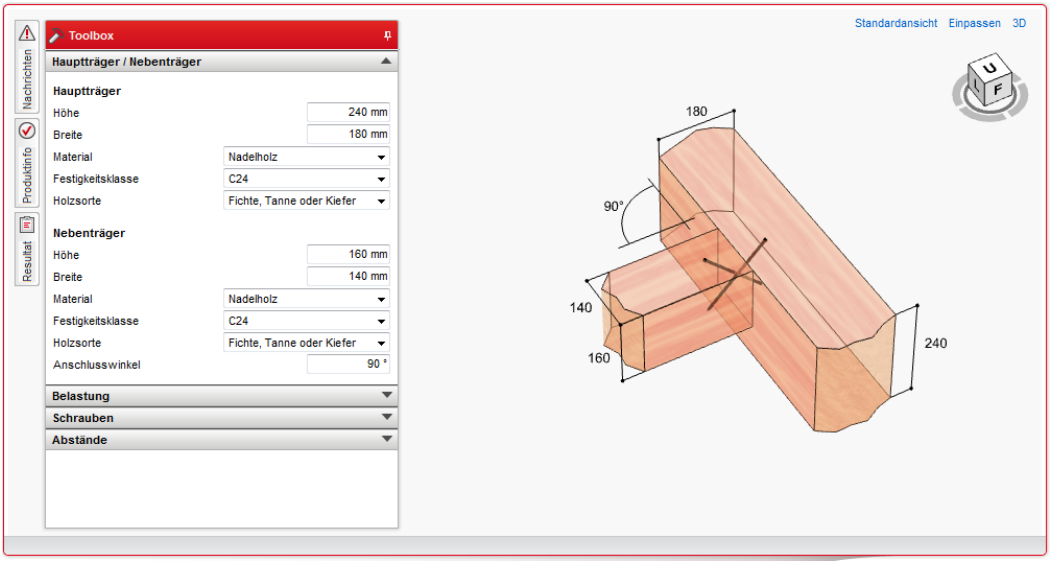


Abbildung 11: Eingabefenster der Bemessungssoftware WOOD FIX von Fischer

2.16 Übersicht über bestehende Holzbauprogramme

Auf den folgenden Seiten ist eine Übersicht über die genauer untersuchten Bemessungsprogramme nach verschiedenen Bewertungskriterien dargestellt. Das Programm SOFiSTiK ist in der Übersicht nicht enthalten, da es sich momentan noch nicht für die Bemessung im Holzbau eignet.

Tabelle 1: Programmvergleich zu Norm, Systemen und Dachformen

Programm	***Anbieter***	***Normen***	***Systeme/Nachweise***	***Dachformen***
Frilo	Nemetschek	• DIN 1052 • EC 5 inkl. NA (DIN, ÖNORM, NTC) • Code of practice UK	• (Verbund-)Träger • Fachwerkträger • Stütze • Aussteifungsverband • Vertikaler K-Verband • Holztafelwand • Leimholzbinder • Gedübelter Balken • Querschnitte • TCC Topfloor Holz Beton Verbund • Ebenes und Räumliches Stabwerk • Trägerrost	• Pfettendach • Sparren- und Kehlbalkendach, einschließlich Anschlussdetails • Pfette • Sparren • Kopfbandbalken
Scia Engineer	Nemetschek	• EC 5 inkl. NA (DIN, BS, CSN, NEN, NF, ÖNORM, PN, STN, NBN, IS, SFS, SIST, ELOT, SR, LU)	• Stab • Balken • Stütze • Ebenes und Räumliches Fachwerk • Platte • Wand • Schale • Ebener und Räumlicher Rahmen • Trägerrost • Allgemeines 3D Tragwerk	-
RX-HOLZ	Dlubal	• DIN 1052 • EC 5 inkl. NA (DIN, CSN, DK, NEN, NF, ÖNORM, PN, SFS, SS, UNI)	• Brettschichtholzträger • Durchlaufträger • Stütze • Rahmen • Verband	• Pfette
RFEM/RSTAB-Zusatzmodule: **RF./HOLZ Pro**	Dlubal	• DIN 1052 • EC 5 inkl. NA (DIN, CSN, NBN, NEN, NF, ÖNORM, PN, SFS, SS, UNI, NA to BS, DK, SIST, I.S., STN) • SIA 265	• Holzstäbe und –stabsätze	-
HoB.Ex EC5	INGENIEURBÜRO HOLZBAU	• EC 5	• Spannungsnachweis • Stabilitätsnachweis • Träger • Aussteifung	• Pfettendach • Sparrendach • Kehlbalkendach • Pfette • Sparren

DC-Statik 12.04	Dietrich's	• DIN 1052 • EC 5 inkl. NA (DIN, ÖNORM, NF) • SIA 261 • NTC	• Durchlaufträger • Stütze • Allgemeines Stabwerk	• Pfettendach • Sparrendach • Kehlbalkendach • Pfetten • Sparren
Harzer-Statik	Harzer Software	• EC 5 inkl. NA (DIN, ÖNORM)	• Spannungsnachweis • Träger • Stütze • Holzbalkendecke • Rahmen • Brettschichtholzbinder • Deckenscheibe • Holzrahmenbau • Aussteifung • Ebenes Stabwerk	• Pfettendach • Sparrendach • Kehlbalkendach • Allgemeines Dach • Pfetten • Sparren
RIBtec – RTholzbau und RTbsholz	RIB	• DIN 1052 • EC 5 inkl. NA (DIN, ÖNorm, für BSH auch CSN und BS)	• Träger • Stütze • Stab • Leimbinder • Brettschichtholzträger	• Sparrendach • Kehlbalkendach • Allgemeines Dach • Pfette • Sparren
CS-STATIK Holzbau	GRAITEC	• DIN 1052 • EC 5 mit/ohne dt. NA • SIA 265	• Träger • Querschnitt • Wandtafeln • Stützen • Brettschichtholzträger • Ebenes und Räumliches Stabwerk	• Pfettendach • Sparrendach • Kehlbalkendach • Pfetten • Sparren
D.I.E.	D.I.E.	• DIN 1052 • EC 5 mit NA (UNI, DIN, ÖNORM)	• Stütze • Träger • Faltwerk • Platte • Scheibe • Ebener und Räumlicher Rahmen • Trägerrost	• Pfettendach • Sparrendach • Kehlbalkendach
mb WorkSuite	mbAEC	• DIN 1052 • EC mit NA (DIN, ÖNORM)	• Träger (mit/ohne Verstärkung) • Stütze • Ebenes und räumliches Stabwerk • Fachwerk • Brettschichtholzbinder • Wandscheibe • Allgemeines Tragwerk	• Pfettendach • Sparrendach • Kehlbalkendach • Pfetten • Sparren

Tabelle 2: Programmvergleich zu Materialbibliothek, automatische Querschnittsoptimierung, Details und Brandschutznachweis und Dachformen

Programm	***Materialbibliothek***	***Automatische Querschnitts-optimierung***	***Details (Anschlüsse, Versatz, …)***	***Brandschutznachweis***
Frilo	Ja, jedoch ohne Einsicht/Zugriff in ges. Materialparameter	Ja	Ausführlich	Ja (Stütze, Holzbemessung, Leimholzbinder)
Scia Engineer	Ja, Erstellung neuer Materialien möglich	Ja	-	Nein
RX-HOLZ	Ja, Erstellung neuer Materialien möglich	Ja	Eingeschränkt	Ja
RFEM/RSTAB-Zusatzmodule: **RF-/HOLZ Pro**	Ja, Erstellung neuer Materialien möglich	Ja	Zusatzmodul STABDÜBEL, sehr ausführlich (nicht nach EC)	Ja
HoB.Ex EC5	Nicht einsehbar	Nein	Teilweise eingeschränkt	Nein
DC-Statik 12.04	Ja	Varianten-berechnung	Umfangreich	Nein (zusätzliches Modul erhältlich)
Harzer-Statik	Ja, jedoch ohne Einsicht/Zugriff in ges. Materialparameter	Nein	Umfangreich	Ja
RIBtec–RTholzbau und RTbsholz	Nicht einsehbar	Ja, mit Variantenwahl des Nutzers	umfangreich	Ja
CS-STATIK Holzbau	Ja, Erstellung neuer Materialien möglich	Ja	Sehr ausführlich	Ja
D.I.E.	Ja	Ja	Nur für Dachanschlüsse und DIN 1052, umfangreich	Ja (für Stützen)
mb WorkSuite	Nicht einsehbar	Ja	Sehr ausführlich	Ja

Tabelle 3: Programmvergleich zu Besonderheiten und Schnittstellen, unterstützten Dateiformaten und Preis

Programm	***Besonderheiten und Schnittstellen***	***Unterstütze Dateiformate***	***Preis (ab)***
Frilo	• ASCII-Schnittstelle für Stabwerke und Dächer • GLASER –isb cad- • ALLPLAN • Step dtH (nur Import) • Lastweiterleitung an andere Positionen möglich	• DXF • DTH • DSTV • EMF • RTF (nur Export) • XML (nur Export) • Enhanced Metafile (nur Export)	FRILO-Paket Holz: 1850€

Scia Engineer	• Plugin
	  • Tekla Structures
	  • Autodesk Revit Structure
		• ESA, ESAD (eigenes Format)
		• XML
		• IFC
		• XLS
		• ASCII
		• RTF
		• (3D)PDF
		• DSTV
		• DWG
		• DXF
		• VRML
		• HTML
		• SDN
		• R2S
		• T2S
			Keine Angaben

RX-HOLZ	• RX-HOLZ-Dateien lassen sich in RSTAB/RFEM öffnen • Export zu Excel/Calc und in CSV-Format	• RH2 (eigenes Dateiformat) • CSV (nur Export) • XLS (nur Export) • ODS (nur Export) • DXF (nur Export)	RX-HOLZ Paket: 1650€ + RX-HOLZ Pfette: 400€ + RX-HOLZ Rahmen: 400€ + RX-HOLZ Verband: 400€

RF-/HOLZ Pro (RFEM/RSTAB-Zusatzmodule)	• Export/ Import von Excel/Calc-Dateien
	• Schnittstelle zu CAD/CAM-Software
	  • Frilo ESK/RS
	  • SEMA GmbH
	  • Bentley ProStructure
	  • S&S Abbund
	  • Cadwork
	  • hsbCAD
	  • Tekla Structures
	  • Intergraph
	  • Autodesk Revit Structure
	  • Autodesk AutoCAD
		Eigenes Dateiformat:
		• RS8, RS7, RS6, RST, RSX
		• RF5, RF4, RF3, RFE, RFX
		Formate für Stabwerke:
		• STP
		Formate für Tabellenkalkulation:
		• XLS
		• ODS
		• CSV (nur Export)
		Allgemeine Formate für CAD-Programme:
		• DXF
		• IFC
		• ISM.DGN
		• DAT
		• DGN
		Formate für Statikprogramme:
			HOLZ Pro: 1250€
			RF-HOLZ Pro: 1250€

	• Direkter Import/Export:
	  • Tekla Structures
	  • Autodesk AutoCAD
	• Programmierbare COM-Schnittstelle
		• ANS
		• XML
		Zusätzliche Formate:
		• STP
		• STEP
		• IGS
		• IGES
		• SAT	

HoB.Ex EC5	• Basiert auf Excel • Nachweise werden aufgeführt	Eigenes Dateiformat-	HoB.Ex-EC5 (inkl. Aussteifung): 749€
DC-Statik 12.04	• Import von DXF- und DXW-Dateien zum Abgreifen von Punkten • Lastübernahme aus anderen Positionen möglich	• DXF (nur Import) • DXW (nur Import)	980€
Harzer-Statik	-	Eigenes Dateiformat	798€
RIBtec – RTholzbau und RTbsholz	• Auflagerdateien anderer Systeme können als Lasten übernommen werden	• HBA (eigenes Dateiformat) • RTBSH (eigenes Dateiformat)	RTholzbau: 2800€ RTbsholz: 800€
CS-STATIK Holzbau	• Import von DWG-Dateien aus CS-CADI in das Stabwerksprogramm • Schnittstelle zu anderen Positionen und CS-STATIK-Programmen • Nachweis- und Bemessungsmodus möglich	• DWG (Import aus CS-CADI) • DXF (Export mittels CAD-Schnittstelle) • GTC (Export mittels CAD-Schnittstelle) • CSX (eigenes Format) • SUS (eigenes Format)	Komplettpaket Holzbau: 2400€ Stabwerke: 800€ +Erw. Holzbemessung: 800€
D.I.E.	• Möglichkeit des Erstellens und Verwenden von Makros • Import von DWG-, GEO- und VEC-Dateien in die FEM-Projekte • Schnelleingabe möglich • Überwachung von Ergebnispunkten	• BAUSTATIK (eigenes Projektformat) • CS (Makro-Datei) • S01-S20(eigenes Dateiformat) • DWG (AutoCAD) • GEO (ISB-CAD) • VEC (ZEIG)	Faltwerk. 3900€ Räuml. Rahmentragwerke: 2075€ Ebene Rahmentragwerke: 1250€ Scheiben: 1375 Platten: 1975€ Trägerroste: 800€ Dach: 850€ Durchlaufträger: 850€
mb WorkSuite	• Import von Daten aus anderen Positionen möglich • Schnittstellen zwischen den einzelnen Programmen	• IFC • (3D) DXF • (3D)DWG	Je nach Umfang variierende Preise: Ing⁺ comfort (BauStatik,MicroFe,ViCADo): 8490 €

2.17 Bewertung und Fazit

Die Recherche hat ergeben, dass inzwischen einige hilfreiche Programme zur Berechnung und Bemessung im Holzbau zur Verfügung stehen. Es gibt sowohl Programme, die speziell für den Holzbau entwickelt wurden, als auch Holzbemessungsmodule zu bereits bestehenden Statikprogrammen. Die Bemessungsprogramme speziell für den Holzbau sind dabei meist einfacher aufgebaut und bieten zahlreiche vordefinierte Tragwerkstypen zur Bemessung an. Bei dieser Art von Programmen stehen zudem immer auch Dachformen für die Bemessung zur Verfügung.

Die Programme beziehen sich bei der Nachweisführung immer auf den Eurocode 5, stellen aber auch oft noch die DIN 1052 und verschiedene nationale Anhänge für den EC 5 zur Verfügung. Bei der Festlegung der zur Bemessung heranzuziehenden Normen sind Unterschiede zwischen den Programmen erkennbar. Bei einigen Programmen muss bereits beim Erstellen einer neuen Position festgelegt werden, nach welcher Norm diese bemessen werden soll. Bei anderen Programmen kann die Norm auch noch im Eingabefenster variiert werden. Dies ermöglicht durch das Beibehalten des Systems einen Vergleich der Ergebnisse. Bei manchen Programmen, wie beispielsweise RX-HOLZ von Dlubal, können die zugehörigen Normparameter dabei nicht nur eingesehen sondern auch variiert werden. Neben der Bemessung im Grenzzustand der Tragfähigkeit und der Gebrauchstauglichkeit ermöglichen die meisten Programme auch eine Brandschutzbemessung.

Meist bieten die Programme Materialbibliotheken, in denen die Eigenschaften der verfügbaren Materialien eingesehen und teilweise variiert werden können, um so eigene Materialien für die Bemessung zu erstellen. Einige Programme stellen Datenbanken für Lasten, beispielsweise aus verschiedenen Deckenaufbauten zur Verfügung und verfügen zudem über eine automatische Lastermittlung.

Neben der Berechnung der vorhandenen Querschnittsausnutzung, ist häufig auch eine automatische Optimierung des Querschnitts innerhalb der Programme möglich. Als Alternative hierzu besitzt DC-Statik die Variantenberechnung und auch die Programme von RIBtec führen die Bemessung für verschiedene Varianten durch, aus denen der Nutzer anschließend wählen kann. CS-Statik bietet zudem die Möglichkeit zwischen dem Nachweis- und Bemessungsmodus zu wählen.

Auch in der Detailbemessung unterscheiden sich die untersuchten Programme sehr. Es gibt einige Programme wie HoB.Ex in welchen die Detailbemessung nur in separaten Modulen stattfinden kann. Die einzige Hilfestellung ist dabei, dass bei der Bemessung eines Systems auch die charakteristischen Schnittgrößen ausgegeben werden. In anderen Programmen wie BauStatik von mbAEC dagegen, können bei den Positionsbemessungen Übergaben erzeugt werden, welche dann später im Detailmodul weiterverwendet werden können. Beim zugehörigen Stabwerksprogramm und beim Stabwerksprogramm in CS-Statik von GRAITEC ist es ähnlich. Hier wird das Detailbemessungsprogramm aus dem Stabwerksprogramm heraus gestartet, Geometrie, Querschnitte, und Anschlussschnittgrößen werden übernommen und es wird eine neue Position erstellt. Eine Schnittstelle zwischen den Modulen ist dabei nur in Form der Rückgabe der Ergebnisse für die Ausgabe vorhanden. Eine Berücksichtigung der Verbindungen ist lediglich durch die manuelle Abänderung der Auflagerbedingungen in einigen Programmen möglich. In den Dachbemessungsmodulen sind die Anschlussbemessungen bereits integriert und auch bei DC-Statik

werden die Stabwerksanschlüsse intern mit dem Stabwerk berechnet. In RX-HOLZ werden bei den Modulen „Rahmen", „Pfetten" und „Verband" ebenfalls die Anschlüsse mit bemessen.

Ein wichtiger Punkt zur Bewertung der Programme sind die vorhandenen Schnittstellen und unterstützen Dateiformate. Hierbei verfügen die nicht auf den Holzbau beschränkten Stabwerks- und FEM-Programme meist über eine größere Auswahl. Sie bieten wie Scia Engineer häufig die Möglichkeit bereits in der Vorplanung entworfene Modelle, beispielsweise als DWG-Dateien zu importieren und anschließend auch wieder zu exportieren. Diese Eigenschaften fördern somit die Interoperabilität zwischen den einzelnen Programmen und ermöglichen eine ganzheitlichen Modellierung, Planung und Bemessung. Um eine umfassende Planung zu ermöglichen, besitzen einige Programme zudem eine Projektverwaltung und ermöglichen es Positionen mehrfach zu verwenden oder Lasten für die Berechnung anderer Positionen zu übernehmen.

Die Preise der einzelnen Programme sind erwartungsgemäß hoch. Oft stehen verschiedene Ausführungen, Komplettpakete und Zusatzmodule zum Erwerb zur Verfügung.

Insgesamt unterscheiden sich die Programme vor allem in ihrer Detailliertheit, bezüglich der einzugebenden Daten, der Ergebnisausgabe, der rechenbaren Systeme und der Anschlüsse. Die Ausgabe der Ergebnisse erfolgt oft umfangreich mit grafischen Darstellungen im RTF-Format, sodass der Nutzer diese, seinen Anforderungen entsprechend gestalten kann. Meist ist die Programmoberfläche so gestaltet, dass dem Nutzer die Eingabe anhand von Skizzen, Beschreibungen und Beispielen vereinfacht wird. Je komplexer die eingegebenen Systeme sind, die das Programm zulässt, desto höher werden dabei meist die Ansprüche an die Vorkenntnisse des Nutzers.

Die Untersuchung verfügbarer Programme hat gezeigt, dass bereits einige benutzerfreundliche und variabel gestaltete Programme auf dem Markt sind, welche eine umfassende computergestützte Planung im Holzbau ermöglichen. Trotzdem war wie am Beispiel von SOFiSTiK immer wieder zu erkennen, dass der Markt für computergestützte Bemessungswerkzeuge momentan noch Mitten in der Entwicklung steckt und in Zukunft sicherlich weitere Werkzeuge zur Verfügung stehen werden. Die vorhandenen Programme verfügen bereits über einen großen Anwendungsbereich. Jede der Bemessungssoftware zeigt jedoch auch Grenzen im Anwendungsbereich auf. Diese Grenzen zeigen sich beispielsweise darin, dass der Nutzer in seinen Handlungen, während der Bemessung eingeschränkt ist. Dies spiegelt sich unter anderem in nicht einsehbaren und veränderbaren Materialien, vordefinierten Tragwerkstypen und Nachweispaketen wieder. Der Nutzer ist in jedem Fall auf die Vollständigkeit des Programms und dessen Bemessungsvorgang angewiesen. Eine benutzerdefinierte Gestaltung des Nachweises ist meist nicht möglich. Somit ist der Anwendungsbereich des Programms immer auf die Vorgaben des Programms beschränkt. Soll ein Nachweis nach einer neuen Zulassung erfolgen oder überhaupt nur ein einzelner Nachweis geführt werden, ist dies anhand der Berechnungsprogramme nicht möglich. Der Nutzer hat auf den Inhalt der Programme keinen Einfluss und ist so in seiner Handlung meist eingeschränkt. Er kann lediglich mit den ihm zu Verfügung gestellten Elementen arbeiten. Dem gegenübergestellt wird im folgenden Kapitel die Idee einer offenen, modularen Programmplattform und deren Anwendungsbereiche erörtert. Diese soll ein multifunktionales Berechnungsprogramm zur Verfügung stellen, das je nach Bedarf erweitert werden kann.

3 Erörterung einer modularen, offenen Programmplattform

Im Folgenden wird genauer auf die einzelnen Programmpunkte eingegangen und erörtert inwieweit eine offene Programmplattform eine Verbesserung zu vorhandenen kommerziellen Bemessungsprogrammen darstellt. Unter einer solchen Plattform soll ein Programm verstanden werden, das aufgrund seines modularen Charakters ständig durch weitere Elemente erweitert werden kann. Aufgrund seiner Offenheit kann diese Erweiterung von verschiedenen Entwicklern vorgenommen werden. Dadurch entsteht ein anpassbares und somit multifunktional anwendbares Bemessungsprogramm.

Die ersten Einschränkungen der untersuchten Programme bestehen bei der Wahl der Systeme. Einige Programme besitzen einzelne Bemessungsmodule für verschiedene Systeme. Oft sind in den Modulen bereits Voreinstellungen getroffen, wodurch keine benutzerdefinierte, individuelle Eingabe mehr möglich ist. Im Gegensatz dazu stehen die Stabwerksmodule, die meist separat in den Programmpaketen oder als völlig eigenständiges Programm vorhanden sind. In diesen kann das Tragwerk ganz nach den Wünschen des Nutzers eingegeben und bemessen werden, und die Festlegung der Tragwerksform muss nicht schon zu Beginn stattfinden. Eine prinzipielle, interne Gliederung der Bemessungsprogramme nach dem System beziehungsweise die Unterteilung nach dem Eurocode 5 scheint durchaus sinnvoll. Dieses Prinzip würde auch in einem multifunktionalen Programm weiter verwendet werden. Dies liegt vor allem daran, dass zur Nachweisführung nach dem Eurocode oft auch der Bauteiltyp entscheidend ist. Schon vor dem Start des Programms die Entscheidung treffen zu müssen, um welchen Bauteiltyp es sich handelt, sollte jedoch nicht notwendig sein. Man sollte vielmehr mehrere Bauteile erstellen und diese zu einem Projekt zusammenschließen können. Einige Programme ermöglichen dies bereits, indem vom Nutzer auch Projekte erstellt werden können, welchen im Verlauf der Anwendung einzelne Positionen hinzugefügt werden können.

In den meisten Modulen findet die Bemessung automatisch nach der gewählten Norm statt. Das Programm schreibt vor, welche Nachweise durchgeführt werden und lässt keine Änderungen zu. Meist wird erst im Ausgabefenster des Ergebnisses für den Benutzer sichtbar, welche Nachweise genau geführt wurden. Bei einem modularen, offenen Programm könnte vom Nutzer selbst entschieden werden, welche Nachweise, nach welcher Norm durchgeführt werden sollen und welche nicht. Dieser Aspekt wurde beispielsweise bereits bei BauStatik von mbAEC berücksichtigt und realisiert. Durch die benutzerdefinierte Zusammenstellung von Nachweisen zu Nachweispaketen sind dem Anwender somit kaum Grenzen gesetzt. Nicht existierende Nachweistypen könnten einer offenen, modularen Programmplattform zudem einfach hinzugefügt werden und stünden anschließendem jedem Nutzer zur Verfügung. Auch Regelungen aus Zulassungen oder Normänderungen könnten dadurch schnell eingefügt und verwendet werden. Der Nutzer wäre damit nicht auf die Versionserneuerung des bestehenden Programms abhängig, sondern könnte die Erneuerung selbst durchführen.

Bei einigen Programmen musste bereits beim Erstellen einer neuen Position die zu verwendete Norm festgelegt werden. Bei anderen Programmen konnte die Norm dagegen auch noch zu einem späteren Zeitpunkt abgeändert werden. So kann die Bemessung nach verschiedenen Normen durchgeführt und die Ergebnisse können verglichen werden. Bei RX-HOLZ beispielsweise konnten die durch die Norm vorgegebenen Parameter eingesehen und sogar variiert werden. Bei einem offenen Programm wäre

zusätzlich denkbar neben einer benutzerdefinierten Abänderung der Normwerte, einen Nachweis parallel nach verschiedenen Normen zu berechnen, um so einen direkten Vergleich zu erhalten.

Die meisten Programme stellen eine Materialbibliothek zur Materialauswahl zur Verfügung. Einige ermöglichen zudem die Kennwerte der Materialien einzusehen und durch Variation dieser, benutzerdefinierte Materialien zu erstellen. In einem multifunktionalen Berechnungsprogramm sollte dies ebenfalls möglich sein. Es macht Sinn die neu erstellten Materialien anschließend in einer zentralen Datenbank zu speichern und somit jedem weiteren Nutzer zur Verfügung zu stellen. Auch die Erstellung und Speicherung von Standardquerschnitten ist in einer solchen Struktur denkbar.

In der Detailbemessung unterscheiden sich die untersuchten Programme ebenfalls stark. Teilweise wird die Bemessung von Details lediglich in eigenen Modulen ermöglicht, teilweise erfolgt die Bemessung automatisch. In jedem Fall ist der Anwender vom Aufbau des Programms und den zur Verfügung gestellten Komponenten abhängig. In dem multifunktionalen Berechnungsprogramm sollte es möglich sein, die Anschlussbemessung nach Wunsch immer bei der Systembemessung mit zu führen, um so eine Statik des kompletten Systems zu erhalten. Nach Bedarf sollte es jedoch auch möglich sein nur einen Anschluss zu bemessen. In der modularen und offenen Programmplattform könnte dies dadurch ermöglicht werden, dass der Nutzer selbst festlegt welche Nachweise er durchführen und damit auch wie detailliert er das System rechnen möchte. In den Programmen waren die zur Verfügung stehenden Anschlussformen und Verbindungsmittel teilweise stark eingeschränkt. In jedem Fall war der Nutzer auch hier wieder vom Angebot des Programms abhängig. Bei einem multifunktionalen Berechnungsprogramm könnten dem Nutzer alle in der Norm oder auch in Zulassungen berücksichtigten Anschlussformen durch entsprechend erstellte Nachweise zur Verfügung stehen. Um eine universelle Anwendung zu gewährleisten wäre es ebenfalls eine Idee darin eigene Verbindungsmittel anhand der benötigten Kenngrößen zu definieren oder die Verbindungsmittel verschiedener Hersteller zu verwenden. Diese Verbindungsmittel könnten dann in einer Datenbank zur Verfügung stehen.

Um ein multifunktionales Berechnungsprogramm zu erschaffen macht es Sinn sich an der händischen Nachweisrechnung zu orientieren. Der Nutzer sollte zu Beginn der Eingabe, ganz wie in der manuellen Berechnung, festlegen welche Nachweise nach welcher Norm geführt werden sollen. Auf der Eingabeoberfläche sollten dann die benötigten Eingangswerte vom Nutzer abgefragt werden. Entsprechende Kenngrößen der Norm sollten anhand der entsprechenden Norm vorbelegt sein. Es könnte jedoch überlegt werden ob auch eine manuelle Änderung der Werte möglich sein sollte. Der Nutzer soll die Möglichkeit haben alle Freiheiten einer eigenständigen Handrechnung zu besitzen und zudem alle Annehmlichkeiten der automatischen Berechnung und der Eingabe in ein Bemessungsprogramm zu nutzen.

Eine modulare, offene Programmplattform hat wie bereits erwähnt den großen Vorteil, dass jederzeit neue Elemente hinzugefügt werden können. Durch zahlreiche Anwender würden die zur Verfügung stehenden Nachweise und Materialien schnell zu einem anschaulichen Katalog heranwachsen. Zudem könnten die hohen Kosten vermieden werden, die die Anschaffung der meisten Bemessungsprogramme mit sich bringen.

Im nun folgenden zweiten Teil der Masterarbeit wurde ein solches multifunktionales Bemessungsprogramm entworfen und anschließend beispielhaft umgesetzt. Diese beispielhafte Untersuchung soll klären, ob die Idee einer modularen und offenen Programmplattform umsetzbar ist. Als Vorlage zur Strukturierung des Programms soll der Eurocode 5 dienen. Hierzu wird zunächst die Systematik der Nachweisführung anhand des Eurocodes untersucht.

4 Untersuchung der Struktur und Systematik des EC 5

Um ein offenes und modulares Bemessungsprogramm umsetzen zu können, muss zunächst ermittelt werden, welche Daten, in welcher Form zur Verfügung gestellt werden müssen. Auch der prinzipielle Berechnungsablauf und dessen Struktur müssen vor dem Entwurf eines solchen Programmes genauer untersucht werden. Da sich das zu entwickelnde Programm am Eurocode 5 orientieren soll, wird dieser vorerst bezüglich seiner Strukturierung und Systematisierung untersucht.

Hierzu wurden zwei anspruchsvolle Beispiele gewählt und in Bezug auf deren Bemessung und Nachweisstruktur genauer betrachtet. Diese Beispiele sollen später auch zur Orientierung bei der exemplarischen Umsetzung der Programmierung dienen. Als Beispiele werden der biegesteife Anschluss eines Zweigelenkrahmens zwischen einer vertikalen Stütze und einem horizontalen Träger, sowie ein Satteldachträger mit gekrümmten unteren Rand und dessen Querzugverstärkung im Grenzzustand der Tragfähigkeit bemessen. Beide Systeme werden mit einer konstanten Linienlast belastet. Die Beispiele werden speziell für den vorgegebenen Lastfall betrachtet, da sich die zu führenden Nachweise unter anderem aus der Belastungsart ergeben. Auf die Verwendung von genauen Zahlenwerten wurde der Übersichtlichkeit halber verzichtet. Die Untersuchung zeigt daher lediglich den theoretischen Ablauf. Im Rahmen der nun folgenden Untersuchung wird primär darauf geachtet, welche Informationen zu welchem Zeitpunkt und in welchem Rahmen zur Verfügung stehen müssen, um anschließend eine, den Anforderungen gerecht werdende Programmstruktur entwerfen zu können. Es wird für jeden Nachweis darauf verwiesen woher diese Werte, Bedingungen und Nachweisformeln entnommen werden, da genau diese Informationen für die Umsetzung in einen vollständigen Programmcode essentiell sind. Der ausführliche Nachweisablauf und die Normbezüge können dem Anhang in Kapitel 13 entnommen werden. Im Folgenden werden lediglich die Untersuchung und die daraus resultierende Schlüsse aufgeführt.

Zu Beginn jeder Nachweisführung werden zuerst das Material und die Geometrie gewählt. Daraus ergeben sich folgende für die Bemessung relevanten Kenngrößen:

- Festigkeiten
- Dichte und andere materialspezifische Kennwerte
- Holzart
- Geometrie und Abmessungen

Neben den aufgeführten Eingangswerten und den aus der Norm entnehmbaren Größen werden zudem die maßgebenden Schnittgrößen benötigt. Diese sollen im späteren Programm als bereits bekannt angenommen werden. Zudem sind vor der Berechnung die für die Bemessung notwendigen Parameter der Nutzungsklasse und Lasteinwirkungsdauer festzulegen. Sind diese aus den Bauteilen und Randbedingungen resultierenden Größen festgelegt, kann mit der Nachweisrechnung begonnen werden. Die Norm ist thematisch nach verschiedenen Spannungszuständen und Tragwerksystemen gegliedert. Die Entscheidung welche Nachweise für die gewählten Systeme und deren Belastung zu führen sind, wird vom Ingenieur selbst getroffen. In der vorherigen Untersuchung der Bemessungsprogramme wurde diese Entscheidung meist vom jeweiligen Programm vorweg genommen. Im späteren Berechnungsprogramm soll jedoch die Wahl der Nachweise wie bei der Handrechnung durch den Anwender selbst stattfinden. Im Anhang sind die für die gewählten Beispiele

durchzuführenden Nachweise aufgeführt. Nachdem die in diesem Fall notwendigen Nachweise und deren Bestandteile genauer betrachtet wurden, können folgende Beobachtungen festgehalten werden. Die Norm stellt zuerst Definitionen und Bedingungen in Formeln und Textform zur Verfügung. Diese Formeln legen fest wie ein Berechnungswert aus den gegebenen Eingangsgrößen berechnet werden soll und stellen zudem Bedingungen zur Erfüllung eines Nachweises auf. Die Formeln scheinen auf den ersten Blick einfach in den späteren Programmcode übertragen werden zu können. Schwieriger scheinen die zusätzlichen Bedingungen und Anweisungen in Textform. Des Weiteren werden in der Norm Fixwerte, beispielsweise für Beiwerte verwendet. Es kann festgehalten werden, dass all diese Norminformationen in direkter Abhängigkeit von den Eingangswerten und Randbedingungen des zu bemessenden Beispiels stehen (siehe Abbildung 12). Viele Bedingungen werden dabei bereits durch die Art des Bauteils und die darauf wirkende Belastung vorgegeben. Es ist für die richtige Normverwendung also wichtig, dass die Eingangsgrößen schon zu Beginn der Berechnung eindeutig und vollständig festgelegt sind. Einige der Eingangsgrößen können direkt aus den Eigenschaften des Bauteils abgeleitet werden, andere wiederum erschließen sich aus den sonstigen Randbedingungen.

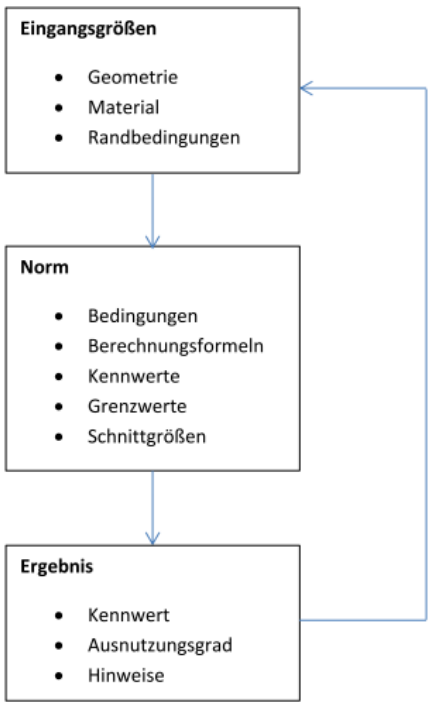


Abbildung 12: Struktur der Nachweisführung

Nachdem aus den Eingangsgrößen nach Vorgabe der Norm Kennwerte berechnet wurden, dienen auch diese oft wieder als Eingangsgröße für eine weitere Bemessung (siehe Abbildung 12). Das Ziel der Nachweisrechnung ist es, den Ausnutzungsgrad des bestehenden Systems zu ermitteln, welcher angibt zu wieviel Prozent das System ausgenutzt wird. Zudem können sich als Ergebnis eines Nachweises auch Hinweise zu weiteren Nachweisführungen oder zur Ausführung des Bauteils ergeben. Auch diese müssen im späteren Programm berücksichtigt werden und dem Anwender auf der grafischen Oberfläche ausgegeben werden.

Wie sich gezeigt hat können die meisten Eingangsgrößen eines Nachweises direkt aus den Eigenschaften des zu bemessenden Bauteils ermittelt werden. Es scheint daher sinnvoll diese Eingangsgrößen einem Bauteil direkt zuzuordnen und dann bei der Berechnung jeweils über das Bauteil darauf zuzugreifen. Diese Vorgehensweise wird ebenfalls, wenn auch unbewusst in der einfachen Handrechnung verwendet. Auch der Nachweis selbst bezieht sich, sobald er verwendet wird immer auf ein oder mehrere bestimmte Bauteile. Das heißt dass auch einem Nachweis bestehende Bauteile eindeutig zugeordnet werden. Eine Verwechslung von den dem Bauteil zugewiesenen Größen kann deshalb nicht mehr auftreten. Der Eurocode selbst stellt die Nachweise als eine Art Liste von Berechnungsformeln zur Verfügung, welche dann jeweils vom Nutzer ausgewählt und für den speziellen Fall angewendet werden.

Durch die Untersuchung der Nachweisführung konnte somit eine wichtige Systematik des Eurocodes erkannt werden. Diese Systematik besteht darin, dass alle im Eurocode enthaltenen Nachweise eine Art Schablone darstellen. Diese Schablone kann für konkrete Anwendungen verwendet werden und ist jederzeit wiederverwendbar. Eine derartige Struktur ist auch für die verschiedenen Bauteile erkennbar, da auch sie je nach Bauteiltyp immer die gleiche Struktur aufweisen. Betrachtet man die Berechnungsformeln für Verbindungsmittel, fällt auf, dass diese von verschiedenen Nachweisen verwendet werden. Trotzdem sind auch diese Formeln nur an einer Stelle im Eurocode vorzufinden. Daran orientiert ist es sinnvoll im späteren Programm mehrmals verwendete Beiwerte und Formeln außerhalb des eigentlichen Nachweises zur Verfügung zu stellen. Da diese Berechnungsformeln ebenfalls in direkter Abhängigkeit mit dem Nachweis stehen, von welchem sie aufgerufen werden, ist auch hier eine eindeutige Zuweisung vorhanden.

Die Struktur der eigentlichen Nachweisführung sowie die Bedingungen sind in der Norm immer im Nachweis selbst vorzufinden. Es macht Sinn, dieses Konzept auch für das spätere Programm zu übernehmen und den Bemessungsablauf selbst in der Nachweisschablone zu speichern.

Zu Beginn der Masterarbeit lag die Umsetzung der Norm in Form einer Datenbank nahe. Die Untersuchung hat jedoch gezeigt, dass der Bemessungsvorgang zu komplex ist und sich nicht für eine solche Speicherung eignet. Um die ermittelte Struktur und den Ablauf der Nachweisführung zu verdeutlichen, zeigt Abbildung 13 einen verallgemeinerten Ablaufplan.

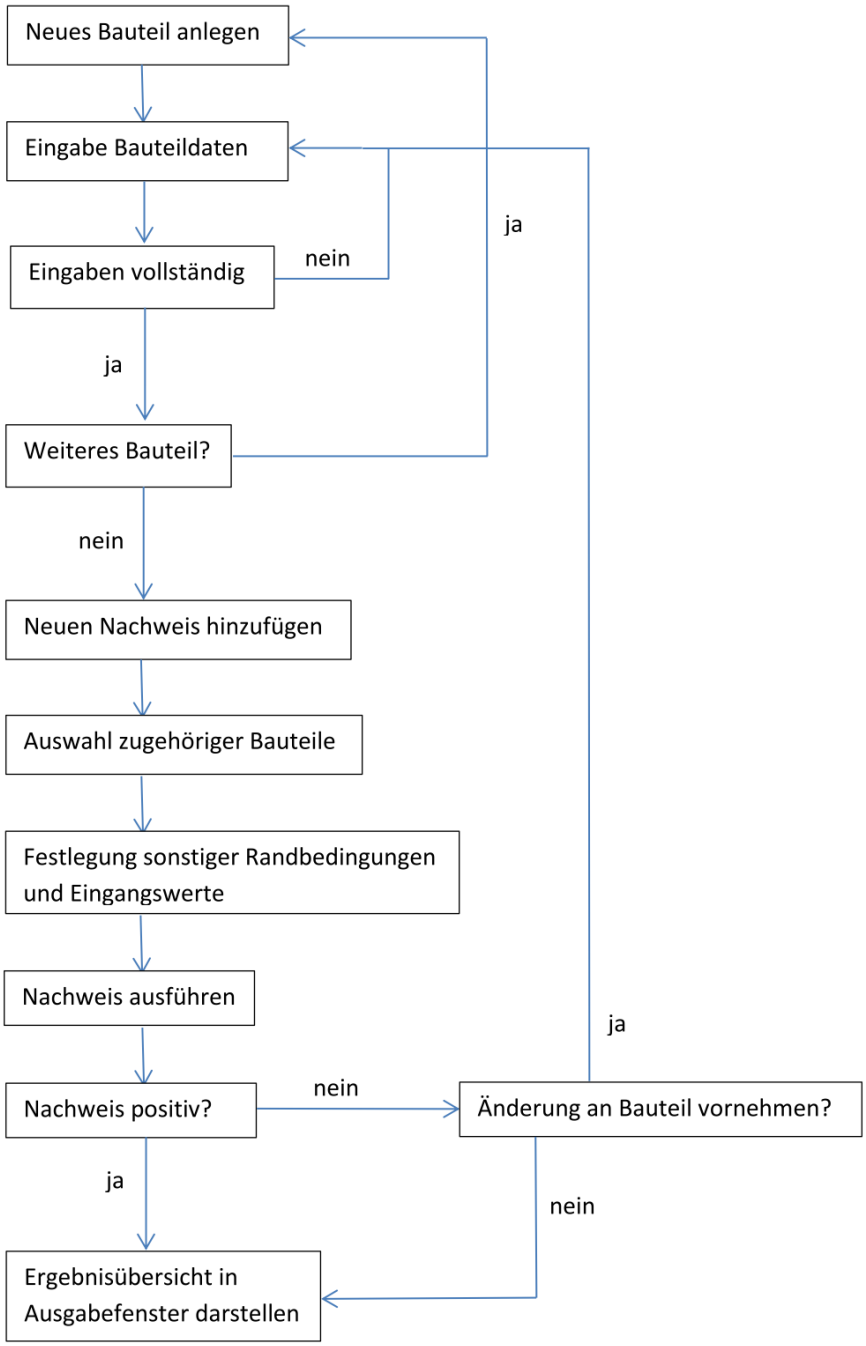


Abbildung 13: Allgemeiner Ablaufplan

Wie in Abbildung 13 zu erkennen ist, werden vorerst ein Bauteil und dessen Eingangsgrößen festgelegt. Um eine erfolgreiche Bemessung durchführen zu können, wird das spätere Programm vor der Berechnung die Vollständigkeit der Eingaben überprüfen. Wenn alle benötigten Bauteile angelegt wurden, wird ein Nachweis hinzugefügt. Diesem müssen die zu verwendenden Bauteile zugeordnet werden. Nachdem die sonstigen Randbedingungen für den Nachweis festgelegt wurden, kann der Nachweis ausgeführt werden. Ist der Nachweis positiv, wird das Ergebnis ausgegeben. Ist der Nachweis nicht erfüllt, können Änderungen am Bauteil vorgenommen werden und der Nachweis wird neu berechnet.

5 Entwurf einer eurocodeorientierten Programmarchitektur

Aus der Struktur der Nachweise und damit des Eurocodes wird nun die Struktur des späteren Programms entwickelt. Die Untersuchung des Eurocodes hat die prinzipielle Systematik und Struktur der Nachweisführung aufgezeigt. Im Folgenden wird die Entwicklung der Programmarchitektur erläutert. Wichtig hierfür sind die bisher erlangten Erkenntnisse bezüglich der einzelnen Komponenten der Bemessung.

5.1 Nachweisstruktur

Neben den Eingangsgrößen aus dem zu bemessenden Bauteil, werden zusätzliche Kenngrößen für die Nachweisführung benötigt. Diese sind von den Kenngrößen des Bauteils und weiteren System- oder Umgebungseigenschaften abhängig. Die Ermittlung dieser Kenngrößen anhand der Norm könnte im späteren Programm mittels Fallunterscheidungen umgesetzt werden. Dies ist möglich, da die Entscheidungen, die für einen Nachweis getroffen werden müssen immer eindeutig sind. Zusätzlich zu den eigentlichen Nachweisen enthält die Norm auch Bedingungen. Diese Bedingungen gilt es im Nachweis ebenfalls zu überprüfen oder deren Erfüllung durch eine entsprechende Textausgabe auf der Eingabeoberfläche sicherzustellen. Um die Übersichtlichkeit bei mehreren Nachweisen zu gewährleisten, soll nicht nur das Ergebnis des Nachweises in Form eines Ausnutzungsgrades, sondern auch ein zugehöriger Status ausgegeben werden. Dieser Status gibt beispielsweise an, ob der Nachweis erfüllt ist oder nicht und ob er nach den aktuellen Eingaben bereits berechnet wurde oder nicht.

Fasst man die ermittelten Eigenschaften des Nachweises zusammen, kann die folgende Struktur erstellt werden.

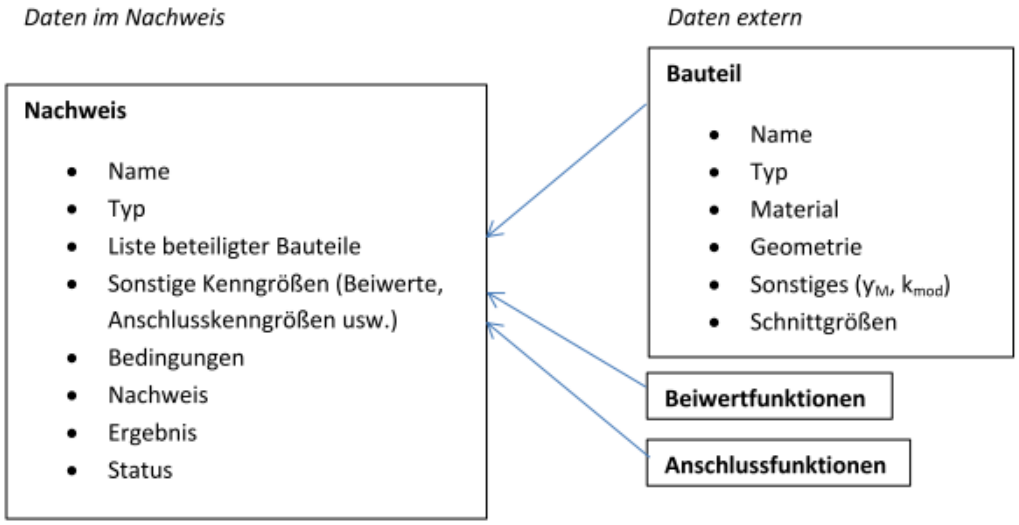


Abbildung 14: Nachweisstrukur

Der Nachweis ist je nach Nachweistyp unterschiedlich umfangreich. Bei Anschlussnachweisen werden beispielsweise weitere Informationen über die Verbindungsmittel benötigt. Der erste Gedanke bei der Umsetzung der Nachweise war, diese in Form von Listen, ähnlich der Norm zu speichern. In diesem Fall werden zum Beispiel Berechnungsformeln in Excel-Dateien gespeichert. Die eigentliche

Bemessung findet ebenfalls in Excel statt. Bei der Nachweisführung wird dann jeweils eine Formel aufgerufen, deren Elemente teilweise durch Verweise wieder anhand anderer Formeln berechnet werden. Durch die Untersuchung wurde jedoch deutlich wie komplex die Norm ist, da sie nicht nur aus Formeln und Fixwerten besteht, sondern auch aus Bedingungen, Erläuterungen und zusätzlichen Anweisungen. Die Arbeit mit Excel erschien daher schnell zu unübersichtlich und damit nicht gut geeignet um die komplexe Logik des Eurocodes umzusetzen. Durch die grundlegenden Gemeinsamkeiten aller Nachweise wurde deutlich, dass es Sinn macht eine allgemeine Schablone für alle Nachweise zu erstellen. In der späteren Programmierung kann dies durch die Implementierung einer Basisklasse realisiert werden. Diese Basis- oder auch Superklasse „Nachweis“ gibt die Grundstruktur aller Nachweise vor. In jedem Nachweis sollten eine Liste der beteiligten Bauteile, die Normwerte, eine Nachweisfunktion, das Ergebnis und der Status vorhanden sein. Zudem besitzt jeder Nachweis einen Typ, der angibt, um welche Nachweisschablone es sich handelt. Da es im späteren Programm durchaus möglich ist, dass mehrere Nachweise des gleichen Typs erstellt werden, sollte der Nachweis zudem einen eindeutigen Namen zur Identifizierung besitzen. Die Nachweisfunktion dient der eigentlichen Berechnung des Nachweises und enthält die Implementierung der Normvorgaben. Als Ergebnis dient der Ausnutzungsgrad. Der Status gibt den aktuellen Stand der Berechnung an.

Um die verschiedenen Nachweistypen im späteren Programm zu repräsentieren, ist es sinnvoll für jeden einzelnen Nachweis der Norm eine eigene von der Basisklasse abgeleitete Klasse zu schreiben. In diesen abgeleiteten Klassen werden die, für den Nachweis spezifischen, bereits erwähnten Bedingungen, Formeln und Beiwerte und Fallunterscheidungen gespeichert. Zudem werden jedem Nachweis ein oder mehrere Objekte vom Typ Bauteil zugewiesen. Eine weitere Gruppierung der Nachweise in der späteren Benutzeroberfläche beispielsweise nach Grenzzuständen der Tragfähigkeit und Gebrauchstauglichkeit oder nach der Bauteilart ist zudem denkbar. Die verwendete objektorientierte Struktur bietet eine robuste Basis für die Berechnung, ermöglicht jedoch auch gleichzeitig jederzeit weitere Nachweise anhand der vorgegebenen Struktur hinzuzufügen.

Die aus der Norm stammenden Tabellen und Textzeilen, die für die Nachweisführung benötigt werden, sollen unabhängig von den Nachweisen an nur einer Stelle des Programms gespeichert werden und mittels Funktionsaufrufen für die Nachweise verfügbar sein. Der Grund liegt darin, dass diese von verschiedenen Nachweisen verwendet werden. Durch die Speicherung aller Beiwerte und der zugehörigen Berechnungen an einer zentralen Stelle, müssen Änderungen nur einmalig vorgenommen werden. Dadurch wird eine robuste Struktur zur Verfügung gestellt, die einer redundanten und damit fehleranfälligen Datenspeicherung vorbeugt. Während den ersten Überlegungen schien es denkbar, dass diese Beiwerte in Form eines Datenblatts in Excel zur Verfügung stehen und jederzeit verändert werden können. Die Liste „Beiwerte“ würde in diesem Fall jeweils zu Beginn des Programms neu geladen werden. So wird sichergestellt, dass immer die aktuellen Beiwertbestimmungen verwendet werden. Sinnvoller ist es jedoch die Funktionen innerhalb einer eigenen Klasse zu speichern. Dies liegt daran, dass die Berechnungen selbst im Rahmen einer objektorientierten Programmstruktur umgesetzt werden sollen. Durch diese Struktur besitzen die verwendeten Variablen oft einen direkten Bezug auf ein bestehendes Objekt. Dieser Bezug würde durch die Verwendung von in Excel zur Verfügung gestellten Formeln verloren gehen. In der Klasse der Funktionen und Berechnungsformeln kann dagegen direkt Bezug auf die Bestandteile des Bauteils genommen werden. Diese Thematik

verdeutlicht, welchen Vorteil eine objektorientierte Programmierung besitzt. Da die Funktionen zur Beiwertberechnung stets im Nachweis selbst aufgerufen werden, besitzen sie einen Bezug auf ein spezifisches Bauteil und können aufgrund der im Nachweis definierten Übergabeparameter nachweisspezifisch gesteuert werden. Die nachweisspezifischen Bedingungen, welche in der Norm oft in Textform vorgegeben sind werden wie bereits erwähnt im Nachweis selbst geregelt.

Anschlussnachweise werden ebenfalls als Einzelnachweise strukturiert. Für jede Anschlussart gibt es eine eigene von der Basisklasse „Nachweis“ abgeleitete Klasse. In dieser Klasse sind ebenfalls die Bedingungen und die eigentliche Berechnung gespeichert. Eine Besonderheit dabei ist, dass durch diese Art der Strukturierung, einige Berechnungsformeln von Anschluss-Kennwerten mehrfach in verschiedenen Klassen verwendet werden. Ähnlich wie bei den anderen Beiwerten soll eine eigene Klasse „AnschlussFunktionen“ erstellt werden, in der die Berechnungsformeln redundanzfrei gespeichert werden. Bei Änderungen der Formeln, können diese in der entsprechenden Klasse vorgenommen werden und werden durch den Aufruf aus den Nachweisen automatisch übernommen. Die Anschlussfunktionen können dann je nach Bedarf von den einzelnen Nachweisen aufgerufen werden. Die Trennung der allgemeinen Beiwerte und deren zur Anschlussberechnung benötigten Beiwerte orientiert sich dabei allein an der Gliederung des Eurocodes und könnte auch anders gestaltet werden.

Damit der Nutzer selbst die auszuführenden Nachweise auswählen kann, wird die Klasse „Nachweispaket“ erstellt. Mithilfe dieser Klasse soll jeder Anwender die Möglichkeit haben die Einzelnachweise benutzerdefiniert zusammenzufassen. Denkbar ist zudem, die Nachweispakete zu speichern um sie später wieder verwenden zu können. Eine mögliche Überlegung hierzu wäre die Speicherung einer reinen Nachweispaketschablone. So kann ein Paket zur Verfügung gestellt werden, das Nachweise jedoch noch keine Eingabedaten enthält.

5.2 Bauteilstruktur

Die richtige Struktur der Bauteile ist vor allem deswegen sehr wichtig, da die meisten für die Bemessung benötigten Informationen direkt vom nachzuweisenden Bauteil stammen oder zumindest in Abhängigkeit zu diesem stehen. Diese Informationen müssen daher übersichtlich und eindeutig zur Verfügung stehen. Um die Eindeutigkeit zu gewährleisten, muss jedem Bauteil ein eindeutiger Name zur Identifizierung zugewiesen werden. Für die Vollständigkeit und Definition eines Bauteils muss ebenfalls dessen Typ bekannt sein. Hierfür wird die Basisklasse „Bauteil“ eingeführt. Sie stellt die gemeinsame Struktur aller Bauteile zur Verfügung.

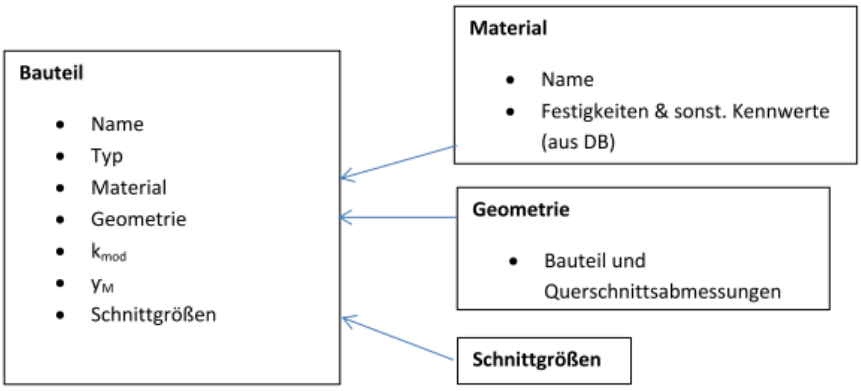


Abbildung 15: Allgemeine Bauteilstruktur

Neben Name und Typ des Bauteils werden auch die für die Tragfähigkeit erforderlichen Beiwerte γ_M und k_{mod} bei der Erstellung des Bauteils anhand der Memberfunktionen der „Beiwerte"-Klasse bestimmt. Da diese Werte mehrmals verwendet werden, werden sie nach vollständiger Eingabe der Bauteildaten einmalig berechnet und anschließend im Bauteil selbst gespeichert. Zusätzlich werden Daten zu Material, Geometrie und Schnittgrößen benötigt, für die ebenfalls jeweils eine eigene Klasse erstellt wird. In der Klasse „Material" werden neben der vom Nutzer im Eingabefenster durch den Namen festgelegten Art des Werkstoffes auch die sonstigen Kenngrößen des Materials gespeichert. Dabei werden Daten wie Festigkeiten, Dichte und E-Modul anhand des Namens automatisch aus einer Tabelle herausgelesen. Um dies gewährleisten zu können muss auch der Name des Materials eindeutig sein. Die materialspezifischen Daten werden in Form von Excel-Datenblättern gespeichert und können somit jederzeit durch weitere Materialien ergänzt werden. Die Klasse „Geometrie" enthält die für die Bemessung relevanten Abmessungen. Gerade bei der Anschlussbemessung wurde deutlich, dass sich manche Nachweise auch direkt auf die Verbindungsmittel, beziehungsweise eine Anschlussart beziehen und hier für die Bemessung entsprechend weitere eindeutige Kenngrößen benötigt werden. Daher werden neben den Bauteilen aus Holz auch Verbindungsmittel als Bauteile zur Verfügung gestellt. Da diese sich in ihrer prinzipiellen Art und dem Umfang der eingebrachten Informationen jedoch deutlich von anderen Bauteilen unterscheiden, scheint es sinnvoll die Bauteile in Holzbauteile und Verbindungsmittelbauteile zu unterscheiden. Diese Idee wird im späteren Programm mit den beiden von der Basisklasse „Bauteil" abgeleiteten Klassen „HolzBauteil" und „VBMBauteil" umgesetzt. Die für das allgemeine Bauteil entwickelte Struktur wird entsprechend weiter unterteilt, um eine Spezialisierung nach Bauteilart zu gewährleisten. Die beiden Klassen „Material" und „Geometrie" dienen daher ebenfalls als Basisklassen, sodass für die abgeleiteten Klassen „HolzBauteil" und „VBMBauteil" jeweils eigene Geometrie- und Materialklassen erstellt werden. Dies liegt daran, dass je nachdem ob es sich um ein Holzbauteil, oder um ein Verbindungsmittel handelt, sehr verschiedene Größen in den beiden Klassen gespeichert werden. Abbildung 16 zeigt die Struktur eines Holzbauteils. Um den Unterschied zwischen allgemeinen Elementen und Spezialisierungen zu symbolisieren, sind alle spezialisierten Elemente kursiv dargestellt.

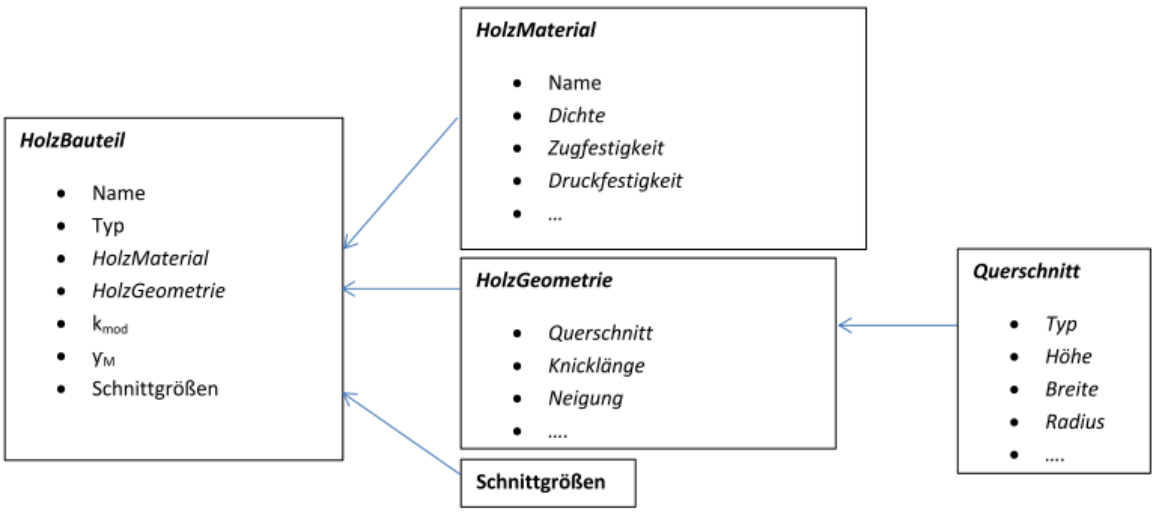


Abbildung 16: Struktur eines Holzbauteils

Die Klasse „HolzGeometrie" neben Parametern, wie der Knicklänge auch noch ein Objekt der Klasse „Querschnitt". Die Klasse „Querschnitt" hat wiederum einen eindeutigen Querschnittstyp. Abhängig vom Typ werden die für die Bemessung benötigten Querschnittsabmessungen entweder mit einem Wert belegt oder, falls für diesen Querschnittstyp nicht benötigt, in der Eingabemaske des Programms ausgegraut und nicht weiter verwendet. In der Praxis heißt dies, dass für ein Bauteil mit einem Rechteckquerschnitt kein Radius einzugeben ist, dafür jedoch Höhe und Breite des Querschnitts. Auch die weiteren Querschnittsgrößen, wie das Trägheitsmoment, berechnen sich je nach Querschnittstyp unterschiedlich und machen somit eine Fallunterscheidung im Programm nach Typ notwendig. Wie bereits angedeutet, sollen in der allgemeinen Klasse Querschnitt alle möglichen Kennwerte zur Verfügung gestellt werden und je nach Bedarf verwendet werden. Eine weitere Spezialisierung mittels abgeleiteter Klassen für die verschiedenen Querschnittstypen ist ebenfalls denkbar. Die entwickelte Struktur ermöglicht zudem die Speicherung der Querschnittstypen und ihrer Abmessungen in Form von Standardquerschnitten in Excel-Datenblättern. Wie auch die Materialtabellen können diese somit einfach eingesehen und ständig um neue Querschnitte erweitert werden. Wichtig ist dabei, dass der Querschnittstyp ebenfalls darin gespeichert ist.

In der Klasse „Geometrie" sind neben dem Querschnitt zusätzliche Daten gespeichert, die je nach Bauteiltyp mit Werten belegt oder leer gelassen werden können. Eine weitere Unterteilung der Verbindungsmittel- und Holzbauteilgeometrien nach Bauteiltyp ist ebenfalls denkbar. Um im Nachweis selbst nicht nach Bauteiltyp unterscheiden zu müssen, wurde die verallgemeinernde Variante gewählt. Sie ermöglicht eine einfache Geometrie und orientiert sich nahe am Eurocode. Die Klasse „Bauteil" besitzt zusätzlich ein Objekt der Klasse „Schnittgrößen" für die Speicherung der Schnittgrößen. Für die Vorgabe der Schnittgrößen ist ebenfalls wieder die Art des Bauteils wichtig. Hier findet erneut eine Verallgemeinerung statt, indem alle möglichen Größen vorgegeben werden und je nach Typ belegt werden. Auch hier ist eine Spezialisierung nach Bauteiltyp denkbar.

In der abgeleiteten Klasse „VBMBauteil" sollen keine zusätzlichen Objekte der Klasse „Querschnitt" gespeichert werden. In den zugehörigen Material- und Geometrieklassen sind jedoch andere Kennwerte gespeichert wie in den Holzbauteilklassen.

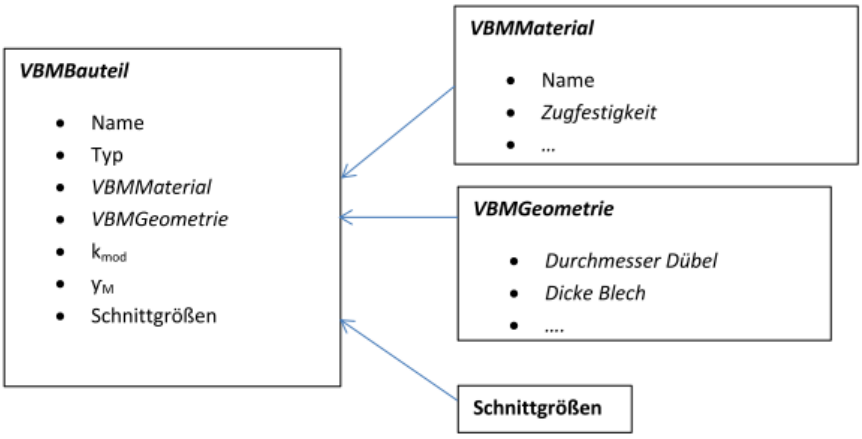


Abbildung 17: Struktur eines Verbindungsmittelbauteils

Bei der Anschlussbemessung werden die vom Anwender eingegebenen Schnittgrößen des Holzbauteils verwendet, um die Kräfte innerhalb des Anschlusses zu berechnen. Dem Verbindungsmittel selbst werden keine eigenen Schnittgrößen mitgegeben, da diese von weiteren Größen, wie der Anzahl der Verbindungsmittel abhängen. Die eigentlich im Verbindungsmittel vorhandenen Schnittgrößen werden daher erst im Nachweis selbst berechnet und können diesem dann zugeordnet werden.

5.3 Programmstruktur

Während der Untersuchung der Beispiele und des prinzipiellen Bemessungsvorgangs wurde deutlich, dass sich die Speicherung ganzer Nachweise oder Bauteile in Excel oder einer Datenbank nicht eignet. Das zu entwickelnde Programm soll über eine benutzerfreundliche Eingabeoberfläche zur Dateneingabe verfügen. Eine Eingabe der Daten in Datenblättern erwies sich als unübersichtlich und fehleranfällig. Nur die Nachweisformeln in Form von Excel-Datenblättern zu speichern würde ebenfalls schnell unübersichtlich und auch die Zuweisung der Bauteile und Zusammenstellung der Bemessung wäre komplex und damit fehleranfällig. Verweise auf Bauteile müssten manuell vorgenommen werden und sämtliche Kennwerte müssten eindeutig definiert werden. Sollte die Berechnung sowieso nicht in Excel sondern im Programm selbst stattfinden, ist die Definition der Nachweise in Excel ungeeignet, da diese dann ohnehin in das Programm eingelesen werden müssten. Ein Programm das auf einer objektorientierten Sprache basiert, scheint dagegen optimal. Die Nachweise und Bauteile erhalten anhand von Basisklassen klare Strukturvorgaben, können jedoch in den abgeleiteten Klassen beliebig erweitert werden und zur Initialisierung beliebig vieler Objekte als Schablone dienen. Durch diese klar vorgegebene Struktur, die jederzeit Erweiterungen zulässt, wird zudem der gewünschte modulare Charakter des Programms wiedergespiegelt. Durch eine eindeutige Zuweisung der Nachweise zu einem oder mehreren Objekten des Typs „Bauteil" ist auch die Verwendung der darin enthaltenen Kenngrößen eindeutig. Damit ist das Programm stabil und verfügt über eine sichere Struktur.

Kritisch zu sehen ist die feste Definition der nachweisspezifischen Formeln, Beiwerte und Randbedingungen innerhalb der Nachweise. Dies ist jedoch notwendig, um der Struktur des Eurocodes zu entsprechen. Mehrfach verwendete Beiwerte und Anschlussfunktionen werden in einer eigenen Klasse festgelegt. Hier sind zum einen fixe Werte gespeichert und zum anderen Funktionen zur Berechnung von benötigten Beiwerten. Durch diese Struktur wird sichergestellt, dass die Formeln

nicht mehrfach in verschiedenen Nachweisen aufgeführt sind, Abänderungen der Funktionen nur an einer Speicherstelle erfolgen müssen und diese sofort für alle Nachweise übernommen werden. Die Funktionen können zum Beispiel die Bestimmung von k_{mod} oder von γ_M, aber auch sonstige, nicht einfach durch einen fixen Zahlenwert definierte Kennwerte sein. In Form von Datenblättern könnten vor allem reine Zahlenwerte wie Schnittgrößen, Materialkennwerte und Querschnittswerte vorgegeben werden. Die Speicherung dieser in leicht zugänglichen Excel-Dateien entspricht dem gewünschten offenen und modularen Charakter des Programms.

Das zu entwickelnde Berechnungsprogramm selbst soll durch eine eigene Klasse repräsentiert werden. Beim Start des Programms wird eine neue Instanz dieser Klasse „Programm" erstellt. Um wie gewünscht ein multifunktionales Berechnungsprogramm umzusetzen, soll der Nutzer nach seinen Anforderungen Bauteile und Nachweise erstellen können. Die Nachweise können zudem zu Nachweispaketen thematisch zusammengefasst werden. Die Klasse „Programm" besitzt zum Speichern der Bauteile und Nachweispakete zwei Listen. Diese sollen nach und nach mit Hilfe klasseneigener Membermethoden aufgefüllt werden. Zudem soll das Programm Methoden zur Ergebnisausgabe und zur Darstellung einer Übersicht der bereits erstellten Bauteile und Nachweispakete besitzen. Um die erstellten Bauteile und Nachweise wiederverwenden zu können, sollte es zudem möglich sein die vorhandenen Elemente zu speichern. Des Weiteren besitzt die Klasse „Programm" zwei Objekte vom Typ „Anschlussfunktionen" und „Beiwerte", welche beim Start des Programms erstellt werden und alle benötigten Zusatzfunktionen zur Verfügung stellen. Wie in den untersuchten Programmen, wird dem Nutzer auch in diesem Programm die Möglichkeit geboten nach der Bemessung die Ergebnisse auszugeben und zu speichern.

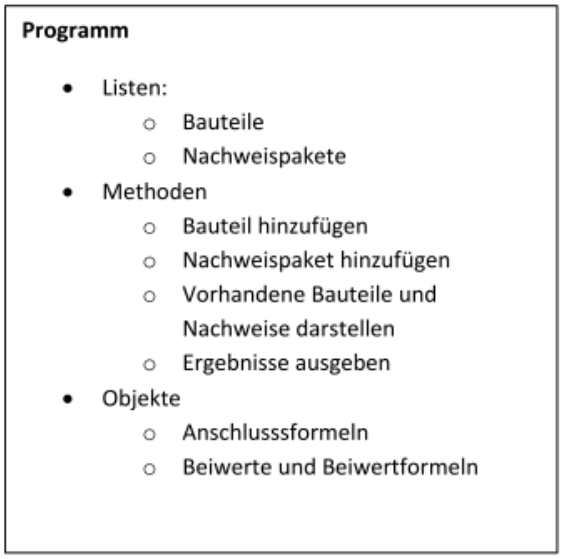


Abbildung 18: Programmstruktur

Aus den entwickelten Strukturen lassen sich damit, die auf den folgenden Seiten als UML-Diagramme dargestellten vorläufigen Klassenstrukturen für das Programm und die Material- und Geometrieklasse als Entwurf für eine Programmarchitektur ableiten. Die einzelnen Klassen werden im UML-Diagramm als Rechtecke dargestellt. In diesem Rechteck können neben dem Namen der Klasse auch deren Eigenschaften, Felder und Methoden dargestellt werden. Wie zu erkennen ist, existiert im UML-

Diagramm des Materials die Klasse „Material“, welche lediglich einen Namen als Eigenschaft besitzt. Davon abgeleitet besitzt das Programm die beiden spezialisierten Klassen „HolzMaterial“ und „VBMMaterial“ (siehe Abbildung 19). Im Klassendiagramm wird die Vererbung durch einen zur Basisklasse gerichteten Pfeil dargestellt. Beide Klassen besitzen neben dem Namen aus der Basisklasse zusätzliche typspezifische Eigenschaften.

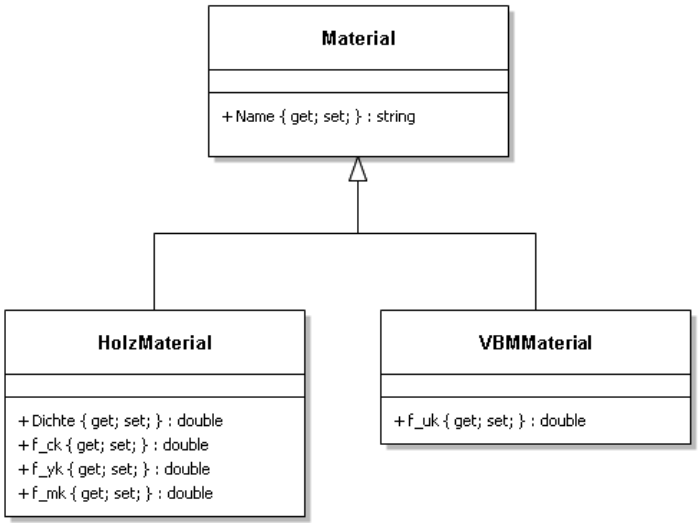


Abbildung 19: UML-Diagramm des Materials

Bei der Klasse „Geometrie“ ist dies ähnlich. Sie wird vor allem als Vorgabe in der Superklasse „Bauteil“ verwendet, wodurch sichergestellt wird, dass jedes Bauteil eine Geometrie besitzt. Die beiden von der Klasse „Geometrie“ abgeleiteten Klassen besitzen wiederum spezifische Eigenschaften für die jeweilige Art von Bauteilen. Die Klasse „HolzGeometrie“ besitzt neben Eigenschaften wie der Knicklänge und der Lamellendicke auch ein Objekt der Klasse „Querschnitt“. Im UML-Diagramm wird diese Beziehung als Aggregation durch eine Raute dargestellt (siehe Abbildung 20). Die Pfeilrichtung gibt an, dass ein Objekt der Klasse „Querschnitt“ einem Objekt der Klasse „HolzGeometrie“ zugeordnet wird und nicht umgekehrt.

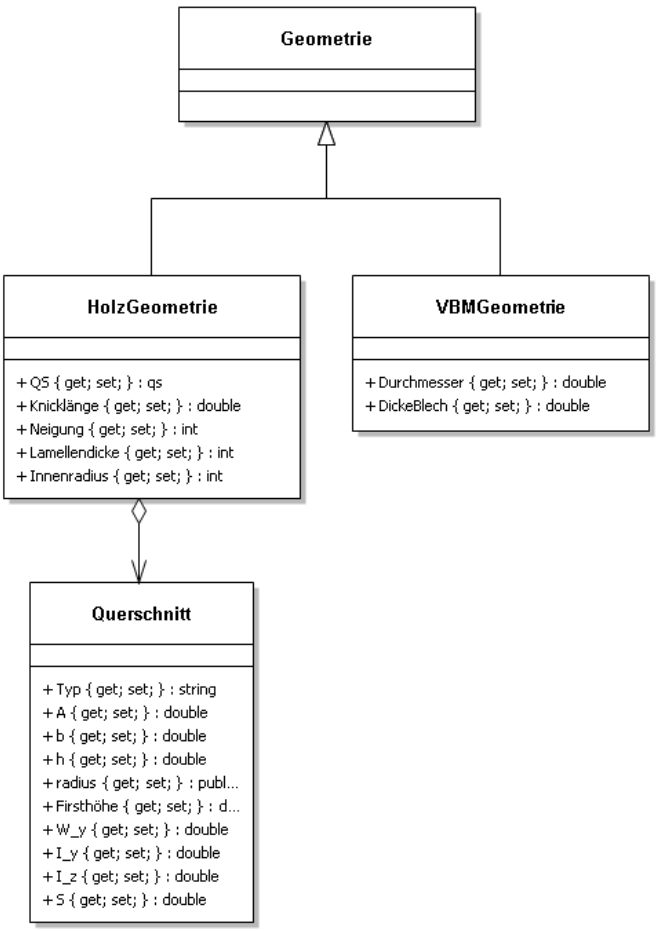


Abbildung 20: UML-Diagramm der Geometrie

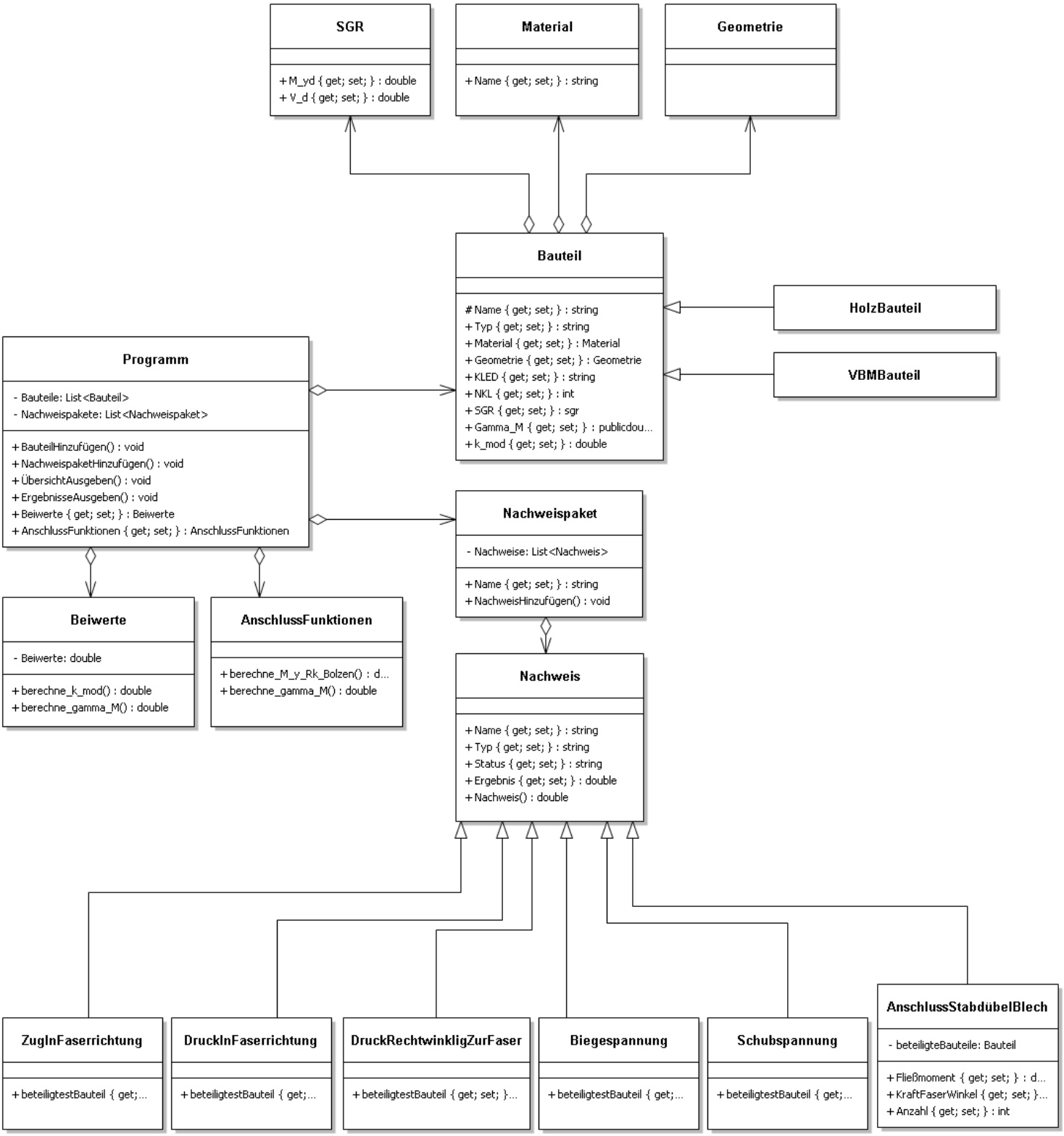


Abbildung 21: UML-Diagramm der Programmstruktur

Abbildung 21 stellt das UML-Diagramm der Programmstruktur dar. Die Klasse" Programm" besitzt eine Liste mit „Bauteil"-Objekten und eine Liste mit „Nachweispaket"-Objekten. Zudem besitzt sie jeweils ein Objekt der beiden Klassen „Beiwerte" und „AnschlussFunktionen". Die Klasse „Beiwerte"

soll neben Fixwerten wie die Klasse „AnschlussFunktionen“ auch über Funktionen zur Berechnung der Normwerte verfügen. Als Membermethoden des Programms sollen zudem noch Funktionen zum Hinzufügen von Bauteilen und Nachweispaketen, sowie zur Darstellung einer Übersicht und Ausgabe der Ergebnisse umgesetzt werden. Die Klasse „Programm“ verwendet durch die Listen auch die beiden Klassen „Bauteil“ und „Nachweispaket“. In dem dargestellten Klassendiagramm ist nur die verallgemeinerte Struktur der Bauteile dargestellt. Wie zu sehen ist besitzt jedes Bauteil neben einem eindeutigen Namen und einem Bauteiltyp auch jeweils ein Objekt der Klassen „SGR“, „Material“ und „Geometrie“. Zusätzlich soll in jedem Bauteil die Nutzungsklasse und die Klasse der Lasteinwirkungsdauer gespeichert werden. Die Beiwerte k_{mod} und y_M werden aus den sonstigen Bauteilkenngrößen anhand der „Beiwerte“-Klasse ermittelt und ebenfalls im „Bauteil“ als Eigenschaft gespeichert. Als von der Klasse „Bauteil“ abgeleitete Klassen existieren die beiden Klassen „HolzBauteil“ und „VBMBauteil“ mit ihren jeweiligen Spezialisierungen. Die Klasse „Nachweispaket“ besitzt neben einer Liste mit Nachweisen einen eindeutigen Namen zur Identifizierung und eine Methode zum Hinzufügen neuer Nachweise. Die Klasse „Nachweis“ dient als Basisklasse für alle Nachweisklassen. Die Nachweise können sich in ihren Eigenschaften stark unterscheiden. Die meisten Nachweise besitzen lediglich ein Objekt vom Typ „Bauteil“. Ein Anschlussnachweis benötigt dagegen zusätzlich noch zahlreiche Informationen. Die Eigenschaften die von allen Nachweisen geteilt werden, sind in der Superklasse „Nachweis“ zu finden. Sie besitzt neben einem Namen auch einen Status, ein Ergebnis und eine Nachweisfunktion.

Nachdem nun eine grundlegende Programmarchitektur entworfen wurde, wird ein Programmablauf aus der Sicht des Anwenders simuliert.

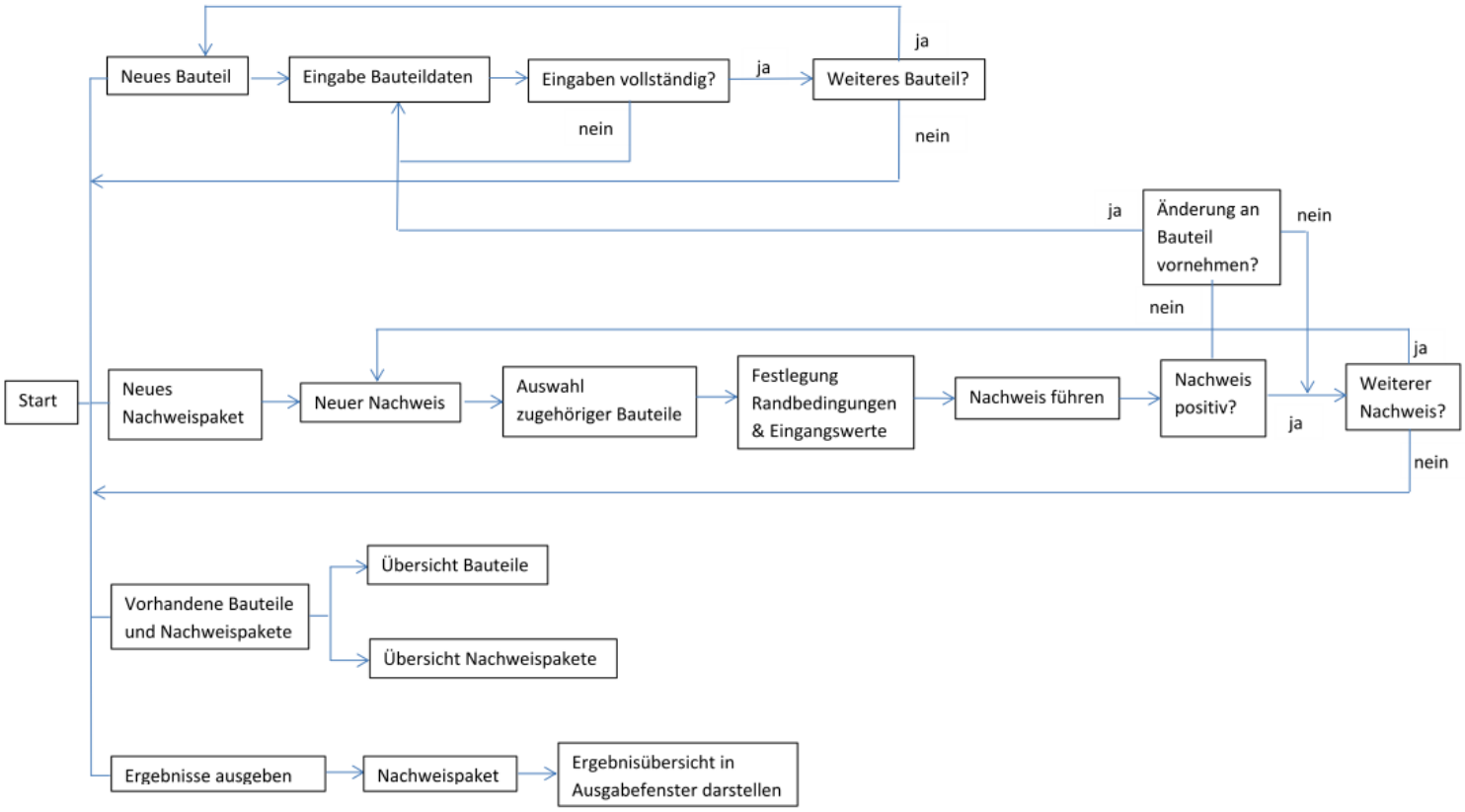


Abbildung 22: Ablaufdiagramm

Beim Start des Programms soll der Anwender folgende Möglichkeiten haben:

1. Ein neues Bauteil hinzufügen
2. Ein neues Nachweispaket hinzufügen
3. Die vorhandenen Bauteile und Nachweispakete anzeigen
4. Die Berechnungsergebnisse ausgeben

1. Soll ein neues Bauteil hinzugeführt werden, erfolgt die Eingabe der Bauteildaten durch den Benutzer auf der Eingabeoberfläche. Nach der Bestätigung der Eingaben werden diese automatisch auf ihr notwendige Vollständigkeit hin überprüft und der Nutzer gelangt bei unvollständiger Dateneingabe wieder bei der Eingabeaufforderung. Sind die eingegebenen Daten vollständig, kann entschieden werden ob ein weiteres Bauteil angelegt werden soll, oder ob der Nutzer zum Hauptmenü zurückgelangen möchte.

2. Bei der Entscheidung für das Anlegen eines neuen Nachweispaketes, können diesem nach Vergabe eines Namens, verschiedene Nachweise hinzugefügt werden. Nach der Wahl eines Nachweises müssen die beteiligten Bauteile aus der Liste der vorhandenen Bauteile ausgewählt werden. Anschließend müssen vom Nutzer zusätzliche Randbedingungen und Eingangswerte vorgegeben werden, bevor die Berechnung gestartet werden kann. Auch hier wird wieder überprüft, ob alle

Eingaben vollständig sind und der Nutzer wird in dem entsprechenden Fall aufgefordert die Eingaben zu vervollständigen. Nach der Berechnung werden die daraus resultierenden Ergebnisse angezeigt. Sollte der Nutzer sich entscheiden aufgrund der Ergebnisse Änderungen am Bauteil vornehmen zu wollen, wird er in das Bauteilmenü weitergeleitet. Sollen keine Änderungen vorgenommen werden, hat der Nutzer die Wahl zwischen dem Hinzufügen wieterer Nachweise zum Nachweispaket und dem Zurückkehren zum Hauptmenü.

3. Die dritte Möglichkeit im Hauptmenü ist, die bereits erstellten Bauteile oder Nachweispakete anzeigen zu lassen. Je nachdem ob der Anwender Bauteile oder Nachweispakete angezeigt bekommen möchte, kann er die einzelnen Bauteile und Nachweise auswählen und sich deren Informationen anzeigen lassen oder in deren Bearbeitungsmenüs gelangen.

4. Zuletzt kann sich der Nutzer die Ergebnisse der Nachweise in übersichtlicher Form ausgeben lassen. Hier findet eine benutzerdefinierte Filterung der Ergebnisse statt, indem zwischen den einzelnen Nachweispaketen gewählt werden kann.

Nachdem nun bereits eine erste Programmstruktur und ein Programmablauf entwickelt wurden, werden im nächsten Schritt die passende Programmiersprache und -oberfläche ermittelt um anschließend mit der Umsetzung des Programms fortzufahren.

6 Programmiersprache und -oberfläche

6.1 Programmiersprache

In der oben beschriebenen geplanten Programmarchitektur wurde ein objektorientierter Ansatz gewählt, um die in der Nachweisberechnung gegebene Struktur optimal umzusetzen. Die objektorientierte Programmierung beruht auf der Idee, dass ein Modell in einzelne Objekte unterteilt werden kann. Dabei ist jedes Objekt eine Instanz einer Klasse. Die Klasse gibt die Struktur und das Verhalten des Objekts vor. Die objektorientierte Programmierung unterstützt das Prinzip der Vererbung, welches in diesem Programm verwendet werden soll. Die Vererbung spiegelt sich durch die Verwendung von Basisklassen und abgeleiteten Klassen wieder. Die Basisklasse stellt dabei die Grundstruktur beispielsweise des Bauteils dar. Die davon abgeleitete Klasse „HolzBauteil“ dagegen ist eine Spezialisierung und enthält neben den Informationen der Basisklasse „Bauteil“ weitere Attribute. Zur Umsetzung des Programmes wird also eine objektorientierte Programmiersprache benötigt. Zu den meistverbreiteten Programmiersprachen gehören C#, C++ und Java. Zur Realisierung des Programms wird die auf Windows-Plattformen meistgenutzte Sprache C# verwendet. C# ist eine .NET-Sprache und stellt damit eine umfangreiche Sammlung an Klassenbibliotheken, Programmierschnittstellen und Dienstprogrammen zur Verfügung. Sie ermöglicht das Verwenden von vorhandenen Bibliotheken wodurch beispielsweise die Gestaltung einer grafischen Benutzeroberfläche stark vereinfacht wird. C# ist eine durchgehend objektorientierte Sprache, die durch ihre mitgelieferten Funktionen eine sichere und intuitive Umgebung liefert und zudem schnell erlernbar ist.

6.2 Programmieroberfläche

Als Programmieroberfläche soll die von Microsoft zur Verfügung gestellte Plattform Microsoft Visual Studio dienen. Sie stellt eine anwenderfreundliche Entwicklungsumgebung für .NET-Programme dar. Neben den zahlreichen vorhandenen Projektvorlagen unterstützt Visual Studio auch verschiedene Programmiersprachen. Visual Studio stellt dem Entwickler zahlreiche Tools und Dienste zur Verfügung um diverse Anwendungen für verschiedene Umgebungen umzusetzen.

Da das Programm ein modulares Programm werden soll, das jederzeit durch weitere Komponenten erweitert werden kann, ist eine einfache Verwendung auch bei geringen Programmierkenntnissen unerlässlich. In Visual Studio werden die einzelnen Komponenten einer Projektmappe, die Projekte und Klassen, in übersichtlicher Form im Projektmappenexplorer dargestellt. Auch der Code selbst ist mit Hilfe entsprechender Farbgebung und automatischer Formatierung klar strukturiert. Nicht verwendete Variablen oder Fehler im Code werden sofort markiert und durch Kommentare des Programms erläutert. Mit dem integrierten Debugger werden Warnungen und Fehlermeldungen direkt im entsprechenden Ausgabefenster aufgelistet und erklärt. Auch die Gestaltung der grafischen Benutzeroberfläche ist einfach und anwenderfreundlich. Visual Studio bietet die Möglichkeit GUIs per Drag & Drop-Prinzip zusammenzustellen. Durch Doppelklick auf die entsprechende Komponente kann automatisch der zugehörige Coderumpf erstellt werden. Dies ermöglicht dem Nutzer einfach neue Module zu implementieren.

7 Umsetzung des ersten Programms

Zur exemplarischen Umsetzung des Programms dienen die beiden oben beschriebenen Beispiele. Dabei ist zu beachten, dass es sich um eine untersuchende Programmentwicklung handelt, die eine beispielhafte Struktur und Idee veranschaulicht. Ziel ist es die Möglichkeit eines selbstentwickelten, multifunktionalen, am Eurocode orientierten Programms zu untersuchen. Deshalb werden neben der exemplarischen Umsetzung auch theoretische Erweiterungsideen vorgestellt.

Die letztendlich umgesetzte Programmstruktur und das entwickelte Programm werden zunächst grob anhand eines UML-Diagramms erläutert. Im Anschluss daran wird das Bemessungsbeispiel des Satteldachträgers verwendet um das Programm und dessen Anwendung genauer zu versanschaulichen. Schließlich werden der programmiertechnische Hintergrund, die Umsetzung und die dabei aufgetretenen Herausforderungen genauer diskutiert.

7.1 Programmstruktur

Abbildung 23 zeigt im UML-Diagramm die umgesetzte Programmstruktur.

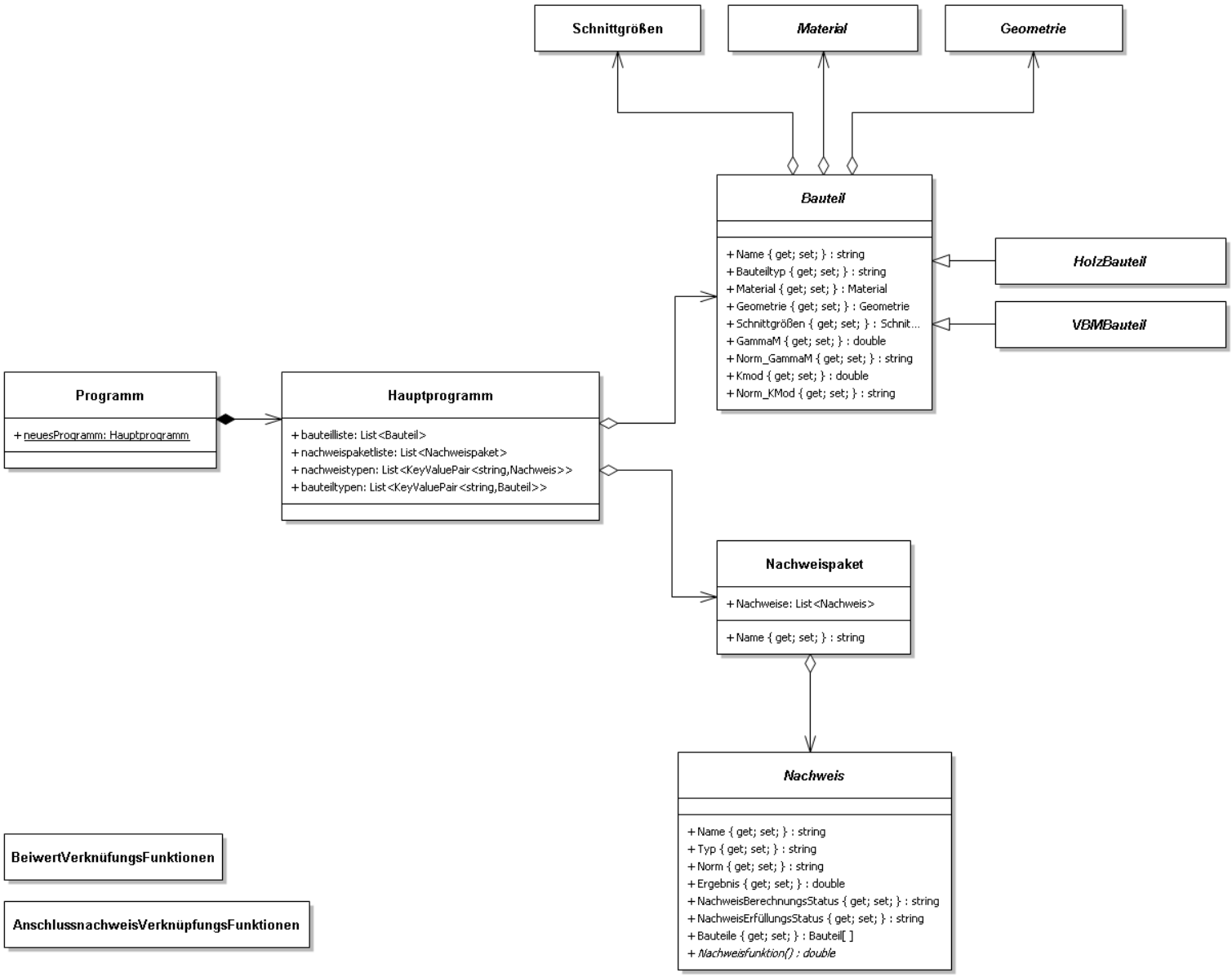


Abbildung 23: UML-Diagramm des umgesetzten Programms

Das „Hauptprogramm“ stellt den eigentlichen Kern des NachweisTools dar. Hier findet die Verwaltung der Bauteile und Nachweise statt. Das Programm startet und erstellt eine neue Instanz der Klasse „Hauptprogramm“. Die Klasse „Hauptprogramm“ besitzt eine Liste mit Bauteilen und eine Liste mit Nachweispaketen und entspricht der Klasse „Programm“ im vorangegangenen Entwurf. Zudem besitzt sie, anders als im Entwurf Listen in welchen die Nachweistypen und Bauteiltypen gespeichert werden. Die Bauteilklasse ist eine abstrakte Klasse, das heißt es kann kein Objekt dieser Klasse erzeugt werden. Im UML-Diagramm wird diese Eigenschaft durch den kursiv geschriebenen Klassennamen symbolisiert. Die Klasse „Bauteil“ dient wie geplant als Basisklasse für die beiden davon abgeleiteten Klassen „HolzBauteil“ und „VBMBauteil“. Die Bauteilklasse gibt damit nur eine verpflichtende

Grundstruktur für die abgeleiteten Klassen vor. Zusätzlich zu den im Entwurf genannten Eigenschaften enthält die Klasse „Bauteil“ die beiden Normverweise für k_{mod} und γ_M um diese später in einer Exportdatei ausgeben zu können. Die Klassen „Material“ und „Geometrie“ sind ebenfalls abstrakte Klassen und dienen den jeweils spezialisierten Klassen für Holzbauteile und Verbindungsmittel als Vorlage. Die beiden Klassen sind deswegen abstrakt, da sie lediglich als allgemeine Vorlage für alle Bauteilklassen des jeweiligen Typs dienen und keine Objekte der Klasse selbst erstellt werden müssen. Die Klasse „Nachweispaket“ besitzt wie im Entwurf einen Namen, welcher das Nachweispaket eindeutig identifiziert und eine Liste mit Nachweisen. Die Klasse „Nachweis“ ist eine abstrakte Klasse und dient als Basisklasse für alle Nachweisklassen. Sie gibt als Struktur den Nachweisnamen und –typ, die zugehörige Norm, das Ergebnis der Nachweisrechnung und den NachweisBerechnungs- und NachweisErfüllungsstatus vor. Zusätzlich legt sie fest, dass jede Nachweisklasse ein Feld mit Bauteilen besitzen muss, in welchem die am Nachweis beteiligten Bauteile gespeichert werden sollen und dass jeder Nachweis eine Nachweisfunktion besitzen muss in welcher die eigentliche Nachweisberechnung stattfindet. Durch das wiederholte Verwenden des Vererbungskonzeptes werden klare Strukturen für die einzelnen Klassen des Programms vorgegeben. Für die Eingabe der Bauteil- und Nachweisdaten sowie für das Hauptfenster und die Warnungen gibt es zusätzlich eigene Form-Klassen, die der Übersichtlichkeit halber nicht im Klassendiagramm aufgenommen wurden. Auch auf die Darstellung der spezialisierten Klassen wurde aus diesem Grund verzichtet. Zusätzlich zu den genannten Klassen existieren die im Entwurf vorgestellten Klassen zur Berechnung der Normkennwerte.

7.2 Anwendung des Programms

Um die Funktionen des Programms aufzuzeigen, werden im Folgenden der Ablauf der Programmanwendung und die Programmoberfläche genauer beschrieben.

7.2.1 Start des Programms

Beim Start des Programms öffnet sich das, während der kompletten Anwendung geöffnete Hauptfenster. Die Idee basiert darauf dem Nutzer jederzeit vier Möglichkeiten zur Verfügung zu stellen. Aus einer Art Hauptmenü soll dieser ein neues Bauteil oder Nachweispaket hinzufügen, vorhandene Bauteile und Nachweispakete anzeigen und die Berechnungsergebnisse ausgeben können. Um diese Idee umzusetzen entstand im Laufe der Programmierung die Idee eines zentralen Hauptfensters, in welchem diese Möglichkeiten und Informationen dem Nutzer sehr übersichtlich in Form von Tabellen dargestellt werden (siehe Abbildung 24). Insgesamt kann das Haupfenster in drei Teile gegliedert werden. Die linke Seite beinhaltet thematisch alle Bauteil-Verknüpfungen. Hier kann durch Auswahl im DropDown-Menü die Art des Bauteils ausgewählt werden, welches dann mit einem Klick auf die „hinzufügen“-Schaltfläche neu erstellt wird. Darunter ist die Bauteil-Tabelle, die eine Übersicht über die bereits erstellten Bauteile gibt. In ihr sind Name und Typ des Bauteils aufgeführt. Soll ein Bauteil genauer betrachtet oder verändert werden, kann es in der Liste ausgewählt werden. Mit einem Klick auf den „Bauteil ansehen“-Button wird das jeweilige Eingabefenster mit allen zugehörigen Bauteilinformationen geöffnet. Wird ein Bauteil nicht mehr benötigt, kann dieses ebenfalls ausgewählt und durch einen Klick auf den zugehörigen Button gelöscht werden.

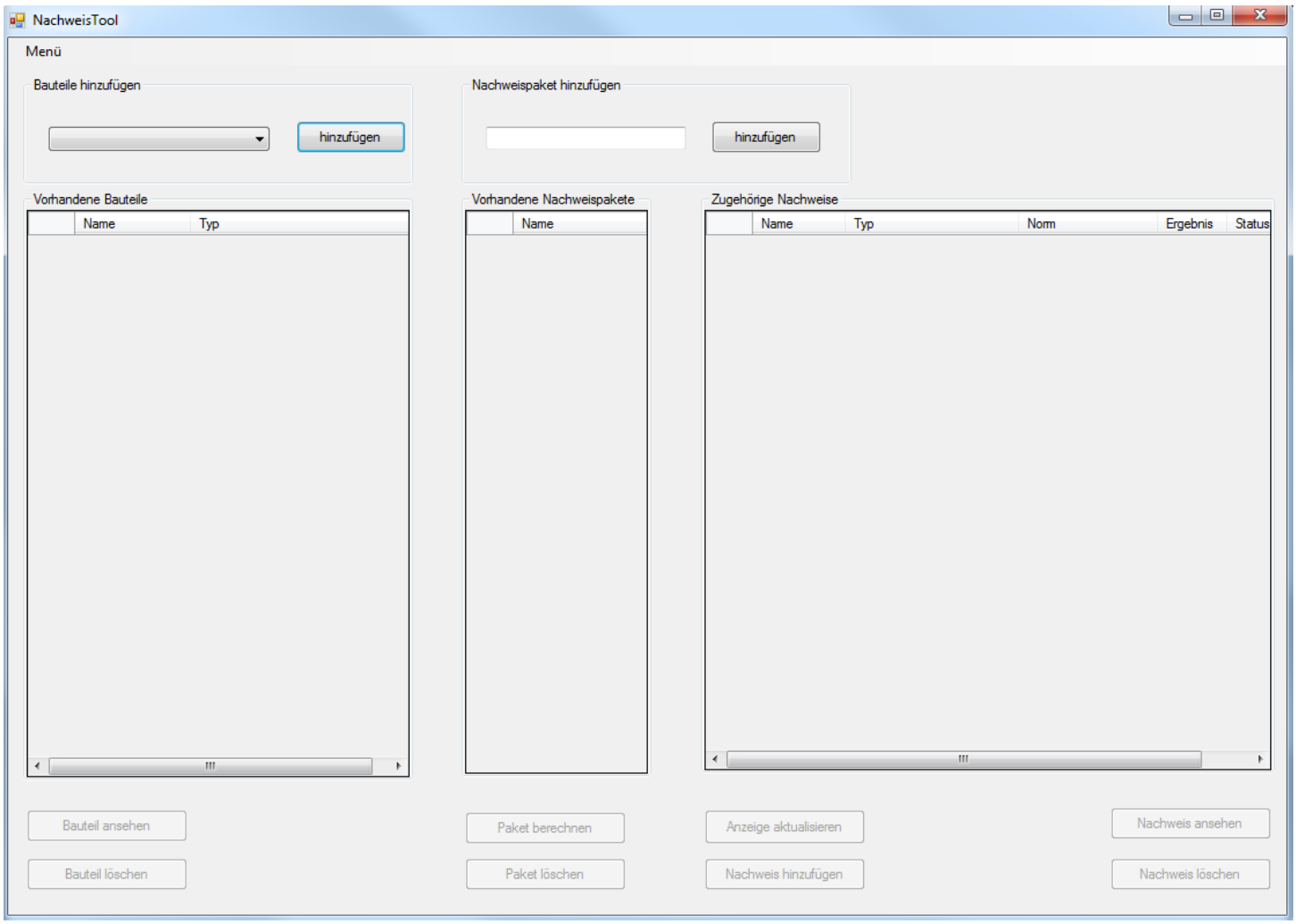


Abbildung 24: Hauptfenster

In der Mitte des Hauptfensters befindet sich der Bereich der Nachweispakete. In der Eingabezeile kann der Name eines neu zu erstellenden Nachweispaketes eingegeben und dann mit einem Klick auf „hinzufügen“ zur Nachweispaketliste des Programms hinzugefügt werden. Unterhalb der Eingabezeile sind die vorhandenen Nachweispakete anhand deren Namen aufgelistet. Unter der Tabelle hat der Nutzer die Möglichkeit, ein in der Tabelle ausgewähltes Nachweispaket entweder zu berechnen oder aus der Nachweispaketliste zu löschen. Wird „Paket berechnen“ ausgewählt, werden alle in diesem Nachweispaket vorhandenen, rechenbaren Nachweise nacheinander berechnet. Die Nachweise selbst werden in der Tabelle auf der rechten Seite dargestellt. Hier fällt die Darstellung ausführlicher aus, weil sie zudem der Ergebnisübersicht dient. Die Tabelle zeigt dabei immer die Nachweise des in der mittleren Tabelle ausgewählten Nachweispakets an. Neben dem Namen und Typ des Nachweises werden die zugehörige Norm, das Ergebnis in Form des Ausnutzungsgrades und sein Status ausgegeben. Unterhalb der Tabelle gibt es eine Schaltfläche zum Aktualisieren der Nachweisanzeige, zum Hinzufügen eines Nachweises zu einem ausgewählten Paket, zum Löschen des ausgewählten Nachweises und zudem einen Button, um das zugehörige Fenster zum ausgewählten Nachweis zu öffnen und dessen Informationen anzeigen und abändern zu können. In diesem Fenster kann der Nachweis auch einzeln neu berechnet werden. Die in der oberen Abbildung ausgegrauten

Schaltflächen werden aktiviert sobald die entsprechenden Elemente, auf die sie zugreifen vorhanden sind. Zusätzlich besitzt das Hauptfenster ein Menü, in welchem Daten als XML exportiert und importiert werden können. In diesem Menü wird das Programm ebenfalls beendet. Auf den Export und Import wird im Verlauf des Beispiels genauer eingegangen. Nachdem nun das Hauptfenster und damit auch die Grundstruktur des Programms aus der Sicht des Anwenders erläutert wurde, wird mit der Anwendung anhand des Beispiels gestartet.

7.2.2 Bauteil hinzufügen

Nachdem das Programm gestartet wurde, soll zuerst der Satteldachträger zur aktuellen Bauteilliste hinzugefügt werden. Dies ist nötig, um ihn später in den Nachweisen verwenden zu können. Um ein Bauteil hinzuzufügen wird aus dem DropDown-Menü links oben im Startfenster zuerst der gewünschte Bauteiltyp ausgewählt.

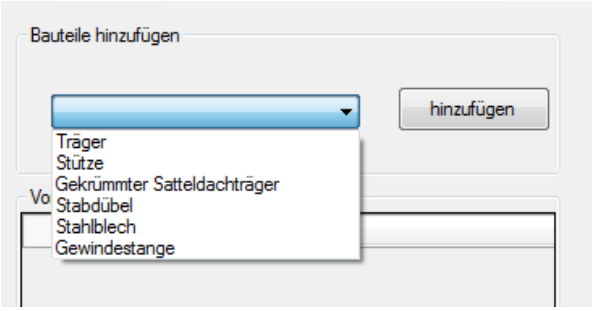


Abbildung 25: Wahl des Bauteiltyps

In diesem Fall wird der gekrümmte Satteldachträger ausgewählt und anschließend der „hinzufügen"-Button geklickt. Dadurch wird das Eingabefenster des zugehörigen Bauteiltyps geöffnet. Die Eingabefenster unterscheiden sich je nach Bauteiltyp stark voneinander. Der Unterschied zwischen der Eingabe für Verbindungsmittel und Holzbauteile ist dabei besonders groß. Bei allen Bauteilen sollte zuerst ein eindeutiger Name festgelegt werden. Die Eindeutigkeit ist in diesem Programm zwingend notwendig, um die Bauteile eindeutig den Nachweisen zuordnen zu können. Die Eindeutigkeit wird beim Drücken des „Eingaben bestätigen"-Buttons im Eingabefenster überprüft. Sollte bereits ein Bauteil mit dem eingegebenen Namen vorhanden sein, wird eine Warnung ausgegeben. Der Nutzer kann dann entscheiden ob das bereits vorhandene Bauteil von dem neuen Bauteil ersetzt werden soll, oder ob er zu dem Eingabefenster des neuen Bauteils zurückkehren möchte, um diesem einen anderen Namen zu geben. Das Eingabefenster der Holzbauteile ist immer in die vier Themen, „Allgemeines", „Material", „Geometrie" und „Schnittgrößen" unterteilt (siehe Abbildung 26). „Allgemeines" und „Material" sind dabei immer identisch. In „Allgemeines" wird der Bauteilname festgelegt und die Nutzungsklasse und die Klasse der Lasteinwirkungsdauer in einem DropDown-Menü ausgewählt.

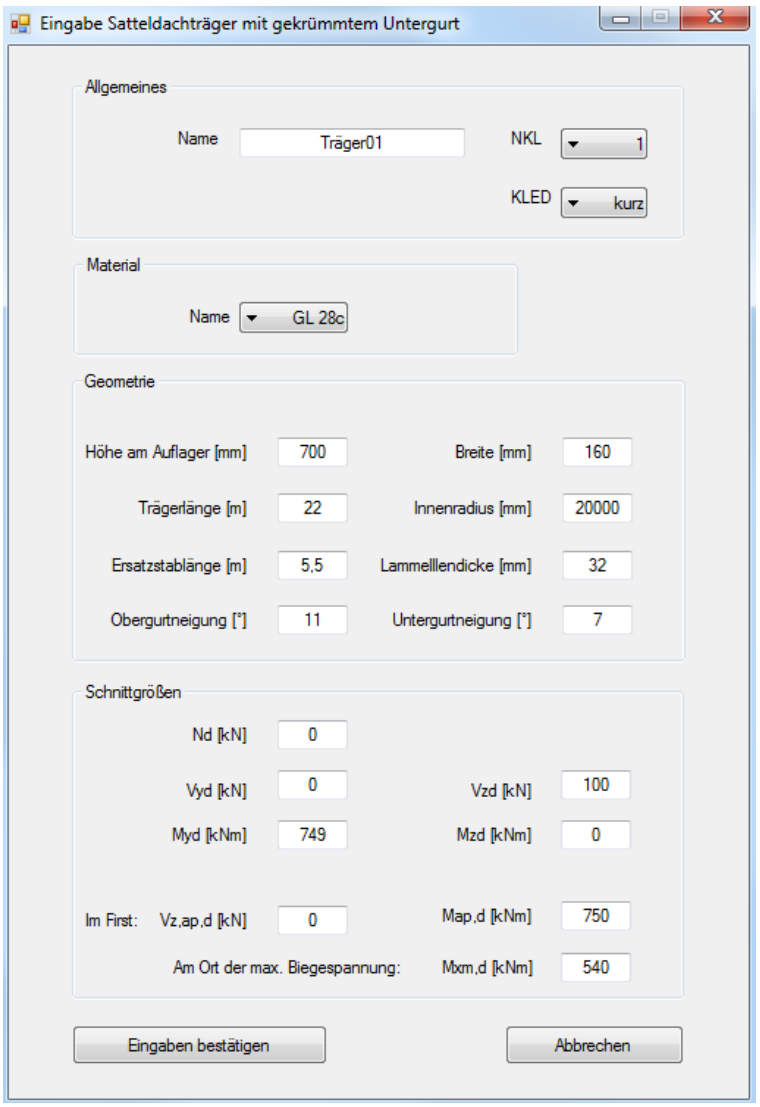


Abbildung 26: Eingabefenster Satteldachträger

Die verfügbaren Materialien werden ebenfalls in einer DropDown-Liste aufgeführt. Die Liste zeigt die Namen der Materialien an und ist mit einem DataTable verknüpft. Dieser DataTable enthält die kompletten Materialdaten mit allen nötigen materialspezifischen Kenngrößen aus dem Excel-Datenblatt. Die Eingabe der Geometriedaten ist wie bereits erwähnt je nach Bauteiltyp sehr unterschiedlich. Im Beispiel des gekrümmten Satteldachträgers müssen unter anderem der Innenradius und die Lamellendicke eingegeben werden. Zusätzlich zu den Geometriedaten müssen auch die Schnittgrößen eingegeben werden. Hier müssen alle maximalen Schnittgrößen beziehungsweise die Schnittgrößen an den vorgegebenen Stellen eingegeben werden (siehe Abbildung 26). Mit der Bestätigung der Eingabe durch das Drücken der entsprechenden Schaltfläche wird überprüft, ob NKL, KLED und Material ausgewählt und ein eindeutiger Name eingegeben wurde. Ist eine dieser Bedingungen nicht erfüllt, wird eine Warnung ausgegeben und der Speichervorgang wird abgebrochen. Der Nutzer kann die Eingabe mit Hilfe der entsprechenden Schaltflächen jederzeit abbrechen, oder die Eingaben bestätigen, um das Fenster zu schließen. Diese Optionen sind vor allem unter der Berücksichtigung relevant, dass genau dieses Fenster wieder geöffnet wird wenn sich der

Nutzer im Hauptfenster dazu entscheidet die Bauteilinformationen zu diesem Bauteil anzusehen oder zu verändern. Da das Fenster jedoch lediglich die Daten übernimmt und keinen direkten Verweis auf das Bauteil darstellt, kann durch bestätigte Änderung des Namens und Materials schnell eine abgewandelte Variante des vorhandenen Bauteils erstellt werden. Nachdem nun alle notwendigen Daten vollständig eingegeben und die Eingaben bestätigt wurden, schließt sich das Fenster und der Satteldachträger wird in die aktuelle Bauteilliste übernommen. Dies wird direkt sichtbar, da er nun im Hauptfenster in der Tabelle der vorhandenen Bauteile aufgeführt wird. Der Vorgang des Hinzufügens eines Bauteils soll nun anhand der Gewindestange zur Verstärkung des Satteldachträgers wiederholt werden. Hierzu wird dieses Mal die Gewindestange als Bauteiltyp aus dem Menü ausgewählt.

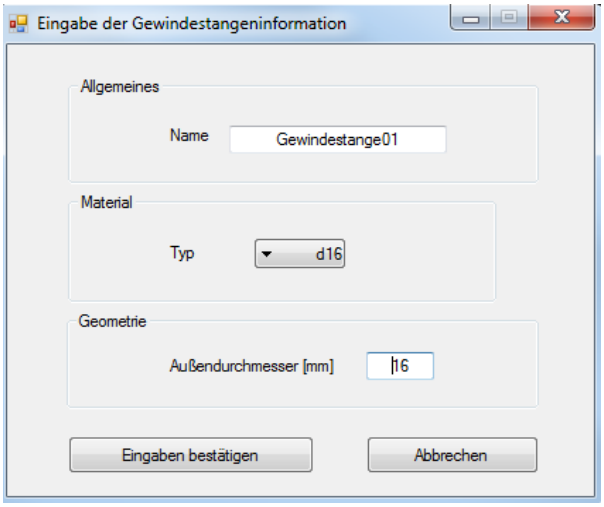


Abbildung 27: Eingabefenster Gewindestange

Wie zu erkennen ist, sind die Eingabefenster zu den Verbindungsmitteln wesentlich einfacher aufgebaut, als die der Holzbauteile (siehe Abbildung 27). Da für die Verbindungsmittel selbst keine Schnittgrößen eingegeben werden, fällt diese Eingabe komplett weg. Auch Nutzungsklasse und Klasse der Lastweinwirkungsdauer werden im Nachweis aus dem Zusammenhang der beteiligten Holzbauteile geschlossen und nicht dem Verbindungsmittel zugeordnet. So wird in den allgemeinen Informationen lediglich ein eindeutiger Name für das Bauteil vergeben. In der ComboBox im Bereich „Material“ werden wieder die Namen der für den Bauteiltyp zur Verfügung stehenden Materialien aus dem internen DataTable angezeigt. Im Fall der Gewindestange muss als geometrische Größe zudem noch der Außendurchmesser in Millimeter festgelegt werden. Auch hier wird beim Bestätigen der Eingaben überprüft ob ein eindeutiger Name eingegeben und ein Material ausgewählt wurde. Nachdem die Eingaben bestätigt wurden, sind alle für dieses Beispiel nötigen Bauteile angelegt. Sie befinden sich in der Bauteilliste des Programms und können nun von den Nachweisen verwendet werden.

7.2.3 Nachweispaket hinzufügen

Um ein Nachweispaket zur aktuellen Nachweispaketliste hinzuzufügen, wird der Name des neuen Pakets in der entsprechenden Eingabezeile eingegeben und dann der „hinzufügen“-Button geklickt.

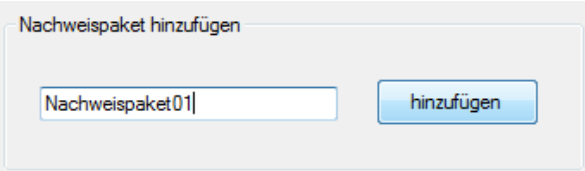


Abbildung 28: Hinzufügen eines neuen Nachweispakets

Wurde ein eindeutiger Name eingegeben wird das Nachweispaket zur Liste der Nachweispakete hinzugefügt. Ansonsten wird eine Warnung ausgegeben und der Name kann geändert werden. Wurde das Nachweispaket hinzugefügt, öffnet sich direkt das Fenster, um einen neuen Nachweis hinzuzufügen. Dieses kann jedoch auch einfach durch Klicken des „Abbrechen"-Buttons geschlossen werden.

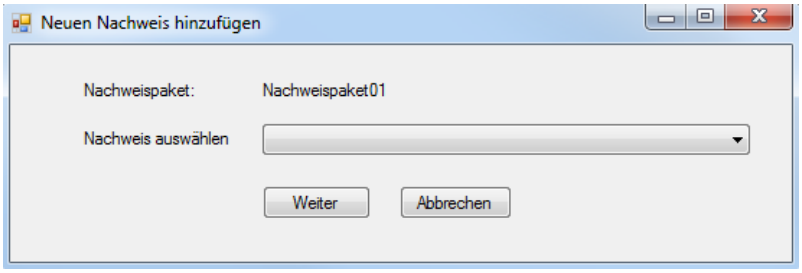


Abbildung 29: Hinzufügen eines neuen Nachweises

7.2.4 Nachweis hinzufügen und berechnen

Es gibt zwei Möglichkeiten einen Nachweis zu einem Nachweispaket hinzuzufügen. Die erste Möglichkeit ist beim Anlegen des neuen Nachweispakets direkt im sich dabei öffnenden Fenster einen Nachweistyp auszuwählen und einen solchen Nachweis anzulegen. Die andere Möglichkeit ist das vorhandene, also bereits erstellte Nachweispaket, in der Liste der vorhandenen Nachweispakete auszuwählen und dann auf den „Nachweis hinzufügen"-Button zu klicken. Beide Ansätze führen zu dem in der folgenden Abbildung dargestellten Fenster.

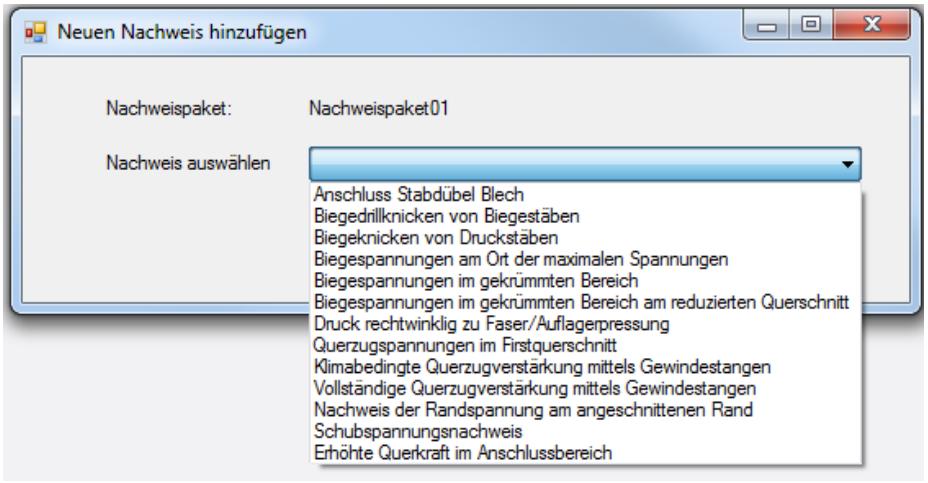


Abbildung 30: Auswahl des Nachweistyps

Das Fenster zeigt den Namen des Nachweispakets an und gibt die Möglichkeit aus dem DropDown-Menü einen der bereits implementierten Nachweise auszuwählen und durch einen Klick auf den „Weiter"-Button zu dessen Eingabefenster zu gelangen. In diesem Fall soll zuerst ein Nachweis der Schubspannungen am Auflager durchgeführt werden.

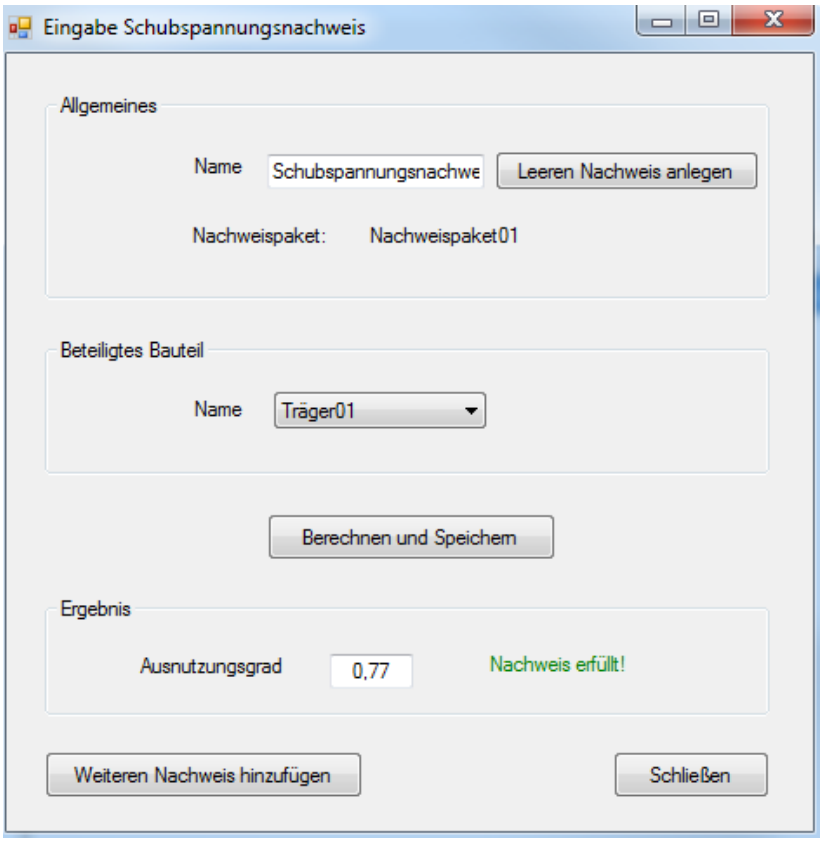


Abbildung 31: Eingabefenster des Schubspannungsnachweises

Im Eingabefenster des Schubspannungsnachweises wird zuerst der Name des neu zu erstellenden Nachweises festgelegt. Nun hat der Nutzer die Möglichkeit den Nachweis als leeren Nachweis anzulegen. Das heißt dem Nachwies wird kein Bauteil zugeordnet und nur Nachweisname und – typ werden gespeichert. Diese Variante ist vor allem dann sinnvoll, wenn ein vordefiniertes Nachweispaket erstellt werden soll, das unabhängig von einem Bauteil gespeichert und je nach Bedarf importiert werden können soll. In diesem Fall soll sich der Nachweis jedoch speziell auf den erstellten Satteldachträger beziehen. Daher wird nun im nächsten Schritt das Bauteil „Träger01" als beteiligtes Bauteil aus der ComboBox ausgewählt. Hier wird bereits im Hintergrund eine sinnvolle Vorauswahl der Bauteile der aktuellen Bauteilliste getroffen, indem nur Holzbauteile angezeigt werden. Die Vorauswahl ist jedoch von Nachweis zu Nachweis verschieden und stellt in den verschiedenen Nachweistypen sicher, dass nur sinnvolle Bauteile ausgewählt werden können. Je nach Nachweisart und damit der Anzahl der beteiligten Bauteile, werden hier mehrere ComboBoxen zur Verfügung gestellt um die Bauteile auszuwählen. Nachdem das Bauteil gewählt wurde, sind in diesem Fall alle nötigen Einstellungen vorgenommen um die Berechnung erfolgreich durchzuführen. Bei anderen Nachweisen können hier noch zusätzliche Eingaben nötig sein, diese werden jedoch an einem anderen Beispiel erläutert. Um den Nachweis zu berechnen und der Nachweisliste des zugehörigen Nachweispaketes

hinzuzufügen genügt nun ein Klick auf den „Berechnen und Speichern"-Button. Hierbei wird überprüft, ob der Nachweisname in dem gewählten Nachweispaket eindeutig ist und alle notwendigen Eingaben vorgenommen wurden. Ist dies nicht der Fall, wird eine Warnung ausgegeben und der Speicher- und Berechnungsvorgang muss abgebrochen werden. Das Ergebnis der Berechnung wird sofort in Form des Ausnutzungsgrades dargestellt. Zudem wird der aktuelle Erfüllungsstatus farbig dargestellt. Dieser gibt an, ob der Nachweis nach aktuellen Berechnungen erfüllt ist oder nicht (siehe Abbildung 31). Dieses Fenster erfüllt ebenfalls die Aufgabe des Informationsfensters, welches sich öffnet, wenn der Nutzer im Hauptfenster einen Nachweis auswählt und auf „Nachweis ansehen" klickt. Der Nutzer hat nun die Möglichkeit entweder diesen Nachweis zu schließen und zum Hauptmenü zurückzukehren oder mit dem „Weiteren Nachweis hinzufügen"-Button noch einen weiteren Nachweis zum ausgewählten Nachweispaket hinzuzufügen. In diesem Fall gelangt er wieder in das Fenster zur Auswahl des gewünschten Nachweises. Nach dem Schließen der Eingabe wird der Nachweis durch Auswahl des zugehörigen Nachweispakets in der Nachweispakettabelle in der Liste der zugehörigen Nachweise auf der rechten Seite dargestellt (siehe Abbildung 32).

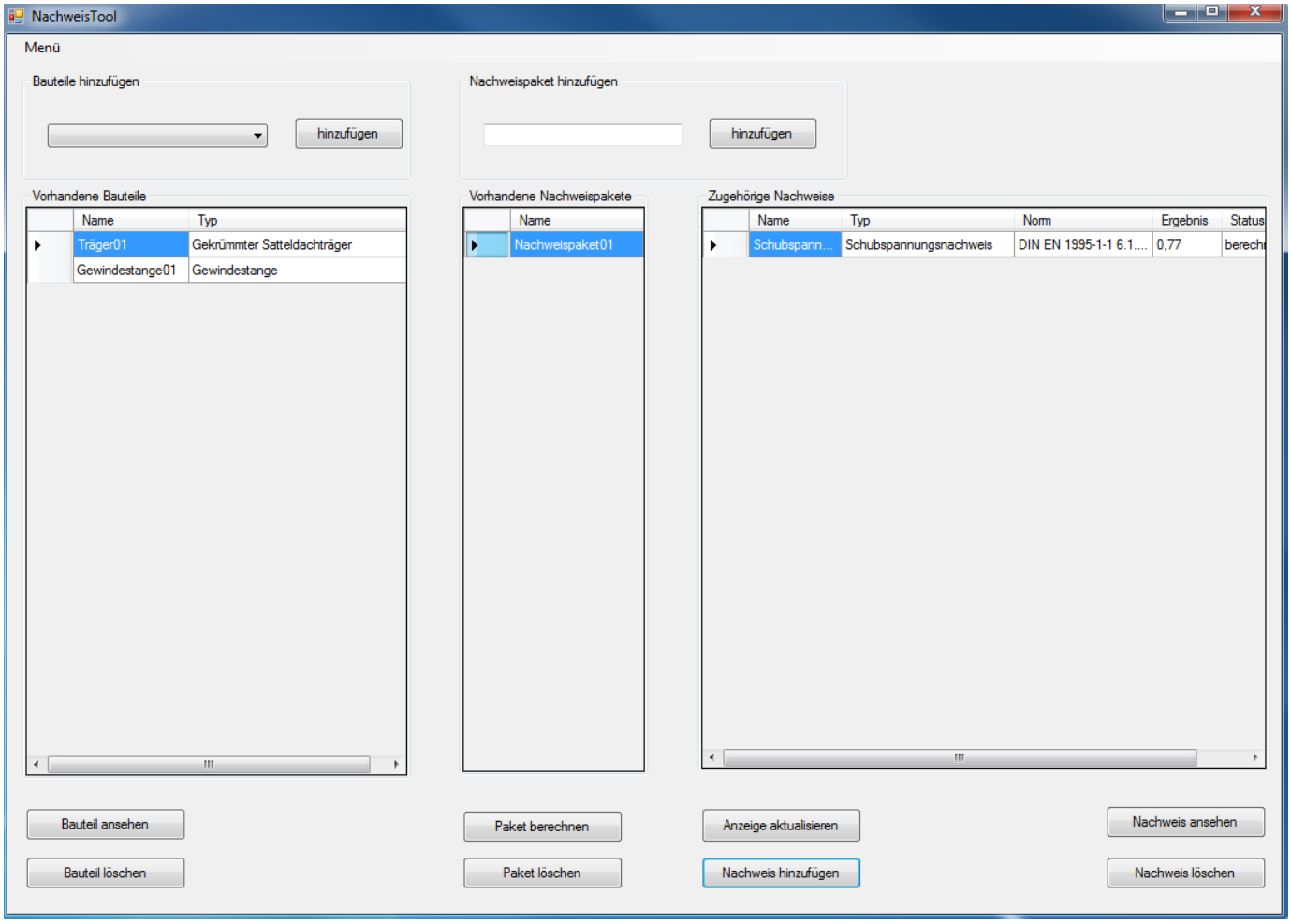


Abbildung 32: Hauptfenster mit aktualisierten Listen

Hier werden der Name und der Typ des Nachweises, die verwendete Norm, das Ergebnis als Ausnutzungsgrad und der Berechnungsstatus dargestellt. Der Berechnungsstatus kann die Werte

„berechnet", „nicht berechnet" und „kein Bauteil" annehmen. Nachdem der Nachweis mit den aktuell gültigen Eingaben bereits berechnet wurde und auch so dargestellt wird, soll nun ein weiterer Nachweis die anderen Varianten verdeutlichen. Als Nächstes soll der Nachweis der Auflagerpressung, also des Drucks senkrecht zur Faser geführt werden. Das Nachweispaket „Nachweispaket01" wird ausgewählt und mit einem Klick auf „Nachweis hinzufügen" wird das Nachweis-Auswahlfenster geöffnet. Der entsprechende Nachweistyp „Auflagerpressung" wird ausgewählt und mit „Weiter" das zugehörige Eingabefenster geöffnet. In der Abbildung wird direkt ersichtlich, dass die Eingabe von Nachweis zu Nachweis sehr unterschiedlich und vor allem verschieden umfangreich ausfallen kann. Im Fall des Nachweises der Auflagerpressung müssen zusätzlich der Lagertyp und die geometrischen Größen der Kontaktflächen eingegeben werden.

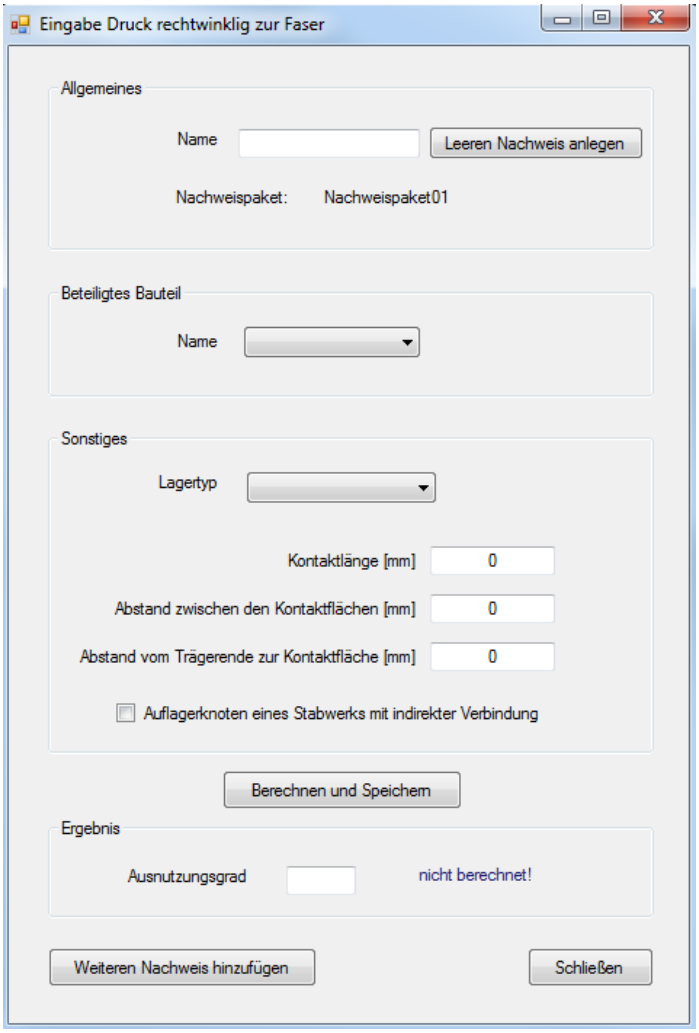


Abbildung 33: Eingabefenster des Nachweises der Auflagerpressung

Hier soll vorerst ein leerer Nachweis hinzugefügt werden. Das heißt, dass lediglich ein Nachweis des Typs „Auflagerpressung" mit einem eindeutigen Namen zum Nachweispaket hinzugefügt wird und alle sonstigen Einstellungen leer bleiben. Daher wird nun ein Name eingegeben und der Button „Leeren Nachweis anlegen" geklickt. Nach der Überprüfung ob der Nachweisname in der Nachweisliste des Nachweispakets eindeutig ist, wird der Nachweis der Liste hinzugefügt und das Fenster wird

geschlossen. Nach dem Aktualisieren der Anzeige wird nun auch der neue Nachweis in der Nachweisliste angezeigt.

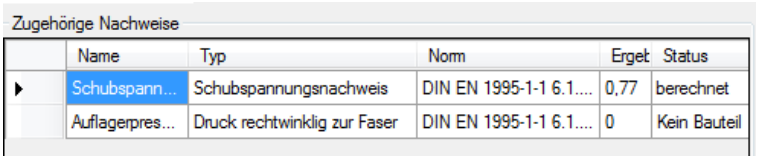


Abbildung 34: Nachweisauflistung

In diesem Fall ist das Ergebnis „0" und der Status „Kein Bauteil" gibt an, dass es sich um einen leeren Nachweis handelt (siehe Abbildung 34).

Der Nachweis wird ausgewählt und mit dem „Nachweis ansehen"-Button wird das Eingabefenster für den Nachweis der Auflagerpressung erneut geöffnet um die sonstigen Daten einzugeben. Beim Klicken auf die „Berechnen und Speichern"-Schaltfläche erscheint dabei folgende Warnung:

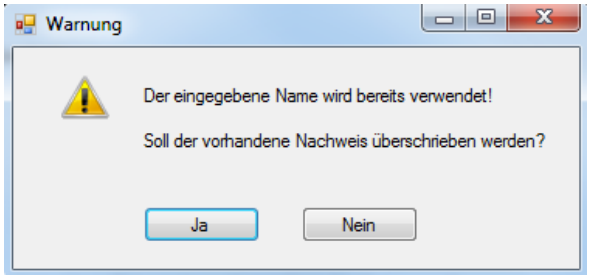


Abbildung 35: Warnung

Die Warnung (siehe Abbildung 35) gibt an, dass der Name bereits verwendet wurde und fragt den Nutzer ob das vorhandene Bauteil überschrieben werden soll. In diesem Fall soll genau dies geschehen, weshalb mit ja bestätigt wird und somit der Nachweis überschrieben und die Berechnung gestartet wird.

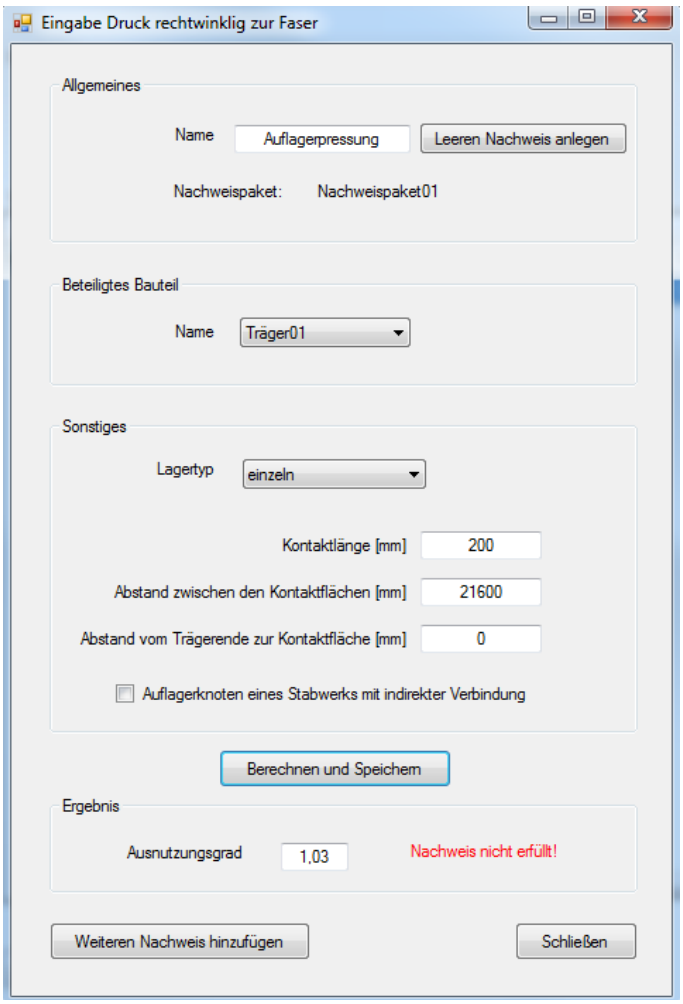


Abbildung 36: Eingabe der Daten des Nachweises der Auflagerpressung

Der Nachweis ist in diesem Fall nicht erfüllt (siehe Abbildung 36) und es liegt nahe die Eingangsdaten des Bauteils so zu verändern, dass der Nachweis aufgeht. Hierzu wird das Eingabefenster des Nachweises geschlossen. Nach dem Aktualisieren der Ansicht werden auch im Hauptfenster die Änderungen am Nachweis der Auflagerpressung sichtbar, indem der Nachweis den Status „berechnet" angenommen hat und das Ergebnis übernommen wurde (siehe Abbildung 37).

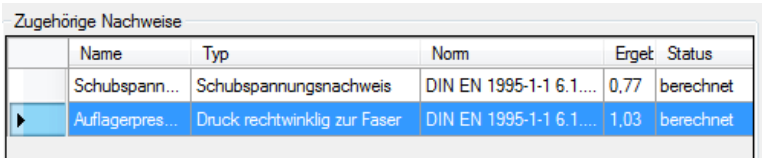


Abbildung 37: Nachweisauflistung

7.2.5 Bauteil ändern und Nachweispaket berechnen

Um das Bauteil zu ändern, muss es auf der linken Seite in der Bauteiltabelle ausgewählt werden. Das zugehörige Eingabefenster wird mit einem Klick auf „Bauteil ansehen" geöffnet.

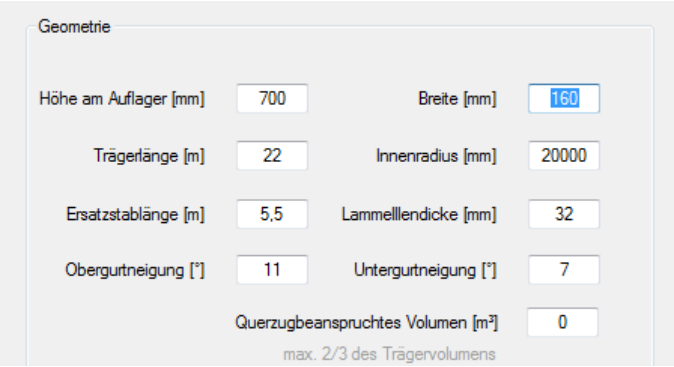


Abbildung 38: Änderungen der Geometriedaten

Die Breite des Satteldachträgers wird von 160 mm auf 180 mm vergrößert und die Daten werden durch das Klicken auf „Eingaben bestätigen" übernommen. Es erscheint eine Warnung, die angibt, dass der Bauteilname bereits vorhanden ist und fragt ob das vorhandene Bauteil überschrieben werden soll (siehe Abbildung 39). Da genau das geschehen soll, wird die Eingabe bestätigt.

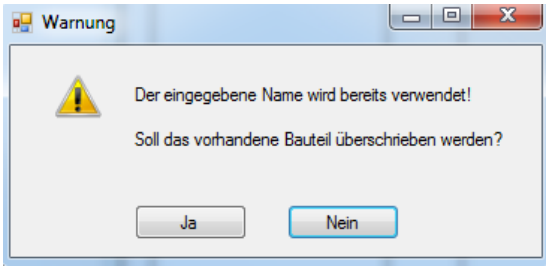


Abbildung 39: Warnung

Die Abänderungen an den Geometriedaten des Satteldachträgers haben unmittelbare Auswirkungen auf die Nachweise, in welchen dieser verwendet wird. Dies wird bei einem Blick auf die Nachweisliste des ausgewählten Nachweispakets sichtbar. Die Nachweise besitzen nun beide den Status „nicht berechnet" (siehe Abbildung 40).

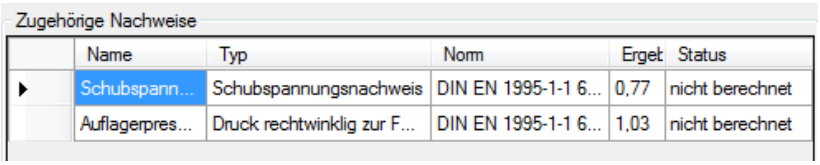


Abbildung 40: Aktualisierte Nachweisliste

Um die Nachweise neu zu berechnen, könnten die zugehörigen Eingabefenster einzeln geöffnet und berechnet werden. Dieser Schritt wird umgangen indem das Nachweispaket ausgewählt wird und der „Paket berechnen"-Button geklickt wird. Dadurch werden alle dem Nachweispaket zugehörigen Nachweise sofort berechnet und die aktualisierte Nachweisliste wird angezeigt.

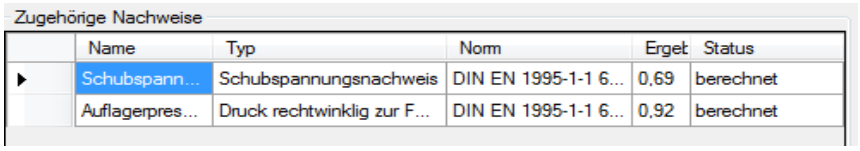


Abbildung 41: Berechnete Nachweisliste

Ein Blick auf die Nachweisliste zeigt, dass nach der Anpassung der Geometrie nun beide Nachweise berechnet und erfüllt sind.

Nach der vollständigen Eingabe aller Nachweise für das Bemessungsbeispiel 2 ergibt sich die Darstellung im Hauptfenster wie in Abbildung 42.

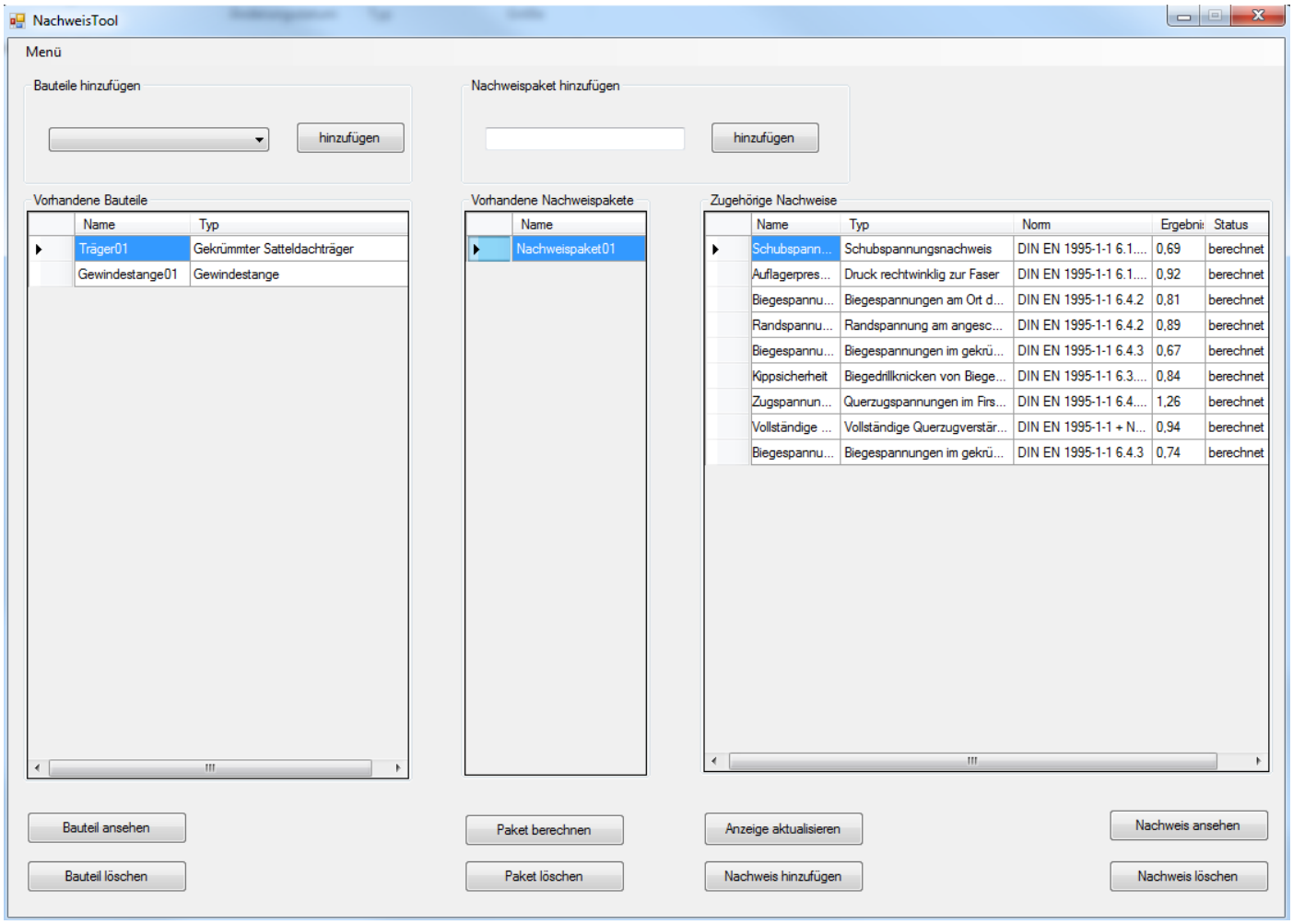


Abbildung 42: Vollständige Eingabe deszweiten Beispiels

Auf die Erklärung der Eingabe des ersten Beispiels wird an dieser Stelle verzichtet. Es wird ebenfalls wie Beispiel 2 als XML-Datei zum Import im Projektordner zur Verfügung gestellt.

7.2.6 Export und Import

Um einmal erstellte Bauteile, Nachweispakete und Nachweise jederzeit wieder verwenden zu können, stellt das entwickelte Programm die Möglichkeit des Exports und Imports der aktuellen Instanzen zur Verfügung. Soll der aktuelle Stand gespeichert werden, genügt ein Klick auf die „Export"-Schaltfläche im Menü des Hauptfensters, links oben (siehe Abbildung 43).

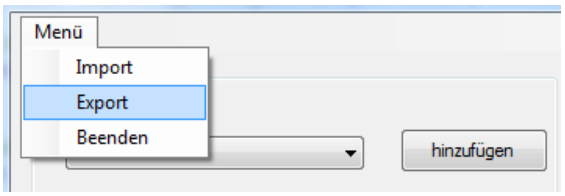


Abbildung 43: Menü des Hauptfensters

Es öffnet sich das XML-Exportfenster, in welchem der Speicherplatz und der Name des zu erstellenden XML-Files festgelegt werden können. In diesem Fall soll die Export-Datei „Beispiel2.xml" heißen und im Debug-Ordner des Projekts gespeichert werden.

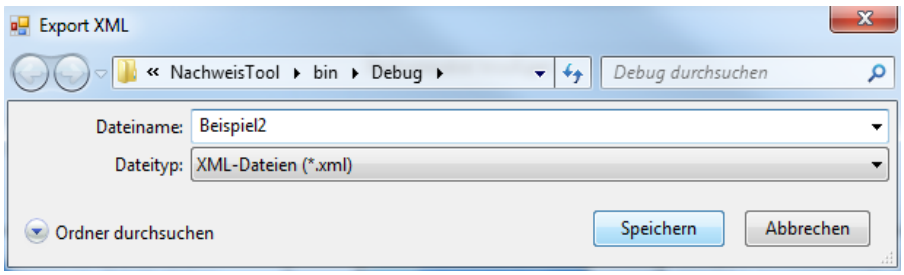


Abbildung 44: XML-Export

Mit dem Klick auf den „Speichern"-Button wird der XmlSerializer im Hintergrund gestartet und die Daten der aktuellen Bauteil- und Nachweispaketliste werden in ein XML-File geschrieben. Das in diesem Fall entstandene XML-File wird in der folgenden Abbildung dargestellt. Es werden alle Daten ausgegeben, die im Code entsprechend definiert wurden. Zudem wird ein Kommentar zu den jeweiligen Einheiten ausgegeben (siehe Abbildung 45).

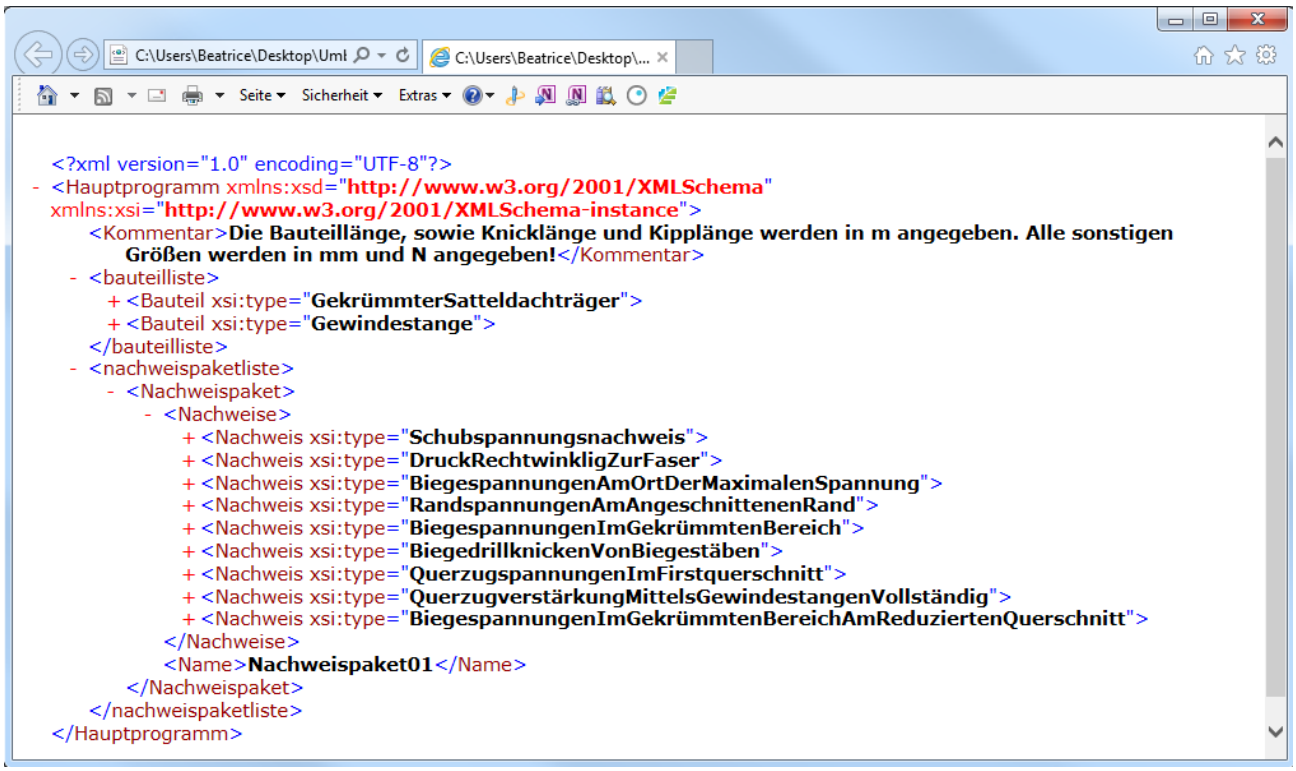


Abbildung 45: Exportierte XML- File

Dieses XML-File ist ausführlich und umfangreich und kann durch weitere Änderungen im Code angepasst werden. Sämtliche Bauteilinformationen wie geometrie- und materialspezifische Kenngrößen werden ausgegeben. Bei den Nachweisen wurde darauf geachtet, dass neben den Daten die in den zugehörigen Eingabefenstern des Nachweises festgelegt werden, auch die bei der Berechnung verwendete Kenngrößen und deren Normbezüge ausgegeben werden (siehe Abbildung 46).

```
- <Nachweis xsi:type="BiegedrillknickenVonBiegestäben">
    <Ergebnis>0.83825777504547117</Ergebnis>
    <Name>Kippsicherheit</Name>
    <Typ>Biegedrillknicken von Biegestäben</Typ>
    <Norm>DIN EN 1995-1-1 6.3.3 + NA NCI zu 6.3.3</Norm>
    <NachweisBerechnungsStatus>berechnet</NachweisBerechnungsStatus>
    <NachweisErfüllungsStatus>true</NachweisErfüllungsStatus>
  + <Bauteile>
    <Sigma_c0d>0</Sigma_c0d>
    <Sigma_myd>15.704279220797812</Sigma_myd>
    <Sigma_mzd>0</Sigma_mzd>
    <Sigma_m_crit>44.707299959987573</Sigma_m_crit>
    <k_cy>1.0030437536478107</k_cy>
    <Norm_k_cy>EC5 6.3.2(3) (6.25)</Norm_k_cy>
    <k_cz>1.03075606783918</k_cz>
    <Norm_k_cz>EC5 6.3.2(3) (6.6)</Norm_k_cz>
    <k_crit>0.96645854353160554</k_crit>
    <Norm_k_crit>EC5 6.3.3(4) (6.34)</Norm_k_crit>
    <Lambda_rel_m>0.791388608624526</Lambda_rel_m>
    <Norm_lambda_rel_m>EC 6.3.3(2) (6.30)</Norm_lambda_rel_m>
    <Lambda_rel_y>0.27190514715094355</Lambda_rel_y>
    <Norm_lambda_rel_y>EC5 6.3.2(1) (6.21)</Norm_lambda_rel_y>
    <Lambda_rel_z>0.0016172314265705898</Lambda_rel_z>
    <Norm_lambda_rel_z>EC5 6.3.2(1) (6.22)</Norm_lambda_rel_z>
    <SeitlichesAusweichenVerhindert>true</SeitlichesAusweichenVerhindert>
```

Abbildung 46: XML-Darstellung des Nachweises der Kippsicherheit

Zwischenergebnisse, wie die Spannungen, aber auch die Werte der Kenngrößen wie k_{crit} und deren Normverweise sind klar strukturiert und nachvollziehbar aufgeführt. So können die Ergebnisse händisch nachgerechnet und der Rechenvorgang zu einem späteren Zeitpunkt nachvollzogen werden.

Die Tatsache, dass es sich dabei um eine XML-Datei handelt stellt sicher, dass die Ausgabedatei gut lesbar und nachvollziehbar strukturiert ist. Der Import vorhandener XMLs ist ebenso einfach wie der Export. Auch hierzu findet sich der entsprechende Button im Menü des Hauptfensters, welcher das Importfenster öffnet.

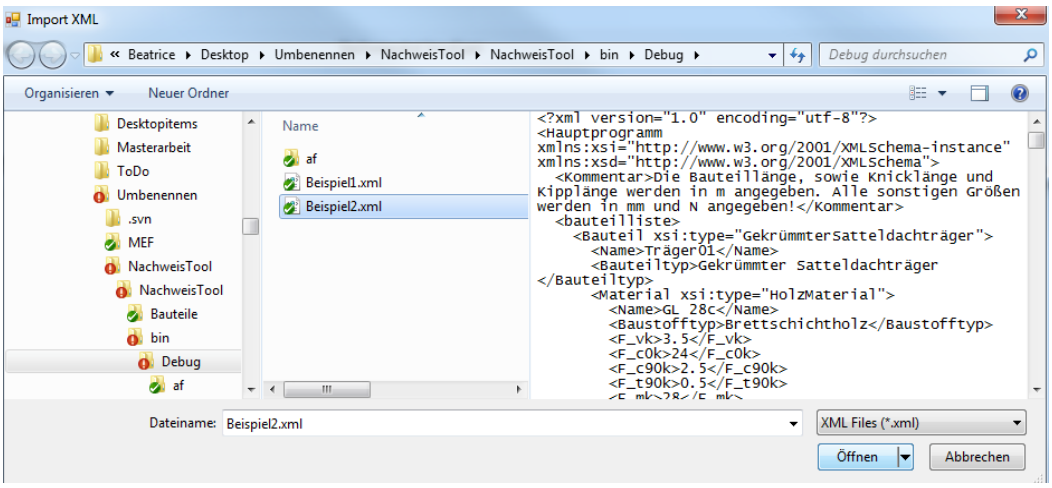


Abbildung 47: XML-Import

Im Importfenster kann die gewünschte XML-Datei ausgewählt und dann mittels „Öffnen" in das Programm geladen werden (siehe Abbildung 47). Um sicherzustellen, dass keine Namensverwechslung stattfindet, sollte der Import zu Beginn der Eingaben stattfinden. Die importierten Bauteile, Nachweispakete und Nachweise können nun in der bekannten

Benutzeroberfläche verwendet werden. Nicht benötigte Elemente können einfach in der zugehörigen Liste ausgewählt und gelöscht werden.

Aus der vorangegangenen Anwendungssystematik ergibt sich das allgemeine Ablaufdiagramm auf der folgenden Seite.

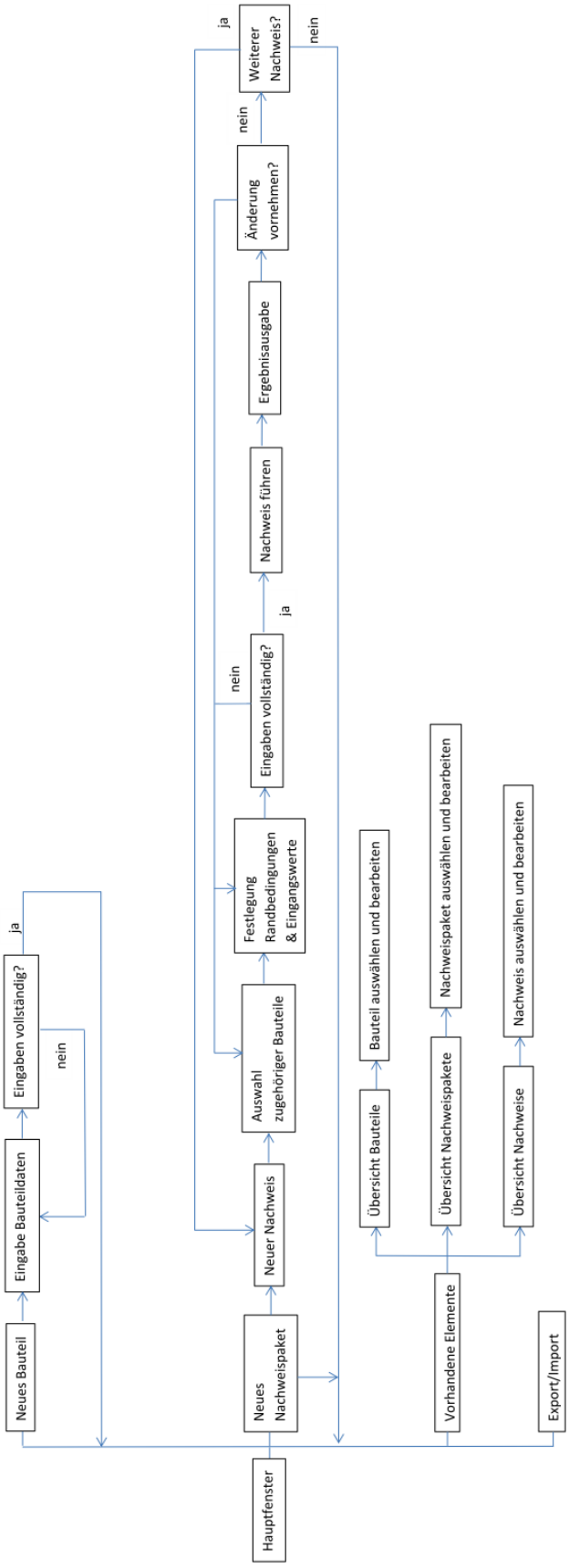


Abbildung 48: Ablaufdiagramm

7.3 Programmiertechnische Umsetzung

Nachdem nun die Funktionen und der Umfang des Programms aus Sicht des Anwenders dargestellt wurden, soll die interne Umsetzung erklärt werden. Die Programmierung wurde im Projekt „NachweisTool“ umgesetzt. Abbildung 49 zeigt den Aufbau des Projektes in der Projektmappe.

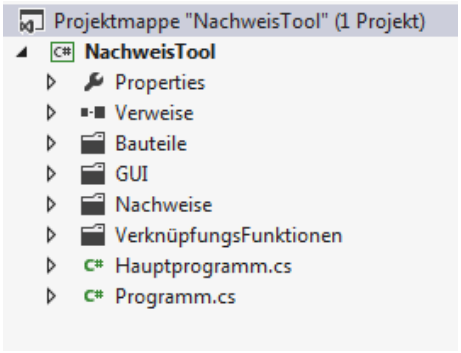


Abbildung 49: Projektmappenexplorer

Neben den Klassen „Hauptprogramm“ und „Programm“ wurden Ordner zu Strukturierung des Codes verwendet. Im Ordner „Bauteil“ sind alle Bauteilklassen sowie die Klassen für die Geometrie- und Materialinformationen gespeichert.

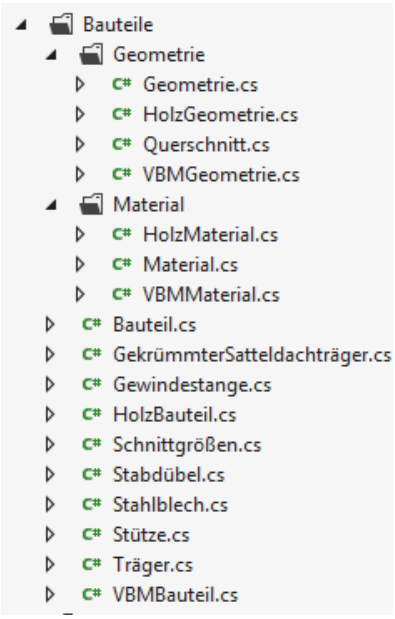


Abbildung 50: „Bauteil“-Ordner

Alle grafischen Elemente für die Bauteil- und Nachweiseingabe sowie die Warnungen und das Hauptfenster befinden sich in dem Ordner „GUI“. Hierzu wurde dieser Ordner zusätzlich in die Ordner „Bauteile“, „Nachweise“ und „Warnungen“ gegliedert. Im Ordner „Nachweise“ befinden sich sowohl die Klasse „Nachweispaket“ wie auch die abstrakte Basisklasse „Nachweis“. Zudem sind hier sämtliche von der Nachweisklasse abgeleiteten Klassen gespeichert, welche die verschiedenen Nachweistypen repräsentieren (siehe Abbildung 51).

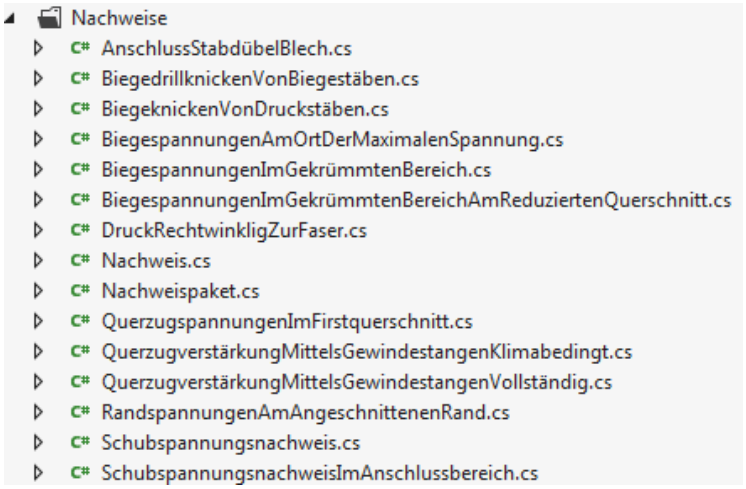


Abbildung 51: „Nachweis"-Ordner

Im Ordner „VerknüpfungFunktionen" sind die beiden Klassen „AnschlussnachweisVerknüpfungsFunktionen" und „BeiwertVerknüpfungsFunktionen" gespeichert, in denen die Kennwerte aus der Norm berechnet werden. Nachdem nun ein kurzer Überblick über die Projektstruktur gegeben wurde, wird im Folgenden genauer auf den Aufbau und die Funktion der einzelnen Klassen eingegangen.

7.3.1 Hauptfenster – Hauptprogramm

Programm.cs

Beim Start des Programms erstellt die Klasse „Programm" eine Instanz der Klasse „Hauptprogramm". Die Klasse „Programm" ist statisch, weshalb der indirekte Weg, diese Instanz aus der Klasse „Programm" zu erstellen gewählt wurde. Definitionsgemäß, kann eine statische Klasse nur statische Member enthalten, weshalb auch der Member vom Typ „Hauptprogramm" statisch ist. Da die Bauteile und Nachweispakete jedoch in Listen gespeichert werden sollen, wird die nicht statische Klasse „Hauptprogramm" benötigt.

```
static class Programm
{
    public static Hauptprogramm NeuesProgramm = new Hauptprogramm();

    /// <summary>
    /// Der Haupteinstiegspunkt für die Anwendung.
    /// </summary>
    [STAThread]
    static void Main()
    {
        Application.EnableVisualStyles();
        Application.SetCompatibleTextRenderingDefault(false);
        Application.Run(new Startfenster_Form());
    }
}
```

Abbildung 52: Klasse „Programm"

Die „Main"-Methode der Klasse „Programm" wird beim Start des Programms automatisch ausgeführt. In ihr wird eine neue Instanz der Klasse „Startfenster_Form" erstellt und die Anwendung aus diesem Fenster heraus gestartet.

Startfenster_Form.cs

Die Klasse „Startfenster_Form" beschreibt die grafische Darstellung und sämtliche Interaktionen des Benutzers im Hauptfenster. Jeder Klick auf eine Schaltfläche oder in eine Tabelle und sämtliche grafische Logik des Hauptfensters steckt in diesem Teil des Codes. Hier wird beispielsweise sichergestellt, dass die Buttons zum Bearbeiten der Bauteilliste erst dann aktiviert werden, wenn Bauteile erstellt wurden. Zudem wird festgelegt in welcher Form die vorhandenen Elemente dargestellt werden. Von hier aus werden auch die Methoden der Bauteil- und Nachweisklassen zum Laden der Eingabefenster aufgerufen. Diese Klasse ist hierfür, wie das Hauptfenster selbst, in die Regionen „Bauteile", „Nachweise" und „Nachweispakete" aufgeteilt und spielt eine zentrale Rolle im Programm.

Hauptprogramm.cs

Die Klasse „Hauptprogramm" ist in die Regionen „Bauteil-", „Nachweis-" und „Nachweispaket-Methoden", „DataTables", „Querschnittswerte" und „XML" aufgeteilt. In diesen Regionen stellt sie zahlreiche Methoden zur Bearbeitung dieser Elemente bereit. Die Klasse verfügt über eine Liste mit Bauteilen, die „bauteilliste", und eine Liste mit Nachweispaketen, die „nachweispaketliste". Zusätzlich besitzt sie zwei Listen in welchen sogenannte KeyValuePairs gespeichert werden. Ein KeyValuePair ist ein abstrakter Datentyp, welcher einen Schlüsselwert mit einem anderen Wert verknüpft. In diesen KeyValuePair-Listen sind wie der Name schon sagt die verschiedenen implementierten Nachweistypen und Bauteiltypen mit ihrem Namen und einem zugehörigen Objekt der Klasse gespeichert. Das Programm arbeitet ab jetzt immer mit dieser Liste um beispielsweise im Hauptfenster die verfügbaren Bauteiltypen anzuzeigen und das zugehörige Eingabefenster zu öffnen. Dies ist damit die einzige Stelle im Hauptprogramm, an welcher die möglichen Typen festgelegt werden müssen. An anderen Stellen des Programmcodes werden die verschiedenen Typen lediglich zur Ermöglichung des XML-Exports durch den XmlSerializer benötigt. Durch diese Gestaltung kann das Programm mit minimalen Änderungen im Code leicht erweitert werden.

```
bauteiltypen = new List<KeyValuePair<string, Bauteil>>();
bauteiltypen.Add(new KeyValuePair<string,Bauteil>("Träger", new Träger()));
bauteiltypen.Add(new KeyValuePair<string,Bauteil>("Stütze", new Stütze()));
bauteiltypen.Add(new KeyValuePair<string,Bauteil>("Gekrümmter Satteldachträger", new GekrümmterSatteldachträger()));
bauteiltypen.Add(new KeyValuePair<string,Bauteil>("Stabdübel", new Stabdübel()));
bauteiltypen.Add(new KeyValuePair<string,Bauteil>("Stahlblech", new Stahlblech()));
bauteiltypen.Add(new KeyValuePair<string,Bauteil>("Gewindestange", new Gewindestange()));
```

Abbildung 53: Zuweisung der KeyValuePairs der Bauteiltypen im Konstruktor der Klasse „Hauptprogramm"

Im Konstruktor der Klasse „Hauptprogramm" werden alle Member initialisiert und die Bauteil- und Nachweistypen zu den Listen hinzugefügt. Neben dem Konstruktor besitzt die Klasse die bereits genannten nach Regionen unterteilten Methoden, welche im Folgenden vorgestellt werden.

Bauteil-Methoden

In der Region „Bauteil-Methoden" sind Methoden zur Bearbeitung der Bauteilliste des Programms gespeichert. Folgende Methoden stehen zur Verfügung:

- BauteilFinden
- BauteilHinzufügen
- BauteilErsetzen
- BauteilEntfernen
- NameBauteilEindeutig

Nachweis-Methoden

Die folgenden „Nachweis-Methoden" bearbeiten die Nachweise in den Nachweislisten der vorhandenen Nachweispakete:

- NachweisHinzufügen
- NachweisFormÖffnen
- NachweisErsetzen
- NachweisEntfernen
- NameNachweisEindeutig
- NachweisInformation

Nachweispaket-Methoden

Die Methoden in dieser Region dienen zum Bearbeiten der aktuellen Nachweispaketliste:

- NachweispaketHinzufügen
- NachweispaketEntfernen
- NameNachweispaketEindeutig
- NachweispaketBerechnen

DataTables

Die Materialien und ihre Kennwerte werden für das NachweisTool in Form von Excel-Tabellen zur Verfügung gestellt. Somit wird sichergestellt, dass jederzeit weitere Materialien hinzugefügt werden können. Dabei ist lediglich zu beachten, dass der Grundaufbau, also die Spaltenbenennung beibehalten werden muss, um eine fehlerlose Verarbeitung der Daten durch das Tool zu gewährleisten. Weitere Materialien können dem Datenblatt jederzeit in Form von Zeilen hinzugefügt werden. Neben dem Namen, dem Baustofftyp und der Holzart sind alle benötigten materialspezifischen Kenndaten in der Datei enthalten. Das Datenblatt wird als CSV-Datei gespeichert, wodurch eine einfache Weiterverarbeitung durch das Programm gewährleistet wird. Abbildung 54 zeigt einen Ausschnitt aus der Materialtabelle der Holzmaterialien.

	A	B	C	D	E	F	G	H	I	J	K
1	Name	Typ	Art	fmk	ft0k	ft90k	fc0k	fc90k	fvk	fRk	E0mean
2	-	-	-	N/mm^2	N/mm^2	N/mm^2	N/mm^2	N/mm^2	N/mm^2	N/mm^2	N/mm^2
3	C14	Vollholz	Nadelholz	14	8	0,4	16	2	3	0	7000
4	C16	Vollholz	Nadelholz	16	10	0,4	17	2,2	3,2	0	8000
5	C18	Vollholz	Nadelholz	18	11	0,4	18	2,2	3,4	0	9000
6	C20	Vollholz	Nadelholz	20	12	0,4	19	2,3	3,6	0	9500
7	C22	Vollholz	Nadelholz	22	13	0,4	20	2,4	3,8	0	10000
8	C24	Vollholz	Nadelholz	24	14	0,4	21	2,5	4	0	11000
9	C27	Vollholz	Nadelholz	27	16	0,4	22	2,6	4	0	11500
10	C30	Vollholz	Nadelholz	30	18	0,4	23	2,7	4	0	12000
11	C35	Vollholz	Nadelholz	35	21	0,4	25	2,8	4	0	13000
12	C40	Vollholz	Nadelholz	40	24	0,4	26	2,9	4	0	14000
13	C45	Vollholz	Nadelholz	45	27	0,4	27	3,1	4	0	15000
14	C50	Vollholz	Nadelholz	50	30	0,4	29	3,2	4	0	16000
15	GL 20c	Brettschichtl	Nadelholz	20	15	0,5	18,5	2,5	3,5	1,2	10400
16	GL 22c	Brettschichtl	Nadelholz	22	16	0,5	20	2,5	3,5	1,2	10400
17	GL 24c	Brettschichtl	Nadelholz	24	17	0,5	21,5	2,5	3,5	1,2	11000
18	GL 26c	Brettschichtl	Nadelholz	26	19	0,5	23,5	2,5	3,5	1,2	12000
19	GL 28c	Brettschichtl	Nadelholz	28	19,5	0,5	24	2,5	3,5	1,2	12500

Abbildung 54: Tabelle der Holzmaterialien

Für Holzbauteile und Verbindungsmittel gibt es separate Datenblätter, da sich die zugehörigen Daten stark unterscheiden. Der Aufbau der Materialtabelle unterscheidet sich für Verbindungmittel von dem der Holzbauteile (siehe Abbildung 55).

	A	B	C	D	E
1	Name	Typ	fuk	dichte_k	R_tuk
2	-	-	N/mm^2	kg/m^3	kN
3	S 235	Stabduebel	360	0	0
4	S 275	Stabduebel	430	0	0
5	S 355	Stabduebel	510	0	0
6	d16	Gewindestange	0	390	63

Abbildung 55: Materialtabelle der Verbindungsmittel

Die im Bereich „DataTables" enthaltenen Methoden dienen der Verarbeitung dieser Excel-Daten. Die Materialdaten werden anhand der „GetHolzMaterialtable"- und der „GetVBMMaterialtable"-Methode aus der CSV-Datei ausgelesen und in einem DataTable gespeichert. Ein DataTable ist eine Tabelle mit speicherinternen Daten. Diese erstellte Tabelle wird von der Methode zurückgegeben und steht somit den Bauteileingabefenstern zur Verfügung. Dort steht sie als Datenquelle für die Material-ComboBox zur Verfügung, in welcher wiederum nur die Namen der Materialien als Ansicht ausgewählt werden. Die Methode „GetVBMMaterialtable" bekommt zudem auch den Verbindungsmitteltyp übergeben. Es werden dann nur die Daten des zugehörigen Typs aus der CSV-Datei in den DataTable geschrieben. Auf diese Weise wird sichergestellt, dass für einen Stabdübel auch nur dessen Materialien zur Verfügung stehen.

Zusätzlich enthält der Bereich die Methoden „MaterialkennwerteHolz" und „MaterialkennwerteVBM". Diese Methoden dienen der Bestimmung der Materialkennwerte und bekommen einen DataTable, den Materialnamen sowie sämtliche Kenndaten des Materials per Referenz übergeben. Die Übergabe per Referenz dient dazu, dass die an den Daten vorgenommenen Änderungen auch außerhalb dieser

Methode übernommen werden. Anhand des Materialnamen wird das Material in dem übergebenen DataTable gesucht und schließlich werden die in der Tabelle gespeicherten Werte den Kenngrößen zugewiesen.

Querschnittswerte

Für die Nachweisrechnungen werden immer wieder Querschnittswerte wie zum Beispiel der Trägheitsradius benötigt. Um diese Werte nicht im Nachweis selbst immer wieder berechnen zu müssen wird hier eine Methode zur Verfügung gestellt, welche beim Erstellen eines Holzbauteils dessen Querschnittswerte einmalig berechnet. Die Querschnittswerte selbst werden im Holzbauteil gespeichert. Dies ermöglicht neben der Einsparung von zusätzlichem Code auch eine übersichtlichere Darstellung und damit eine bessere Lesbarkeit des Codes. Zudem stellen solche Rechnungen eine mögliche Fehlerquelle dar, da sie innerhalb eines Nachweises schnell unübersichtlich werden.

XML

Das Programm „NachweisTool" bietet die Möglichkeit Bauteile, Nachweise und Nachweispakete als XML-Datei zu exportieren, um diese zu einem späteren Zeitpunkt wieder zu importieren und somit wiederzuverwenden. Um dies zu realisieren dienen die Methoden „ExportXML" und „ImportXML". Die Methode „ExportXML" exportiert ein XML an eine angegebene Adresse. Diese Adresse wird vom Nutzer im Exportfenster eingegeben und der Methode bei deren Aufruf übergeben. Zum Erstellen der XML-Datei wird der XmlSerializer verwendet. Dieser erstellt die XMLs automatisch. Hierzu muss beachtet werden, dass nur öffentliche, also als public definiert Felder und Eigenschaften serialisiert werden und die Eigenschaften einen lesenden und schreibenden Zugriff zulassen müssen. Zudem müssen die Klassen aller zu serialisierten Objekte als public definiert sein und über einen öffentlichen, parameterlosen Konstruktor verfügen. Im Beispiel des „NachweisTools" ist vor allem der Umgang mit den abgeleiteten Klassen wichtig. Da die Bauteil- und Nachweislisten jeweils nur als Listen der abstrakten Basisklassen definiert sind, muss allen Basisklassen zusätzlich eine „[XmlInclude(typeof())]"-Anweisung für jede davon abgeleitete Klasse vorangehen.

```
[XmlInclude(typeof(Träger))]
[XmlInclude(typeof(Stütze))]
[XmlInclude(typeof(GekrümmterSatteldachträger))]
public abstract class HolzBauteil:Bauteil
```

Abbildung 56: „[XmlInclude(typeof())]"-Anweisung für die Klasse „HolzBauteil"

In der „ImportXML"-Methode wird das XML an der übergebenen Adresse anhand des XmlSerializers deserialisiert. Die Deserialisierung erstellt dabei ein neues Objekt vom Typ „Hauptprogramm" aus den importierten XML-Daten. Anschließend werden die aktuellen Bauteil- und Nachweispaketlisten um die Listen des neu erstellten Objekts erweitert.

7.3.2 Bauteile

Die folgende Abbildung zeigt noch einmal den grundlegenden Aufbau des Ordners „Bauteile". Hierin sind alle Klassen enthalten, welche mit dem Bauteil assoziiert werden.

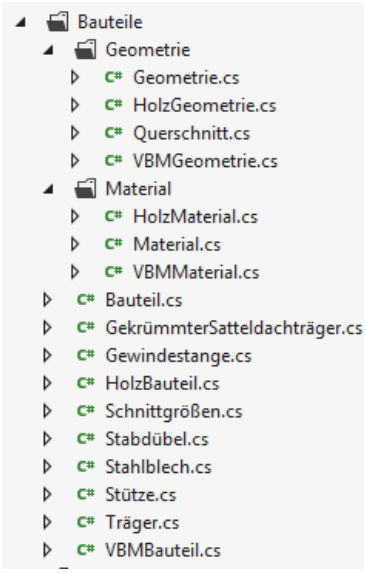


Abbildung 57: „Bauteil"-Ordner

Als Basisklasse für alle Bauteile dient die abstrakte Klasse „Bauteil". Sie gibt die Grundstruktur aller davon abgeleiteten Bauteilklassen vor. Direkt von der Klasse „Bauteil" abgeleitet sind die beiden Klassen „HolzBauteil" und „VBMBauteil". Sie spezialisieren die Eigenschaften der Bauteile zusätzlich. Von den beiden ebenfalls abstrakten Klassen sind dann die eigentlichen Bauteilklassen abgeleitet. Die Bauteilklassen sind dabei stets nach dem Bauteiltyp benannt, den sie erstellen. Welche der beiden abstrakten Klassen als Basisklasse für die Bauteilklasse dient, hängt entsprechend davon ab, ob es sich um ein Holzbauteil oder Verbindungsmittel handelt. Die folgende Grafik zeigt diese Struktur anhand eines einfachen Klassendiagramms der in diesem Programm umgesetzten Bauteiltypen.

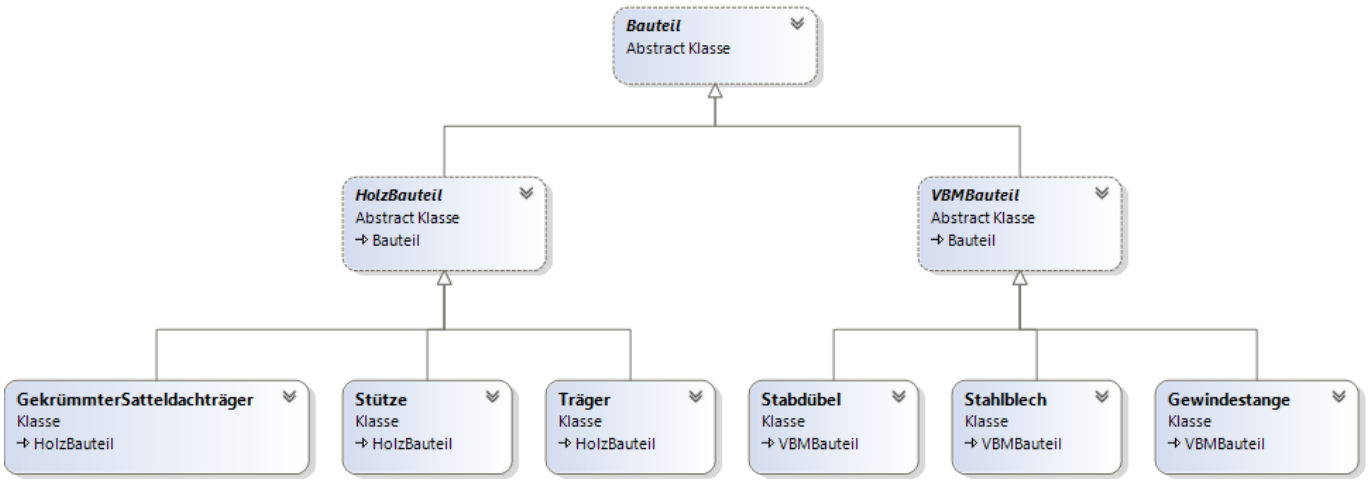


Abbildung 58: Klassendiagramm der Bauteile

Im Folgenden sollen die einzelnen Klassen und deren Umsetzung genauer beschrieben werden.

Bauteil.cs

Die Basisklasse „Bauteil" ist abstrakt, sodass keine Instanz dieser Klasse erstellt werden kann. Dadurch wird sichergestellt, dass jedes Bauteil eindeutig seinem Typ zugeordnet wird. Die Variablen der Klasse „Bauteil" sind alle als protected definiert. Dies stellt sicher, dass auf diese Variablen nur innerhalb der Klasse und in den von der Klasse abgeleiteten Klassen zugegriffen werden kann. Da jedoch beispielsweise auch in den zugehörigen GUIs auf die Variablen eines Bauteils zugegriffen werden muss, besitzt das Bauteil zudem Eigenschaften, welche den Zugriff auf diese Felder regeln. Diese sind als public definiert, da auf sie von außen zugegriffen werden soll. Diese als public definierten Eigenschaften sind auch für die XML-Serialisierung wichtig, da nur als public definierte Eigenschaften automatisch vom XmlSerializer serialisiert und damit exportiert beziehungsweise importiert werden können. Wichtig ist zudem, dass Eigenschaften nicht als Referenz übergeben werden können, da sie nicht als Variablen klassifiziert werden. Hierzu müssen immer die Felder verwendet werden. Die allgemeine Bauteilstruktur gibt die wichtigsten Eigenschaften aller Bauteile vor. Die im Programm umgesetzte Struktur ist den Vorüberlegungen zur Programmentwicklung sehr ähnlich.

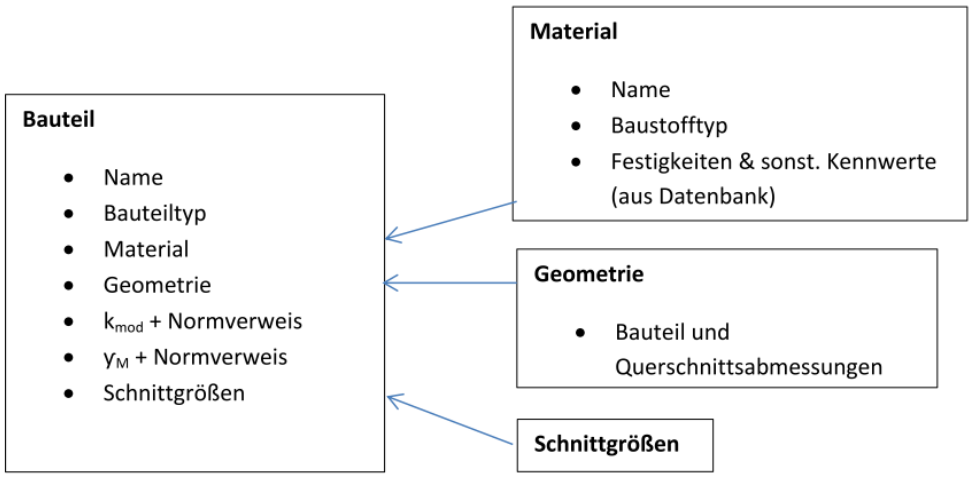


Abbildung 59: Bauteilstruktur

Zu den Bauteileigenschaften gehört ein eindeutiger Name zur Identifizierung des Bauteils, ein Bauteiltyp, jeweils ein Objekt vom Typ „Material", „Geometrie" und „Schnittgrößen" sowie die beiden Beiwerte k_{mod} und γ_M und deren zugehörige Normverweise (siehe Abbildung 59). Die Normverweise werden gespeichert um diese später in der XML-Ausgabedatei anzeigen zu können. Durch die Speicherung der Normverweise, kann die Berechnungen im Nachhinein nachvollzogen werden. Weil der XmlSerializer Objekte vom Typ „Bauteil" serialisiert, müssen vor der Deklaration der Klasse „XmlInclude"-Anweisungen stehen, welche die von der Klasse abgeleiteten Klassen vorstellt.

```
[XmlInclude(typeof(VBMBauteil))]
[XmlInclude(typeof(HolzBauteil))]
public abstract class Bauteil
{
```

Abbildung 60: „XmlInclude"-Anweisung

Der Konstruktor der Klasse „Bauteil" muss für die erfolgreiche Serialisierung zudem parameterlos und als public definiert sein. Der Konstruktor ist leer und enthält keine Daten oder Anweisungen. Die Klasse „Bauteil" gibt zudem zwei abstrakte Methoden vor, welche von den abgeleiteten Klassen implementiert werden müssen. Diese dienen zum Laden des zugehörigen Bauteileingabefensters im leeren und ausgefüllten Zustand.

```
//Methoden
//Laden der GUI
public abstract void LadeGUILeer(System.EventHandler BauteilTabelleUpdate);
public abstract void LadeGUIKomplett(FormClosedEventHandler BauteilTabelleUpdate, FormClosedEventHandler NachweisTabelleUpdate);
```

Abbildung 61: „Bauteil"-Methoden

Die Methoden sind als public definiert, so dass sie jederzeit von außen aufgerufen werden können. Dass die Methoden als abstrakt definiert sind, heißt, dass sie keinen Körper mit Anweisungen besitzen. Sie geben lediglich den Zugriffsmodifizierer und den Rückgabetyp vor. Die Methoden sollen jeweils EventHandler übergeben bekommen. Diese führen ein Update der Bauteiltabelle und der Nachweistabelle im Hauptfenster durch.

HolzBauteil.cs

Die Klasse „HolzBauteil" gibt die zu den Eigenschaften der Basisklasse zusätzlich für Holzbauteile benötigte Struktur vor. Die Klasse ist dabei ebenfalls als abstrakte Klasse definiert, da sie lediglich als Vorlage dient und von ihr keine Instanzen erstellt werden sollen. Alle Holzbauteile sollen zusätzlich zur allgemeinen Bauteilstruktur die Klasse der Lasteinwirkungsdauer KLED und ihre Nutzungsklasse speichern.

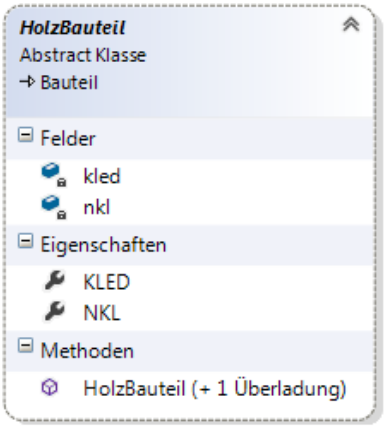


Abbildung 62: Klasse „HolzBauteil"

Die Klasse besitzt neben einem öffentlichen, parameterlosen Standardkonstruktor auch einen Konstruktor, welcher von allen Holzbauteilen zur Erstellung aufgerufen wird. Dem Konstruktor werden beim Aufruf neben dem Bauteilnamen und –typ auch Objekte vom Typ „HolzMaterial", „HolzGeometrie" und „Schnittgrößen", sowie die Nutzungsklasse und die Klasse der Lasteinwirkungsdauer des zu erstellenden Bauteils übergeben. Im Konstruktor werden die Daten den jeweiligen Feldern zugewiesen. Da die Klasse „Bauteil" vorgibt, dass jedes Bauteil ein Objekt der Klassen „Geometrie" und „Material" besitzen muss, bei diesen jedoch auch vom Prinzip der Vererbung

Gebrauch gemacht wurde, können auch Objekte von deren spezialisierten Klassen zugewiesen werden.

```
//Konstruktor
public HolzBauteil( string name, string bauteiltyp, HolzMaterial material, HolzGeometrie geometrie,
                    Schnittgrößen schnittgrößen, int NKL, string KLED)
{
    this.name = name;
    this.bauteiltyp = bauteiltyp;
    this.material = material;
    this.geometrie = geometrie;
    this.schnittgrößen = schnittgrößen;
    this.nkl = NKL;
    this.kled = KLED;

    //Gamma_M und K_mod aus den Eingangsgrößen NKL, KLED und dem Baustofftyp mit Hilfe der Funktionen der Klasse
    //BeiwertVerknüfungsFunktionen berechnen
    this.kMod = BeiwertVerknüfungsFunktionen.KModBestimmen(nkl, kled, material.Baustofftyp, ref norm_kMod);
    this.gammaM = BeiwertVerknüfungsFunktionen.GammaMBestimmenNA(material.Baustofftyp, ref norm_GammaM);
}
```

Abbildung 63: Konstruktor der abstrakten Klasse „HolzBauteil"

Da die Beiwerte k_{mod} und γ_M in den Nachweisen immer wieder verwendet werden, werden sie bei der Erstellung des Bauteils im Konstruktor einmalig bestimmt und im Bauteil selbst gespeichert (siehe Abbildung 63). Hierzu werden die Methoden der statischen Klasse „BeiwertVerknüpfungsFunktionen" aufgerufen. Den Methoden werden alle zur Berechnung benötigten Kennwerte sowie per Referenz auch der entsprechende Normverweis übergeben.

Die von „HolzBauteil" abgeleitete Klasse „Stütze" besitzt wie alle anderen spezialisierten Klassen keine weiteren Eigenschaften. Dieses Konzept wurde verwendet um sicher zu stellen, dass alle Holzbauteile den gleichen Grundaufbau besitzen. Die Unterschiede in der Geometrie wurden anhand einer Verallgemeinerung gelöst, um bei der Berechnung allgemeingültig mit einem Holzbauteil rechnen zu können. Der Konstruktor der Klasse „Stütze" ruft anhand des Schlüsselworts base den Konstruktor der Basisklasse „HolzBauteil" auf und übergibt diesem seine Parameter (siehe Abbildung 64).

```
//Konstruktor
public Stütze(string name, string bauteiltyp, HolzMaterial material, HolzGeometrie geometrie,
              Schnittgrößen schnittgrößen, int NKL, string KLED)
:base (name,bauteiltyp, material,geometrie,schnittgrößen, NKL,KLED)
{

}
```

Abbildung 64: Konstruktor der Klasse „Stütze"

Zudem müssen alle abgeleiteten Klassen die abstrakten Methoden „LadeGUILeer" und „LadeGUIKomplett" der Basisklasse implementieren und überschreiben diese. In den Methoden wird jeweils ein neues Objekt des zugehörigen Eingabefensters, in diesem Fall „StützeEingabe_Form" erstellt und geöffnet. Zudem werden einem Event der Form die übergebenen EventHandler zugeordnet. Bei der „LadeGUILeer"-Methode hat das die Auswirkung, dass beim Klick auf den „Eingabe bestätigen"-Button die Bauteiltabelle upgedatet wird. Bei der „LadeGUIKomplett"-Methode wird beim Schließen des Eingabefensters ein Update der Bauteil- und der Nachweistabelle ausgeführt.

Wenn die Methode „LadeGUIKomplett" aufgerufen wird, und somit ein ausgefülltes Fenster dargestellt wird, wird zudem eine Methode der GUI aufgerufen, welche das aktuelle Objekt übergeben bekommt. Die GUI-Methode weist dann den Elementen des Eingabefensters die Bauteilinformationen des übergebenen Bauteils zu.

VBMBauteil.cs

Die Klasse „VBMBauteil" dient der Spezialisierung der Verbindungsmittelbauteile. Auch diese Klasse ist eine abstrakte Klasse und besitzt keine zusätzlichen Eigenschaften.

Abbildung 65: Klasse „VBMBauteil"

Neben dem öffentlichen, parameterlosen Standardkonstruktor besitzt auch diese Klasse einen öffentlichen Konstruktor, welcher von allen abgeleiteten Verbindungsmittelklassen aufgerufen wird. Dem Konstruktor werden ebenfalls Bauteilname- und typ, sowie Material und Geometrie von den spezialisierten Klassen „VBMMaterial" und „VBMGeometrie" übergeben. Die Übergabe der Schnittgrößen bleibt jedoch aus. Dies liegt daran, dass die Verbindungsmittelschnittgrößen beispielsweise von der Anzahl der Verbindungsmittel abhängig sind und erst im Nachweis selbst aus der Belastung ermittelt und dann zugewiesen werden. Im Konstruktor finden wieder die Variablenzuweisung und die Berechnung des Teilsicherheitsbeiwerts γ_M statt. Abbildung 66 zeigt die Struktur einer spezialisierten Verbindungsmittelklasse am Beispiel der Klasse „Stabdübel".

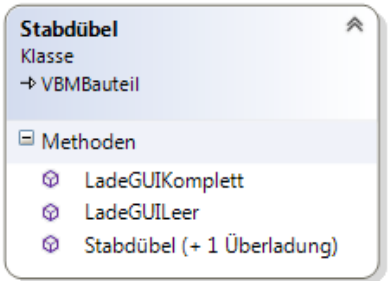


Abbildung 66: Klasse „Stabdübel"

Neben dem parameterlosen Standardkonstruktor besitzt diese Klasse wieder einen öffentlichen Konstruktor, welcher den Konstruktor der Basisklasse „VBMBauteil" aufruft. Zudem werden auch hier die beiden Methoden „LadeGUILeer" und „LadeGUIKomplett" überschrieben. Das Konzept ist dabei analog zur Klasse „Stütze" aufgebaut.

Geometrie.cs

Bei der Struktur der Bauteilgeometrie wurde ebenfalls das Konzept der Vererbung verwendet. Es gibt eine abstrakte Klasse „Geometrie", welche als Basisklasse dient. Die Klasse „Geometrie" selbst besitzt dabei keine Eigenschaften. Sie dient lediglich zur Verallgemeinerung aller Geometrien, wodurch der Basisklasse „Bauteil" vorgegeben werden kann, eine Geometrie zu besitzen.

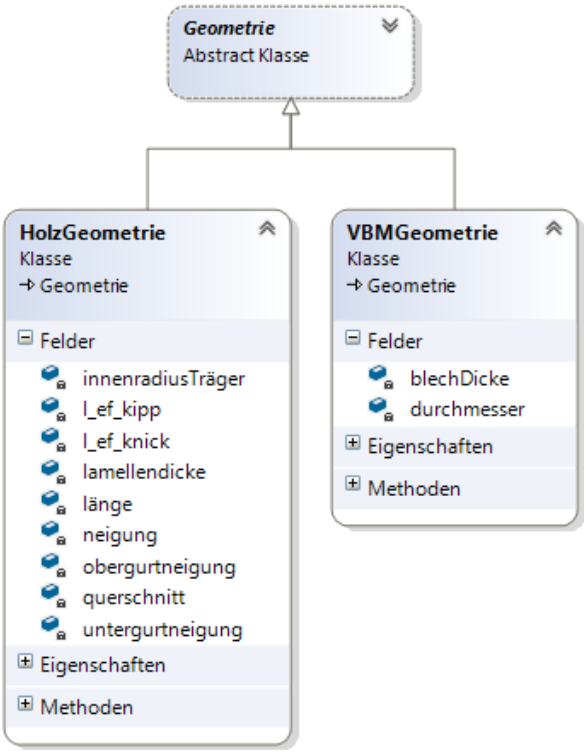


Abbildung 67: Klassendiagramm „Geometrie"

Von der Klasse „Geometrie" abgeleitet wurden eigene Geometrieklassen für die Holzbauteile und Verbindungsmittel verwendet, um den unterschiedlichen Geometriedaten der Bauteilgruppen gerecht zu werden.

HolzGeometrie.cs

Die Klasse „HolzGeometrie" gibt die Geometrie aller Objekte der von „HolzBauteil" abgeleiteten Klassen vor. Neben einem Objekt der Klasse „Querschnitt" besitzt sie zudem zahlreiche, für Holzbauteile typische Geometriedaten. Diese werden als private Felder und zur Zugriffshandhabung und für die XML-Serialisierung auch als Eigenschaften zur Verfügung gestellt. Neben dem parameterlosen Standardkonstruktor besitzt die Klasse einen öffentlichen Konstruktor, welchem bei der Geometrieerstellung alle benötigten Daten als Parameter übergeben werden. Von einer weiteren Spezialisierung wurde abgesehen, da in den Nachweisen verallgemeinert mit Holzbauteil- und Verbindungsmittelgeometrien gerechnet wird.

```
//Konstruktor
public HolzGeometrie(Querschnitt querschnitt, double länge, double l_ef_knick, double l_ef_kipp,
                     int neigung, int lamellendicke, int innenradiusGekrRand,
                     double obergurtneigung, double untergurtneigung)
{
    this.querschnitt = querschnitt;
    this.länge = länge;
    this.l_ef_knick = l_ef_knick;
    this.l_ef_kipp = l_ef_kipp;
    this.neigung = neigung;
    this.lamellendicke = lamellendicke;
    this.innenradiusTräger = innenradiusGekrRand;
    //Zuweisung der Winkel sowie deren Umrechnung vom Gradmaß ins Bogenmaß
    this.obergurtneigung = obergurtneigung * 2 * Math.PI / 360;
    this.untergurtneigung = untergurtneigung * 2 * Math.PI / 360;
}
```

Abbildung 68: Konstruktor der Klasse" HolzGeometrie"

Querschnitt.cs

In der Klasse „Querschnitt" werden alle Querschnittsdaten der Holzbauteile gespeichert. Die Kennwerte werden wieder als Felder und Eigenschaften zur Verfügung gestellt. Neben einem parameterlosen Konstruktor besitzt die Klasse „Querschnitt" einen Konstruktor, welchem die in der GUI eingegebenen Geometriedaten übergeben werden. In diesem Konstruktor werden sie den entsprechenden Feldern zugeordnet. Zudem wird im Konstruktor die „Hauptprogramm"-Methode zur Bestimmung der Querschnittswerte aufgerufen.

VBMGeometrie.cs

Die Klasse „VBMGeometrie" speichert lediglich den Durchmesser des Verbindungsmittels und die Blechdicke. Die Klasse kann jederzeit um zusätzliche Parameter erweitert werden. Da in den Nachweisen bekannt ist, um welche Art von Verbindungsmittel es sich handelt, ist sichergestellt, dass nur auf gültige Eigenschaften zugegriffen wird.

Material.cs

Auch bei der Klassenstruktur der Materialien wurde das Prinzip der Vererbung verwendet. Die abstrakte Klasse „Material" gibt vor, dass jedes Material einen Namen und einen Baustofftyp besitzen muss. Ansonsten besitzt die Klasse lediglich einen öffentlichen, parameterlosen Standardkonstruktor ohne Anweisungen.

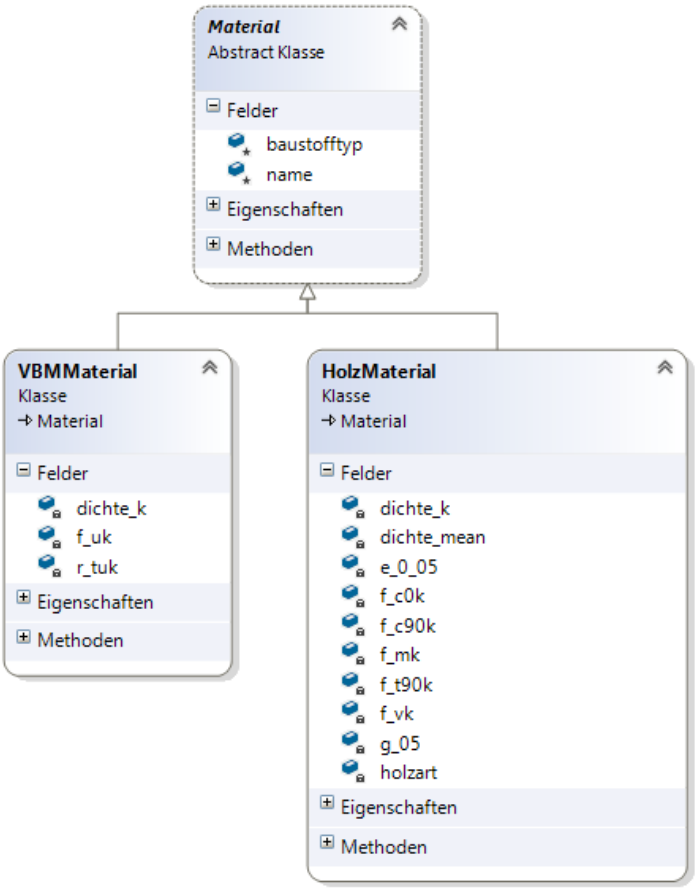


Abbildung 69: Klassendiagramm „Material"

Von der Klasse „Material" abgeleitet existieren die Klassen „VBMMaterial" und „HolzMaterial" (siehe Abbildung 69).

HolzMaterial.cs

Die Klasse „HolzMaterial" beinhalten alle für Holzbauteile typischen Materialeigenschaften. Sie werden als Felder und Eigenschaften zur Verfügung gestellt. Die Klasse „HolzMaterial" wird von allen Holzbauteilen, unabhängig von deren Typ verwendet. Eine weitere Spezialisierung in Form von abgeleiteten Klassen findet nicht statt, da die Materialeigenschaften unabhängig von Bauteiltyp sind. Neben dem Standardkonstruktor besitzt die Klasse einen öffentlichen Konstruktor, der von allen Holzbauteilen verwendet wird (siehe Abbildung 70).

```
public HolzMaterial(DataTable materialtable,string name)
{
    this.name = name;

    //Abrufen der Materialkennwerte aus dem zugehörigen DataTable
    Programm.NeuesProgramm.MaterialkennwerteHolz(materialtable, name, ref baustofftyp, ref holzart,
                                                 ref f_vk, ref f_c0k, ref f_c90k, ref f_t90k, ref f_mk,
                                                 ref e_0_05, ref g_05, ref dichte_k, ref dichte_mean);
}
```

Abbildung 70: Konstruktor der Klasse „HolzMaterial"

Als Parameter werden dem Konstruktor der in der Eingabemaske verwendete DataTable sowie der eindeutige Materialname übergeben. Nach der Zuweisung des Namens wird die Methode „MaterialkennwerteHolz" der aktuellen Instanz des Programms aufgerufen. Alle Parameter außer dem DataTable und dem Namen werden dabei per Referenz übergeben, sodass die in der Methode vorgenommenen Änderungen auch außerhalb der Methode als gültig übernommen werden. Die Methode selbst dient dazu die Daten aus dem DataTable für das anhand des Namens identifizierte Material auszulesen und zuzuweisen.

VBMMaterial.cs

Die Klasse „VBMMaterial" ist von der abstrakten Superklasse „Material" abgeleitet und dient zur Speicherung der Materialien der Verbindungsmittel. Die Klasse besitzt dabei weniger Elemente als die Klasse „HolzMaterial". Der Konstruktor der Klasse ist jedoch ähnlich aufgebaut. Neben der Namenszuweisung findet auch noch die Festlegung des Baustofftyps als Verbindung statt. Die Materialkennwerte werden anhand der „MaterialkennwerteVBM"-Methode ebenfalls aus dem übergebenen DataTable ausgelesen und zugewiesen.

Schnittgrößen.cs

In der Klasse „Schnittgrößen" werden die Schnittgrößen eines Bauteils gespeichert. Die Klasse besitzt zahlreiche Felder beziehungsweise Eigenschaften, um die benötigten Schnittgrößen eines Bauteils zu speichern. Neben dem Standardkonstruktor verfügt die Klasse über den unten abgebildeten von außen aufrufbaren Konstruktor. Dem Konstruktor werden die im Bauteileingabefenster festgelegten Schnittgrößenwerte übergeben. Neben der Wertzuweisung findet im Konstruktor zudem eine Umrechnung der Schnittgrößen von kN in N und kNm in Nmm statt um eine einfache Berechnung im Nachweis zu ermöglichen.

```
//Konstruktor
public Schnittgrößen(double m_yd, double v_yd, double m_zd, double v_zd, double n_d, double m_yd_ap,
                     double v_zd_ap, double m_yd_xm, double m_yd_an, double v_zd_an)
{
    //Umrechnung der Schnittgrößen von KN in N und KNm in Nmm
    this.m_yd = m_yd * 1000000;
    this.v_yd = v_yd * 1000;
    this.m_zd = m_zd * 1000000;
    this.v_zd = v_zd * 1000;
    this.n_d = n_d * 1000;
    this.m_yd_xm = m_yd_xm * 1000000;
    this.m_yd_ap = m_yd_ap * 1000000;
    this.v_zd_ap = v_zd_ap * 1000;
    this.m_yd_an = m_yd_an * 1000000;
    this.v_zd_an = v_zd_an * 1000;
    this.v_zd_erhoeht = v_zd * 1000;
}
```

Abbildung 71: Konstruktor der Klasse „Schnittgrößen"

7.3.3 Nachweise

Die Nachweise sind so gegliedert, dass der Nutzer neue Nachweise immer nur in einem zugehörigen Nachweispaket hinzufügen kann. Das Nachweispaket besitzt eine Liste an Nachweisen, welche jederzeit erweitert oder reduziert werden kann. Die jeweiligen Klassen der verschiedenen Nachweistypen sind alle von der Klasse „Nachweis“ abgeleitet.

Nachweispaket.cs

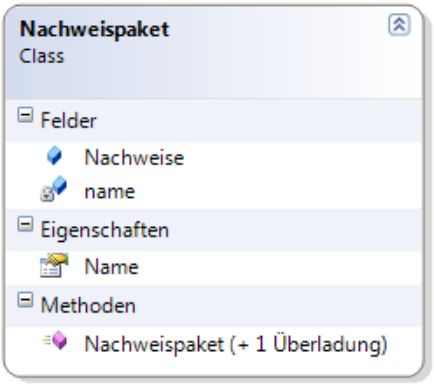


Abbildung 72: Klasse „Nachweispaket“

Zum Anlegen neuer Nachweispakete steht diese Klasse zur Verfügung. Jedes Nachweispaket besitzt einen Namen, welcher eindeutig sein muss, um das Paket in der Nachweispaketliste des Programms identifizieren zu können. Zudem enthält die Klasse eine öffentliche Liste von Nachweisen. Neben einem Standardkonstruktor besitzt die Klasse den untenstehenden öffentlichen Konstruktor.

```
public Nachweispaket(string name)
{
    this.Nachweise = new List<Nachweis>();
    this.name = name;
}
```

Abbildung 73: Konstruktor der Klasse „Nachweispaket“

Dem Konstruktor wird der Nachweispaketname übergeben. Im Konstruktor selbst wird dann der Name zugewiesen und eine neue Nachweisliste erstellt.

Nachweis.cs

Die abstrakte Klasse „Nachweis“ dient als Basisklasse für alle implementierten Nachweistypen (siehe Abbildung 74).

Abbildung 74: Klassendiagramm „Nachweis“

Die Klasse gibt die Grundstruktur eines jeden Nachweises vor. Jeder Nachweis muss zur Identifizierung einen in seinem Nachweispaket eindeutigen Namen besitzen. Zur Beschreibung der Art des Nachweises dient der Nachweistyp. Um die Berechnungen im Nachhinein nachvollziehen zu können, wird zudem die Norm gespeichert, nach welcher die Bemessung stattfindet. Zusätzlich besteht ein Feld, mit den am Nachweis beteiligten Bauteilen. Das Feld enthält immer Objekte der abstrakten Klasse „Bauteil“ um alle Bauteilarten speichern zu können. Neben dem Ergebnis des Nachweises, dem Ausnutzungsgrad besitzt jeder Nachweis einen Berechnungsstatus, der angibt ob ein Bauteil zugeordnet wurde und ob der Nachweis mit den aktuellen Eingaben berechnet wurde oder nicht, sowie einen Erfüllungsstatus, der auf einen Blick angibt, ob der Nachweis erfüllt ist oder nicht. Da jeder Nachweistyp seinen eigenen Konstruktor besitzt, verfügt diese Klasse lediglich über den parameterlosen, öffentlichen Standardkonstruktor.

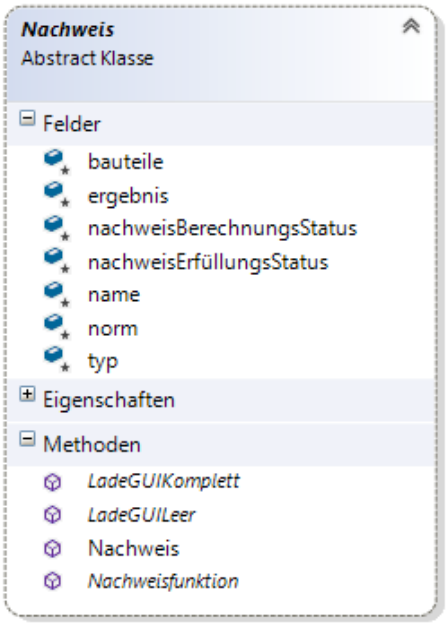


Abbildung 75: Klasse „Nachweis“

Die „Nachweis“-Klasse gibt für alle Nachweise die abstrakte, parameterlose Methode „Nachweisfunktion“ vor. In dieser Methode soll die eigentliche Nachweisführung stattfinden. Da die Methode in der Basisklasse enthalten und als public definiert ist, kann sie jederzeit von außen, unabhängig vom Nachweistyp aufgerufen werden.

```
//Methoden
//Berechnen des Nachweises
public abstract void Nachweisfunktion();

//Laden der GUI
public abstract void LadeGUILeer(string nachweispaketname);
public abstract void LadeGUIKomplett(string nachweispaketname);
```

Abbildung 76: „Nachweis“-Methoden

Sämtliche Nachweistypklassen besitzen durch die Nachweisklasse zudem, ähnlich wie die Bauteilklassen, die Methoden „LadeGUILeer“ und „LadeGUIKomplett“.

Die Struktur einer spezialisierten, also von der Klasse „Nachweis“ abgeleiteten Klasse wird am Beispiel des Schubspannungsnachweises erläutert. Für die Berechnung werden zusätzliche Variablen benötigt. Sie werden in der Klasse ebenfalls als Felder definiert. Alle Variablen, die zusätzlich von außen zugreifbar sein sollen oder in der XML-Ausgabe berücksichtigt werden sollen, werden zudem als Eigenschaften definiert. Die Klasse „Schubspannungsnachweis“ verfügt wie alle andere Nachweistypklassen über insgesamt drei öffentliche Konstruktoren (siehe Abbildung 77).

```
//Standardkonstruktor
public Schubspannungsnachweis()
{ }

//Konstruktor ohne ein Bauteil und die dazugehörigen Daten zu übergeben
public Schubspannungsnachweis(string name)
{
    this.typ = "Schubspannungsnachweis";
    this.norm = "DIN EN 1995-1-1 6.1.7 + NA NDP/NCI zu 6.1.7";
    this.name = name;
    this.NachweisBerechnungsStatus = "Kein Bauteil";
}

//Konstruktor komplett
public Schubspannungsnachweis(HolzBauteil bauteil, string name)
{
    this.typ = "Schubspannungsnachweis";
    this.name = name;
    this.norm = "DIN EN 1995-1-1 6.1.7 + NA NDP/NCI zu 6.1.7";
    this.bauteile = new Bauteil[1];
    this.bauteile[0] = (HolzBauteil)bauteil;
}
```

Abbildung 77: Konstruktoren der Klasse „Schubspannungsnachweis"

Neben dem parameterlosen Standardkonstruktor gibt es einen Konstruktor für das Anlegen eines leeren Nachweises ohne Bauteil oder sonstige Daten. In diesem Fall wird in der Eingabemaske der Name des Nachweises vom Anwender festgelegt. Im Konstruktor werden der eingegebene Name, der Nachweistyp und die die zugehörige Norm den Feldern zugewiesen. Da von außen, in der Nachweistabelle des Hauptfensters, ersichtlich sein soll, dass es sich um einen leeren Nachweis handelt wird der Berechnungsstatus zudem auf „Kein Bauteil" gesetzt. Der zusätzliche Konstruktor dient der Erstellung eines neuen Nachweises mit vollständigen Eingabedaten. Hierzu müssen dem Konstruktor alle zum Nachweisnamen zusätzlich in der GUI eingegebenen Daten als Parameter übergeben werden. Im Fall des Schubspannungsnachweises ist das, das am Nachweis beteiligte Bauteil vom Typ „HolzBauteil". Im Konstruktor finden neben der Zuweisung des Nachweisnamens, -typs und der zugehörigen Norm dann auch die Erstellung des Bauteilfeldes und das Einfügen des Bauteils in dieses statt.

Die nachfolgende Abbildung zeigt die Umsetzung der Nachweisfunktion am Beispiel des Schubspannungsnachweises. Zu Beginn werden Geometrie, Material und Schnittgrößen des Bauteils Feldern zugewiesen, um einen einfacheren Zugriff und die Übersichtlichkeit der Nachweisführung zu gewährleisten. Alle für den Nachweis benötigten Größen werden nacheinander berechnet. Hierzu kann über das Bauteil direkt auf dessen Eingangsparameter zugegriffen werden. Innerhalb der Nachweisfunktionen können die Parameter des Bauteils zudem direkt verändert werden. Während die Logik dieses speziellen Nachweises direkt in der Methode umgesetzt wird, werden mehrmals genutzte Normbeiwerte in den „VerknüpfungsFunktionen" gespeichert und von der Nachweisfunktion aus aufgerufen. Die Normreferenzen werden dabei stets per Referenz übergeben und in der Verknüpfungsfunktion festgelegt. Die normativen Bedingungen und Fallunterscheidungen des Nachweises werden beispielsweise durch die Verwendung von if-Abfragen realisiert. Im Beispiel wird zwischen den verschiedenen Lastfällen unterschieden um den gültigen Ausnutzungsgrad zu berechnen und diesen dann dem Feld „Ergebnis" zuzuweisen. Wurde der Ausnutzungsgrad berechnet,

wird der Berechnungsstatus auf „berechnet" und der Erfüllungsstatus je nach Ergebnis auf true oder false gesetzt.

```
//Nachweisfunktion
public override void Nachweisfunktion()
{
    material = (HolzMaterial)bauteile[0].Material;
    geometrie = (HolzGeometrie)bauteile[0].Geometrie;
    schnittgrößen = bauteile[0].Schnittgrößen;

    //Effektive Breite berechnen
    bef = BeiwertVerknüfungsFunktionen.BefBestimmen(material.Baustofftyp, material.Holzart,
                                geometrie.Querschnitt.Breite, material.F_vk, ref norm_bef);

    //Berechnung des Bemessungswert der Schubfestigkeit
    f_vd = bauteile[0].Kmod * material.F_vk / bauteile[0].GammaM;

    //Berechnung dere Bemessungswert der Schubspannung
    t_yd = ((schnittgrößen.V_zd * geometrie.Querschnitt.S_y) / (geometrie.Querschnitt.I_y * bef));
    t_zd = ((schnittgrößen.V_yd * geometrie.Querschnitt.S_z) / (geometrie.Querschnitt.I_z * bef));

    //Fallunterscheidung ob einfache oder Doppelbiegung
    //Einfache Biegung
    if (t_zd!=0)
    {
        //Berechnung des Ausnutzungsgrads//Nachweisführung
        ergebnis = t_zd / f_vd;
    }
    //Einfache Biegung
    else if (t_yd != 0)
    {
        //Berechnung des Ausnutzungsgrads//Nachweisführung
        ergebnis = t_yd / f_vd;
    }
    if (geometrie.Querschnitt.Querschnittstyp == "Rechteck")
    {
        //Doppelbiegung
        if ((t_yd != 0) && (t_zd != 0))
        {
            //Berechnung des Ausnutzungsgrads//Nachweisführung
            ergebnis = Math.Pow(t_yd / f_vd, 2) + Math.Pow(t_zd / f_vd, 2);
        }
    }

    //Nachweis ist "berechnet"
    NachweisBerechnungsStatus = "berechnet";

    //Bestimmen des NachweisErfüllungsStatus
    if (ergebnis <= 1)
    {
        NachweisErfüllungsStatus = true;
    }
    else
    {
        NachweisErfüllungsStatus = false;
    }
}
```

Abbildung 78: „Nachweisfunktion"

Neben der Nachweisfunktion werden in den spezialisierten Nachweisklassen auch die beiden Methoden „LadeGUILeer“ und „LadeGUIKomplett“ überschrieben. In den Methoden wird ein neues Eingabefenster des jeweiligen Nachweistyps erstellt und angezeigt. Dem Eingabefenster wird der entsprechende Paketname übergeben. Im Falle der „LadeGUIKomplett“-Methode wird zudem das aktuelle Objekt übergeben, von welchem aus die Methode aufgerufen wird, so dass die zugehörigen Nachweisinformationen angezeigt werden können.

INachweisGUI.cs

Wie auch die Bauteilklassen besitzen alle abgeleiteten Nachweisklassen ein eigenes Eingabefenster. Da das Nachweisfenster mehr Methoden besitzen muss als ein Bauteilfenster, beispielsweise um den Nachweis zu berechnen, oder einen leeren Nachweis anzulegen, wird das Interface „INachweiseGUI“ erstellt. Diese Schnittstelle wird von allen Nachweisfenstern implementiert und dient dazu die Existenz der benötigten Methoden sicherzustellen.

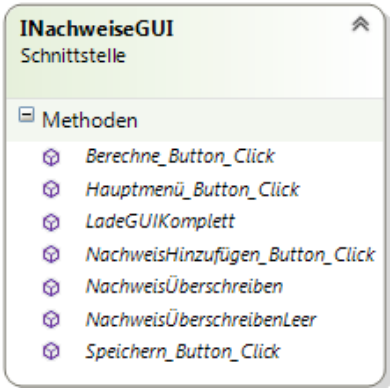


Abbildung 79: Schnittstelle „INachweisGUI“

7.3.4 Verknüpfungsfunktionen

Alle nach der Norm berechneten Beiwerte werden zentral gespeichert. Sie sind thematisch in die beiden Klassen „BeiwertVerknüpfungsFunktionen“ und „AnschlussnachweisVerknüpfungsFunktionen“ aufgeteilt und innerhalb der Klassen weiter nach Themen gegliedert. Weitere Aufteilungen, zum Beispiel nach Norm sind denkbar. Die beiden Klassen sind im Namespace „VerknüpfungsFunktionen“ gespeichert. Die Klassen sind statische Klassen, sie können nicht instanziiert werden. Der Zugriff auf die Methoden erfolgt direkt über den Klassennamen. Die Klassen stellen lediglich Methoden zur Verfügung, welche ebenfalls alle statisch sind. Die folgenden Abbildungen zeigen alle momentan in den beiden Klassen vorhandenen Methoden.

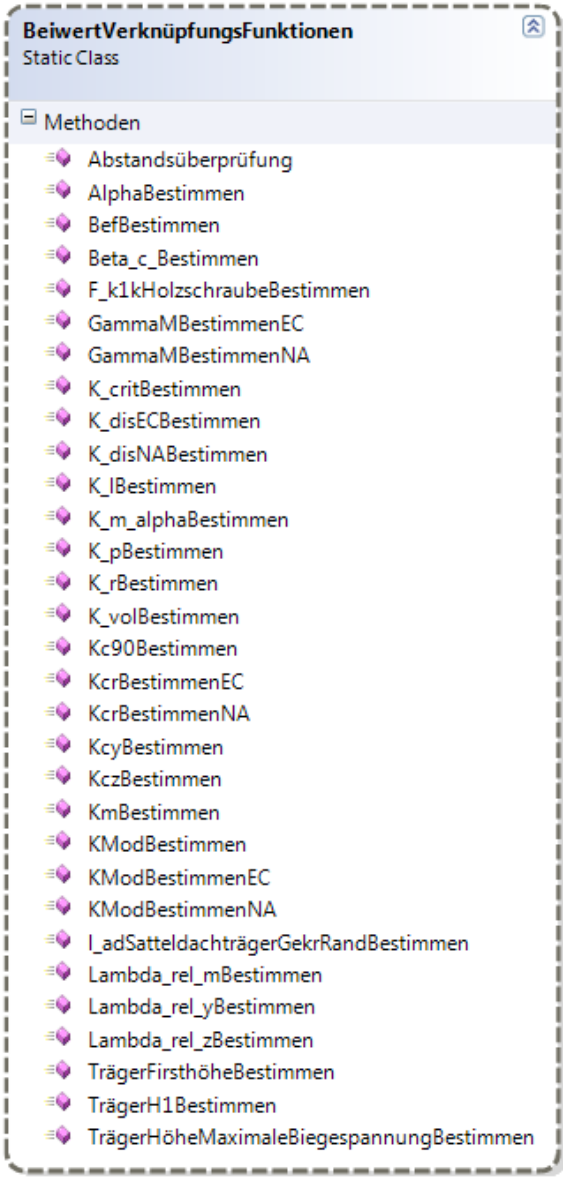


Abbildung 80: Klasse „BeiwertVerknüpfungsFunktionen“

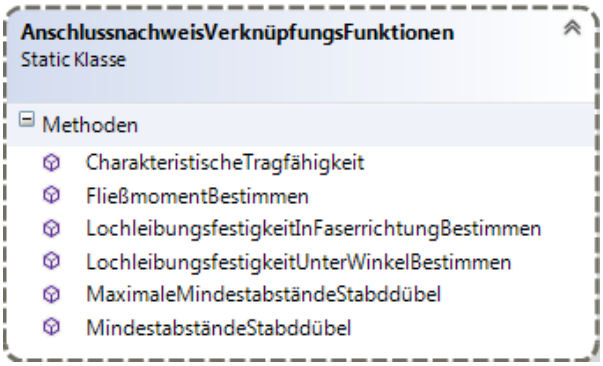


Abbildung 81: Klasse „AnschlussnachweisVerknüpfungsFunktionen“

7.3.5 GUI

Im Ordner „GUI“ sind alle Klassen zur Bearbeitung der grafischen Oberfläche zusammengefasst. Innerhalb dieser Klassen sind sowohl die Darstellung der Fenster, wie auch die Reaktion auf die verschiedenen Aktionen des Benutzers geregelt. Beim Erstellen einer Stütze wird in der hier gespeicherten Klasse „StützeEingabe_Form“ die Vollständigkeit der Eingaben und die Eindeutigkeit des eingegebenen Namens überprüft. Sind diese Bedingungen erfüllt werden neue Objekte von den Typen „Querschnitt“, „HolzGeometrie“, „HolzMaterial“ und „Schnittgrößen“ erstellt. Anschließend wird mit diesen Parametern ein neues Objekt vom Typ „Stütze“ erstellt und dieses mit der zugehörigen Membermethode der Klasse „Hauptprogramm“ der aktuellen Bauteilliste hinzugefügt. Innerhalb des Codes der Eingabefenster befindet sich nicht nur die Darstellungslogik, sondern auch die Verarbeitung der Daten.

7.4 Fazit des ersten Programms

7.4.1 Umsetzung und Problemstellungen

Bei der Untersuchung der Norm wurde eine gewisse Grundstruktur für das Programm entwickelt und im Programmcode umgesetzt. Wie in der vorangegangenen Beschreibung der Programmstruktur zu erkennen ist, wurde die Grundidee eines Programms mit einer Bauteil- und einer Nachweispaketliste realisiert. Vorerst war vorgesehen, ein Startfenster zu erstellen, welches lediglich vier Schaltflächen mit den verschiedenen Optionen besitzen sollte. Je nach Wunsch des Benutzers sollte sich dann durch einen Klick auf den entsprechenden Button das zugehörige Fenster öffnen und nach der Eingabe auch wieder schließen. Um die Ergebnisse eines Nachweispakets anschauen zu können, oder ein Bauteil zu ändern, hätte erst ein Fenster geöffnet werden müssen, in dem alle Elemente angezeigt werden. Aus diesem Fenster heraus hätte sich dann nach der Wahl des Elements das jeweilige Eingabefenster geöffnet. Durch die Umsetzung der Idee ergab sich jedoch eine sehr verschachtelte und vor allem unübersichtliche Ablaufstruktur für den Nutzer.

Das umgesetzte Konzept ist deutlich benutzerfreundlicher. Es ermöglicht die zentrale Darstellung, Auswahl und Bearbeitung der Elemente innerhalb des Hauptfensters. Das Hauptfenster ist groß, übersichtlich, thematisch strukturiert aufgebaut und bleibt während der ganzen Anwendung im Hintergrund geöffnet. Durch die Wahl des zu erstellenden Bauteiltyps und der Eingabe des Nachweispaketnamens innerhalb des Hauptfensters werden Zwischenschritte vermieden. Auch bei der grafischen Umsetzung wurde darauf geachtet, dass die Benutzerfreundlichkeit und Verständlichkeit im Vordergrund steht. Die Eingabefenster sind thematisch strukturiert und mit selbsterklärenden Beschriftungen versehen. Für den Fall, dass Eingaben fälschlicherweise ungenau oder unvollständig sind, wurden Warnungen mit vordefinierten Texten erstellt und die Speicherung oder Berechnung verhindert. Für die Umsetzung der überlegten Programmstruktur war es wichtig, dass die Objekte immer eindeutig identifizierbar sind. Wie bereits im Entwurf wurde für das Projekt der Name zur Identifikation gewählt. Die hierfür benötigte Eindeutigkeit des Namens wird bei der Speicherung überprüft. Falls ein bereits existierenden Namens verwendet wird, wird eine Warnung ausgegeben und der Speicherprozess abgebrochen. Dieses Prinzip wurde bei den Bauteilen, bei den Nachweispaketen und bei den Nachweisen innerhalb eines Nachweispakets umgesetzt. Es stellt sicher, dass diese Elemente innerhalb des Programmcodes allein anhand ihres Namens identifiziert und bearbeitet werden können.

Im Bereich der Bauteile wurde das Konzept der Spezialisierung durch Vererbung realisiert. Jedes Bauteil bekommt die gleiche Grundstruktur durch die Klasse „Bauteil" vorgegeben. Bei dieser Struktur wurde, während der Programmierung entschieden, welche Eigenschaften den Bauteilen vorgegeben werden. Dabei wurde berücksichtigt, dass jeder Nachweis ein Feld mit Objekten der Basisklasse „Bauteil" enthält und daher ohne eine Typumwandlung nur auf deren Eigenschaften zugegriffen werden kann. Daher besitzt die Basisklasse „Bauteil" auch den Beiwert k_{mod}, obwohl dieser bei Verbindungsmitteln nicht genutzt wird. Die Nachweis- sowie die allgemeine Programmstruktur hatte somit Einfluss auf die Bauteilarchitektur. Die spezifischen Unterschiede zwischen den Holzbauteilen und den Verbindungsmitteln wurde durch die eigenen Basisklassen umgesetzt. Hier findet auch die Spezialisierung des Materials und der Geometrie statt. Zu Beginn der Entwicklung, hatte ein Bauteiltyp keine eigene Klasse, um auf eine verallgemeinerte Weise auf die Bauteildaten zuzugreifen. Diese Idee wurde verworfen, da je nach Typ auch eigene Eingabefenster erstellt wurden und die Unterscheidung von Bauteilen nach der Eigenschaft des Bauteiltyps unübersichtlich erschien. Der Entwurf wurde optimiert, wobei für jede Bauteilart eine eigene Bauteilklasse entworfen wurde. Diese enthält keine spezialisierten Eigenschaften. Ziel der allgemeinen Datenspeicherung ist die Anwendbarkeit von allgemein gültigen Nachweisfunktionen. Im Falle einer Erweiterung um einen neuen Bauteiltyp muss damit nicht noch eine Abänderung des Nachweises für diesen Bauteiltyp erfolgen. Aus diesem Grund wird innerhalb des Nachweises hauptsächlich mit Bauteilen des Typs „HolzBauteil" oder VBMBauteil" gearbeitet.

Die Herausforderung während der Programmentwicklung bestand insbesondere darin, die Daten möglichst allgemeingültig zu halten und trotzdem in einigen Fällen zu spezialisieren. Deutlich wird dies an der verallgemeinerten Umsetzung der Klassen „Geometrie" und „Schnittgrößen". Die Eigenschaften dieser Klassen wurden möglichst umfangreich gestaltet, um alle Bauteile abzudecken. Einige Eigenschaften finden hierbei bei manchen Bauteilen keine Verwendung und werden mit Null belegt. Obwohl dies dem eigentlichen Konzept der Vererbung und Spezialisierung wiederspricht, stellt diese Umsetzung eine geeignete Lösung für die Herausforderung dar.

In den ersten Phasen der Programmentwicklung wurden die verschiedenen Bauteiltypen an zahlreiche Stellen im Code verwendet. Später wurde erreicht, dass die Bauteiltypen einmalig im Konstruktor des Hauptprogramms in Form der KeyValuePair-Liste definiert werden. Die verschiedenen Typen werden außerdem in den eigenen Klassen und in den Klassen der Eingabefenster sowie innerhalb der „XmlInclude"-Anweisungen verwendet. Dadurch wird der Zugriff auf den bestehenden Programmcode im Falle einer Erweiterung minimal gehalten.

Eine weitere Herausforderung stellte die Handhabung mit den im Bauteil vorhandenen Schnittgrößen dar. Dem Bemessungsprogramm geht kein Berechnungsprogramm voran, weshalb kein Schnittgrößenverlauf vorliegt. Welche Schnittgrößen des Bauteils bekannt sein müssen, hängt vom Nachweis ab, der geführt werden soll. Bereits in der Eingabemaske eines Holzbauteils muss festgelegt werden, welche Schnittgrößen an welcher Stelle wirken. Die Schnittgrößenklasse wurde deshalb umfangreich gestaltet. Es liegt aber weiterhin beim Nutzer, die Schnittgrößen im Hinblick auf die Nachweise richtig im Eingabefenster festzulegen.

Die Entwicklung der Nachweisstruktur orientiert sich an der händischen Bemessung. Da die eigentliche Nachweisführung in der Nachweisfunktion stattfindet, werden an dieser Stelle alle Schritte

der Bemessung logisch umgesetzt. Hierzu wurden vor allem Fallunterscheidungen eingesetzt. Da die Nachweise sehr unterschiedlich sind, wurde jeder Nachweistyp als eigene Klasse umgesetzt. Die allgemeine Grundstruktur wird dabei durch eine abstrakte Basisklasse realisiert. Ein Interface für die Eingabefenster stellt zudem das Vorhandensein notwendiger Methoden sicher. Welche Bemessungen als ein Nachweis ausgeführt werden, liegt beim Programmierer. In diesem Fall stellen die Anschluss- und Verstärkungsbemessung jeweils einen einzigen Nachweis dar. Detailliertere Nachweise, wie die Bemessung eines einzigen Verbindungsmittels sind ebenfalls möglich. In dieser Situation spiegelt sich der modulare und multifunktionale Charakter des Programmes wieder. Jeder Nutzer kann einen Nachweis mit beliebigem Umfang erstellen. Lediglich die Basisstruktur ist durch die gemeinsame Basisklasse der Nachweise vordefiniert.

Eine besondere Herausforderung lag darin einen Nachweis nicht nur für einen bestimmten Fall, sondern auch für verschiedene Bauteile umzusetzen. Schritte, die in der Handrechnung einfach umzusetzen sind, müssen hier für alle Varianten durchdacht werden. Um dies zu ermöglichen, wurde das Feld mit den am Nachweis beteiligten Bauteilen als Objekte der Basisklasse „Bauteil“ umgesetzt. Um den Zugriff zu vereinfachen, eine gewisse Übersichtlichkeit zu erhalten und vor allem auf typenspezifische Informationen zugreifen zu können, wurden jedoch Geometrie und Material je nach Typ weiter spezialisierte Objekten zugewiesen. Ziel war es, möglichst allgemein auf alle wichtigen Daten zugreifen zu können und unnötige Fallunterscheidungen zu vermeiden. Gleichzeitig war es teilweise nötig in einem Nachweis mit speziellen bauteiltypischen Daten zu arbeiten. Da die Spannungsermittlung und die Nachweislogik speziell für einen Fall gelten und direkt zu einem Nachweis gehören, wurden diese innerhalb der Nachweisfunktion fest definiert. Die mehrfach verwendeten Beiwertberechnungsformeln wurden dagegen zentral an einer einzigen Stelle gespeichert, um eine nicht redundante Speicherung zu gewährleisten. Neue Beiwertberechnungsformeln sind in der entsprechenden Klasse einzufügen. Um die Wiederverwendung der reinen Nachweispakete zu gewährleiste wurde sichergestellt, dass auch leere Nachweise gespeichert werden können. In diesem Fall werden außer dem Namen keine weiteren Daten festgelegt. Diese Pakete können erstellt und exportiert werden und durch den Import jederzeit als Vorlage wiederverwendet werden.

Aufgrund der einfachen Handhabung werden die Materialien als Tabellen im Excel-Format zur Verfügung gestellt. Für das NachweisTool stehen zwei vordefinierte Datenblätter für die Materialien der Holzbauteile und Verbindungsmittel zur Verfügung. Diese Materialtabellen können beliebig durch weitere Elemente ergänzt werden, indem weitere Zeilen hinzugefügt werden. Wichtig ist, dass die Tabelle nach der Bearbeitung im CSV-Format gespeichert wird. Die Daten im CSV-Format können einfach eingelesen und im Fall des NachweisTools in einen DataTable geladen werden. Dieser DataTable kann dann wie eine echte Tabelle mit speicherinternen Daten innerhalb des Programms gehandhabt werden. Spalten und Zeilen können selektiert und ausgelesen werden.

Die Ausgabe der Ergebnisse erfolgt in Form einer XML-Datei. Das XML-Format überzeugt neben der klaren und hierarchischen Struktur vor allem durch seine einfache Lesbarkeit. Das Dateiformat kann ebenfalls auf unkomplizierte Weise für den Import und Export verwendet werden. So wird sowohl die Ergebnisausgabe als auch die Wiederverwendung gespeicherter Elemente umgesetzt. Der Import und Export der erstellten Elemente wurde mit Hilfe des XmlSerializers realisiert. Dieser schreibt auf einfache Art und Weise die ihm übergebenen Elemente in eine XML-Datei und erstellt die Objekte beim

Import anhand der Informationen im importierten XML. Bei der Umsetzung war zu beachten, dass der XmlSerializer für alle Objekte einen als public definiert, parameterlosen Konstruktor benötigt und die zugehörigen Klassen ebenfalls öffentlich zugreifbar sein müssen. Zudem serialisiert der XmlSerializer nur als public definierte Felder und Eigenschaften und die Eigenschaften müssen über eine get- und eine set-Methode verfügen. Um mit dem XmlSerializer arbeiten zu können, sind im Code noch einige Eigenschaften hinzugefügt und Änderungen vorgenommen worden. Alle Variablen sind prinzipiell als private Felder definiert. Soll zudem die Exportmöglichkeit bestehen oder ein Zugriff von außen möglich sein, werden sie zusätzlich als public definierten Eigenschaften zur Verfügung gestellt. Neben der einfachen Handhabung der Serialisierung und Deserialisierung kann das XML auch einfach ergänzt oder benutzerdefiniert gesteuert werden. Die Bauteilliste enthält Objekte der Basisklasse „Bauteil“. Um die Objekte der abgeleiteten Klasse serialisieren und deserialisieren zu können, müssen deren Typen jedoch bekannt sein. Daher werden „XmlInclude“-Anweisungen der abgeleiteten Klassen vor den jeweiligen Basisklassen eingefügt. Dadurch wird bei der Erweiterung des Programms durch einen neuen Bauteil- oder Nachweistyp der Zugriff auf dessen Basisklasse und damit auf den bestehenden Code benötigt.

Wichtig ist neben der Funktionalität und Modularität die Offenheit und Erweiterbarkeit des Programms. Durch die vorhandene Struktur hat das Programm einen sehr modularen Charakter. Die Möglichkeit der Erweiterung und somit auch die Multifunktionalität wurden umgesetzt. Es wurde bei der Entwicklung und Umsetzung des Programmcodes eine beispielhafte Struktur geschaffen, welche die genannten Schwerpunkte erfüllt. Als Beispiel dienen dabei die erweiterbaren Materialtabellen und Verknüpfungsfunktionen, aber vor allem auch die Grundstruktur der Basisklassen und spezialisierten Nachweis- und Bauteilklassen. Beim Hinzufügen, eines neuen Nachweis- oder Bauteiltyps müssen jedoch Änderungen an mehreren Punkten im Programm vorgenommen werden. Um diesen Fall zu veranschaulichen soll im folgenden Abschnitt das Hinzufügen eines solchen Nachweistyps und eines Bauteiltyps zur bestehenden Programmarchitektur schrittweise beschrieben werden.

Hinzufügen eines neuen Nachweistyps

1. Kopieren einer bestehenden, möglichst ähnlichen Nachweisklasse
2. Umbenennen der Klassenkopie und anpassen der Klasse
    a. Anpassen der Felder und Eigenschaften
    b. Umbenennen des Nachweistyps und der zugehörigen Norm in den Konstruktoren
    c. Zusätzliche vom Nutzer einzugebende Variablen in den Konstruktor aufnehmen und zuweisen
    d. Umschreiben der Nachweisfunktion anhand der Norm
3. Kopieren einer bestehenden möglichst ähnlichen Nachweiseingabefensterklasse
4. Umbenennen der Klassenkopie und anpassen der Klasse.
    a. Anpassen der grafischen Oberfläche
    b. Anpassen der „Load“-Methode sowie der Methoden „BerechneButtonClick“, „NachweisÜberschreiben“, „NachweisÜberschreibenLeer“ und „LadeGUIKomplett“
5. In den „LadeGUI“-Methoden der Nachweisklasse, Eingabefenster in Objekte der neuen Klasse umändern
6. Einfügen des neuen Nachweistyps in die KeyValuePair-Liste „nachweistypen“ im Konstruktor der Klasse „Hauptprogramm“

7. Einfügen der „XmlInclude“-Anweisung oberhalb der Superklasse „Nachweis“

Hinzufügen eines neuen Bauteiltyps

1. Kopieren einer bestehenden, möglichst ähnlichen Bauteilklasse
2. Umbenennen der Klassenkopie und der Konstruktoren
3. Kopieren einer bestehenden möglichst ähnlichen Bauteileingabefensterklasse
4. Umbenennen der Klassenkopie und anpassen der Klasse
    a. Anpassen der grafischen Oberfläche
    b. Anpassen der Methoden „EingabeBestätigen_Button_Click“, „BauteilÜberschreiben“ und „LadeGUIKomplett“
5. In den „LadeGUI“-Methoden der Nachweisklasse, Eingabefenster in Objekte der neuen Klasse umändern
6. Einfügen des neuen Bauteiltyps in die KeyValuePair-Liste „bauteiltypen“ im Konstruktor der Klasse „Hauptprogramm“
7. Einfügen der „XmlInclude“-Anweisung oberhalb der Superklasse „HolzBauteil“ oder „VBMBauteil“

Trotz der Minimierung der Zugriffe auf den bestehenden Code sind mehrere Änderungen nötig. In jedem Fall müssen ebenfalls Änderungen in der Klasse „Hauptprogramm“ vorgenommen werden. Die Erweiterung um einen neuen Bauteil- oder Nachweistyp verlangt deshalb auch immer einen Zugriff auf den bestehenden Code. Dieser Zugriff setzt zum einen ein Verständnis des bestehenden Codes voraus und stellt zudem eine mögliche Fehlerquelle dar.

7.4.2 Vorteile

- Übersichtliche, zentrale Darstellung im Hauptfenster
- Benutzerfreundliche Anwendung
- Programmstruktur intuitiv und nahe an der händischen Berechnung
- Zentrale Speicherung der Normbestimmungen
- Modularer Programmcharakter
- Einfache Erweiterung der vorhandenen Materialien
- Einfacher Export und Import
- Übersichtliche, gut lesbare Ausgabedatei
- Ausgabedatei ermöglicht Nachvollziehung der Bemessung
- Einfache Variantenerstellung möglich

7.4.3 Nachteile

- Erweiterung aufwendig
- Erweiterung benötigt Zugriff auf bestehenden Code
- Keine dynamische Erweiterung möglich
- Verschachtelte Programmstruktur schwierig nachzuvollziehen
- Nachträgliche Namensänderungen nicht möglich

7.4.4 Ausblick

Während der Entwicklung wurde deutlich, wie verschieden selbst die Geometrieeigenschaften von Bauteilen der gleichen abgeleiteten Gruppe sein können. So sind die Geometriedaten und Schnittgrößen für einen Satteldachträger wesentlich umfangreicher, als für eine einfache Stütze und die Geometriedaten eines Stahlblechs unterscheiden sich trotz gleicher Basisklasse vollständig von denen eines Stahldübels. Die momentane Struktur speichert sämtliche Geometrieinformationen einer Bauteilunterklasse in einer Klasse. Nicht benötigte Werte werden nicht belegt. Da das System der Vererbung jedoch vorsieht, dass nur allgemeine Daten in einer Basisklasse gespeichert werden und für Spezialisierungen abgeleitete Klassen notwendig sind, wäre gut denkbar, dass bei einer Weiterentwicklung weitere Spezialisierungen in Richtung der Bauteiltypen stattfinden. Dies würde bedeuten, dass ein Satteldachträger beispielsweise nicht den Basiskonstruktor der „HolzBauteil“-Klasse verwendet, sondern einen eigenen Konstruktor besitzt in welchem ihm eine „Satteldachträgergeometrie“ und „Satteldachträgerschnittgrößen“ zugewiesen werden. Diese spezialisierten Klassen wären jedoch wieder von denen des Holzbauteils abgeleitet. Dies würde bedeuten, dass es zwei Vererbungsebenen in Geometrie, Schnittgrößen und eventuell auch Material gibt. Dabei würde versucht alle Informationen nach wie vor so allgemein wie möglich zu halten und diese in der ersten Vererbungsebene, den Klassen „VBMBauteil“ und „HolzBauteil“ oder sogar in den Basisklassen zu speichern. Nur Eigenschaften, die ausschließlich in den jeweiligen Bauteiltypen auftreten, würden in den zweifach abgeleiteten Klassen gespeichert werden. Dadurch müsste bei Nachweisen, die allgemein für Holzbauteile gleich zu führen sind, keine weitere Spezialisierung im Nachweis durchgeführt werden. In Nachweisen, welche beispielsweise nur für Satteldachträger zu führen sind, könnte dagegen eine spezielle Typkonvertierung des beteiligten Bauteils durchgeführt und somit zusätzlich auf dessen spezifische Eigenschaften zugegriffen werden.

Eine mögliche Weiterentwicklung wäre, neben dem Bereitstellen von Materialien, auch vordefinierte Standardquerschnitte zur Verfügung zu stellen. In diesem Fall hätte der Nutzer innerhalb des Programms die Möglichkeit, verwendete Querschnitte optional der Querschnitttabelle hinzuzufügen. Die Standardquerschnitte könnten entweder innerhalb des Programms oder manuell in der Excel-Tabelle laufend erweitert werden. Innerhalb der Anwendung hätte der Benutzer dann die Wahl entweder einen eigenen Querschnitt zu definieren, oder einen verfügbaren Standardquerschnitt zu verwenden.

Beim Export und Import in der bestehenden Programmierung wird die Eindeutigkeit der Namen der importierten Elemente nicht überprüft. Dies scheint kein großes Problem zu sein, da der Import vorhandener Daten in der Regel zu Beginn einer Programmanwendung stattfindet. Da in diesem Fall noch keine Elemente in der aktuellen Anwendung vorliegen, sind auch keine Namen zu überprüfen. In einer Weiterentwicklung könnte diese Tatsache durch die Implementierung einer Eindeutigkeitsprüfung und gegebenenfalls Namensänderung optimiert werden. Wichtig ist dabei, bei einer Namensänderung von zu importierenden Bauteilen auch deren Verweise innerhalb der Nachweise abzuändern. Zusätzlich könnte auch die Idee umgesetzt werden, nur eine bestimmte Auswahl innerhalb des Programms exportieren zu können.

Eine weitere Idee zur Weiterentwicklung wäre es einen Schnittgrößenverlauf zu speichern. Der Nachweis müsste dann je nach Nachweistyp einen Maximalwert oder den Wert an einer bestimmten

Stelle suchen. Nach dem bisherigen Stand werden Maximalwerte, Werte im Anschlussbereich, im First oder am Ort der maximalen Spannungen verwendet. Da die Bauteile für verschiedene Nachweise eingesetzt werden, müssen im Eingabefenster und der Schnittgrößenklasse alle möglichen Nachweissituationen berücksichtigt werden. Durch die Speicherung eines kompletten Schnittgrößenverlaufs würden automatisch alle verfügbaren Schnittgrößen zur Verfügung stehen. In diesem Fall müsste jedoch zusätzlich noch die Speicherung von erhöhten Schnittgrößen berücksichtigt werden.

Alle Formeln der Nachweisberechnung werden im Namespace „VerknüpfungsFunktionen“ zusammengefasst und sind zudem thematisch in Klassen unterteilt. Eine mögliche Erweiterung oder Unterteilung der beiden bisherigen Klassen „AnschlussnachweisVerknüpfungsFunktionen“ und „BeiwertVerknüpfungsFunktionen“ nach Norm oder Zulassung ist in einer Weiterentwicklung des Programms ebenfalls denkbar.

Eine weitere Idee ist, vordefinierte Nachweispakete für bestimmte Bauteile und Belastungsarten zu erstellen. Ein Schritt in diese Richtung wurde bereits getan, indem die Speicherung leerer Nachweispakete umgesetzt wurde. Die Vorsortierung durch eine thematische Gliederung würde zur Übersichtlichkeit beitragen. Zudem wäre es denkbar die Nachweise weiter zu detaillieren. Aktuell wird der Stabdübelnachweis als ein kompletter Nachweis behandelt. Es wäre aber auch durchaus denkbar diesen Nachweis weiter in Einzelnachweise aufzuteilen.

Bisher lässt sich die entwickelte Programmstruktur nur durch Eingaben an mehreren Stellen im Programmcode erweitern. Um neue Typen hinzuzufügen ist es deshalb notwendig auf den aktuellen Code zuzugreifen und diesen abzuändern. Dieser Punkt macht einen wichtigen Teil der Programmidee aus und ist Grundlage für die Weiterentwicklung des bestehenden Codes.

Um eine dynamische Erweiterung zuzulassen, wird das Programm im nächsten Kapitel umstrukturiert. Neue Nachweis- und Bauteiltypklassen sollen dabei auch außerhalb des bisherigen Programmcodes erstellt werden können.

8 Dynamische Gestaltung des Programmcodes

In einem zweiten Programm wurde versucht, das bereits entwickelte Programm so umzustrukturieren, dass es eine dynamische Erweiterung durch weitere spezialisierte Klassen zulässt. Ziel war es, diese Erweiterung möglichst einfach und unabhängig vom bestehenden Programmcode zu ermöglichen. Für die Umsetzung wurde das Managed Extensibility Framework (MEF) angewendet, das im Folgenden erklärt wird.

8.1 MEF

MEF ist eine Bibliothek des .Net-Framework, die das Erstellen von erweiterbaren Anwendungen zulässt. Durch die Verwendung des Frameworks lassen sich weitere, externe Komponenten, in diesem Fall Nachweis- und Bauteiltypen zum Programm hinzufügen ohne den Quellcode des Programms zu verändern. Harte Abhängigkeiten werden dadurch vermieden und der Code kann sinnvoll gekapselt werden. Das Prinzip des MEF stützt sich dabei auf das Importieren und Exportieren von Schnittstellen.

Das MEF besteht immer aus den folgenden drei Teilen:

1. Parts
2. Catalogs
3. Container

Die Funktionalitäten, die von den Anwendungskomponenten zur Verfügung gestellt werden, werden als Parts deklariert. Diese Parts werden dann in Catalogs zur Verfügung gestellt. Einem Container stehen dann wiederum die Catalogs zur Verfügung. Der Container sucht dann in den vorhandenen Catalogs und Parts nach der passenden Schnittstelle. Im Fall des NachweisTools ist das MEF-Prinzip nach der dargestellten Struktur umgesetzt.

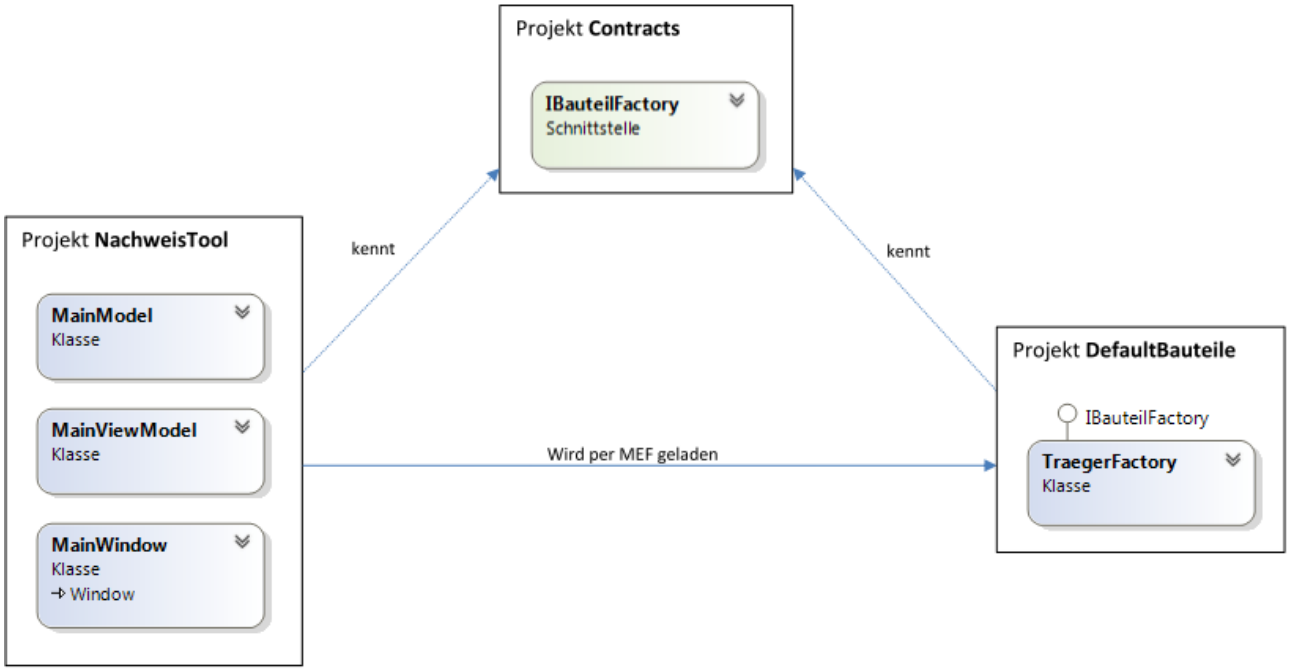


Abbildung 82: MEF-Prinzip am Beispiel der Bauteile

Das Projekt „NachweisTool", in welchem sich die Umsetzung des ehemaligen Hauptprogramms in Form der Klassen „MainWindow", „MainModel" und deren Verknüpfung die Klasse „MainViewModel" befinden, kennt dass Projekt „Contracts" (siehe Abbildung 82). In diesem Projekt befinden sich die Interfaces „IBauteilFactory" und „INachweisFactory". Eine Klasse, im obigen Beispiel „TraegerFactory" der Klassebibliothek „DefaultBauteile", implementiert dann eines dieser Interfaces, hier „IBauteilFactory". Per MEF wird dann die Klassenbibliothek dynamisch zur Laufzeit geladen. Da zwischen „NachweisTool" und „DefaultBauteile" sonst keine Verbindung besteht, kann das NachweisTool nur auf Eigenschaften zugreifen, die durch das Interface vorgegeben wurden. Der Speicherort der Klassenbibliothek ist somit beliebig, solange die Ausgabeadresse für das Laden per MEF bekannt ist. Die Bauteil- und NachweisFactories müssen lediglich genauso wie das Projekt „NachweisTool" über einen Verweis auf das Projekt „Contracts" verfügen. Dieses Prinzip wurde im zweiten Programm für die verschiedenen Nachweis- und Bauteiltypen angewendet.

Um das MEF im NachweisTool verwenden zu können wurde ein grundlegender Neuaufbau des Programms vorgenommen. Der Inhalt wurde entsprechend dem MEF-Prinzip in verschiedene Projekte gegliedert. Alle Projekte wurden dabei in der Projektmappe „NachweisTool" gespeichert (siehe Abbildung 83). Dieser gemeinsame Speicherort ist jedoch nicht notwendig, ein Hinzufügen der entsprechenden Verweise ist ausreichend.

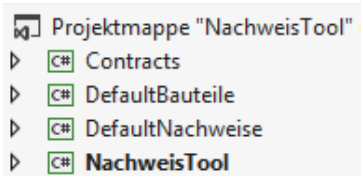


Abbildung 83: Projektmappe

In dem Projekt „Contracts" sind sämtliche von den Erweiterungen verwendete Vorgaben gespeichert. Die Projekte „DefaultBauteile" und „DefaultNachweise" werden zur Speicherung der spezialisierten Typklassen verwendet. In dem Projekt „NachweisTool" sind das Hauptfenster sowie das Hauptprogramm und deren Verknüpfung gespeichert. Hierbei wurde das MVVM-Pattern umgesetzt, bei welchem es darum geht Logik und Design klar voneinander zu trennen. Im zweiten Programm wurde zudem anstelle einer Windows Forms-Anwendung eine Windows Presentation Foundation (WPF)-Anwendung verwendet. Dieser direkte Nachfolger von Windows Forms, WPF ermöglicht die klare Trennung von Logik und Design.

8.2 Model View ViewModel (MVVM)

Da der ursprüngliche Code sehr interaktiv ist, besitzen verschiedene Klassen immer wieder Zugriff auf verschiedene Elemente anderer Klassen. Die Grundstruktur dieser Interaktionen wurde im Laufe der Programmierung für einen späteren Anwender, der den Code verstehen oder auch abändern möchte, sehr komplex. Aus diesem Grund wurde in der Weiterentwicklung das MVVM-Pattern verwendet. Diese drei Komponenten stecken hinter dem MVVM-Konzept.

1. Model
   In diesem Teil des Programms werden die zu bearbeitenden Daten repräsentiert. Das Model hat jedoch keinerlei Zugriff auf die Darstellung in der GUI.

2. View
   In der View-Komponente wird die Ansicht in der GUI bearbeitet. Dieser Teil enthält wiederum keinerlei Logik zur Weiterbearbeitung der Daten und beschäftigt sich lediglich mit der Darstellung.
3. ViewModel
   Das ViewModel ist das Zwischenstück zwischen der View und dem Model. Es nimmt Ereignisse und Eingaben aus dem View entgegen und kann diesem Daten zur Darstellung zur Verfügung stellen. Die eingegebenen Daten werden vom ViewModel an das Model zur Verarbeitung weitergeleitet.

Durch die Umsetzung des MVVM-Patterns im NachweisTool entsteht eine Trennung von grafischen und logischen Elementen und damit eine klare Struktur des Projekts. Sämtliche Bearbeitung der Nachweise, Nachweispakete und Bauteile und deren Listen finden nun in der Klasse „MainModel" statt. Auch die Methoden der Klasse „MainModel" werden nicht mehr außerhalb des Projekts „NachweisTool" aufgerufen. Die Struktur der verschiedenen Projekte ist somit so unabhängig wie möglich gestaltet. Die folgende Abbildung stellt dar wie das MVVM-Pattern im Projekt „NachweisTool" umgesetzt wurde.

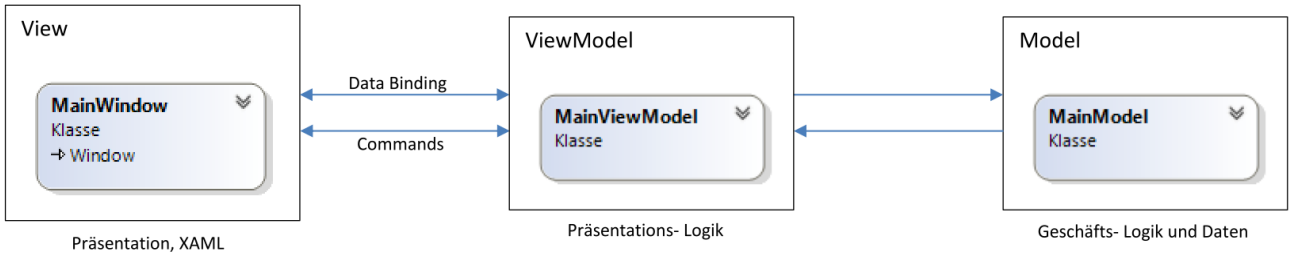


Abbildung 84: Umgesetztes MVVM-Pattern

In der Klasse „MainWindow" wird die Darstellung im Hauptfenster beschrieben. Das „MainModel" übernimmt die meisten Aufgaben der ursprünglichen Klasse „Hauptprogramm" und bearbeitet damit die eigentlichen Daten. Die Klasse „MainViewModel" ist das Verbindungsstück zwischen den beiden Klassen. Über DataBinding und Commands werden Informationen und Logik der GUI zwischen den Klassen „MainWindow" und „MainViewModel" gesteuert.

Nachdem nun auf die wesentlichen beiden umgesetzten Prinzipien eingegangen wurde, wird im Folgenden die Umsetzung der einzelnen Projekte und deren Klassen erläutert. Dabei wird vor allem auf die Unterschiede zur ersten Programmvariante eingegangen. Da sich für den Ablauf aus der Sicht des Anwenders nur geringe Änderungen ergeben haben, soll darauf nicht gesondert eingegangen werden. Diese Anwendungsunterschiede werden im Laufe der Programmbeschreibung und im anschließenden Vergleich der beiden Programme beschrieben.

8.3 NachweisTool

Das Projekt „NachweisTool" ist das Startprojekt der Projektmappe und stellt das eigentliche Datenverarbeitungsprogramm dar. Um auf die Inhalte des Projekts „Contracts" zugreifen zu können, besitzt das NachweisTool einen Verweis auf „Contracts".

MainWindow.xaml und MainWindow.xaml.cs

In diesen beiden Klassen befindet sich das Layout und Data Binding des ständig geöffneten Hauptfensters. In der XAML-Datei werden das Layout und die Datenverknüpfung festgelegt. Wie bei Windows Forms können auch in WPF Elemente mithilfe der Entwurfsdarstellung eingefügt und angepasst werden. Im XAML können dann deren Eigenschaften weiter variiert werden und durch ein Binding direkt an Daten gebunden werden. Diese Daten werden durch den DataContext des Fensters, ein Objekt der Klasse „MainViewModel“ zur Verfügung gestellt. Während in der ersten Version im Code zur GUI direkt auf die Daten des Programms zugegriffen wurde, arbeitet der Code Behind des „MainWindows“ mit den Informationen und Methoden des zugehörigen „MainViewModels“.

MainViewModel.cs

Die Klasse „MainViewModel“ dient zur Kommunikation zwischen der GUI und der eigentlichen Datenverarbeitung und den Daten. Hierzu besitzt sie ein Objekt vom Typ „MainModel“. In der GUI kann lediglich auf die Elemente der Klasse „MainViewModel“ zugegriffen werden. Die meisten Methoden sind sehr ähnlich, wie im „MainModel“ benannt. Sie rufen die Methoden des Members „mainModel“ auf und reichen Informationen an diese weiter. Um die verschiedenen vorhandenen Bauteil- und Nachweistypen in der GUI darstellen zu können, werden die, diese repräsentierenden Factories, aus dem „mainModel“ ausgelesen und gespeichert. Im Konstruktor der Klasse werden die Materialien aus den CSV-Dateien in den internen Speicher geladen. Im ersten Programm wurden die Materialien beim Erstellen der Bauteileingabefenster immer neu geladen. Durch die nun umgesetzte Variante wird sichergestellt, dass die Materialien nur einmal, beim Erstellen des „MainViewModels“, geladen werden. In der Klasse „MainViewModel“ werden zudem Methoden zur Verfügung gestellt, welche später Events der Eingabefenster zugeordnet werden.

```
void WindowBauteilErstellung(object sender, BuildingElementCreationEventArgs e)
{

    try
    {
        mainModel.BauteilHinzufügen(e.Bauteil);
    }

    catch (NameNichtEindeutig ex)
    {
        var warnung = new WarnungNameNichtEindeutig();
        warnung.TextBlock.Text = ex.Message;
        warnung.Show();
        e.Erstellt = false;
        return;
    }

    var window = (IBauteilWindow)sender;
    e.Erstellt = true;
    window.BuildingElementCreation -= WindowBauteilErstellung;
    bauteilViewItems.Clear();

    foreach (var buildingElement in mainModel.Bauteile())
        bauteilViewItems.Add(buildingElement);
}
```

Abbildung 85: Methode „WindowBauteilErstellung“

Ein Beispiel hierfür ist die Methode „WindowBauteilErstellung" (siehe Abbildung 85). Sie wird ausgeführt, wenn ein Bauteil erstellt werden soll. Innerhalb dieser Methode wird anhand eines Try-Catch-Blocks versucht das übergebene Bauteil dem „mainModel" anhand dessen Membermethode „BauteilHinzufügen" hinzuzufügen.

```
public void BauteilHinzufügen(IBauteil bauteil)
{
    foreach (IBauteil element in Bauteile())
    {
        if (element.Name == bauteil.Name)
            throw new NameNichtEindeutig();
    }
```

Abbildung 86: „BauteilHinzufügen"-Methode der Klasse „MainModel"

Wird innerhalb der Methode eine Exception ausgelöst, weil der Name nicht eindeutig ist, wird der zugehörige Catch-Block ausgeführt (siehe Abbildung 86). In diesem Fall wird eine Warnung mit dem Text der entsprechenden Exception ausgegeben. Konnte das Bauteil hinzugefügt werden, wird der Code nach dem Catch-Block ausgeführt und die Bauteilliste in der Ansicht wird aktualisiert.

MainModel.cs

In der Klasse „MainModel" finden die eigentliche Verarbeitung der Informationen und die Bearbeitung der Bauteil-, Nachweis- und Nachweispaketlisten statt. Diese Klasse stellt damit die Umsetzung der Klasse „Hauptprogramm" des ersten Programms dar. Im Konstruktor des MainModels wird anders als im ersten Programm lediglich die Methode „SnoopForFactories" aufgerufen.

```
//Snoop
private void SnoopForFactories()
{
    var aggreagateCatalog = new AggregateCatalog();
    aggreagateCatalog.Catalogs.Add(new DirectoryCatalog("."));
    var compositionContainer = new CompositionContainer(aggreagateCatalog);
    compositionContainer.ComposeParts(this);
}
```

Abbildung 87: „SnoopForFactories"-Methode

In dieser Methode werden nach dem MEF-Prinzip alle im Debug-Ordner vorhandenen Klassenbibliotheken als Catalog dem neuen Container zur Verfügung gestellt. Da das Programm wissen muss welche Interfaces importiert werden sollen, müssen zudem sogenannte Import-Attribute im Code vorhanden sein. Da in diesem Fall verschiedene Exporte zur Erweiterung verfügbar sind und all diese verwendet werden sollen, wird hier das „ImportMany"-Attribut verwendet (siehe Abbildung 88).

```
[ImportMany]
public IEnumerable<Lazy<IBauteilFactory>> BauteilFactories { get; set; }


[ImportMany]
public IEnumerable<Lazy<INachweisFactory>> NachweisFactories { get; set; }
```

Abbildung 88: „ImportMany"-Attribut

Der „ImportMany“-Anweisung folgt die Angabe des zu importierenden Typs in Form des MEF-Typs „Lazy<T, TMetadata>“. In diesem Fall sollen Objekte des Typs „IBauteilFactory“ und „INachweisFactory“ importiert werden und anschließend in den IEnumerable-Auflistungen zur Verfügung stehen. Die Klasse besitzt zudem ähnlich wie beim ersten Programm eine Liste mit Bauteilen und eine Liste mit Nachweispaketen. Hierbei handelt es sich jedoch um IList-Interfaces und auch in der Bauteilliste selbst sind Objekte vom Interfacetyp „IBauteil“ gespeichert. Zudem sind diese Auflistungen als private und readonly definiert, wodurch sichergestellt wird, dass keine unrechtmäßigen Änderungen daran vorgenommen werden können. Durch spezielle Methoden der Klassen können die Listen als IEnumerable-Auflistungen an die aufrufenden Funktionen zurückgegeben werden.

Die weiteren Methoden der Klasse haben sich aufgrund der neuen Programmstruktur ebenfalls verändert. Im ersten Programm sind zur Bearbeitung der Bauteile insgesamt fünf Methoden vorhanden. Diese Methoden dienen dem Finden, Hinzufügen, Entfernen und Ersetzen eines Bauteils und zur Überprüfung ob dessen Name eindeutig ist. Neben der „Bauteile“-Methode, welche den Inhalt der aktuellen Bauteilliste zurückgibt, bestehen drei weitere Methoden. Die beiden Methoden zum Hinzufügen und Entfernen eines Bauteils, welche dem alten Programm sehr ähnlich sind, sowie eine Methode, die alle Nachweise an welchen das Bauteil beteiligt ist neu berechnet. Der Grund für die Umstrukturierung liegt am DataBinding des MVVM-Patterns. Wird ein Fenster mit Bauteilinformationen geöffnet, besteht ein direkter DataContext zu diesem Bauteilobjekt. Durch das Binding werden Änderungen in der Eingabemaske direkt im Bauteil übernommen. Somit wird die Option des ersten Programms, Änderungen in der Bauteilmaske vorzunehmen und dann zu entscheiden ob das bestehende Bauteil überschrieben werden sollen, überflüssig. Da auf das Bauteil nur in der Klasse „MainModel“ und direkt durch das Binding zugegriffen wird, wird auch die Methode „BauteilFinden“ nicht länger benötigt. Die Überprüfung ob der Bauteilname eindeutig ist wird im zweiten Programm in der bereits vorgestellten „BauteilHinzufügen“-Methode durch eine if-Abfrage implementiert. Ist der Name bereits vorhanden, wird eine entsprechende Exception geworfen, die eine Warnung an den Nutzer ausgibt. Diese Art der Umstrukturierung aufgrund des MVVM-Patterns wurde auch in den anderen Methoden umgesetzt. Abbildung 89 zeigt die Struktur der Klasse „MainModel“.

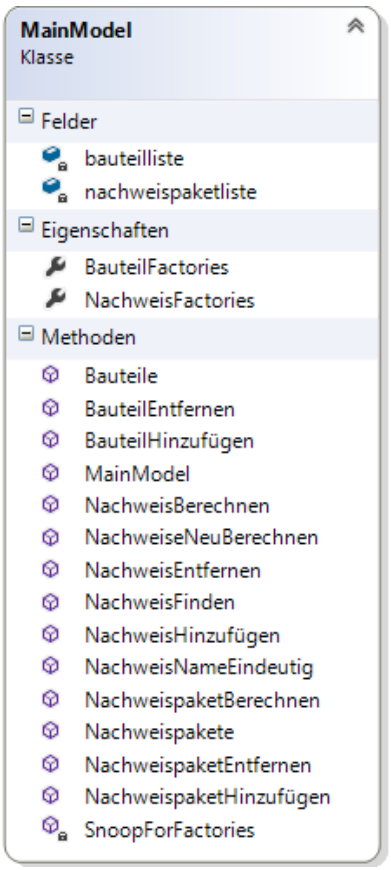


Abbildung 89: Klasse „MainModel“

Auch in dieser Klasse wurden die Methoden in Regionen gegliedert. Neben den Regionen für Bauteile, Nachweise und Nachweispakete, besteht ebenfalls eine Region für den XML-Export und Import. Der Code in diesem Teil ist auskommentiert, da der Export und Import in der aktuellen Programmversion noch nicht zur Verfügung steht. Anders als im ersten Programm wurden die Querschnitts- und Materialmethoden in der eigenen Klasse „StorageService“ gespeichert. Diese Klasse ist in dem Project „Contracts“ gespeichert.

8.4 Contracts

Im Projekt „Contracts“ sind alle allgemeinen Vorgaben gespeichert. Dazu zählen neben den Bauteil-, Geometrie- und Materialklassen auch der „StorageService“ und die für die Nachweisrechnung verwendeten „Verknüpfungsfunktionen“. Durch einen Verweis auf dieses Projekt stehen die darin enthaltenen Funktionen auch jedem anderen Projekt zur Verfügung.

8.4.1 Bauteile

Die prinzipielle Bauteilstruktur ist dem ersten Programm sehr ähnlich. Auch in diesem Programm gibt eine abstrakte Basisklasse „Bauteil“ die prinzipielle Struktur aller Bauteile vor. Im Unterschied zum ersten Programm sind die beiden abgeleiteten Klassen „HolzBauteil“ und „VBMBauteil“ jedoch nicht mehr abstrakt. Der Grund liegt darin, dass in der Nachweisfunktion ein Kopierkonstruktor dieser Klassen aufgerufen werden muss und keine Instanzen einer abstrakten Klasse erstellt werden können. Der genaue Hintergrund hierzu wird im Kapitel „HolzBauteil.cs und VBMBauteil.cs“ erläutert.

Bauteil.cs

Die Bauteilklasse besitzt kaum Felder sondern hauptsächlich Eigenschaften (siehe Abbildung 90). Der Grund liegt darin, dass das DataBinding nur mit Properties funktioniert. Als Felder sind lediglich die Normverweise vorhanden, die später im XML exportiert und importiert werden sollen. Dieses Konzept wurde ebenfalls auf die anderen Klassen des Projekts angewandt.

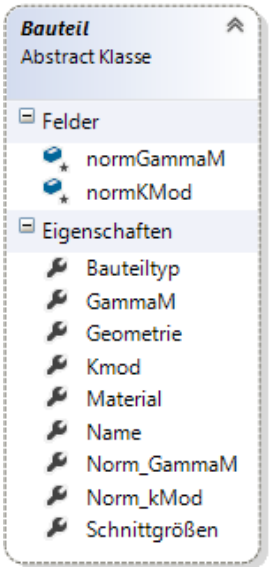


Abbildung 90: Klasse „Bauteil“

Im Unterschied zum ersten Programm ist weiterhin zu erkennen, dass diese Klasse nicht über Methoden verfügt. Die Umsetzung der Methoden zum Laden der Eingabefenster der Bauteile wurde mittels der Schnittstelle „IBauteilWindow“ umgesetzt.

HolzBauteil.cs und VBMBauteil.cs

Die beiden von der Klasse „Bauteil“ abgeleiteten Basisklassen unterscheiden sich kaum von denen im ersten Programm. Sie sind nicht mehr abstrakt und besitzen ausschließlich Eigenschaften. Die Informationen wie verfügbare Nutzungsklassen, Klassen der Lasteinwirkungsdauer und die Materialien sind nicht mehr im Code zum Eingabefenster festgelegt, sondern als öffentliche, statische Eigenschaften in den Klassen deklariert. Das jeweilige Eingabefenster greift dann über das Binding auf die Eigenschaften zu. Die Initialisierung der Eigenschaften erfolgt im Konstruktor der jeweiligen Klasse. Im Fall des Materials wird hierzu die entsprechende Methode der statischen Klasse „StorageService“ aufgerufen (siehe Abbildung 91).

```
public static IList<string> materialien { get; set; }

//Standardkonstruktor
public VBMBauteil()
{ }

public VBMBauteil(string bauteiltyp)
{
    Bauteiltyp = bauteiltyp;
    Geometrie = new VBMGeometrie();
    Material = new VBMMaterial();
    materialien = StorageService.GetVBMMaterial(bauteiltyp);
}
```

Abbildung 91: Codeausschnitt der Klasse „VBMBauteil"

Im Fall der Verbindungsmittel wird der Bauteiltyp der Methode übergeben. In dieser werden dann die Namen der für diesen Bauteiltyp verfügbaren Materialien einer Liste zugeordnet. Die Liste wird zurückgegeben und steht über DataBinding in der Eingabemaske als Materialien zur Verfügung. Den Konstruktoren der abgeleiteten Bauteilklassen wird nur der Bauteiltyp übergeben. Durch das direkte DataBinding wird schon beim Öffnen des Fensters ein neues Bauteil angelegt. Dieses besitzt jedoch neben dem Typ keine weiteren Informationen. So werden im Konstruktor leere Objekte der jeweiligen Member erstellt. Da ohne diese Informationen eine Bestimmung von k_{mod} und γ_M nicht möglich ist, wird die Berechnung der Kennwerte auch nicht mehr innerhalb des Konstruktor aufgerufen. Durch das DataBinding werden die Änderungen immer direkt am Bauteil und dessen Komponenten vorgenommen. Beim Klicken des „Speichern"-Buttons wird das verwendete Bauteil dann lediglich zur Bauteilliste hinzugefügt. Durch diese Umstrukturierung können nun auch unvollständige Bauteile gespeichert werden. Wenn fälschlicherweise unvollständige Bauteile in Nachweisfunktionen verwendet werden, wird die Berechnung abgebrochen. Zusätzlich enthalten die beiden Klassen anders als im vorherigen Programm auch einen Kopierkonstruktor (siehe Abbildung 92).

```
//Kopierkonstruktor
public VBMBauteil(VBMBauteil bauteil)
{
    Name = bauteil.Name;
    Bauteiltyp = bauteil.Bauteiltyp;
    Material = new VBMMaterial((VBMMaterial)bauteil.Material);
    Geometrie = new VBMGeometrie((VBMGeometrie)bauteil.Geometrie);

    //Gamma_M aus dem Baustofftyp mit Hilfe der Funktionen der Klasse BeiwertVerknüfungsFunktionen berechnen
    GammaM = BeiwertVerknüfungsFunktionen.GammaMBestimmenNA(Material.Baustofftyp, ref normGammaM);
}
```

Abbildung 92: Kopierkonstruktor der Klasse „VBMBauteil"

Dieser Kopierkonstruktor wird zu Beginn der Nachweisrechnung aufgerufen. Hierin werden die Daten eines übergebenen Bauteils übernommen und die jeweiligen Beiwerte berechnet. Für die Zuweisung von Material, Geometrie und Schnittgrößen wird ebenfalls deren Kopierkonstruktor aufgerufen. Da die Beiwerte erst zur Nachweisführung benötigt werden, ist der Aufrufzeitpunkt ideal. Auch die Schnittgrößen werden vor der Berechnung einmalig in andere Einheiten umgerechnet. Da sich die Darstellung in der Eingabemaske nicht ändern soll, wird eine Kopie zur Verwendung des Bauteils innerhalb des Nachweises erstellt. Dies verhindert unter anderem, dass sich die Einheit der Schnittgrößen im Eingabefenster durch die Umrechnung ändert. Im ersten Programm fand die

Erstellung des Bauteils erst nach der vollständigen Eingabe der Informationen statt. Nachdem überprüft wurde, dass alle Eingaben vollständig sind, wurde das Bauteil erstellt und alle Werte berechnet. Die Rückrechnung der Schnittgrößen fand direkt im zur Eingabemaske gehörenden Code statt. Die neue Umsetzung mithilfe des WPF und DataBinding ermöglicht eine einfachere Codegestaltung und flexiblere Eingabe, muss dadurch jedoch auch in diesen Schritten berücksichtigt werden. Wichtig ist, dass falls Änderungen am Bauteil innerhalb des Nachweises vorgenommen werden sollen, diese immer am Bauteil direkt vorgenommen werden, um außerhalb des Nachweises übernommen zu werden. Um mögliche Fehler während der Berechnung durch unvollständige Eingangsdaten zu vermeiden, wurden entsprechende Exceptions in die Nachweisfunktion eingebaut.

IBauteilWindow.cs

Die Schnittstelle „IBauteilWindow" wird von allen Bauteileingabefenstern implementiert und gibt deren Grundstruktur vor. Wie in Abbildung 93 zu erkennen ist, besitzt jedes Eingabefenster eine Methode „Show" zum Anzeigen des Fensters, eine Eigenschaft vom Typ „object" als DataContext und ein Event, das ausgelöst wird, wenn im Bauteileingabefenster der Button „Speichern und Schließen" geklickt wird.

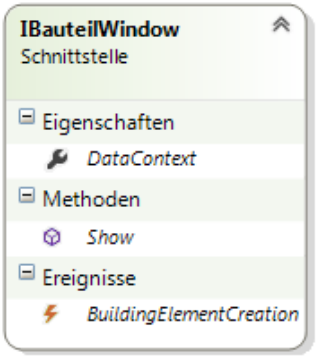


Abbildung 93: Struktur der „IBauteilWindow"-Schnittstelle

Passend hierzu wurde eine eigene von EventArgs abgeleitete Klasse „BuildingElementCreationEventArgs" erstellt, welche die Ereignisdaten enthält.

```
public class BuildingElementCreationEventArgs : System.EventArgs
{
    public readonly Bauteil Bauteil;
    public bool Erstellt = true;

    public BuildingElementCreationEventArgs(Bauteil bauteil)
    {
        Bauteil = bauteil;
    }
}
```

Abbildung 94: Klasse zum Kapseln der Eventdaten

Diese Klasse besitzt ein Objekt vom Typ „Bauteil", welchem im Konstruktor ein übergebendes Bauteil zugewiesen wird, sowie eine bool-Variable, welche angibt ob ein Bauteil erfolgreich zur Bauteilliste hinzugefügt werden konnte oder nicht. In der Praxis heißt das, dass beim Klicken auf den „Speichern

und Schließen"-Button im Bauteileingabefenster ein neues Objekt vom Typ „BuildingElementCreationEventArgs" erstellt wird.

```
private void EingabenBestätigenButtonClick(object sender, RoutedEventArgs e)
{
    BuildingElementCreationEventArgs neu = new BuildingElementCreationEventArgs((Bauteil)DataContext);
    OnBuildingElementCreation(neu);
    if (neu.Erstellt == true)
    {
        Close();
    }
}
```

Abbildung 95: EventHandler des „Speichern und Schließen"-Buttons

Dabei wird diesem der aktuelle DataContext des Fensters, also das Bauteil übergeben. Anschließend wird einer Methode dieses Objekt übergeben. In dieser Methode wird überprüft ob dem zugehörigen EventHandler-Delegat des Events Methoden zugewiesen wurden. Ist dies der Fall, werden diese Methoden ausgeführt. Der Methode werden das aktuelle Bauteil, sowie das „BuildingElementCreationEventArgs"-Objekt übergeben. Die Zuweisung der Methode selbst, findet in der Klasse „MainViewModel", beim Erstellen des Fensters statt. In der hier zugewiesenen Methode „WindowBauteilErstellung" wird versucht das Bauteil der aktuellen Bauteilliste hinzuzufügen. Ist dies nicht möglich wird eine Exception aufgerufen, eine Warnung ausgegeben und die Variable „Erstellt" auf false gesetzt. Der Wert der Variable Erstellt wird im Code des Eingabefensters abgefragt. Ist der Wert true, wird das Fenster geschlossen, ansonsten bleibt es zur Änderung der Eingabedaten geöffnet.

IBauteilFactory.cs

Die Schnittstelle „IBauteilFactory" wird von allen Bauteiltyp-Factories implementiert und gibt damit deren Struktur vor. Dieses Interface wird zudem zum MEF-Export und -Import verwendet. Alle Eigenschaften und Methoden, welche der Klasse „MainModel" bekannt sein müssen, müssen in diesem Interface deklariert sein.

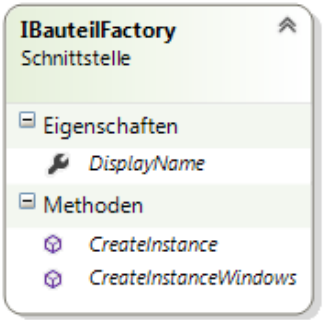


Abbildung 96: Struktur der „IBauteilFactory"-Schnittstelle

Alle Factories besitzen eine Eigenschaft „DisplayName", welche den angezeigten Namen des Bauteiltyps repräsentiert (siehe Abbildung 96). Zudem gibt die Schnittstelle Methoden zum Erstellen einer neuen Bauteilinstanz und einer neuen Instanz des Bauteileingabefensters vor.

Erweiterung – DefaultBauteile

Um einen neuen Bauteiltyp zu erstellen, braucht es wie im ersten Programm ein Eingabefenster für das Bauteil sowie eine eigene Bauteilklasse und um das Bauteil zu exportieren und in das „MainModel" importieren zu können eine zugehörige Factory. Die Komponenten sollen hier anhand des Hinzufügens des Bauteiltyps Träger beispielhaft erläutert werden.

```
public class Traeger : HolzBauteil
{
    //Standardkonstruktor
    public Traeger():base("Traeger")
    {
    }
    //Kopierkonstruktor
    public Traeger(Traeger bauteil)
        : base(bauteil)
    {
    }
}
```

Abbildung 97: Klasse „Traeger"

Die Bauteiltypklasse selbst ist einfacher gefasst, als die des ersten Programms, was vor allem an der neuen Struktur und der Verwendung von WPF liegt. Die Klasse enthält einen Standard- sowie einen Kopierkonstruktor. Beide rufen mit dem Schlüsselwort base den entsprechenden Konstruktor der Basisklasse auf. Die Klasse „TraegerWindow" implementiert die Schnittstelle „IBauteilWindow" und verfügt damit über die benötigten Eigenschaften. Die Factory implementiert das vorgestellte Interface „IBauteilFactory". Vor der Deklaration der Factory-Klasse muss die „Export"-Anweisung mit dem zu exportierenden Typ stehen. Wie in der unten stehenden Abbildung zu sehen ist, werden alle Factories als Schnittstellen vom Typ „IBauteilFactory" importiert. In der Factory selbst werden die von „IBauteilFactory" vorgegebene Eigenschaft und die Methoden implementiert.

```
[Export(typeof(IBauteilFactory))]
public class TraegerFactory : IBauteilFactory
{
    public string DisplayName
    {
        get { return "Traeger"; }
    }

    public Bauteil CreateInstance()
    {
        return new Traeger();
    }

    public IBauteilWindow CreateInstanceWindows()
    {
        return new TraegerWindow();
    }
}
```

Abbildung 98: Klasse „TraegerFactory"

Der DisplayName wird dabei mit „Traeger" belegt und die beiden Methoden geben eine neue Instanz des Trägers und des zugehörigen Eingabefensters zurück. Durch die Verwendung der Factories wird ermöglicht, dass das „MainModel" ohne den Träger selbst zu kennen einen solchen erstellen und das

zugehörige Fenster öffnen kann. Diese Beispielhafte Umsetzung wurde im Programmcode auch für ein Verbindungsmittel, den Stabdübel durchgeführt. Um das Einfügen neuer Typen zu erleichtern, wurden sogenannte Templates erstellt, welche als Vorlage dienen. Diese existieren für die verschiedenen Elemente und wurden anhand der bestehenden Bauteile entworfen.

8.4.2 Nachweise

Die prinzipielle Struktur der Nachweise wurde wie bei den Bauteilen ebenfalls durch Interfaces für die Factory und das Eingabefenster erweitert. Ansonsten ist die Struktur dem Aufbau des ersten Programms sehr ähnlich. Die Nachweise werden weiterhin in Nachweispaketen gespeichert.

Nachweis.cs

Die abstrakte Klasse „Nachweis“, welche als Vorlage für alle Nachweise dient, hat sich durch die Umstrukturierung geringfügig verändert. Die Klasse implementiert jetzt das „INotifyPropertyChanged“-Interface. Diese Schnittstelle benachrichtigt Clients, in diesem Fall das Eingabefenster, dass ein Eigenschaftswert geändert wurde.

```
public double Ergebnis
{
    get { return ergebnis; }

    set
    {
        ergebnis = value;
        OnPropertyChanged("Ergebnis");
    }
}
```

Abbildung 99: Aufruf der „OnPropertyChanged“-Methode

Durch die „OnPropertyChanged“-Methode, welche in der Set-Methode einer Eigenschaft aufgerufen wird, wird das Event „PropertyChanged“ aufgerufen, welches im XAML des Eingabefensters verknüpft ist (siehe Abbildung 99). Dadurch wird sichergestellt, dass einer Änderung der Eigenschaft direkt in die Eingabemaske übernommen wird. Diese Implementierung hat ihren Grund in dem in WPF verwendeten DataBinding.

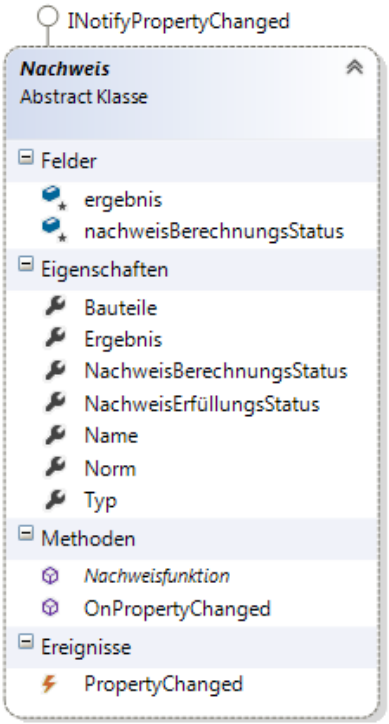


Abbildung 100: Struktur der Klasse „Nachweis“

Neben dieser Änderung verfügt die Klasse „Nachweis“ des zweiten Programms nicht mehr über die Methoden zum Laden der GUI.

INachweisFactory.cs

Die Factory-Schnittstelle für die Nachweise ist ähnlich aufgebaut wie das „IBauteilFactory“-Interface. Sie besitzt ebenfalls eine Eigenschaft für den dargestellten Nachweisnamen und Methoden zum Erstellen eines neuen Nachweises und eines neuen Nachweisfensters.

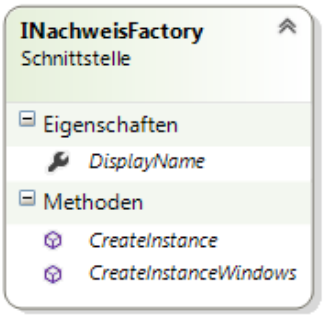


Abbildung 101: Struktur der Schnittstelle „INachweisFactory“

INachweisWindow.cs

Auch für die Nachweiseingabefenster gibt es eine Schnittstelle „INachweisWindow“. Diese besitzt neben der Eigenschaft „DataContext“ und einer Methode zum Anzeigen des Fensters zudem zwei Events. Das eine Event wird ausgelöst, wenn ein neuer Nachweis erstellt werden, das andere wenn er

berechnet werden soll. Die folgende Abbildung zeigt die dafür erstellte, von EventArgs abgeleitete Klasse „VerificationElementCreationArgs", welche die mit dem Ereignis in Zusammenhang stehenden Daten kapselt.

```
public class VerificationElementCreationEventArgs : System.EventArgs
{
    public readonly Nachweis Nachweis;
    public readonly string NachweispaketName;
    public bool Erstellt = true;

    public VerificationElementCreationEventArgs(Nachweis nachweis, string nachweispaketName)
    {
        Nachweis = nachweis;
        NachweispaketName = nachweispaketName;
    }
}
```

Abbildung 102: „VerificationElementCreationArgs"

Die Klasse besitzt neben einem Nachweis und einem Nachweispaketnamen, welche dem Konstruktor übergeben und darin zugewiesen werden, zudem auch eine bool-Variable. Diese gibt an ob der Nachweis erstellt werden konnte oder nicht. Die Verwendung dieser Events ist ähnlich der des „IBauteilWindow"-Interfaces. Wird der „Speichern und Schließen"-Button geklickt, wird ein Objekt dieser Klasse erstellt und einer Methode übergeben, die überprüft ob dem EventHandler-Delegat eine Methode zugewiesen wurde. Wurde eine Methode zugewiesen, wird diese ausgeführt. Konnte der Nachweis nicht der Nachweisliste hinzugefügt werden, wird die Variable „Erstellt" auf false gesetzt. In diesem Fall wird eine Exception geworfen und eine Warnung an den Nutzer ausgegeben. Wurde der „Speichern und Schließen"-Button geklickt, wird das Fenster bei erfolgreichem Erstellen geschlossen. Konnte der Nachweis nicht in das zugehörige Nachweispaket aufgenommen werden, bleibt das Fenster dagegen geöffnet. Wurde der „Berechnen"-Button geklickt, ist der Vorgang bis zur Abfrage des Erstellungsstatus identisch. Konnte der Nachweis nicht erstellt werden, bleibt auch hier das Fenster geöffnet. Hat die Variable jedoch den Wert true, wird ein Objekt vom Typ „VerificationElementCreationArgs" erstellt und einer Methode übergeben, die das Event zum Berechnen eines Nachweises ausführt.

Erweiterung – DefaultNachweise

Zum Hinzufügen eines neuen Nachweistyps sind die Schritte sehr ähnlich dem Hinzufügen eines neuen Bauteiltyps. Für die Nachweise wird die Systematik des Hinzufügens am Beispiel des Nachweises für Biegeknicken von Druckstäben erklärt. Wie bei den Bauteilen muss auch hier die Klasse des neuen Typs, die des zugehörigen Eingabefensters und die der Factory erstellt werden. In der Klasse „BiegeknickenVonDruckstaeben" gibt es anders als im ersten Programm nur noch einen Konstruktor.

```
public BiegeknickenVonDruckstaeben()
{
    Typ = "Biegeknicken von Druckstaeben";
    Norm = "DIN EN 1995-1-1 6.3.2";
    NachweisBerechnungsStatus = "nicht berechnet!";
    Bauteile = new Bauteil[1];
    Ausgabetext = "";
}
```

Abbildung 103: Konstruktor

Der Konstruktor besitzt keine Übergabeparameter, da auch der Nachweis wie die Bauteile schon beim Erstellen des Fensters erstellt wird. Im Konstruktor werden die fixen Variablen zugewiesen und dem Bauteilfeld wird ein Feld mit der Länge eins zugewiesen (siehe Abbildung 103). Der im ersten Programm zusätzlich vorhandene Konstruktor eines leeren Nachweises mit dem Berechnungsstatus „Kein Bauteil" ist jetzt unnötig, da ein Nachweis durch das DataBinding in jedem beliebigen Zustand gespeichert werden kann. Das hat zur Folge, dass ein Nachweis nur den Berechnungsstatus „berechnet" und „nicht berechnet" annehmen kann. Da die Nachweisfunktion jedoch ausgeführt werden kann, wenn die Daten nicht vollständig sind, wird dieser Fall in der Nachweisfunktion durch Exceptions berücksichtigt. Hierzu wird zunächst überprüft ob ein Bauteil ausgewählt wurde. Sollte dies nicht der Fall sein, wird eine entsprechende Exception geworfen, die den Bemessungsvorgang abbricht und eine Warnung ausgibt (siehe Abbildung 104). Wurde ein Bauteil ausgewählt, wird das Bauteil kopiert, wobei alle Umrechnungen und Beiwertbestimmungen ausgeführt werden. Auch um die sonstigen Berechnungen in der Nachweisfunktion wurde ein Try-Catch-Block eingefügt, der für den Fall, dass aufgrund von Unvollständigkeiten ein Fehler auftritt, eine Exception wirft.

```
//Nachweisfunktion
public override void Nachweisfunktion()
{
    if (Bauteile[0] == null)
        throw new KeinBauteilAusgewaehlt();

    HolzBauteil bauteil=new HolzBauteil((HolzBauteil)Bauteile[0]);
```

Abbildung 104: Überprüfung ob ein Bauteil dem Nachweis zugewiesen wurde

Die Nachweisklasse selbst kann, wie auch im ersten Programm, sehr unterschiedlich ausfallen und muss entsprechend des Nachweistyps angepasst werden. Das Gleiche gilt auch für die Klasse „BiegeknickenVonDruckstaebenWindow", die wie alle Nachweiseingabefenster die Schnittstelle „INachweisWindow" implementiert. Die Eingabefensterklassen der Nachweise enthalten wie die der Bauteile lediglich die Informationen zur Darstellung der Informationen und keine Logik zur Datenbearbeitung. In der Klasse „BiegeknickenVonDruckstaebenFactory" erfolgt zunächst wieder die „Export"-Anweisung mit dem Typ „INachweisFactory". Die Zuweisung des angezeigten Namens und der „CreateInstance"-Methode sind identisch mit der von „IBauteilFactory". Die „CreateInstanceWindows"-Methode ist wesentlich umfangreicher. Der Methode werden ein Nachweispaketname und alle vorhandene Bauteile übergeben. Nachdem ein neues Eingabefenster des Typs erstellt wurde, werden der Name des Nachweispakets sowie in diesem Fall alle Holzbauteile in das neu erstellte Fenster zur Darstellung und Auswahl des beteiligten Bauteils übernommen.

```
[Export(typeof(INachweisFactory))]
public class BiegeknickenVonDruckstaebenFactory : INachweisFactory
{
    public string DisplayName
    {
        get { return "Biegeknicken von Druckstaeben"; }
    }

    public Nachweis CreateInstance()
    {
        return new BiegeknickenVonDruckstaeben();
    }

    public INachweisWindow CreateInstanceWindows(string NachweispaketName, ObservableCollection<Bauteil> Bauteile)
    {
        var window=new BiegeknickenVonDruckstaebenWindow();
        window.NachweispaketName_Label.Content = NachweispaketName;
        foreach (var element in Bauteile)
        {
            if (element is HolzBauteil)
            {
                window.BauteilCombobox.Items.Add(element);
            }
        }
        return window;
    }
```

Abbildung 105: Klasse „BiegeknickenVonDruckstaebenFactory“

Zur Erweiterung der Nachweise um neue Nachweistypen wurde ein Template für das Eingabefenster und ein Template für die Factory und die eigentliche Nachweisklasse erstellt. Die Anleitungen zur Verwendung der Templates befinden sich in Kapitel 8.5.1.

8.4.3 Verknüpfungsfunktionen

Die Umsetzung der Verknüpfungsfunktionen ist identisch mit der des ersten Programms, weshalb im Folgenden nicht näher darauf eingegangen wird.

8.4.4 StorageService

Die statische Klasse „StorageService“ beinhaltet alle Methoden zum Einlesen und der Kennwertbestimmung von Material und Geometrie (siehe Abbildung 106).

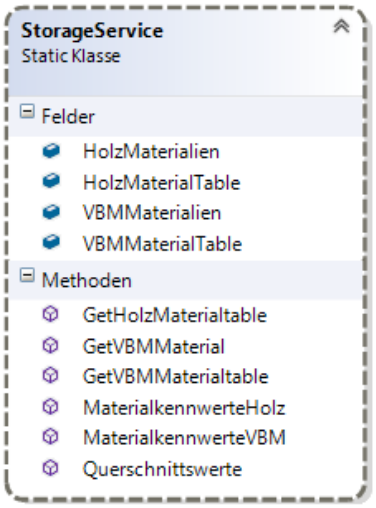


Abbildung 106: Klasse „StorageService“

Um den Code weiter zu vereinfachen, werden im Gegensatz zur ersten Programmversion die Materialtabellen nur einmalig, beim Erstellen des „MainViewModels“ aus der CSV-Datei in die DataTables geladen (siehe Abbildung 107). Im ersten Programm dagegen wurden die Materialien bei jedem Erstellen eines Eingabefensters neu geladen.

```
public MainViewModel()
{
    StorageService.GetHolzMaterialtable();
    StorageService.GetVBMMaterialtable();
}
```

Abbildung 107: Konstruktor der Klasse „MainViewModel“

Zur Bestimmung der Materialkennwerte bekommen die Methoden als Vereinfachung gegenüber der vorherigen Programmversion das bestehende Material als Übergabeparameter und weisen die fehlenden Kennwerte aus dem DataTable direkt den Eigenschaften des passenden Materials zu.

8.5 Fazit des zweiten Programms

8.5.1 Vergleich der beiden Programme

Wie sich in der Vorstellung des zweiten Programms gezeigt hat, wurde durch die Umsetzung des MVVM-Pattern, der MEF-Struktur und der Verwendung von Windows Presentation Foundation eine neue Programmstruktur entwickelt. Das Grundprinzip der Nachweis- und Bauteilstrukturierung wurde erhalten und nur geringfügig der restlichen Struktur angepasst. Dieser Umstand wird bei der Betrachtung des UML-Diagramms deutlich (siehe Abbildung 108).

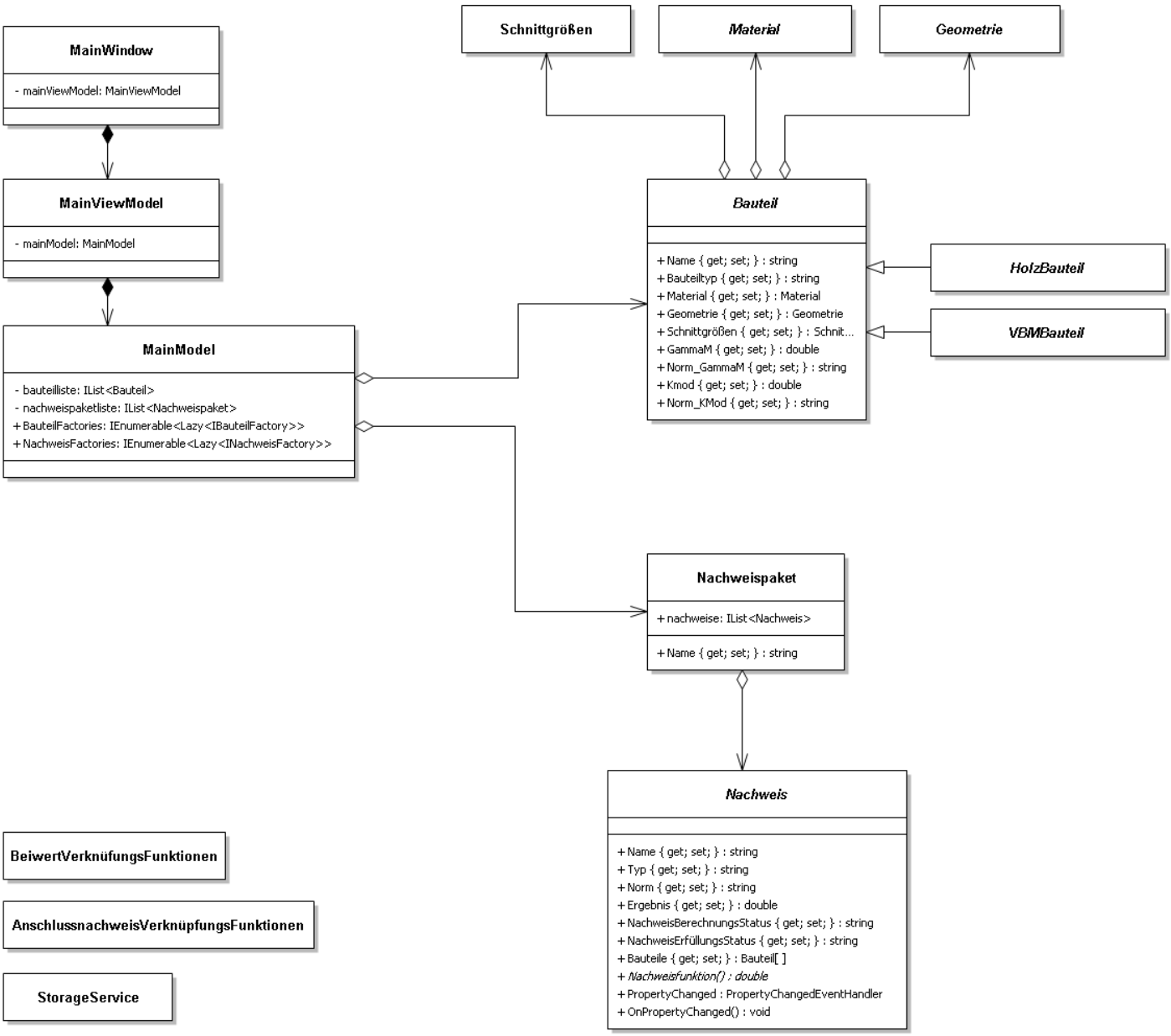


Abbildung 108: UML-Diagramm des zweiten Programms

Das UML-Diagramm ist dem des ersten Programms sehr ähnlich. Lediglich die Umsetzung des MVVM-Patterns am Hauptprogramm sowie die Auslagerung der Material- und Geometriemethoden in

die Klasse „StorageService“ sind auf den ersten Blick verschieden. Ergänzt wurde das bestehende Prinzip zusätzlich durch die Factory-Interfaces, welche der Umsetzung des MEF-Prinzips dienen. Daraus ergibt sich letztendlich auch die Unterteilung des Programmcodes in verschiedene Projekte. Dies ist der erste maßgebliche Unterschied zum ersten Programm.

Ein Nachweis- oder Bauteiltyp besitzt neben dem eigenen Eingabefenster und der eigenen Typklasse in der zweiten Programmversion zudem eine Factory. Im ersten Programm wurde bereits versucht durch das Interface „INachweisGUI“ eine Schablone für alle Nachweiseingabefenster einzuführen. Im zweiten Programm existieren Interfaces für die Eingabefenster der Bauteile und Nachweise, sowie für deren Factories. Die unterschiedlichen Factories ermöglichen erst die Umsetzung des MEF, da sie alle durch die Implementierung des „IBauteilFactory“- oder „INachweisFactory“-Interfaces als gleicher Typ exportiert und importiert werden können. Innerhalb dieser Factories unterscheiden sich diese und stellen je nach Typ die zugehörigen Klassen zur Verfügung. Durch diese Umsetzung des MEF-Prinzips können in dieser zweiten Programmversion Erweiterungen hinzugefügt werden, ohne auf den bisherigen Code zuzugreifen. Diese Tatsache und vor allem die daraus entstehende Erleichterung beim Hinzufügen neuer Elemente werden im Folgenden durch die Anleitung zum Hinzufügen neuer Typen deutlich.

Hinzufügen eines neuen Bauteiltyps

1. Neues Projekt vom Typ „Klassenbibliothek“ erstellen
2. Properties des Projekts durch einen Doppelklick öffnen → Erstellen → Als Ausgabepfad „..\NachweisTool\bin\Debug\“ angeben
3. Verweis auf Projekt „Contracts“ hinzufügen
4. Rechtsklick auf Projekt im Projektmappenexplorer → Hinzufügen → Neues Element...
5. Wahl des Elements „DefaultHolzBauteil“/ „DefaultVBMBauteil“→ Eingabe eines neuen Bauteiltypnamen→ Bestätigung der Eingabe
6. Schritt 4 Wiederholen
7. Wahl des Elements „DefaultHolzBauteilWindow“/ „DefaultVBMBauteilWindow“ → Eingabe des Bauteiltypnamen → Bestätigung der Eingabe
8. Eventuell Anpassen der grafischen Oberfläche
9. Rechtsklick auf Projektmappe → Projektabhängigkeiten → „NachweisTool“ in Abhängigkeit des neuen Projekts setzen um die richtige Buildreihenfolge sicherzustellen

Hinzufügen eines neuen Nachweistyps

1. Neues Projekt vom Typ „Klassenbibliothek“ erstellen
2. Properties des Projekts durch einen Doppelklick öffnen → Erstellen → Als Ausgabepfad „..\NachweisTool\bin\Debug\“ angeben
3. Verweis auf Projekt „Contracts“ hinzufügen
4. Rechtsklick auf Projekt im Projektmappenexplorer → Hinzufügen → Neues Element...

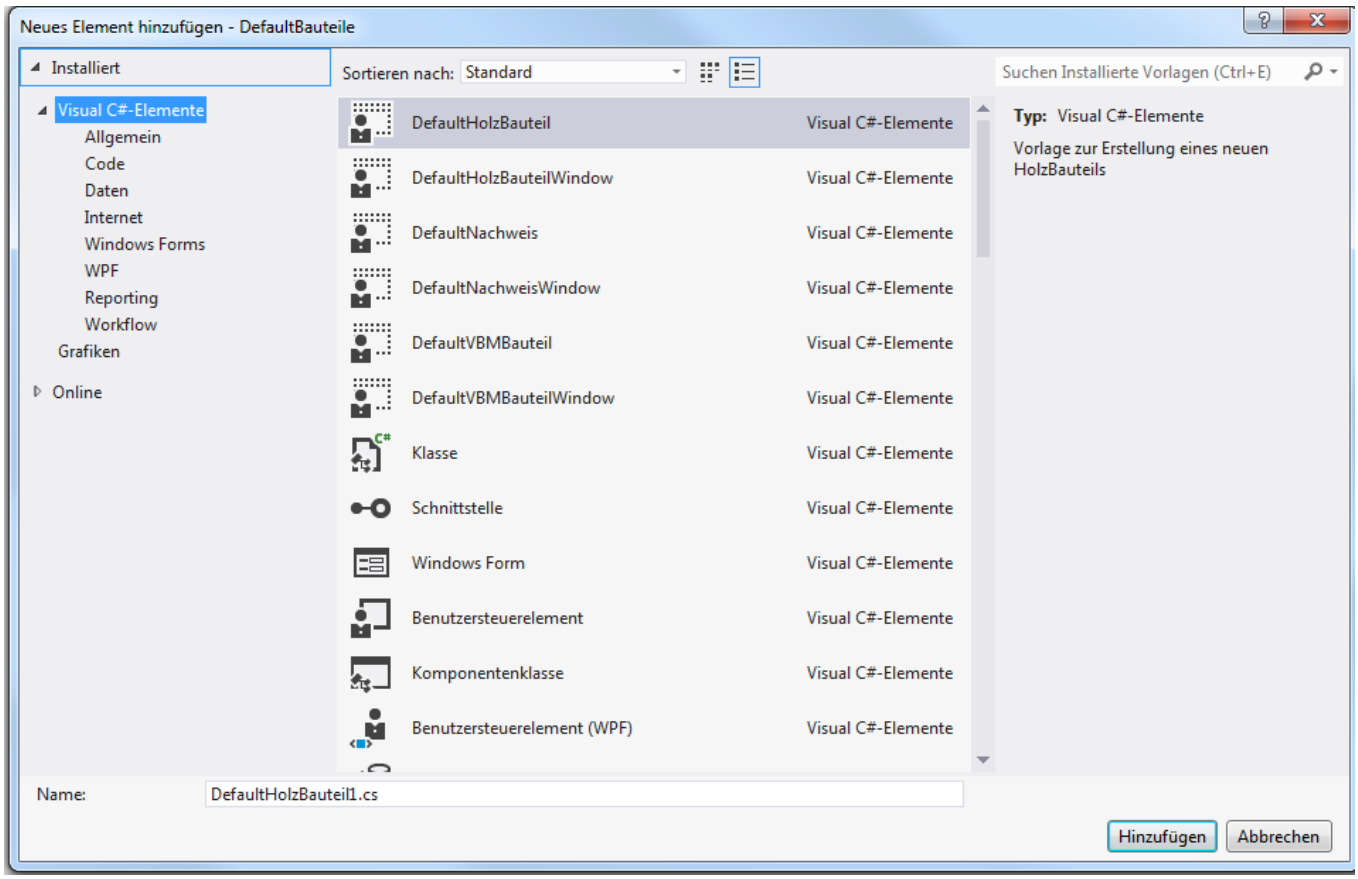


Abbildung 109: Verfügbare Default-Elemente

5. Wahl des Elements „DefaultNachweis“ → Eingabe eines neuen Nachweistypnamen→ Bestätigung der Eingabe
6. Anpassen der Felder, Eigenschaften und des Konstruktors
7. Umschreiben der Nachweisfunktion anhand der Norm
8. Schritt 4 Wiederholen
9. Wahl des Elements „DefaultNachweisWindow“ → Eingabe des Nachweistypnamen→ Bestätigung der Eingabe
10. Eventuell Anpassen der grafischen Oberfläche
11. Rechtsklick auf Projektmappe → Projektabhängigkeiten → „NachweisTool“ in Abhängigkeit des neuen Projekts setzen um die richtige Buildreihenfolge sicherzustellen

Anhand der erstellten Templates wird das Hinzufügen neuer Elemente im Vergleich zum ersten Programm wesentlich vereinfacht. Die Anleitung zeigt, dass keine Änderungen im bestehenden Programmcode vorgenommen werden müssen. Im Gegensatz dazu musste im ersten Programm der Code des Hauptprogramms um die hinzugefügten Typen manuell erweitert werden. Die Verknüpfung zu den anderen Projekten des Programms besteht hier lediglich darin, dass der Verweis auf das Projekt „Contracts“ eingefügt wird und als Ausgabepfad „..\NachweisTool\bin\Debug\“ angeben wird. Durch das MEF-Prinzip findet die Klasse „MainModel“ alle an dieser Stelle gespeicherten Klassenbibliotheken und importiert die vorhandenen, der Vorgabe entsprechenden Factory-Interfaces. Die neuen Elemente müssen dabei nicht zwangsläufig in einem neuen Projekt angelegt werden. Es

kann ebenfalls ein bereits vorhandenes Projekt vom Typ „Klassenbibliothek" verwendet werden. Mit der Umsetzung dieser Erweiterungsmöglichkeit wurde das Hauptziel der Weiterentwicklung des ersten Programms erreicht. Der gewünschte modulare Charakter des Bemessungsprogramms konnte somit umgesetzt und weiterentwickelt werden.

Einen weiteren maßgeblichen Unterschied des zweiten Programms zur ersten Variante stellt die Verwendung des MVVM-Patterns in der Umsetzung mit WPF anstelle von Windows Forms dar. In der ersten Variante waren Logik und Darstellung eng miteinander verknüpft. Im Code der GUIs waren Darstellungslogik und Datenverarbeitung gespeichert. Eine Weiterentwicklung durch einen anderen Entwickler war aufgrund der nicht eindeutigen Struktur schwierig. Im zweiten Programm wurde die MVVM-Struktur für das Projekt „NachweisTool" umgesetzt, wodurch die Darstellungs- und Geschäftslogik und die Daten klar voneinander getrennt wurden. Zudem wurde darauf geachtet, dass auch die Bearbeitung der Daten lediglich im Projekt „NachweisTool" stattfindet. In der ersten Programmversion fand die Bearbeitung der Bauteiltabellen dagegen teilweise sogar im Code der zugehörigen Eingabemaske statt. Durch diese Umsetzung wurde also nicht nur die eigentliche Programmstruktur klarer, sondern auch der Code zu den Eingabefenstern und in den jeweiligen Bauteil- und Nachweisklassen sind nun einfacher und übersichtlicher strukturiert. Um dieses Prinzip zu veranschaulichen zeigen die folgenden Grafiken den internen Programmablauf der beiden Programme beim Erstellen und Hinzufügen eines neuen Bauteils in die Bauteilliste.

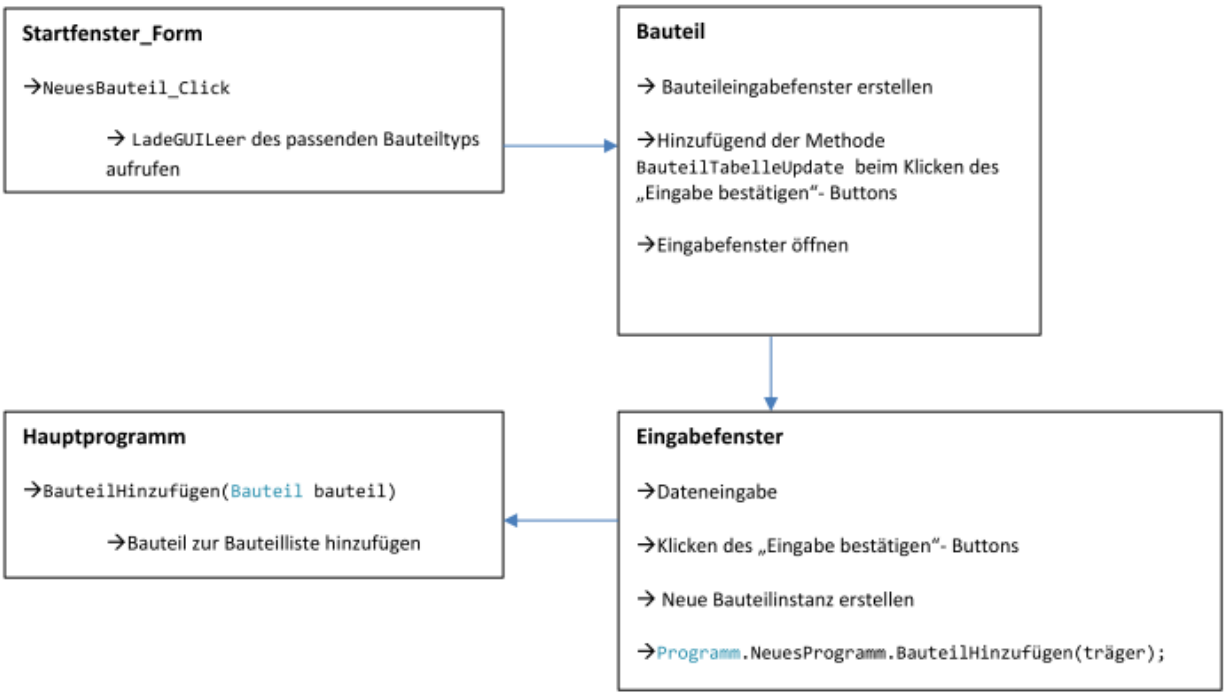


Abbildung 110: Interner Ablauf des ersten Programms

Beim ersten Programm sind Darstellung und Logik sichtbar eng vermischt. Aus dem Code des Hauptfensters wird ein neues Eingabefenster des ausgewählten Bauteiltyps erstellt. Hierzu wird eine Methode der Bauteilklasse aufgerufen. Nach der Eingabe der Bauteildaten wird im Code des Eingabefensters ein neues Bauteil erstellt und dieses durch den Aufruf der entsprechenden Methode

des Hauptprogramms der Bauteilliste hinzugefügt. In dieser Variante steckt in der Bauteil-Klasse Logik der GUI und im Code der GUI, Logik zur Verarbeitung der Daten.

Im zweiten Programm wird im Code des Hauptfensters ein Fenster der gewünschten Factory erstellt und geöffnet. Factory und Fenster werden dann der „ErstelleBauteilClick"-Methode des „MainViewModels" übergeben. Innerhalb dieser Methode wird eine neue Instanz des Bauteiltyps erstellt, welche dann zum DataContext des Eingabefensters wird. Dem Event, welches beim Klicken des „Speichern und Schließen"-Buttons des Eingabefensters ausgelöst wird, wird zudem die Methode „WindowBauteilErstellung" der Klasse „MainViewModel" übergeben. Wird nach der Dateneingabe eben dieser Button geklickt, wird diese Methode ausgeführt. Darin wird wiederum der „BauteilHinzufügen"-Methode des „MainModels" das Bauteil übergeben und zu dessen Bauteilliste hinzugefügt. Es besteht eine klare und eindeutige Struktur und die Bearbeitung der Daten findet außerhalb der GUI statt.

Abbildung 111: Interner Ablauf zweites Programm

Das neue Programm besitzt eine klare Struktur, welche das Nachvollziehen und Weiterentwickeln deutlich vereinfacht. In den Bauteilklassen und vor allem deren GUIs steckt keine Logik des eigentlichen Programms und auch auf die Methoden des „MainModels" wird nicht zugegriffen. Durch das direkte DataBinding wird der Code zusätzlich vereinfacht. Die Objekte können direkt mit den Eingabemasken verknüpft werden, wodurch die bisherige Zuweisung der Werte entfällt.

Durch das Binding hat sich auch die Anwendung des Programms dahingehend verändert, dass bei einer erneuten Bearbeitung der Elemente im Eingabefenster, ein Bezug auf das aktuelle Objekt besteht und somit alle getätigten Änderungen sofort übernommen werden. Dies hat im Vergleich zum ersten Programm weitere Folgen innerhalb des Codes wie beispielsweise das Verwenden von leeren Konstruktoren und des Kopierkonstruktors. Zudem fällt der Zwischenschritt des Speicherns eines leeren Nachweises weg, da er in jedem Zustand gespeichert werden kann. Im ersten Programm dagegen konnte ein Nachweis entweder leer, also nur dessen Typ und Name oder mit sämtlichen Eingaben gespeichert werden. Die Vollständigkeit der Eingaben wurde vor dem Speichern überprüft. Diese Überprüfung wurde nun im Eingabeablauf nach hinten verschoben und findet erst in der Nachweisfunktion selbst statt. Diese Überprüfung wurde ebenfalls mit Exceptions umgesetzt. Im ersten Programm dagegen wurde die Überprüfung beim Klick auf den „Eingabe bestätigen“-Button beziehungsweise dem „Berechnen und Speichern“-Button anhand von manuell eingefügten If-Abfragen umgesetzt. Dabei wurde im Code des Eingabefensters jeweils der benötigte Inhalt der Eingabefelder überprüft. Die Exceptions dagegen wurden direkt mit Try-Catch-Anweisungen um die Berechnungsformeln eingefügt und werden im Falle eines Fehlers aufgrund fehlender Daten ausgelöst. Durch diese neue Umsetzung wurden der Code und damit auch die Erweiterung zusätzlich stark vereinfacht. Vergleicht man den Code der neuen Eingabefenster mit dem der alten, besitzt der neue Code lediglich Präsentationslogik und ist deutlich gekürzt und vereinfacht. Diese Erneuerungen machen die weitere Arbeit mit dem Programmcode des zweiten Programms wesentlich einfacher.

Wie bereits erwähnt, kann der Nutzer aufgrund des DataBinding innerhalb der Eingabefenster nur direkt an einem existenten Bauteil oder Nachweis arbeiten. Diese Eigenschaft ermöglicht es dem Nutzer Bauteile und Nachweise in jedem Bearbeitungsstand zu speichern. Durch diesen direkten Bezug der Eingabefenster auf ein bestehendes Objekt ist es nicht mehr möglich ein bestehendes Bauteil zu verändern und die veränderte Variante unter einem anderen Namen als Kopie zu speichern. Die direkte Verwendung der Bauteile bringt jedoch auch positive Auswirkungen mit sich. So kann beispielsweise der Name eines Bauteils im Nachhinein geändert werden. Wurde dieses Bauteil bereits in einem Nachweis verwendet, wird auch der Name des am Nachweis beteiligten Bauteils automatisch geändert. Das ist möglich, da im Nachweis ein Verweis auf das bestehende Bauteil besteht. Auch die Namen der Nachweise können daher nach dem Erstellen geändert werden. Im ersten Programm, in welchem mit Kopien gearbeitet wurde, wird dagegen bei einer Namensänderung ein neues Bauteil als Kopie des vorherigen Bauteils erstellt. Das alte Bauteil muss manuell gelöscht werden. Durch den Vorgang des Löschens wiederum wird der Berechnungsstatus aller Nachweise, die dieses Bauteil verwenden, auf „kein Bauteil“ gesetzt. Im Nachweis selbst muss dann das neue Bauteil ausgewählt werden um die gewünschten Änderungen zu erhalten. Hier zeigt das zweite Programm deutliche Vorteile.

Des Weiteren haben kleinere Änderungen und Weiterentwicklungen, wie zum Beispiel das Verwenden von eigens erstellten Events und Exceptions und damit verknüpfte Warnungen stattgefunden. In der Anwendung selbst haben neben den genannten Unterschieden nur geringe Änderungen stattgefunden. Im zweiten Programm ist der Import und Export der Daten noch nicht implementiert, da dieser nicht auf ähnliche Weise wie im ersten Programm zu realisieren war. Um das ehemalige Prinzip umzusetzen, müssen vor allen Basisklassen die Typen der davon abgeleiteten Klassen deklariert werden. Diese Umsetzung wiederspricht jedoch der dynamischen Erweiterungsmöglichkeit und der

damit vorhandenen Unabhängigkeit vom bestehenden Code. Die prinzipielle Implementierung der Funktionalität des Imports und Exports ist jedoch durchaus möglich.

8.5.2 Vorteile

- Dynamische Erweiterung ohne Zugriff auf bestehenden Code möglich
- Erweiterung mithilfe der Templates einfach gestaltet
- Klare, verständliche Programmstruktur durch MVVM
- Verständlichkeit des internen Datenflusses
- Nachvollziehbare Programmstruktur macht Weiterentwicklung des Programms einfach
- DataBinding macht ausführliche Variablenzuweisung unnötig
- Events und EventHandler ermöglichen Reaktion auf Ereignisse in der GUI an anderer Stelle
- Vereinfachung des Code und geeignete Reaktion auf auftretende Fehler mittels Exceptions
- Speicherung der Bauteile und Nachweise in jedem Zustand möglich
- Nachträgliche Abänderung der Objektnamen möglich

8.5.3 Nachteile

- Grundlegende Kenntnisse von WPF für Programmierer erforderlich
- Keine einfache Erstellung von Varianten möglich
- Mögliche Implementierung des Exports und Imports schwieriger

8.5.4 Ausblick

Ein wichtiger Schritt um auch das zweite Programm in seiner Vollständigkeit nutzen zu können ist die Implementierung weiterer Nachweis- und Bauteiltypen sowie die Implementierung des Exports und Imports. Bisher wurde die Umsetzung des neuen Konzepts zur Veranschaulichung nur an einem Holzbauteil, einem Verbindungsmittel und einem Nachweis durchgeführt. Wie die anderen Weiterentwicklungsideen des ersten Programm ist auch die Einführung von in Excel-Dateien zur Verfügung gestellten Standardquerschnitten in dieser Programmversion denkbar, um die Wiederverwendbarkeit und den modularen Charakter des Programms zu stärken. Auch die Idee einer weiteren Spezialisierung der Geometrien und Schnittgrößen nach Bauteiltyp sollte weiter untersucht werden. Eine weitere Idee wäre der Einbau von grafischen Elementen, beispielsweise in Form von Bauteil- und Systemskizzen, um die Eingabe und das Programm anschaulicher und damit noch benutzerfreundlicher zu gestalten.

Ein möglicher Kritikpunkt des zweiten Programms ist, dass sich ein Eingabefenster durch das DataBinding immer auf ein direktes Objekt bezieht. So können anders als im ersten Programm nicht einfach Bauteilvarianten durch Abänderung des Namens erstellt werden. Eine Lösung hierfür wäre die Implementierung einer Kopierfunktion. Es ist vorstellbar, dass hierzu ein Bauteil in der bestehenden Bauteilliste ausgewählt wird und durch einen Klick auf einen entsprechenden Button eine Kopie von diesem erstellt wird. Diese Kopierfunktion würde dem Bauteil automatisch einen anderen Namen zuweisen und das Bauteil könnte dann beliebig variiert werden.

Zusammengefasst kann festgehalten werden, dass das zweite Programm durch die umgesetzte dynamische und modulare Weiterentwicklung und der Verwendung der Grundstruktur des ersten Programms das Prinzip des gewünschten Berechnungsprogramms erfüllt.

9 Fazit

Im Rahmen der vorliegenden Masterarbeit wurde sowohl die aktuelle Marktlage bezüglich bereits erhältlicher Bemessungsprogramme für den Holzbau untersucht und bewertet, als auch die Machbarkeit eines modularen, offenen und multifunktionalen Berechnungsprogramm anhand eigener Programmierung untersucht.

Im ersten Teil der Arbeit wurden die verfügbaren Berechnungsprogramme untersucht. Es hat sich gezeigt, dass bereits einige für den Holzbau brauchbare Programme zur Verfügung stehen. Der Umfang der Programme ist jedoch wesentlich geringer als für andere Bauarten und steckt wie am Beispiel von SOFiSTiK gezeigt wurde noch in der Entwicklungsphase. Innerhalb der verfügbaren Bemessungsprogramme konnte zwischen allgemeinen Stabwerks- und FEM-Berechnungsprogrammen mit zusätzlichem Holzbemessungsmodul und speziellen für den Holzbau entwickelten Programmen unterschieden werden. Bei allen Programmen wurden Grenzen in deren Anwendbarkeit festgestellt. Aus diesem Grund wurden die Idee und der Anwendungsbereich einer modularen und offenen Programmplattform erörtert. Es konnte dabei festgestellt werden, dass ein offenes und von verschiedenen Entwicklern ständig erweiterbares Programm optimal wäre, weil es nicht nur kostenfrei verfügbar, sondern auch universell anpassbar und anwendbar ist.

Im nächsten Schritt wurde der Eurocode 5 bezüglich seines Inhalts und Systematik genauer untersucht um darauf aufbauend eine Programmstruktur zu entwickeln. Es hat sich gezeigt, dass eine objektorientierte Programmierung einer Datenbankstruktur vorzuziehen ist, da diese dem Nachweischarakter am besten entspricht. Eine Umsetzung innerhalb von Excel wurde aufgrund der Komplexität der Nachweisführung ausgeschlossen. Auf Basis dieses Ergebnisses wurde die objektorientierte .NET-Sprache C# als Programmiersprache für die Umsetzung des Programms ausgewählt. Als vor allem sehr anwenderfreundliche Programmieroberfläche wurde Visual Studio von Microsoft gewählt.

Im nächsten Schritt wurde eine erste beispielhafte Programmversion umgesetzt. In dieser wurde eine prinzipielle Programmstruktur für ein multifunktionales Berechnungsprogramm entwickelt. Dabei wurde mehrfach das Konzept der Vererbung verwendet. Während der Entwicklung musste immer wieder abgewägt werden, welche Informationen spezialisiert werden müssen und welche verallgemeinert werden können, um somit den modularen Charakter des Programms widerzuspiegeln. Im ersten Programm konnte eine gute, grundlegende Bauteil- und Nachweisstruktur umgesetzt werden, die Erweiterung hingegen war nicht ohne Zugriff auf den bestehenden Programmcode möglich.

Um den direkten Zugriff auf den Programmcode bei der Erweiterung zu umgehen, wurde ein zweites Programm entwickelt. In dieser Programmversion wurde das Managed Extensibility Framework verwendet, um eine dynamische Erweiterung des Programms durch weitere Module zu ermöglichen. Zudem wurde das MVVM-Pattern und WPF verwendet, womit eine bessere Strukturierung und vor allem die Trennung von Logik und Darstellung innerhalb des Codes möglich waren. Durch die beispielhafte Umsetzung einer dynamischen Gestaltung des Codes wurde die grundlegende Nachweisstruktur des ersten Programms erhalten, eine einfache Erweiterbarkeit mittels Templates ermöglicht und eine gut nachvollziehbare Programmstruktur erreicht.

Als Fazit der Untersuchungen bleibt festzuhalten, dass es bereits einige computergestützte Werkzeuge zur Tragwerksplanung im Holzbau gibt, der Markt bezüglich dieser Programme jedoch noch in der Entwicklungsphase steckt und deren Umfang und Einsetzbarkeit in den nächsten Jahren noch zunehmen werden. Ein modulares und offenes Programm stellt vor allem aufgrund seiner Multifunktionalität und ständigen Erweiterbarkeit eine gute Alternative zu den bestehenden Programmen dar. In dieser Masterarbeit wurde durch die beispielhafte Programmierung gezeigt, wie mögliche Strukturen eines solchen Programmes aussehen und es wurde bewiesen, dass dessen Umsetzung möglich ist.

10 Literaturverzeichnis

Bentley Systems (2013): http://www.bentley.com/de-DE/Products/STAAD.Pro/ (Dezember 2013)

Christian Moser (2011): http://www.wpftutorial.net/ (April 2014)

CODE PROJECT (2011): http://www.codeproject.com/Articles/188054/An-Introduction-to-Managed-Extensibility-Framework (April 2014)

developer-zone@pno (2014): http://dev.pno-inkasso.de/tag/mvvm (April 2014)

D.I.E. CAD und Statik Software (2013): http://www.die.de/ (Dezember 2013)

Dietrich's DC Statik (2013): http://www.dc-statik.com/seiten/home_de.htm (Dezember 2013)

Dlubal Software (2013): http://www.dlubal.de/statik-software-fuer-den-holzbau.aspx (Dezember 2013)

Galileo Computing (2010): http://openbook.galileocomputing.de/visual_csharp_2010/ (April 2014)

Graitec Innovation (2013): http://www.graitec.com/ge/cs-statik.asp (Dezember 2013)

Harzer-Statik-Software (2013): http://www.harzerstatik.de/ (Dezember 2013)

HECO-Schrauben (2013): http: //www.heco-schrauben.de/de/Home (Dezember 2013)

INGENIEURBÜRO HOLZBAU (2013): http://www.hobex.net/ (Dezember 2013)

Kunz, A. (2010): MEF – Das Managed Extensibility Framework http: //www.edvsz.hs-osnabrueck.de/fileadmin/user/huelsmann/05_MEF_Anleitung.pdf

mbAEC Software (2013):http://www.mb-programme.de/ (Dezember 2013)

Microsoft Developer Network (2014): http://msdn.microsoft.com/en-us/library/dd460648 (April 2014)

Nemetschek Frilo (2013): http://www.frilo.eu/de/produkte.html (Dezember 2013)

Nemetschek Scia (2013): http://nemetschek-scia.com/de/software/product-selection/scia-engineer (Dezember 2013)

Normenausschuss Bauwesen im DIN (2010): Eurocode 5: Bemessung und Konstruktion von Holzbauten, Berlin: Beuth Verlag GmbH 2010.

Normenausschuss Bauwesen im DIN (2010): Nationaler Anhang – National festgelegte Parameter - Eurocode 5: Bemessung und Konstruktion von Holzbauten, Berlin: Beuth Verlag GmbH 2010.

RIB Software AG (2013): http://www.rib-software.com/de/loesungen/tragwerksplanung/ribtec/bauteilnachweise-im-hochbau/holzbaupaket.html (Dezember 2013)

SOFiSTiK (2013): http://www.sofistik.de/ (Dezember 2013)

SPAX (2013) http: //www.spax.com/de/ (Dezember 2013)

Veit Christoph (2013): http://www.vcmaster.com/de/Start/VCmaster.html (Dezember 2013)

Würth (2013): http://www.wuerth.de/web/de/awkg/meine_branche/downloadcenter/software/technical_sw/technicalsoftware.php (Dezember 2013)

11 Abbildungsverzeichnis

12 Tabellenverzeichnis

13 Anhang

13.1 Beispiel 1: Anschluss zwischen Träger und vertikaler Stütze

13.1.1 Träger

Material und Geometrie

Zunächst müssen hier wie auch bei allen zukünftigen neuen Bauteilen das Material und die Abmessungen gewählt werden. Daraus ergeben sich dann folgende für die Bemessung relevanten Werte:

- Festigkeiten
- Dichte und andere materialspezifische Kennwerte
- Holzart
- Geometrie und Abmessungen

Weitere Eingangsgrößen

Neben den bereits erwähnten Eingangswerten und den aus der Norm entnehmbaren Größen werden ebenfalls die maßgebenden Schnittgrößen benötigt. Diese sollen im späteren Programm als bereits ermittelt und somit bekannt angenommen werden. Ob diese in Form von Tabellenblättern oder durch manuelle Eingabe des Nutzers in der Eingabemaske zur Verfügung gestellt werden bleibt vorerst offen. Zudem sind vor der Berechnung die für die Bemessung notwendigen Parameter der Nutzungsklasse und Lasteinwirkungsdauer festzulegen.

Nachweise

Bei allen Nachweisen muss der Beanspruchung die durch das Bauteil gewährleistete Tragfähigkeit gegenübergestellt werden. Die Ermittlung dieser, anhand der Nutzungsklasse, Baustoffart und Lasteinwirkungsdauer erfolgt immer auf die gleiche Weise und wird deshalb in den folgenden Bemessungen nicht weiter erläutert.

Tragfähigkeit

$$R_d = k_{mod} \frac{R_k}{\gamma_M} \qquad (2.17)$$

γ_M *Teilsicherheitsbeiwert für eine Baustoffeigenschaft aus (2.4.1)*

Tabelle 2.3 — Empfohlene Teilsicherheitsbeiwerte γ_M für Baustoffeigenschaften und Beanspruchbarkeiten

Grundkombinationen:	
Vollholz	1,3
Brettschichtholz	1,25
LVL, Sperrholz, OSB,	1,2
Spanplatten	1,3
Harte Faserplatten	1,3
Mittelharte Faserplatten	1,3
MDF-Faserplatten	1,3
Weiche Faserplatten	1,3
Verbindungen	1,3
Nagelplatten (Stahleigenschaften)	1,25
Außergewöhnliche Kombinationen	1,0

Abbildung 112: y_M-Werte nach EC5

k_{mod} *Modifikationsbeiwert für Lasteinwirkungsdauer und Feuchtegehalt aus (3.1.3)*

Tabelle 3.1 — Werte für k_{mod}

Baustoff	Norm	Nutzungsklasse	Klasse der Lasteinwirkungsdauer				
			ständige Einwirkung	lange Einwirkung	mittlere Einwirkung	kurze Einwirkung	sehr kurze Einwirkung
Vollholz	EN 14081-1	1	0,60	0,70	0,80	0,90	1,10
		2	0,60	0,70	0,80	0,90	1,10
		3	0,50	0,55	0,65	0,70	0,90
Brettschichtholz	EN 14080	1	0,60	0,70	0,80	0,90	1,10
		2	0,60	0,70	0,80	0,90	1,10
		3	0,50	0,55	0,65	0,70	0,90
Furnierschichtholz (LVL)	EN 14374, EN 14279	1	0,60	0,70	0,80	0,90	1,10
		2	0,60	0,70	0,80	0,90	1,10
		3	0,50	0,55	0,65	0,70	0,90
Sperrholz	EN 636						
	Typ EN 636-1	1	0,60	0,70	0,80	0,90	1,10
	Typ EN 636-2	2	0,60	0,70	0,80	0,90	1,10
	Typ EN 636-3	3	0,50	0,55	0,65	0,70	0,90
OSB	EN 300						
	OSB/2	1	0,30	0,45	0,65	0,85	1,10
	OSB/3, OSB/4	1	0,40	0,50	0,70	0,90	1,10
	OSB/3, OSB/4	2	0,30	0,40	0,55	0,70	0,90
Spanplatten	EN 312						
	Typ P4, Typ P5	1	0,30	0,45	0,65	0,85	1,10
	Typ P5	2	0,20	0,30	0,45	0,60	0,80
	Typ P6, Typ P7	1	0,40	0,50	0,70	0,90	1,10
	Typ P7	2	0,30	0,40	0,55	0,70	0,90
Holzfaserplatten, hart	EN 622-2						
	HB.LA, HB.HLA1 oder 2	1	0,30	0,45	0,65	0,85	1,10
	HB.HLA1 oder 2	2	0,20	0,30	0,45	0,60	0,80
Holzfaserplatten, mittelhart	EN 622-3						
	MBH.LA1 oder 2	1	0,20	0,40	0,60	0,80	1,10
	MBH.HLS1 oder 2	1	0,20	0,40	0,60	0,80	1,10
	MBH.HLS1 oder 2	2	–	–	–	0,45	0,80
Holzfaserplatten, MDF	EN 622-5						
	MDF.LA, MDF.HLS	1	0,20	0,40	0,60	0,80	1,10
	MDF.HLS	2	–	–	–	0,45	0,80

Abbildung 113: k_{mod}-Werte nach EC5

R_k *der charakteristische Wert der Beanspruchbarkeit aus Materialkennwerten*

Schubspannungsnachweis

$$\tau_d \leq f_{v,d} \qquad (6.13)$$

$f_{v,d}$ *Bemessungswert der Schubfestigkeit, siehe Tragfähigkeit*

τ_d *Bemessungswert der Schubspannung* $\tau_d = \frac{V*S}{I*b_{ef}}$ *mit V aus der Angabe und S und I aus Querschnittsabmessungen berechnet.* b_{ef} *ergibt sich nach (6.1.7 (2))*

$$b_{ef} = k_{cr}\, b \qquad (6.13a)$$

wobei b die Breite des entsprechenden Abschnitts des Bauteils ist.

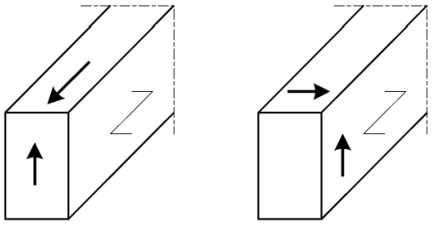
ANMERKUNG Der empfohlene Wert für k_{cr} ist gegeben durch:

k_{cr} = 0,67 für Vollholz

k_{cr} = 0,67 für Brettschichtholz

k_{cr} = 1,0 für andere holzbasierte Produkte in Übereinstimmung mit EN 13986 und EN 14374.

Angaben hinsichtlich der Nationalen Auswahl sind im Nationalen Anhang zu finden.



(a) **(b)**

Bild 6.5 — (a) Bauteil mit einer Schubspannungskomponente in Faserrichtung (b) Bauteil mit beiden Spannungskomponenten rechtwinklig zur Faserrichtung (Rollschub)

NDP Zu 6.1.7(2) Schub

Für Holzwerkstoffe nach DIN EN 13986 und DIN EN 14374 und für Vollholz aus Laubholz gelten die in DIN EN 1995-1-1 empfohlenen Werte.

Für Vollholz und Balkenschichtholz aus Nadelholz gilt $k_{cr} = \frac{2{,}0}{f_{v,k}}$ mit $f_{v,k}$ in N/mm².

Für Brettschichtholz gilt $k_{cr} = \frac{2{,}5}{f_{v,k}}$ mit $f_{v,k}$ in N/mm².

Für Brettsperrholz gilt $k_{cr} = 1{,}0$.

Bei Stäben aus Nadelschnittholz dürfen die Werte für k_{cr} in Bereichen die mindestens 1,50 m vom Hirnholzende des Holzes entfernt liegen, um 30 % erhöht werden.

ANMERKUNG Der k_{cr} – Faktor berücksichtigt den Unterschied der Tragfähigkeit der Bauteile nach längerer Standdauer zu Bauteilen bei Auslieferung, z.B. infolge Rissbildung unter Berücksichtigung der statistischen Verteilung über die Bauteiloberfläche. Er kann nicht mit einer zulässigen Risstiefe im Endzustand gleich gesetzt werden.

Druck rechtwinklig zur Faserrichtung

$$\sigma_{c,90,d} = \frac{F_{c,90,d}}{A_{ef}} \leq k_{c,90} * f_{c,90,d} \qquad (6.3)$$

$F_{c,90,d}$ Bemessungswert der Druckkraft rechtwinklig zur Faserrichtung aus Angabe

A_{ef} wirksame Kontaktfläche

Die wirksame Kontaktfläche rechtwinklig zur Faserrichtung, A_{ef}, sollte unter Berücksichtigung einer wirksamen Kontaktlänge parallel zur Faserrichtung bestimmt werden, wobei die tatsächliche Kontaktlänge ℓ, auf jeder Seite um 30 mm erhöht wird, jedoch nicht mehr als a, ℓ oder $\ell_1/2$, siehe Bild 6.2.

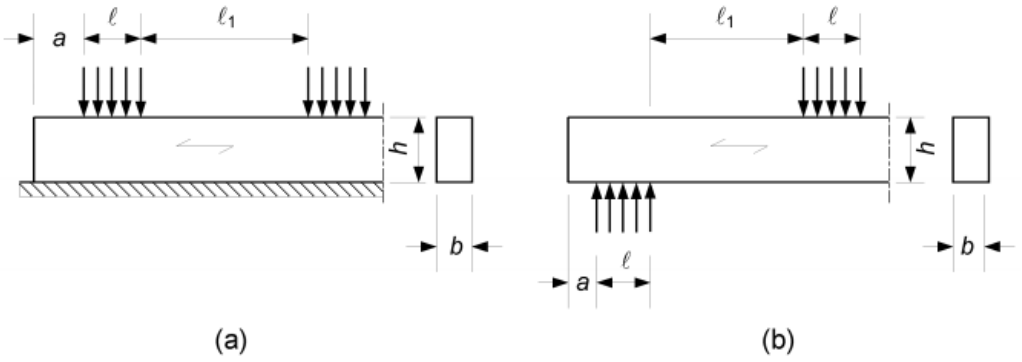


Bild 6.2 — Bauteil auf (a) kontinuierlicher Lagerung und (b) Einzellagerung

Abbildung 114: Bild 6.2 zu A_{ef}

NCI Zu 6.1.5 „Druck rechtwinklig zur Faserrichtung“

ANMERKUNG zu Bild 6.2 "Kontinuierliche Lagerung“ entspricht Schwellendruck. „Einzellagerung“ ist gleichbedeutend mit Auflagerdruck.

(NA.5) Für Bauteile auf Einzellagerung mit $\ell_1 \geq 2$ h ist für Auflagerlängen $\ell > 400$ mm bei Brettschichtholz aus Nadelholz der Wert $k_{c,90} = 1{,}75$ anzunehmen.

(NA.6) Bei Auflagerknoten von Stabwerken mit indirekten Verbindungen gilt $k_{c,90}=1{,}5$.

$f_{c,90,d}$ *Bemessungswert der Druckfestigkeit rechtwinklig zur Faserrichtung, siehe Tragfähigkeit*

$k_{c,90}$ *Beiwert zur Berücksichtigung der Art der Einwirkung, der Spaltgefahr und des Grades der Druckverformung*

(2) Der Wert für $k_{c,90}$ ist in der Regel zu 1,0 anzunehmen, es sei denn, es gelten die Bedingungen der folgenden Absätze. In diesen Fällen darf ein höherer Wert für $k_{c,90}$ bis zu einem Höchstwert von $k_{c,90} = 1{,}75$ angenommen werden.

(3) Für Bauteile auf kontinuierlicher Unterstützung, bei denen $\ell_1 \geq 2h$, siehe Bild 6.2(a), ist in der Regel der Wert für $k_{c,90}$ anzunehmen zu:

— $k_{c,90} = 1{,}25$ bei Vollholz aus Nadelholz;

— $k_{c,90} = 1{,}5$ bei Brettschichtholz aus Nadelholz

wobei h die Höhe des Bauteils und ℓ die Kontaktlänge ist.

(4) Für Bauteile auf Einzelabstützungen, bei denen $\ell_1 \geq 2h$, siehe Bild 6.2(b), ist in der Regel der Wert für $k_{c,90}$ anzunehmen zu:

— $k_{c,90} = 1{,}5$ bei Vollholz aus Nadelholz;

— $k_{c,90} = 1{,}75$ bei Brettschichtholz aus Nadelholz, vorausgesetzt, es gilt: $\ell \leq 400$ mm

wobei h die Höhe des Bauteils und ℓ die Kontaktlänge ist.

Biegeknicken von Druckstäben

$$\frac{\sigma_{c,0,d}}{k_{c,y} f_{c,0,d}} + \frac{\sigma_{m,y,d}}{f_{m,y,d}} + k_m \frac{\sigma_{m,z,d}}{f_{m,z,d}} \leq 1 \quad (6.23)$$

$$\frac{\sigma_{c,0,d}}{k_{c,z} f_{c,0,d}} + k_m \frac{\sigma_{m,y,d}}{f_{m,y,d}} + \frac{\sigma_{m,z,d}}{f_{m,z,d}} \leq 1 \quad (6.24)$$

$\sigma_{c,0,d} = \frac{N}{A}$ *Bemessungswert der Druckspannung in Faserrichtung, N und A aus Angabe und Querschnittsabmessungen*

$f_{c,0,d}$ *Bemessungswert der Druckfestigkeit in Faserrichtung, siehe Tragfähigkeit*

$\sigma_{m,y,d}$ und $\sigma_{m,z,d}$ Bemessungswerte der Biegespannung $\sigma_{m,d} = \frac{M}{W}$ mit M aus Angabe und W aus Querschnittsabmessungen berechnet

$f_{m,y,d}$ und $f_{m,z,d}$ Bemessungswerte der Biegefestigkeit, siehe Tragfähigkeit

k_m Beiwert aus (6.1.6 (2))

(2) Der Wert für den Beiwert k_m ist in der Regel anzunehmen zu:

— für Vollholz, Brettschichtholz und Furnierschichtholz:

bei Rechteckquerschnitten: k_m = 0,7

bei anderen Querschnitten: k_m = 1,0

— für andere tragende Holzwerkstoffe, bei allen Querschnitten: k_m = 1,0.

$k_{c,y}$ und $k_{c,z}$

$$k_{c,y} = \frac{1}{k_y + \sqrt{\left(k_y^2 - \lambda_{rel,y}^2\right)}} \quad (6.53)$$

$$k_{c,z} = \frac{1}{k_z + \sqrt{\left(k_z^2 - \lambda_{rel,z}^2\right)}} \quad (6.26)$$

$$k_y = 0{,}5 * \left(1 + \beta_c\left(\lambda_{rel,y} - 0{,}3\right) + \lambda_{rel,y}^2\right) \quad (6.27)$$

$$k_z = 0{,}5 * \left(1 + \beta_c\left(\lambda_{rel,z} - 0{,}3\right) + \lambda_{rel,z}^2\right) \quad (6.28)$$

β_c Imperfektionsbeiwert

$$\beta_c = \begin{cases} 0{,}2 & \text{für Vollholz;} \\ 0{,}1 & \text{für Brettschichtholz und Furnierholz;} \end{cases} \quad (6.29)$$

$\lambda_{rel,y}$ und $\lambda_{rel,z}$ bezogener Schlankheitsgrad

$$\lambda_{rel,y} = \frac{\lambda_y}{\pi}\sqrt{\frac{f_{c,0,k}}{E_{0,05}}} \quad (6.21)$$

$$\lambda_{rel,z} = \frac{\lambda_z}{\pi}\sqrt{\frac{f_{c,0,k}}{E_{0,05}}} \quad (6.22)$$

λ_y und$\lambda_z = \frac{l_{ef}}{i}$ Schlankheitsgrad für Biegung um zugehörige Achse

Wobei:

i *Trägheitsradius aus Querschnittsabmessungen*

Ersatzstablänge $l_{ef} = \beta * h$ *mit h als Stablänge aus Bauteilabmessungen und β als Knicklängenbeiwert aus NA.13.2*

Tabelle NA.23 — Knicklängenbeiwerte β für Stäbe

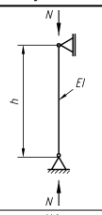
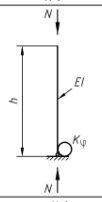
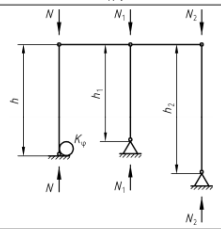
	1 System	2 Knicklängenbeiwert
1		$\beta = 1$
2		$\beta = \sqrt{4 + \frac{\pi^2 \cdot E \cdot I}{h \cdot K_\varphi}}$ K_φ: Federkonstante der elastischen Einspannung (Kraft · Länge/Winkel)
3		$\beta = \sqrt{\left(4 + \frac{\pi^2 \cdot E \cdot I}{h \cdot K_\varphi}\right) \cdot (1 + \alpha)}$ für eingespannte Stütze, mit: $\alpha = \frac{h}{N} \cdot \sum \frac{N_i}{h_i}$

Abbildung 115: Knicklängenbeiwerte β nach NA

$E_{0,05}$ *5%-Quantil des Elastizitätsmodels in Faserrichtung aus Materialangaben*

13.1.2 Stütze

Material und Geometrie sowie die weiteren Eingangsgrößen werden wie beim Träger behandelt.

Nachweise

Biegeknicken von Druckstäben

Auf die genauere Beschreibung des Knicknachweises für die auf Druck belastete Stütze wird hier verzichtet und auf den Biegeknicknachweis des vorherigen Trägers verwiesen.

Schubspannungsnachweis

Auf die genauere Beschreibung des Schubspannungsnachweises wird hier verzichtet und auf den Schubspannungsnachweis des vorherigen Trägers verwiesen.

13.1.3 Anschluss

Bei dem zu bemessenden biegesteifen Anschluss soll es sich um eine Stabdübel-Verbindung mit innen liegendem Stahlblech handeln.

Eingangswerte

Bei der Bemessung des Anschlusses zwischen Träger und Stütze müssen zusätzlich zu den Kennwerten der beiden beteiligten Bauteile auch weitere Eingangsgrößen der Verbindung zur Verfügung stehen:

- Art des Anschlusses
- Stahlgüte VBM
- Durchmesser und Anordnung VBM
- Dicke Blech
- Winkel zwischen Kraft und Faserrichtung

Nachweise

Die Nachweisführung eines solchen gewählten Anschlusses setzt sich aus einzelnen Nachweisen für Verbindungsmittel, Bauteile und deren Interaktion zwischen einander zusammen. Im Folgenden werden diese einzeln aufgeführt, sollen jedoch im Späteren als ein Nachweis betrachtet werden. Natürlich ist aber auch eine Verwendung der Nachweise als Einzelnachweise denkbar.

Anschluss

(Abscheren) Fließmoment des Stabdübels

$$M_{y,Rk} = 0{,}3 f_{u,k} d^{2,6} \qquad (8.30)$$

$M_{y,Rk}$ *chakteristischer Wert des Fließmomentes [Nmm]*

$f_{u,k}$ *charakteristischer Wert der Zugfestigkeit in [N/mm²] aus Angabe*

d *Durchmesser des Stabdübels [mm] aus Angabe*

Falls d≤30mm gilt in Holz und Furnierschichtholz LVL:

charakteristischer Wert der Lochleibungsfestigkeit in Faserrichtung:

$$f_{h,0,k} = 0{,}082(1 - 0{,}01d)\rho_k \qquad (8.32)$$

ρ_k *charakteristischer Wert der Rohdichte des Holzes [kg/m³]*

charakteristischer Wert der Lochleibungsfestigkeit bei einem Winkel α zur Faserrichtung:

$$f_{h,\alpha,k} = \frac{f_{h,0,k}}{k_{90}\sin^2\alpha + \cos^2\alpha} \qquad (8.31)$$

α Winkel zwischen Kraft-und Faserrichtung

$f_{h,0,k}$ *charakteristischer Wert der Lochleibungsfestigkeit in Faserrichtung, s.o.*

k_{90} *Beiwert siehe (8.33)*

$$k_{90} = \begin{cases} 1{,}35 + 0{,}015\,d & \text{für Nadelhölzer} \\ 1{,}30 + 0{,}015\,d & \text{für Furnierschnittholz LVL} \\ 0{,}90 + 0{,}015\,d & \text{für Laubhölzer} \end{cases} \qquad (8.33)$$

wirksame Verbindungsmittelanzahl in Faserrichtung für Kräfte in dieser Richtung:

a_1 *Abstand in Faserrichtung aus Angabe*

n *Anzahl der Verbindungsmittel in Reihe*

$$n_{ef} = min\left\{ \begin{matrix} n \\ n^{0{,}9^4}\sqrt{\frac{a_1}{13d}} \end{matrix} \right. \qquad (8.34)$$

wirksame Verbindungsmittelanzahl für Kräfte rechtwinklig zur Faserrichtung:

$$n_{ef} = n \qquad (8.35)$$

Stabdübeldurchmesser

(2) Der Stabdübeldurchmesser sollte kleiner als 30 mm und größer als 6 mm sein.

Mindestabstände

Tabelle 8.5 — Mindestabstände von Stabdübeln

Abstände (siehe Bild 8.7)	Winkel	Mindestabstände
a_1 (in Faserrichtung)	$0° \le \alpha \le 360°$	$(3 + 2 \lvert \cos \alpha \rvert) d$
a_2 (rechtwinklig zur Faserrichtung)	$0° \le \alpha \le 360°$	$3 d$
$a_{3,t}$ (beanspruchtes Hirnholzende)	$-90° \le \alpha \le 90°$	max (7 d; 80 mm)
$a_{3,c}$ (unbeanspruchtes Hirnholzende)	$90° \le \alpha < 150°$ $150° \le \alpha < 210°$ $210° \le \alpha \le 270°$	max ($a_{3,t} \lvert \sin \alpha \rvert$) d; 3 d) 3 d max ($a_{3,t} \lvert \sin \alpha \rvert$) d; 3 d)
$a_{4,t}$ (beanspruchter Rand)	$0° \le \alpha \le 180°$	max [(2 + 2 sin α) d; 3 d]
$a_{4,c}$ (unbeanspruchter Rand)	$180° \le \alpha \le 360°$	3 d

Abbildung 116: Mindestabstände nach EC5 Tabelle 8.5

Charakteristischer Wert der Tragfähigkeit $F_{v,Rk}$ je Scherfuge und Verbindungsmittel

$$F_{v,Rk} = \sqrt{2} * \sqrt{2 * M_{y,Rk} * f_{h,k} * d} \qquad (NA.\ 108)$$

Mindestholzdicke t_{req} für Mittelhölzer mit zweischnittig beanspruchten Verbindungsmitteln

$$t_{req} = 1{,}15 * 4 * \sqrt{\frac{M_{y,Rk}}{f_{h,k} * d}} \qquad (NA.109)$$

Bemessungswerte der Tragfähigkeit

$$F_{v,Rd} = \frac{k_{mod} * F_{v,Rk}}{\gamma_M} \qquad (NA.106)$$

Mit $\gamma_M = 1{,}1$

Nachweis der erhöhten Querkraft im Anschlussbereich

Der oben bereits erläuterte Schubspannungsnachweis ist erneut mit erhöhter Querkraft durchzuführen. Dies bedeutet, dass aus der Anschlussbemessung neue Eingangsgrößen für diesen Nachweis entstanden sind.

13.2 Beispiel 2: Satteldachträger mit gekrümmten unteren Rand

Material und Geometrie

Zunächst muss auch hier wieder das Material gewählt werden und die Abmessungen festgelegt werden. Die Abmessungen des Satteldachträgers sind dabei jedoch wesentlich komplexer als die eines einfachen Trägers. Aus der Eingabe des Anwenders ergeben sich folgende Eingangswerte:

- Festigkeiten
- Dichte und andere materialspezifische Kennwerte
- Holzart
- Geometrie und Abmessungen

Weitere Eingangsgrößen

Die maßgebenden Schnittgrößen sollen im späteren Programm wieder als bereits bekannt angenommen werden. Zusätzlich hierzu sind für die Querzugverstärkung Angaben zu den für die Verstärkung verwendete Verbindungsmittel wie beispielsweise die Art, Anzahl, Anordnung, Festigkeit, Durchmesser und Länge festzulegen.

Nachweise

Nachweis der Biegespannungen am Ort der maximalen Spannung

Bemessungswerte der Biegespannungen nach (6.4.2(2)):

$$\sigma_{m,\alpha,d} = \sigma_{m,0,d} = \frac{6M_d}{bh^2} \qquad (6.37)$$

Wobei gilt:

$$\sigma_{m,\alpha,d} \leq k_{m,\alpha} f_{m,d} \qquad (6.38)$$

$\sigma_{m,\alpha,d}$ *Bemessungswert der Biegebeanspruchung unter Berücksichtigung des Trägeranschnittes*

Trägerbreite b *und Trägerhöhe* h *am Ort der maximalen Spannungen aus Nebenrechnungen*

$f_{m,d}$ *Bemessungswert der Biegefestigkeit s.o.*

$k_{m,\alpha}$ *für Zugspannungen entlang des angeschnittenen Randes:*

$$k_{m,\alpha} = \frac{1}{\sqrt{1+\left(\frac{f_{m,d}}{0{,}75 f_{v,d}}\tan\alpha\right)^2+\left(\frac{f_{m,d}}{f_{t,90,d}}\tan^2\alpha\right)^2}} \qquad (6.39)$$

$k_{m,\alpha}$ *für Druckspannungen entlang des angeschnittenen Randes:*

$$k_{m,\alpha} = \frac{1}{\sqrt{1+\left(\frac{f_{m,d}}{1,5 f_{v,d}}\tan\alpha\right)^2+\left(\frac{f_{m,d}}{f_{c,90,d}}\tan^2\alpha\right)^2}} \qquad (6.40)$$

Nachweis der Biegespannungen im gekrümmten Bereich

Bemessungswerte der Biegespannungen im Firstbereich nach (6.4.3(4)):

$$\sigma_{m,d} = k_l \frac{6 M_{ap,d}}{b * h_{ap}^2} \qquad (6.42)$$

Trägerhöhe im First h_{ap} *aus Angabe*

$M_{ap,d}$ *Bemessungsmoment im Firstquerschnitt*

k_l *aus*

$$k_\ell = k_1 + k_2\left(\frac{h_{ap}}{r}\right) + k_3\left(\frac{h_{ap}}{r}\right)^2 + k_4\left(\frac{h_{ap}}{r}\right)^3 \qquad (6.43)$$

mit

$$k_1 = 1 + 1{,}4 \tan\alpha_{ap} + 5{,}4 \tan^2\alpha_{ap} \qquad (6.44)$$

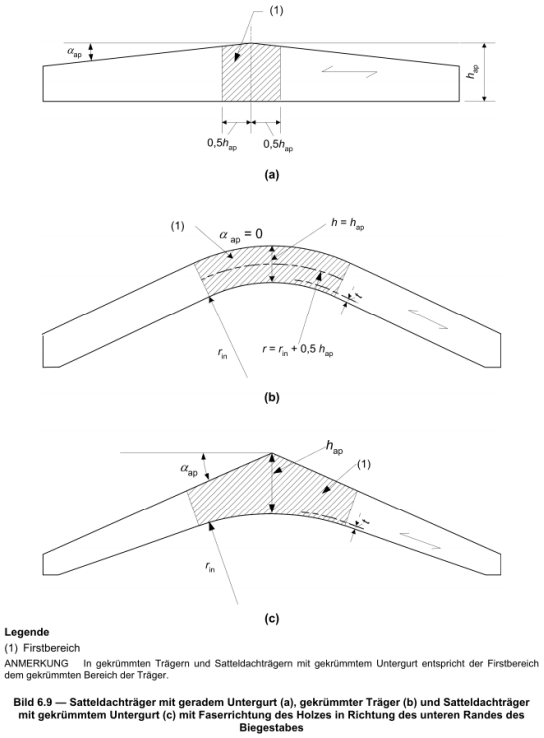
$$k_2 = 0{,}35 - 8 \tan\alpha_{ap} \qquad (6.45)$$

$$k_3 = 0{,}6 + 8{,}3 \tan\alpha_{ap} - 7{,}8 \tan^2\alpha_{ap} \qquad (6.46)$$

$$k_4 = 6 \tan^2\alpha_{ap} \qquad (6.47)$$

$$r = r_{in} + 0{,}5\, h_{ap} \qquad (6.48)$$

r_{in} *Innenradius nach Bild 6.9*



Legende
(1) Firstbereich
ANMERKUNG In gekrümmten Trägern und Satteldachträgern mit gekrümmtem Untergurt entspricht der Firstbereich dem gekrümmten Bereich der Träger.

Bild 6.9 — Satteldachträger mit geradem Untergurt (a), gekrümmter Träger (b) und Satteldachträger mit gekrümmtem Untergurt (c) mit Faserrichtung des Holzes in Richtung des unteren Randes des Biegestabes

Abbildung 117: Definition des Innenradius r_{in}

Im Firstbereich sollten die Biegespannungen nach (6.4.3(3)) folgende Bedingung erfüllen:

$$\sigma_{m,d} \leq k_r f_{m,d} \qquad (6.41)$$

k_r *Beiwert zur Berücksichtigung der Spannungen infolge des Biegens der Lamellen während der Herstellung nach (6.4.3(5)):*

(5) Für Satteldachträger mit geradem Untergurt ist $k_r = 1{,}0$. Für gekrümmte Träger (mit konstantem Querschnitt) und für Satteldachträger mit gekrümmtem Untergurt sollte k_r angenommen werden zu:

$$k_r = \begin{cases} 1 & \text{für } \frac{r_{in}}{t} \geq 240 \\ 0{,}76 + 0{,}001\frac{r_{in}}{t} & \text{für } \frac{r_{in}}{t} < 240 \end{cases} \qquad (6.49)$$

Dabei ist

r_{in} der innere Radius, siehe Bild 6.9;

t die Lamellendicke.

Schubspannungsnachweis am Auflager

Auf die genauere Beschreibung des Schubspannungsnachweises wird hier verzichtet und auf den Schubspannungsnachweis des Trägers im vorherigen Beispiel verwiesen.

Nachweis der Auflagerpressung/Druck rechtwinklig zur Faser

Auf die genauere Beschreibung des Nachweises der Auflagerpressung wird hier verzichtet und auf den Drucknachweis rechtwinklig zur Faser des Trägers im vorherigen Beispiel verwiesen.

Nachweis der Kippsicherheit (Biegedrillknicken)

Bezogener Kippschlankheitsgrad:

$$\lambda_{rel,m} = \sqrt{\frac{f_{m,k}}{\sigma_{m,crit}}} \qquad (6.30)$$

$\sigma_{m,crit}$ *kritische Biegespannung nach der klassischen Stabilitätstheorie, berechnet mit den 5%-Quantilwerten der Steifigkeiten*

$$\sigma_{m,crit} = \frac{M_{y,crit}}{W_y} = \frac{\pi\sqrt{E_{0,05} I_z G_{0,05} I_{tor}}}{l_{ef} W_y} \qquad (6.31)$$

$E_{0,05}$ *5%-Quantilwert des Elastizitätsmoduls in Faserrichtung*

$G_{0,05}$ *5%-Quantilwert des Schubmoduls in Faserrichtung*

I_z *Flächenmoment 2. Grades um die schwache Achse z*

I_{tor} *Torsionsträgheitsmoment*

W_y *Widerstandsmoment um die starke Achse y*

l_{ef} *wirksame Länge des Biegestabs nach Tabelle 6.1*

Tabelle 6.1 — Wirksame Länge als Quotient der Stützweite

Art des Biegestabes	Art der Belastung	ℓ_{ef}/ℓ [a]
Einfach unterstützt	Konstantes Biegemoment	1,0
	Gleichmäßig verteilte Belastung	0,9
	Einzellast in Feldmitte	0,8
Auskragend	Gleichmäßig verteilte Belastung	0,5
	Einzellast am freien Kragende	0,8

[a] Der Quotient aus wirksamer Länge ℓ_{ef} und der Stützweite ℓ gilt für einen Biegestab, der an den Auflagern ausreichend gegen Verdrehen gesichert ist, und Lasteintragung in der Schwerachse des Querschnitts. Greift die Last am Druckrand des Biegestabes an, dann sollte ℓ_{ef} um $2h$ erhöht werden. ℓ_{ef} darf um $0{,}5h$ verringert werden, wenn die Last am Zugrand des Biegestabes angreift.

Abbildung 118: Wirksame Länge l_{ef}

Für Nadelholz gilt nach *(6.32):*

$$\sigma_{m,crit} = \frac{0{,}78b^2}{hl_{ef}}E_{0,05} \qquad (6.32)$$

Ist nur ein Biegemoment um die starke Achse y vorhanden sollte folgende Bedingung gelten:

$\sigma_{m,d} \leq k_{crit} f_{m,d}$

k_{crit} *Beiwert zur Berücksichtigung der zusätzlichen Spannungen infolge seitlichen Ausweichens nach (6.34)*

(4) Bei Biegestäben mit spannungsloser AC seitlicher AC Vorkrümmung innerhalb der in Abschnitt 10 festgelegten Grenzen darf k_{crit} nach Gleichung (6.34) bestimmt werden.

$$k_{crit} = \begin{cases} 1 & \text{für} \quad \lambda_{rel,m} \leq 0{,}75 \\ 1{,}56 - 0{,}75\lambda_{rel,m} & \text{für} \quad 0{,}75 < \lambda_{rel,m} \leq 1{,}4 \\ \dfrac{1}{\lambda^2_{rel,m}} & \text{für} \quad 1{,}4 < \lambda_{rel,m} \end{cases} \qquad (6.34)$$

(5) Bei Biegestäben, bei denen ein seitliches Ausweichen des Druckgurtes über die gesamte Länge verhindert wird und an den Auflagern eine Gabellagerung besteht, darf der Beiwert k_{crit} zu 1,0 angenommen werden.

Zusätzlich gilt nach (NA.7):

$$\frac{\sigma_{c,0,d}}{k_{c,y} * f_{c,0,d}} + \frac{\sigma_{m,y,d}}{k_{crit} * f_{m,y,d}} + \left(\frac{\sigma_{m,z,d}}{f_{m,z,d}}\right)^2 \leq 1 \qquad (NA.58)$$

$$\frac{\sigma_{c,0,d}}{k_{c,z} * f_{c,0,d}} + \left(\frac{\sigma_{m,y,d}}{k_{crit} * f_{m,y,d}}\right)^2 + \frac{\sigma_{m,z,d}}{f_{m,z,d}} \leq 1 \qquad (NA.59)$$

Nachweis der Querzugspannungen im Firstquerschnitt

Maximale Zugspannung rechtwinklig zur Faserrichtung:

$$\sigma_{t,90,d} = k_p \frac{6M_{ap,d}}{b\,h_{ap}^2} \qquad (6.54)$$

$M_{ap,d}$ *Bemessungswert des Biegemoments im First, das zu Querzugspannungen führt*

k_p *nach (6.56):*

$$k_p = k_5 + k_6\left(\frac{h_{ap}}{r}\right) + k_7\left(\frac{h_{ap}}{r}\right)^2 \qquad (6.56)$$

mit

$$k_5 = 0{,}2 \tan \alpha_{ap} \qquad (6.57)$$

$$k_6 = 0{,}25 - 1{,}5 \tan \alpha_{ap} + 2{,}6 \tan^2 \alpha_{ap} \qquad (6.58)$$

$$k_7 = 2{,}1 \tan \alpha_{ap} - 4 \tan^2 \alpha_{ap} \qquad (6.59)$$

Dabei sollte im Firstbereich folgende Bedingung erfüllt sein. Ist diese Bedingung nicht erfüllt, ist eine Verstärkung zur Aufnahme der Querzugspannungen erforderlich!

$$\sigma_{t,90,d} \leq k_{dis} k_{vol} f_{t,90,d} \qquad (6.50)$$

Für eine kombinierte Beanspruchung aus Querzug und Schub gilt folgende Bedingung:

$$\frac{\tau_d}{f_{v,d}} + \frac{\sigma_{t,90,d}}{k_{dis} k_{vol} f_{t,90,d}} \leq 1 \qquad (6.53)$$

k_{vol} nach (6.51) und k_{dis} nach (6.52), Trägerart und Holzart aus Angabe

$$k_{vol} = \begin{cases} 1{,}0 & \text{für Vollholz} \\ \left(\frac{V_0}{V}\right)^{0{,}2} & \text{für Brettschichtholz und Furnierschichtholz mit allen Furnieren in Richtung der Stabachse} \end{cases} \quad (6.51)$$

AC $k_{dis} = \begin{cases} 1{,}4 & \text{für Satteldachträger mit geradem Untergurt und konzentrisch gekrümmte Träger mit gekrümmten Untergurt} \\ 1{,}7 & \text{für Satteldachträger mit gekrümmten Untergurt} \end{cases}$ (6.52) AC

Dabei ist

k_{dis} ein Beiwert zur Berücksichtigung der Spannungsverteilung im Firstbereich;

k_{vol} ein Volumenfaktor;

$f_{t,90,d}$ ein Bemessungswert der Zugfestigkeit rechtwinklig zur Faserrichtung;

V_0 das Bezugsvolumen von 0,01 m^3;

V das querzugbeanspruchtes Volumen im Firstbereich, in m^3 (siehe Bild 6.9), sollte nicht größer als $2V_b/3$, mit V_b als Gesamtvolumen des Biegestabes, angenommen werden.

Querzugverstärkung mittels Gewindestangen

Die oben genannten Bedingungen für die maximal zulässigen Querzugspannungen im First dürfen nach den folgenden Bedingungen unberücksichtigt bleiben:

NCI NA.6.8.5 Verstärkungen für die Aufnahme zusätzlicher klimabedingter Querzugspannungen für Satteldachträger mit geradem Untergurt, gekrümmte Träger und Satteldachträger mit gekrümmten Untergurt

(NA.1) Für Satteldachträger mit geradem Untergurt, gekrümmte Träger und Satteldachträger mit gekrümmtem Untergurt mit Verstärkungen nach Absätzen (NA.2) bis (NA.6) für die Aufnahme zusätzlicher, klimabedingter Querzugspannungen dürfen in den Nutzungsklassen 1 und 2 die Bedingungen nach Gleichung (6.50) und Gleichung (6.53) unbeachtet bleiben, sofern die maximale Zugspannung rechtwinklig zur Faserrichtung des Holzes im Firstquerschnitt Gleichung (NA.91) erfüllt:

$$\frac{\sigma_{t,90,d}}{k_{dis} \cdot \left(\frac{h_o}{h_{ap}}\right)^{0{,}3} \cdot f_{t,90,d}} + \left(\frac{\tau_d}{f_{v,d}}\right)^2 \le 1 \quad (NA.91)$$

dabei ist

k_{dis} = 1,3 für Satteldachträger mit geradem oder gekrümmten Untergurt;

k_{dis} = 1,15 für gekrümmte Träger;

h_o = 600 mm;

$\sigma_{t,90,d}$ = Bemessungswert der Querzugspannungen nach DIN EN 1995-1-1:2010-12, 6.4.3(8).

Zugkraft für die Verstärkungen zur Aufnahme zusätzlicher klimabedingter Querzugspannungen zu bemessen sind:

$$F_{t,90,d} = \frac{\sigma_{t,90,d} * b^2 * a_1}{640 * n} \qquad (NA.92)$$

Fugenspannung:

$$\frac{\tau_{ef,d}}{f_{k1,d}} \leq 1 \qquad (NA.93)$$

Mit

$$\tau_{ef,d} = \frac{2 * F_{t,90,d}}{\pi * l_{ad} * d_r} \qquad (NA.94)$$

$F_{t,90,d}$ *Bemessungswert der Zugkraft je Stahlstab aus Angabe*

l_{ad} *wirksame Verankerungslänge des Stahlstabs oberhalb und unterhalb der Trägerachse, aus Angabe*

d_r *Stahlstabaußendurchmesser aus Angabe*

Und dem Bemessungswert de Ausziehparameter der Holzschrauben:

$f_{k,1,k} = 22 * 10^{-6} * \rho_k^2$

Zu bemessende Zugkraft zur vollständigen Aufnahme der Querzugkräfte durch Verstärkungen

➔ In den beiden inneren Vierteln des querzugbeanspruchten Bereichs:

$$F_{t,90,d} = \sigma_{t,90,d} * b * \frac{a_1}{n} \qquad (NA.99)$$

➔ In den äußeren Vierteln:

$$F_{t,90,d} = \frac{2}{3}\sigma_{t,90,d} * b * \frac{a_1}{n} \qquad (NA.100)$$

b *Trägerbreite, aus Angabe*

a_1 *Abstand der Verstärkungen in Trägerlängsrichtung in Höhe der Trägerachse, aus Angabe*

n *Anzahl der Verstärkungselemente im Bereich innerhalb der Länge* a_1 *aus Angabe*

Abstände und Anordnung der Verstärkungen

Es gilt:

$$250mm \leq a_1 \leq 0{,}75 * h_{ap}$$

Nach (NCI NA.6.8.6(NA.5))

> (NA.4) Die Stahlstäbe müssen mit Ausnahme einer Randlamelle über die gesamte Trägerhöhe durchgehen und sollten im querzugbeanspruchten Bereich an der Trägeroberkante untereinander mindestens 250 mm jedoch nicht mehr als 0,75 h_{ap} Abstand zueinander haben.

Biegespannungsnachweis am reduzierten Querschnitt

Hierzu wird der bereits erläuterte Nachweis der Biegespannungen im gekrümmten Bereich des Trägers erneut mit dem durch die Bohrungen für die Gewindestangen reduzierten Querschnittswerten durchgeführt. Das heißt wiederum, dass der Satteldachträger neben seinen normalen geometrischen Größen auch reduzierte Querschnittsgrößen speichern können muss.

Selbständigkeitserklärung

Hiermit versichere ich, dass ich die vorliegende Arbeit selbständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel benutzt habe.

……..

Vorname Name

München, x. Juni 2014