Maximilian Müller

Konzipierung einer Automatisierten Methode zur Generierung von Toolmanagement Stammdaten

Masterarbeit

zur Erlangung des akademischen Grades

Diplomingenieur

Wirtschaftsingenieurwesen-Maschinenbau

eingereicht an der

Technischen Universität Graz

Betreuer: Dipl.-Ing. Johannes Schmid, BSc
Begutachter: Univ.-Prof. Dipl.-Ing. Dr.techn. Franz Haas

Institut für Fertigungstechnik

Graz, 25. März 2021

EIDESSTATTLICHE ERKLÄRUNG

AFFIDAVIT

Ich erkläre an Eides statt, dass ich die vorliegende Arbeit selbstständig verfasst, andere als die angegebenen Quellen/Hilfsmittel nicht benutzt, und die den benutzten Quellen wörtlich und inhaltlich entnommenen Stellen als solche kenntlich gemacht habe. Das in TUGRAZonline hochgeladene Textdokument ist mit der vorliegenden Masterarbeit identisch.

I declare that I have authored this thesis independently, that I have not used other than the declared sources/resources, and that I have explicitly indicated all material which has been quoted either literally or by content from the sources used. The text document uploaded to TUGRAZonline is identical to the present master‘s thesis.

Datum / Date

Unterschrift / Signature

Danksagung

An dieser Stelle möchte ich mich bei all jenen bedanken, die an der erfolgreichen Durchführung dieser Arbeit beteiligt waren.

Mein ganz besonderer Dank gilt Herrn Univ.-Prof. Dipl.-Ing. Dr.techn. Franz Haas, welcher Institutsvorstand am Institut für Fertigungstechnik ist, und Herrn Dipl.-Ing. Dr.techn. Rudolf Pichler, dem Leiter der smartfactory@tugraz. Sie beide ermöglichten mir die Durchführung dieser spannenden Arbeit.

Des Weiteren gebührt mein Dank Herrn Dipl.-Ing. Johannes Schmid, der die Betreuung dieser interessanten Arbeit übernahm. Seine fachliche Kompetenz, sein intensiver, zeitlicher Einsatz und seine hilfreichen Anregungen haben maßgeblich zum Erfolg dieser Arbeit beigetragen. Nicht ohne Grund bleibt mir durch ihn eine spannende Zeit in Erinnerung. Außerdem möchte ich mich bei allen Mitarbeiterinnen und Mitarbeitern der smartfactory@tugraz bedanken, welche mir alle in positiver Erinnerung bleiben werden.

Bei Herrn Dipl.-Ing. Gerhard Krubner möchte ich mich für die hilfreichen Tipps bedanken, welche mir den Einstieg in NX Open erleichtert haben. Auch bei Herrn Univ.Doz. Hon.Prof. Dipl.-Ing. Dr.techn. Michael Heiss möchte ich mich für die Bereitstellung benötigter Mittel bedanken.

Zu guter Letzt möchte ich allen nahestehenden Personen meinen Dank für die benötigte Motivation und Unterstützung zum Abschließen meines Studiums ausdrücken.

Maximilian Müller

Graz, 25. März 2021

Kurzfassung

Die vorliegende Arbeit beschäftigt sich mit der Optimierung eines Prozesses der CAD/CAM/CNC-Prozesskette der smartfactory@tugraz an der Technischen Universität Graz. Die Methode zur Überführung von physischen Werkzeugen in digitale Werkzeugmodelle und die anschließende Integration in eine Werkzeugdatenbank sind Ausgangssituation für die wissenschaftlichen Untersuchungen.

Das Ziel dieser Arbeit ist die automatisierte Generierung digitaler Werkzeugmodelle für Schaftfräser. Ausgehend von einem Werkzeugscan des Komplettwerkzeugs mittels Voreinstellgerät ist die zentrale Aufgabe dieser Arbeit die Aufbereitung des Werkzeugmodells für eine CAM-Planung sowie die Integration in eine Werkzeugdatenbank. Der Werkzeugscan muss hierfür in dessen Einzelkomponenten separiert sowie charakteristische Größen für die Integration in die Werkzeugdatenbank Siemens MRL definiert werden. Hierbei kommt die Programmierschnittstelle NX Open in Kombination mit der Software Siemens NX zum Einsatz.

Ausgehend vom Resultat des Werkzeugscans, welcher als Konturskizze des Komplettwerkzeugs im 2D-Format ausgegeben wird, erfolgt eine Analyse. Beruhend auf den Ergebnissen wird eine eigene Skizze erstellt, die den Scan hinreichend genau abbildet. Dazu wird eine innerhalb dieser Arbeit entwickelte Methode eingesetzt, welche die Linien des gescannten Konturzugs auf Relevanz für weitere Untersuchungen prüft. Der Vergleich der Steigung jeder Linie mit einem festgelegten Toleranzbereich entscheidet über ihre weiterführende Verwendung. Anschließend werden die Endpunkte der verbliebenen Linien ausgelesen und unter Anwendung einer festgelegten Methodik auf die wesentlichen reduziert. Die finale Skizzenerstellung erfolgt durch Kombination der Koordinaten der verbliebenen Punkte, die als Ergebnis einen Polygonzug in Stufenform liefert. Danach wird die Separierung in die Einzelkomponenten (Halter und Werkzeug) vorgenommen, indem die jeweiligen Linien den zugehörigen Skizzen hinzugefügt werden. Darüber hinaus finden die Erstellung von Rotationskörpern und deren Aufbereitung für die Werkzeugdatenbank statt. Abschließend werden relevante Parameter in ein Tabellenblatt exportiert. Das durch diese Methode erstellte Werkzeug kann anschließend in einer Werkzeugdatenbank abgespeichert werden und steht für eine CAM-Planung zur Verfügung.

Die experimentelle Durchführung erfolgte anhand einer Versuchsreihe von fünf verschiedenen Durchmessern eines Schaftfräsers zu jeweils drei Versuchen. Die verwendete Methodik zur Skizzenerstellung liefert grundsätzlich die erwarteten Ergebnisse. Einzelne Versuche ergaben jedoch minimale Abweichungen, welche durch Nachbesserungen zum erwarteten digitalen Werkzeugmodell führten.

Abstract

The present thesis deals with the optimization of a process of the CAD/CAM/CNC process chain of the smartfactory@tugraz at the University of Technology in Graz. The method for transferring physical tools into digital tool models and the subsequent integration into a tool management software are the initial situation for the scientific investigations.

The aim of this thesis is the automated generation of digital tool models for end mills. Starting from a tool scan of the complete tool by means of a presetter, the central task of this work is the preparation of the tool model for a CAM planning as well as the integration into a tool management software. For this purpose, the tool scan has to be separated into its individual components and characteristic values have to be defined for the integration into the Siemens MRL tool management software. For this purpose, the programming interface NX Open in combination with the software Siemens NX is used.

Based on the result of the tool scan, which results in a contour sketch of the complete tool in 2D format, an analysis is performed. As a result, an own sketch is created, which represents the scan with sufficient accuracy. For this purpose, a method developed within this thesis is used, which checks the lines of the scanned contour train for relevance for further investigations. The comparison of the slope of each line with a defined tolerance range decides on its further use. Subsequently, the end points of the remaining lines are read out and reduced to the essential ones using a specified methodology. The final sketching is done by combining the coordinates of the remaining points, which gives as a result a polygon course in step form. Then, the separation into the individual components (holder and tool) is performed by adding the relevant lines to the corresponding sketches. Furthermore, the creation of rotational bodies and their preparation for the tool management software is performed. Finally, relevant parameters are exported to a spreadsheet. The tool created by this method can be stored in a tool management software and is available for CAM planning.

The experimental performance was based on a test series of five different diameters of an end mill at three trials each. The methodology used for sketch generation basically delivers the expected results. However, individual tests resulted in minimal deviations, which led to the expected digital tool model by rework.

Inhaltsverzeichnis

Abbildungen

Tabellen

Abkürzungen

API	Application Programming Interface
CAD	Computer Aided Design
CAM	Computer Aided Manufacturing
CLDATA	Cutter Location Data
CNC	Computerized Numerical Control
CSW	Coordinate System Workpiece
DMC	DataMatrix-Code
DNC	Direct Numerical Control
DXF	Drawing Exchange Format
GUI	Graphical User Interface
IDE	Integrated Development Environment
IGES	Initial Graphics Exchange Specification
MCS	Mounting Coordinate System
MRL	Manufacturing Resource Library
NC	Numerical Control
PCS	Primary Coordinate System
PLM	Product Lifecycle Management
RFID	Radio Frequency Identification
SPS	Speicherprogrammierbare Steuerung
STEP	Standard for the Exchange of Product Model Data
VB	Visual Basic

1 Einleitung

Aufgrund der Möglichkeit zur intelligenten Vernetzung aller produktionsrelevanten Elemente, sowohl innerhalb eines Unternehmens als auch über die Unternehmensgrenzen hinaus, findet in der Industrie aktuell ein Wandel statt. Vielfach angestrebte Ziele einer modernen Produktionsstätte sind die Erreichung fehlerfreier Produktionsprozesse, die Optimierung der Bearbeitungszeiten sowie eine generell verbesserte Effizienz des gesamten Produktionsprozesses. Der permanente Datenaustausch bietet Unternehmen nicht nur unzählige Möglichkeiten, sondern führt auch zu großen Herausforderungen. Oftmals stellt sich der Umgang mit enormen Datenmengen als komplex und kompliziert dar, sodass Optimierungspotentiale nicht vollständig genutzt werden können. Vor allem Simulationen gelten aufgrund der steigenden Rechnerleistung als zukünftige Schlüsseltechnologie für die Produktentwicklung und Produktionsplanung. Hierfür sind digitale Objekte erforderlich, welche die realen Objekte virtuell abbilden, sodass ein Prozess bereits im Vorhinein auf dessen Ausführbarkeit geprüft werden kann. Die Einführung von computergestützten Systemen wie beispielsweise Computer Aided Design (CAD), Computer Aided Manufacturing (CAM), Computerized Numerical Control (CNC) und Werkzeugmanagement stellt für Unternehmen, insbesondere durch die Variantenvielfalt und die letztlich gewünschte Interaktion dieser Produkte, eine herausfordernde Entscheidung dar.

1.1 Forschungsziel

In produzierenden Unternehmen entsteht durch zunehmende Produktvarianten eine gleichzeitig wachsende Werkzeugvielfalt. Dies bringt sowohl physisch als auch digital einen zusätzlich belastenden Verwaltungsaufwand mit sich, welcher neben den operativen Tätigkeiten bewältigt werden muss. Übersteigt der Aufwand die verfügbaren Ressourcen, können entdeckte Verbesserungspotentiale nicht mehr umgesetzt werden. Dabei kommt vor allem der Bereitstellung von digitalen Objekten für eine CAM-Simulation eine entscheidende Rolle zu. Nur durch exakte virtuelle Abbildungen und deren Austausch über Schnittstellen ohne Informationsverlust kann eine Simulation ihre volle Leistungsfähigkeit erreichen. Dazu wird in dieser Arbeit der Fokus speziell auf die Überführung eines physischen Werkzeugs in eine Werkzeugdatenbank gelegt. Das Ziel dieser Arbeit ist es, ausgehend von einem gescannten Komplettwerkzeug mittels Voreinstellgerät eine automatisierte Methode zur Generierung eines digitalen Werkzeugs zu entwickeln. Dabei ist speziell die richtige Aufbereitung für eine CAM-Simulation zu berücksichtigen. Hierzu müssen das Komplettwerkzeug in seine Einzelkomponenten zerlegt und charakteristische Größen für die Integration in die Werkzeugdatenbank festgelegt werden. Das Ergebnis soll vollständige 3D-Modelle der Einzelkomponenten des physischen Komplettwerkzeugs darstellen.

1.2 Begriffsdefinitionen

Sowohl in der Literatur als auch in der Praxis werden die Begriffe Digitaler Zwilling, Digitaler Schatten und Digitales Modell einerseits unterschiedlich interpretiert und andererseits gleichbedeutend verwendet. Bei genauerer Betrachtung unterscheiden sie sich bezüglich des Grads der Datenintegration zwischen dem physischen Objekt und dem digitalen Pendant. Aufgrund dessen wird im nachstehenden Abschnitt eine für diese Arbeit gültige Definition festgelegt. Außerdem wird beschrieben, um welchen dieser Begriffe es sich in der vorliegenden Arbeit handelt.

1.2.1 Digitaler Zwilling

Ein Digitaler Zwilling liefert ein realitätsgetreues, virtuelles Abbild eines realen Produkts oder Prozesses. Dazu werden Prozessmodelle und Simulationen verwendet.[1] Der Digitale Zwilling verfügt über einen automatischen Datenfluss zwischen einem physischen und einem digitalen Objekt in beide Richtungen. Diese bidirektionale Datenübertragung zwischen den beiden Objekten ist vollständig integriert, sodass das digitale Objekt als kontrollierende Instanz des physischen Objekts angesehen werden kann. Eine Veränderung des physischen Objekts verursacht automatisch eine Änderung des digitalen Objekts. Analog dazu kann der umgekehrte Datenfluss zu einer Veränderung des physischen Objekts führen.[2] In Abbildung 1 wird dies veranschaulicht.

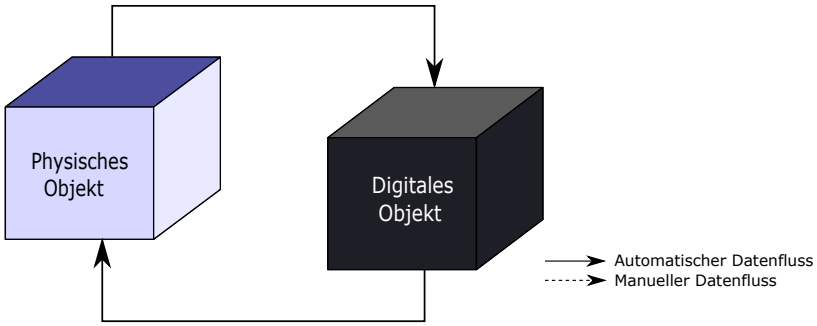


Abbildung 1: Definition Digitaler Zwilling
Quelle: Kritzinger et al. (2018), (leicht modifiziert).

1.2.2 Digitaler Schatten

Ein Digitaler Schatten ist ein hinreichend genaues Abbild der Prozesse einer entsprechenden physischen Einheit. Er dient zur Beschreibung von Daten in der Entwicklung, der Produktion sowie in anderen anliegenden Bereichen. Der Digitale Schatten überführt den realen Prozess in die virtuelle Welt.[3] Er speichert alle Daten und spiegelt die Gestalt sowie den vergangenen, aktuellen und zukünftigen Status des physischen Gegenstücks hinreichend genau wider. Die Daten des Digitalen Schattens sind in einer Datenbank gespeichert und werden von

[1] Vgl. Bauernhansl et al. (2016), S. 23.
[2] Vgl. Kritzinger et al. (2018).
[3] Vgl. Schuh et al. (2016), S. 48.

einer Software verwaltet. Diese Art der Datenintegration verbessert die Generierung von aussagekräftigen Informationen.[4] Der Hauptzweck eines Digitalen Schattens dient der Entscheidungsfindung und führt damit zur Steigerung der Verwendbarkeit und Effizienz von physischen Ressourcen.[5]

Der Grad der Datenintegration eines Digitalen Schattens ist im Vergleich zum Digitalen Zwilling in einem weniger ausgeprägten Maß verfügbar. Der Digitale Schatten verfügt über einen automatischen Datenaustausch vom physischen zum digitalen Objekt. Das bedeutet, dass das digitale Objekt immer auf dem aktuellen Stand des physischen Objekts gehalten wird. Der Datenaustausch vom digitalen zum physischen Objekt erfolgt durch einen manuellen Vorgang.[6] Der Grad der Datenintegration eines Digitalen Schattens wird in Abbildung 2 dargestellt.

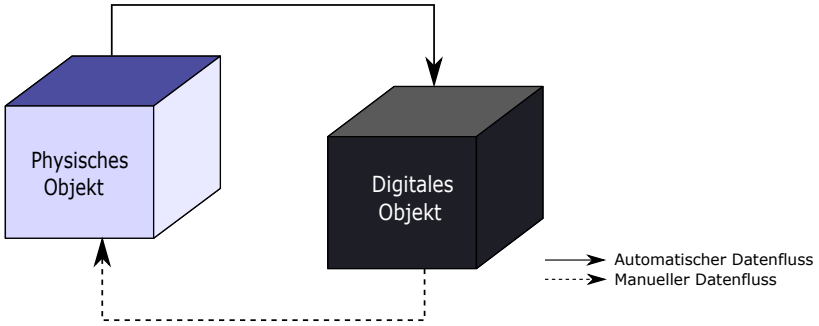


Abbildung 2: Definition Digitaler Schatten
Quelle: Kritzinger et al. (2018), (leicht modifiziert).

1.2.3 Digitales Modell

Ein Digitales Modell ist die digitale Abbildung eines geplanten oder bereits existierenden Objekts, ohne dabei in irgendeiner Form einen automatischen Datenaustausch zwischen dem digitalen und dem physischen Objekt zu verwenden. Das Digitale Modell kann ein umfassendes Abbild des physischen Objekts durch mathematische Modelle darstellen. Die Daten von bereits realisierten physischen Systemen können für eine Weiterentwicklung

[4] Vgl. Tao et al. (2019), S. 21.
[5] Vgl. Dalmolen et al. (2012).
[6] Vgl. Kritzinger et al. (2018).

dieser Modelle verwendet werden. Der Datenaustausch erfolgt dabei stets manuell und entsprechende Änderungen werden nicht automatisch aktualisiert.[7] In Abbildung 3 wird der manuelle Datenaustausch eines Digitalen Modells abgebildet.

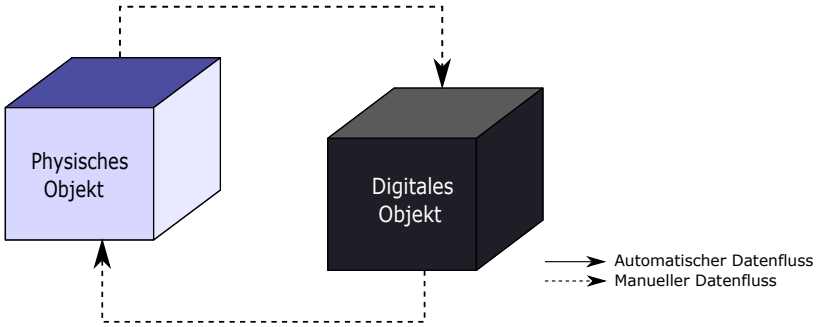


Abbildung 3: Definition Digitales Modell
Quelle: Kritzinger et al. (2018), (leicht modifiziert).

Nachdem die Begriffe eindeutig nach deren Grad der Datenintegration definiert wurden, kann weiterführend eine Einteilung für das Objekt, welches im Zuge der vorliegenden Arbeit erstellt wird, getroffen werden. Dieses verfügt weder über einen automatischen Datenaustausch vom physischen zum digitalen Objekt noch vom digitalen zum physischen. Schlussfolgernd ist für die weiteren Untersuchungen von einem Digitalen Modell auszugehen.

[7] Vgl. Kritzinger et al. (2018).

2 Stand der Technik einer CAD/CAM/CNC-Prozesskette

In diesem Kapitel wird der Stand der Technik einer CAD/CAM/CNC-Prozesskette nähergebracht und ein Überblick des Gesamtprozesses vermittelt. Im Zuge der Beschreibung der einzelnen Elemente der Prozesskette werden verschiedene Varianten von CAD/CAM-Systemen vorgestellt und unterschiedliche Simulationsvarianten für eine Fertigung behandelt. Außerdem werden die Gründe für die Einführung einer Werkzeugdatenbank aufgezeigt sowie die Erstellung und der Zusammenbau von digitalen Werkzeugen untersucht. Zum Abschluss wird das physische Werkzeugmanagement behandelt und die in dieser Arbeit betrachtete CAD/CAM/CNC-Prozesskette vorgestellt.

Die CAD/CAM/CNC-Prozesskette verfolgt das Ziel einer durchgängigen Informationsweitergabe von der Entwicklung eines Produkts bis zu dessen Fertigung. Die Rückführung von Fertigungsdaten ist für eine kontinuierliche Prozessoptimierung entscheidend. Dabei ist auf die Vollständigkeit und den Umfang der Daten sowie auf eine detaillierte Abstimmung der Prozesse zueinander zu achten. Infolgedessen kann der Planungsprozess reibungslos und schließlich auch die Fertigung des Bauteils problemlos stattfinden. Zusätzlich muss die CAD/CAM/CNC-Prozesskette in die organisatorischen Prozesse eingebettet werden. Somit werden die Verfügbarkeit von benötigten Ressourcen sowie die Datenübertragung bis in den Shopfloor berücksichtigt.[8]

[8] Vgl. Dietrich/Richter (2020), S. 426.

Grundsätzlich ist die CAD/CAM/CNC-Prozesskette in drei Teile zu gliedern, wie in Abbildung 4 veranschaulicht, nämlich in den vorbereitenden, den planenden und den ausführenden Teil. Die Vorbereitung, welche die Konstruktion und Entwicklung umfasst, hat die Aufgabe, ein Digitales Modell des Produkts zu erstellen und an die planende Stelle zu übergeben. Die Herausforderung der Fertigungsplanung liegt in der Definition der Bearbeitungsschritte zur Erstellung eines fertigen Produkts. Der abschließende Teil ist für die Durchführung der Bearbeitungsschritte an der CNC-Maschine verantwortlich. Alle drei Teile der Prozesskette haben einen entscheidenden Einfluss auf das Endergebnis. Nur durch eine gute Abstimmung der Prozessschritte ist eine effiziente und fehlerfreie Produkterstellung möglich.

Abbildung 4: Gliederung einer CAD/CAM/CNC-Prozesskette
Quelle: In Anlehnung an Hehenberger (2011), S. 122.

2.1 Computer Aided Design

CAD, auch rechnergestütztes Konstruieren genannt, ist die Unterstützung durch den Computer bei sämtlichen Konstruktionstätigkeiten eines Produkts. Bei der Einführung von CAD war das Hauptaugenmerk auf die Erzeugung von Geometriedaten gerichtet. Neben dieser Hauptaufgabe beschäftigt sich CAD mit der Berechnung, der Simulation und der Informationsgewinnung, um ein fertigungsnahes 2D- oder 3D-Modell zu erstellen. Bei der Konstruktion und Entwicklung technischer Produkte wird zwischen einer Neuentwicklung und einer Änderung eines existierenden Produkts nicht unterschieden. Unabhängig von der Einstufung des zu bearbeitenden Produkts fallen alle diese Tätigkeiten unter den Begriff CAD. Die Darstellung eines Produkts in einem 3D-Modell im Vorhinein bietet der Anwenderin oder dem Anwender die Möglichkeit einfach Änderungen vorzunehmen. Heutzutage reichen die geometrischen Daten eines Produkts nicht mehr aus, vielmehr werden numerische und

technologische Daten mit in das CAD-System eingebunden.[9] Diesbezüglich sollen bereits in der Produktentwicklung mögliche Fehler beim Kunden ausgeschlossen werden.

Assoziativität in der Konstruktion ist die Erstellung von Abhängigkeiten zu vorangehenden Arbeitsschritten. Dem Anfang dieses Vorgehens kommt die größte Bedeutung zu. Davon ausgehend werden weitere Verknüpfungen erstellt, sodass bei Änderungen alle darauffolgenden Verbindungen angepasst werden. Zu den größten Änderungen führen auch Anpassungen am Grundmodell. Die vollständig assoziative Konstruktion kann durch eine modulübergreifende Verknüpfung ergänzt werden. Dies führt zu einer Aktualisierung aller verknüpften Modelle, solange keine Trennung der Abhängigkeiten vollzogen wird.

Unter Parametrisierung versteht man die Speicherung von eingegebenen Werten in einer Software als Parameter. Diese können jederzeit geändert oder verwendet werden. Mit deren Hilfe ist eine klare, vollständige Definition der Konstruktion zu erstellen.[10]

Aktuell geht der Trend in der Konstruktion zu parametrisch-assoziativen Beziehungen. Diese ermöglichen eine erhebliche Zeiteinsparung bei Modifikationen im Nachhinein. Durch die Änderung eines Parameters und der Abhängigkeiten passt sich das Modell an die neue Eingabe an. Dieses Vorgehen ist bei einer großen Variantenvielfalt von Produkten ein praktisches Werkzeug. Heutige CAD-Systeme bieten außerdem die Möglichkeit, mehrere Teile gleicher Gestalt mit verschiedenen Parametern auf Knopfdruck zu erzeugen. Zusammenfassend kann festgehalten werden, dass die angewandte Parametrik in den CAD-Systemen zur Erleichterung bei der Verwendung ähnlicher Modelle, bei Modifikationen und bei großer Variantenvielfalt führt.[11] Bekannte CAD-Systeme, die sich im Maschinenbau etabliert haben, sind unter anderem Siemens NX, CATIA und PTC Creo. Im Rahmen dieser Arbeit wird Siemens NX verwendet. In diesem CAD-System werden parametrisch-assoziative Beziehungen mit Hilfe von Expressions erstellt. Diese definieren durch Formeln die Charakteristiken von Bauteileigenschaften.[12]

[9] Vgl. Hehenberger (2011), S. 120.
[10] Vgl. Scheidegger (2016), S. 190.
[11] Vgl. Kief et al. (2017), S. 705.
[12] Vgl. Siemens PLM Software (2021a).

2.2 Computer Aided Manufacturing

CAM, auch rechnergestützte Fertigung, befasst sich mit der Planung und Erstellung von Fertigungsabläufen. Hierzu zählen unter anderem die Erstellung des Bearbeitungsplans, die Definition von Spannmitteln und Werkzeugen sowie die finale Generierung eines Numerical Control (NC)-Codes durch den Postprozessor.

Der Grundgedanke eines CAM–Systems, die Erstellung eines NC-Programms, wird durch weitere Funktionen ergänzt. Die Anwenderin beziehungsweise der Anwender wird bereits beim Import des CAD-Modells vom CAM-System unterstützt. Dazu werden Standarddateiformate wie Initial Graphics Exchange Specification (IGES), Standard for the Exchange of Product Model Data (STEP) oder Drawing Exchange Format (DXF) vom CAM-System unterstützt.[13] Der konventionelle CAM-Planungsprozess beginnt mit einem Werkstück und Produktionsdaten. Dabei werden die zu bearbeitenden Flächen und Konturen des Werkstücks selektiert und mit einer Bearbeitungsoperation verknüpft. Beruhend auf diesen Operationen werden ein passendes Werkzeug und Technologieparameter, wie Vorschub und Drehzahl, festgelegt. Diese Entscheidung wird basierend auf Erfahrungen sowie Herstellerinformationen getroffen. Des Weiteren können die Werkzeugbahnen durch eine Simulation des Materialabtrags untersucht werden.[14] [15]

Einem CAM-System sind die exakten Spezifikationen der gewünschten Werkzeugmaschine nicht bekannt. Es erstellt ein abstraktes Bearbeitungsprogramm eines Rohteils im Cutter Location Data (CLDATA)-Format. Das CLDATA-Format ist ein allgemein gültiges Ausgabeformat für Werkzeugbahnen, das unabhängig von einer Werkzeugmaschine oder deren Steuerung ist.[16] Ein Postprozessor hat die Aufgabe, dieses Bearbeitungsprogramm vom CLDATA-Format in die Maschinensprache einer Werkzeugmaschine zu übersetzen, damit es für diese lesbar ist. Der Output eines Postprozessors wird NC-Code oder oftmals G-Code genannt. Der spezifische NC-Code, der ausführbare Programmcode, ist nur an die ausgewählte Werkzeugmaschine angepasst und kann nicht für beliebige Maschinen eingesetzt werden. Der Postprozessor vollführt sozusagen einen Übersetzungsvorgang.[17]

[13] Vgl. Dietrich/Richter (2020), S. 420.
[14] Vgl. Brecher et al. (2011).
[15] Vgl. Reinhart (2017), S. 344.
[16] Vgl. Kief et al. (2017), S. 614.
[17] Vgl. Dangelmaier/Gausemeier (2019), S. 8.

Im Anschluss kann das erzeugte NC-Programm auf Kollisionen überprüft werden. Die benötigten Mittel für eine Kollisionsüberprüfung sind unter anderem die 3D-Modelle des Komplettwerkzeugs, der Werkzeugmaschine und des Werkstücks.[18] Abbildung 5 veranschaulicht dies exemplarisch.

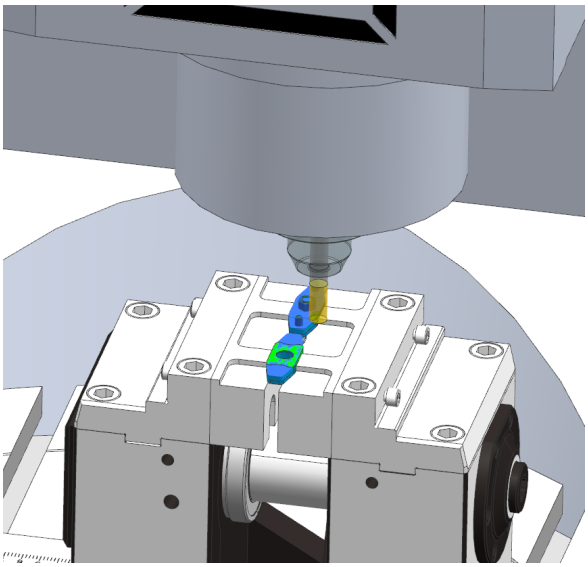


Abbildung 5: Simulation einer Fräsbearbeitung
Quelle: Schmid et al. (2020).

2.2.1 Kopplungsvarianten von CAD/CAM-Systemen

Heutzutage werden eigenständige CAD- und CAM-Systeme zu modernen CAD/CAM-Systemen zusammengefasst. Dies reduziert das Risiko von Prozessfehlern, weil der Zugriff auf dasselbe Datenmodell sichergestellt ist. Dadurch kann ein möglicher Informationsverlust beim Import von CAD-Modellen im Standarddateiformat vermieden werden. Das im CAD-System erzeugte Modell kann ohne Datenverlust für die CAM-Programmierung verwendet werden.[19]

[18] Vgl. Oehler (2016), S. 5.
[19] Vgl. Kief et al. (2017), S. 707.

Bei den Kopplungsvarianten wird zwischen drei verschiedenen Typen unterschieden:[20] [21]

- Gekoppeltes CAD/CAM-System
- Integriertes CAD/CAM-System
- STEP-NC basiertes CAD/CAM-System

Die Beschreibung jeder einzelnen Variante findet in den nachfolgenden Unterabschnitten statt. Bildlich sind in Abbildung 6 die Kopplungsvarianten gegenübergestellt.

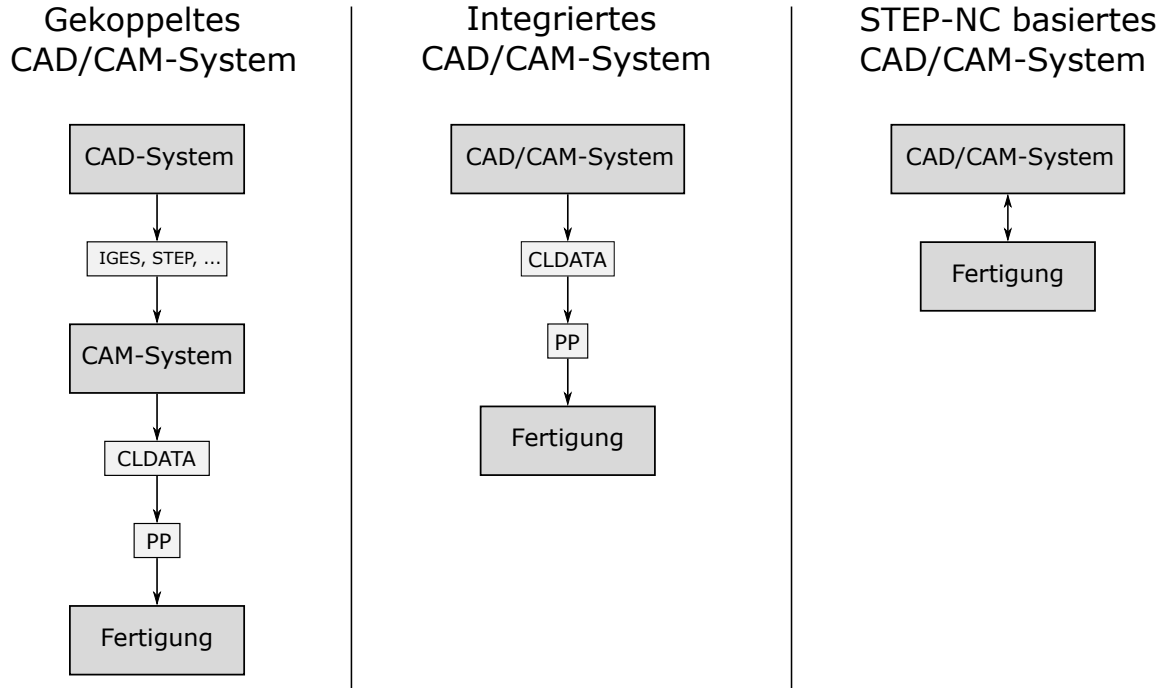


Abbildung 6: Kopplungsvarianten von CAD/CAM-Systemen
Quelle: In Anlehnung an Hehenberger (2011), S. 147.

[20] Vgl. Kief et al. (2017), S. 715.
[21] Vgl. Kretzschmann (2010), S. 16.

2.2.1.1 Gekoppeltes CAD/CAM-System

Das eigenständige CAM-System kann mit gängigen 3D-Datenformaten umgehen und erstellt unter Auswahl der Rohteilkonturen durch die Benutzerin oder den Benutzer die Bearbeitungsoperationen. Ein solches CAM-System wird für die Erstellung komplexer CNC-Programme eingesetzt, bietet jedoch keine Möglichkeit, zurück in das CAD-System zu wechseln. Es besteht keine direkte Verbindung zwischen den beiden Systemen, sodass kein Datenfluss vom CAM-System zum CAD-System verfügbar ist. Das 3D-Modell muss im CAD-System geändert und der Prozess erneut durchlaufen werden.[22]

2.2.1.2 Integriertes CAD/CAM-System

Das integrierte CAD/CAM-System bietet den Vorteil, dass das 3D-Modell in derselben Software verwendet werden kann. Somit ist die Konvertierung in ein anderes Dateiformat nicht notwendig, wodurch die Datenübertragung erleichtert wird. Dies führt zusätzlich zu einer verbesserten Datenkonsistenz und es gehen keine Informationen durch die Umwandlung in ein anderes Datenformat verloren. Die Fertigungsinformationen können direkt aus dem 3D-Modell entnommen werden.[23]

2.2.1.3 STEP-NC basiertes CAD/CAM-System

Das Ausgangsmodell für ein STEP-NC basiertes CAD/CAM-System ist das CAD-Modell im standardisierten STEP-Format. Um auch die Verwendung eines einheitlichen Datenformats für Bearbeitungsschritte sicherzustellen, wird auf das international standardisierte Datenmodell STEP-NC zurückgegriffen. Bei diesem handelt es sich grundsätzlich um eine Erweiterung des STEP-Modells um Bearbeitungsbahnen. Die für die Bearbeitung notwendigen Daten der Werkzeugmaschine und die Informationen für einen Übersetzungsvorgang werden getrennt vom STEP-NC-Modell in einer separaten Datei gespeichert. Das STEP-NC-Modell beinhaltet eine Bearbeitungsfolge, welche für alle Werkzeugmaschinen gleich ist. Dieses neutrale Ausgangsmodell wird

[22] Vgl. Kief et al. (2017), S. 715 f.
[23] Vgl. Kief et al. (2017), S. 716.

anschließend durch einen Übersetzungsvorgang in ein NC-Programm für die jeweilige Werkzeugmaschine überführt. Hierzu werden die für den Postprozessor benötigten Daten in einer gesonderten Datei in tabellarischer Form abgelegt. In einem finalen Schritt wird der NC-Code für die Steuerung der jeweiligen Werkzeugmaschine erstellt. Beim STEP-NC basierten CAD/CAM-System besteht die Option der Rückführung eines NC-Programms in ein STEP-NC-Modell. Dadurch kann ein neues NC-Programm für eine andere Werkzeugmaschine generiert werden.[24]

2.2.2 CAM-Simulationsmethoden

Für die Sicherstellung der Fehlerfreiheit eines NC-Programms an einer Werkzeugmaschine werden Simulationen in der Vorbereitungsphase durchgeführt. Mit Hilfe dieser Simulationen kann eine erhöhte Prozesssicherheit während der Fertigung erzielt werden.[25] Dieser Unterabschnitt behandelt die verschiedenen Arten von CAM-Simulationsmethoden. Außerdem wird auf die Vor- und Nachteile der Simulationen eingegangen. Die CAM-Simulationsmethoden können wie folgt eingeteilt werden:

- CAM-Simulation ohne Steuerungsnachbildung
- CAM-Simulation mit emulierter Steuerung
- CAM-Simulation mit simulierter Steuerung

Durch die Integration von 3D-Modellen der Werkzeugmaschine, des Werkzeugs und der eingesetzten Spannmittel können mögliche Kollisionen an der Werkzeugmaschine beim Bearbeitungsprozess im Vorhinein erkannt werden. Diese sind für eine aussagekräftige CAM-Simulation notwendig.[26] [27] In den folgenden Unterabschnitten wird auf die verschiedenen Simulationsmethoden näher eingegangen.

[24] Vgl. Brecher et al. (2013).
[25] Vgl. Westkämper/Löffler (2016), S. 242.
[26] Vgl. Hofmann (2017), S. 83.
[27] Vgl. Deng et al. (2018).

2.2.2.1 CAM-Simulation ohne Steuerungsnachbildung

Bei der CAM-Simulation ohne Steuerungsnachbildung werden die Bearbeitungsbahnen des Werkzeugs im CLDATA-Format generiert. Wie bereits in Abschnitt 2.2 erwähnt, handelt es sich dabei um ein maschinenneutrales Dateiformat, welches von der CNC-Steuerung nicht gelesen werden kann. Diese Art der Simulation dient der reinen Beschreibung von Werkzeugbahnen. Sie werden vor allem durch die beiden Maße des Komplettwerkzeugs in Abbildung 7 definiert. Je nach Ausführung des Werkzeugs ergänzen aber auch beispielsweise die Maße einer Rundung oder einer Fase die Definition. Außerdem besteht die Möglichkeit, dem Werkstück ein Spannmittel zuzuordnen und das Komplettwerkzeug inklusive Werkzeughalter darzustellen. Bei dieser Simulation können Fehler durch den Postprozessor nicht erkannt werden.[28]

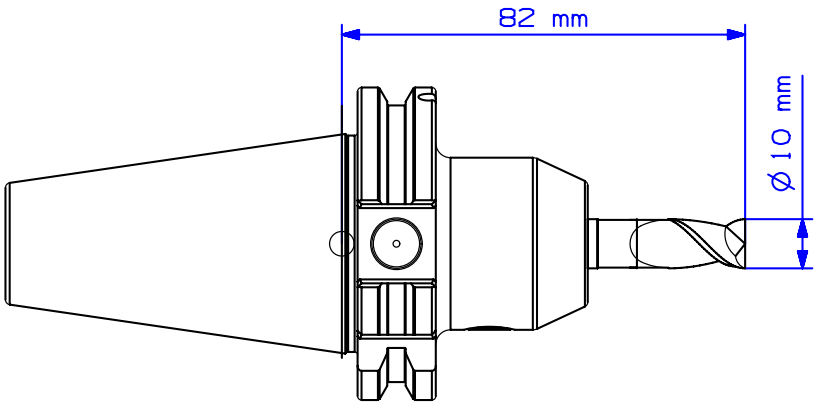


Abbildung 7: Relevante Maße zur Beschreibung von Werkzeugbahnen
Quelle: Schmid (2021).

Diese Simulationsmethode ist für eine erste Kontrolle der Bearbeitungsschritte geeignet. Für eine Untersuchung und zur genaueren Laufzeitbestimmung des Bearbeitungsprogramms sollte auf Simulationen des NC-Codes zurückgegriffen werden. Somit können auch Fehler durch den Übersetzungsvorgang des Postprozessors bereits bei der Simulation erkannt werden.[29] Die wichtigsten Vor- und Nachteile der Simulation ohne Steuerungsnachbildung sind in Tabelle 1 zusammengefasst.

[28] Vgl. Oehler (2016), S. 8.
[29] Vgl. Kief et al. (2017), S. 649.

Tabelle 1: Vor- und Nachteile einer CAM-Simulation ohne Steuerungsnachbildung
Quelle: Kief et al. (2017), S. 650.

Vorteile	**Nachteile**
Simulation im frühen Stadium möglich	Fehler durch den Postprozessor bleiben unbemerkt
Festlegung auf eine Werkzeugmaschine nicht notwendig	Randbedingungen der Kinematik bleiben unberücksichtigt
Auswahl eines Spannmittels nicht erforderlich	Keine Simulation des klassischen NC-Codes

2.2.2.2 CAM-Simulation mit emulierter Steuerung

Die CAM-Simulation mit emulierter Steuerung ist eine Erweiterung der zuvor beschriebenen Simulation. Bei dieser Simulationsmethode wird die Übersetzung des CLDATA-Formats in einen maschinenlesbaren Code durch den Postprozessor durchgeführt. Die Emulation beinhaltet den NC-Code, Nullpunktverschiebungen, Transformationen sowie den spezifischen Sprachumfang der Steuerung. Für die Berechnung des Zeitablaufs des NC-Programms können im Vorhinein individuelle Verzögerungs- und Beschleunigungsbefehle für die jeweiligen Achsen eingepflegt werden. Diese Laufzeitberechnung anhand des NC-Codes ist als nicht exakt anzusehen, da die Bewegungsdynamik der Maschine nicht berücksichtigt wird. Dennoch liefert sie sehr gute Ergebnisse, welche in der Praxis ausreichend sind. Bei zu hohen Lasten können Begrenzungsbefehle eine Überlastung verhindern, sodass eine Abweichung der Laufzeit entsteht.[30] Für die Kollisions- und Fehlerüberprüfung wird die Hinterlegung eines Modells der Werkzeugmaschine und des Komplettwerkzeugs empfohlen. Hierdurch kann der gesamte Fertigungsprozess als 3D-Simulation dargestellt werden.[31] Die 3D-Darstellung, die Zerspanungssimulation sowie die Kollisionsüberprüfung wurden jahrelang durch die Rechnerleistung begrenzt. Durch die stetig steigende Leistung von Computern wurde dieser hindernde Faktor beseitigt und die CAM-Simulation mit emulierter Steuerung findet ihre Anwendung auf leistungsstarken Laptops und PCs.

[30] Vgl. Altintas/Tulsyan (2015).
[31] Vgl. Oehler (2016), S. 9 f.

In Tabelle 2 sind die Vor- und Nachteile einer CAM-Simulation mit emulierter Steuerungsnachbildung zusammengefasst. Durch die Tatsache, dass die Bearbeitungsfolge für die Simulation nicht im CLDATA-Format, sondern im maschinenlesbaren Code simuliert wird, ergeben sich deutliche Vorteile im Vergleich zur CAM-Simulation ohne Steuerungsnachbildung.

Tabelle 2: Vor- und Nachteile einer CAM-Simulation mit emulierter Steuerung
Quelle: Kief et al. (2017), S. 650.

Vorteile	**Nachteile**
Darstellung unterschiedlicher Maschinen und Steuerungen	Steuerungsnachbildung komplexer Werkzeugmaschinen nicht ausreichend
Kollisionserkennung	Zeitanalyse basiert auf eingepflegten Verzögerungs- und Beschleunigungswegen
Protokollierung auftretender Fehler	
Visualisierung des Fertigungsprozesses und der Werkzeugmaschine	

2.2.2.3 CAM-Simulation mit simulierter Steuerung

Der steigende Funktionsumfang einer CNC-Steuerung erschwert die Abbildung aller Funktionen in der CAM-Simulation mit emulierter Steuerung. Die CAM-Simulation mit simulierter Steuerung kann die reale Maschinensteuerung in der virtuellen Umgebung mit vollem Funktionsumfang abbilden. Es handelt sich um eine virtuelle Maschinensteuerung.[32] Diese beinhaltet alle Funktionen der realen Steuerung, welche am PC ausgeführt werden können. Diese virtuelle CNC-Steuerung wird mit den realen Daten betrieben und ermöglicht die Bedienung über die Bedienoberfläche, wie sie auch an der realen Steuerung aufgebaut ist.[33] In Abbildung 8 ist eine virtuelle Maschinensteuerung dargestellt.

[32] Vgl. Denkena/Ammermann (2010).
[33] Vgl. Kief et al. (2017), S. 651.

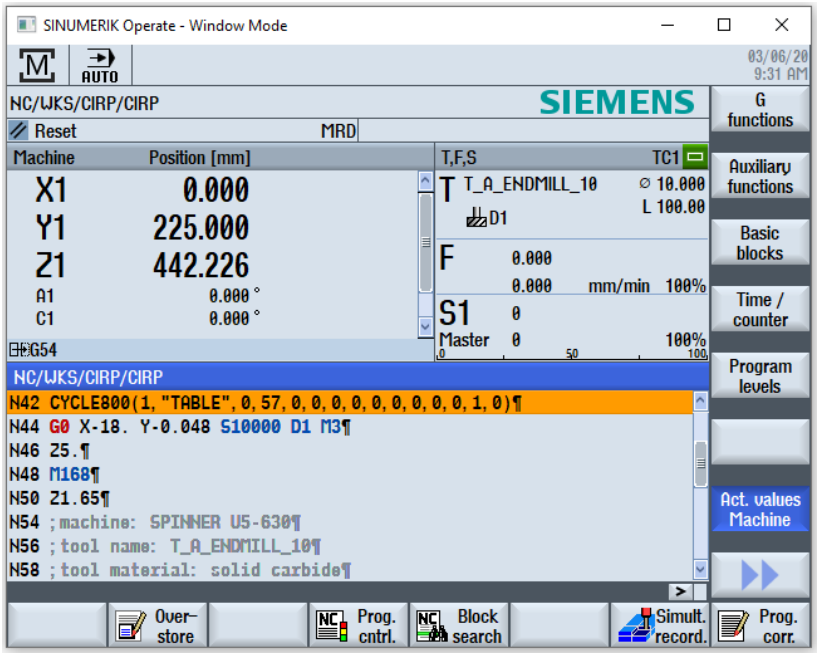


Abbildung 8: Oberfläche einer virtuellen Maschinensteuerung
Quelle: Schmid et al. (2020).

Diese Art der Simulation hat eine hohe Genauigkeit und bietet daher höchste Prozesssicherheit. Der NC-Code wird durch die virtuelle Steuerung auf Syntaxfehler geprüft.[34] Durch den größeren Funktionsumfang sinkt die Performance gegenüber der CAM-Simulation mit emulierter Steuerung. Im Gegenzug ist jedoch eine exakte Laufzeitberechnung durchführbar. In Tabelle 3 werden die Vor- und Nachteile der CAM-Simulation mit simulierter Steuerungsnachbildung aufgezählt.

[34] Vgl. Oehler (2016), S. 11.

Tabelle 3: Vor- und Nachteile einer CAM-Simulation mit simulierter Steuerung
Quelle: Kief et al. (2017), S. 650.

Vorteile	**Nachteile**
Voller Funktionsumfang der CNC-Steuerung	Schlechtere Performance
Bedienung identisch mit realer Steuerung	Umständliches Installationsverfahren
Simulation verhält sich exakt zur realen Fertigung	
Exakte Laufzeitmessung	
Prüfung des NC-Codes auf korrekte Syntax	

Damit eine Simulation möglichst realistisch wird, ist eine CAM-Simulation mit emulierter beziehungsweise simulierter Steuerung zu bevorzugen. Dabei werden der NC-Code, welcher später auf der Werkzeugmaschine ausgeführt wird, simuliert und mögliche Fehler durch den Übersetzungsvorgang des Postprozessors bereits in der Simulationsphase erkannt.[35]

2.2.3 CAM Automatisierungsmethoden

Moderne CAM-Systeme helfen der Anwenderin oder dem Anwender bei der Erstellung von Bearbeitungsstrategien, indem sie teil- oder vollautomatisierte Mechanismen für sich wiederholende Tätigkeiten anbieten. Der Grundgedanke dabei ist die Anwendung von ähnlichen Bearbeitungsschritten auf Bauteil-Features. Grundsätzlich können diese Automatisierungsmechanismen in drei Gruppen eingeteilt werden:

- Templates
- Feature-Makro-Mapping
- Programmierschnittstellen

[35] Vgl. Brillinger et al. (2019).

Die auf Templates basierende Methode ist nicht als Automatisierung anzusehen. Dennoch bietet sie eine Möglichkeit, vorgefertigte Bearbeitungsoperationen von ähnlichen, bereits erstellten Bauteilen auf neue anzuwenden. Die damit einhergehende Datenpflege der Templates ist nicht zu vernachlässigen. Bei einer großen Variantenvielfalt von Bauteilen übersteigt der Pflegeaufwand den Vorteil des Templates. Die zweite Variante ist das Feature-Makro-Mapping. Sie ordnet Bauteil-Features mit definierten Parametern mit Hilfe von Makros einer definierten Bearbeitungsabfolge sowie Werkzeugen zu. Dadurch ist eine automatisierte Erstellung der Bearbeitungsschritte für diese definierten Geometrien möglich. In Zusammenhang mit einer automatisierten Feature-Erkennung bieten aktuelle CAM-Systeme eine Möglichkeit, auch bei verschiedenen Datenformaten die Features richtig zu erkennen. Die dritte Möglichkeit stellt eine Programmierschnittstelle, auch als Application Programming Interface (API) bezeichnet, dar. Sie ermöglicht Analysen der Geometrie am CAD- oder CAM-Modell. Außerdem kann ein automatisiertes Anlegen und Parametrieren von Bearbeitungsoperationen durchgeführt werden. Durch eine Vielzahl von individuellen Anwendungsfällen führt die API zu einem hohen Pflegeaufwand.[36]

Durch die Kopplung der Bauteilgeometrie mit den Werkzeugbewegungen werden die Werkzeugwege bei einer Geometrieänderung neu erstellt. Nicht immer ist die automatisierte Erstellung der Werkzeugwege zufriedenstellend. Dann muss eine manuelle Nachbearbeitung durch die Benutzerin oder den Benutzer erfolgen. Die Erstellung von Werkzeugwegen basiert auf einem umfangreichen Bestand an Werkzeug-, Technologie-, Werkzeugmaschinen-, Werkstoff- und Spannmitteldaten. CAM-Systemen wird eine immer größere Bedeutung zuteil, da die gesamten Fertigungsabläufe in der virtuellen Umgebung analysiert und korrigiert werden können. Diese virtuelle Bearbeitung im Vorhinein bietet einen deutlichen Vorteil. Dennoch ist die Anwenderin oder der Anwender ein essentieller Bestandteil der Fertigung und ist für die erfolgreiche Durchführung der Bearbeitungsfolgen neben der CAM-Simulation ebenso bedeutend.[37]

[36] Vgl. Reinhart (2017), S. 344 ff.
[37] Vgl. Dietrich/Richter (2020), S. 422.

2.3 Computerized Numerical Control

CNC, auch computerunterstützte numerische Maschinensteuerung genannt, ist die numerische Steuerung von Werkzeugmaschinen. Moderne Maschinen sind mit einer CNC-Steuerung ausgestattet, welche die veraltete NC-Steuerung verdrängt. Die CNC-Steuerung ist ein elektronisches Eingabegerät, mit welchem eine Werkzeugmaschine bedient und gesteuert werden kann. Die Informationen von Positions-, Dreh- und Zustandssensoren werden von der Steuerung analysiert und ausgewertet. Damit kann ein ständiger Vergleich zwischen dem Ist-Zustand und dem Soll-Zustand des programmierten Befehls durchgeführt werden. Ein Computer, welcher sich in der Steuerung befindet, berechnet in Folge des Vergleichs die von der CNC-Steuerung ausgeführten Interpolations- und Korrekturbewegungen. Die Ansteuerung von mehreren Achsen gleichzeitig ist ein besonderes Kennzeichen von modernen CNC-Steuerungen.[38] Dadurch können komplexe Bearbeitungen wie die Fünf-Achs-Simultanbearbeitung realisiert werden. Die kurzen Zykluszeiten und die hohe Anzahl an zu interpolierenden Achsen erhöhen den Rechenaufwand, der eine CNC-Steuerung auch heute an ihr Limit bringen kann.[39] Jede Maschine benötigt neben der CNC-Steuerung Anpassungsprogramme, welche vom Maschinenhersteller geliefert werden. Die Speicherprogrammierbare Steuerung (SPS) verwaltet diese Programme. Es handelt sich dabei um maschinenspezifische Abläufe, die zum Beispiel den Werkzeugwechsel umfassen. Direct Numerical Control (DNC) führt zur Reduzierung von Stillstandszeiten, indem eine Trennung von programmerstellenden und programmausführenden Tätigkeiten eingeführt wird. Bei DNC handelt es sich um eine Methode zur direkten Übertragung an die Steuerungseinheit der Werkzeugmaschine über ein gemeinsames Netzwerk. Dies ermöglicht die Programmierung, Simulation, Korrektur und Optimierung von NC-Programmen an einem PC. Die Werkzeugmaschine kann während der Programmierung eines NC-Programms im Büro ein anderes Werkstück fertigen.[40]

[38] Vgl. Hehenberger (2011), S. 79.
[39] Vgl. Oehler (2016), S. 6.
[40] Vgl. Hehenberger (2011), S. 80.

2.4 Software für Werkzeugmanagement

Für die Verwaltung der Werkzeuge in der Fertigung wird eine Werkzeugdatenbank eingesetzt. Manchmal steht eine Werkzeugdatenbank bereits zur Verfügung, aber deren Nutzung wurde nicht in die bestehenden Prozesse integriert. Dabei benötigen heutige CAD/CAM-Systeme für eine aussagekräftige CAM-Simulation detailgetreue 3D-Modelle der realen Werkzeuge. Eine große Hürde stellt dabei die Integration des vorhandenen Werkzeugbestands in die Werkzeugdatenbank dar. Dieser Prozess muss sorgfältig und durchdacht durchgeführt werden, da sonst von keinem Mehrwert profitiert werden kann.[41] An die Werkzeugdatenbank werden bei der Einführung einige Anforderungen gestellt. Dabei ist zu erwähnen, dass die Vorteile einer Werkzeugdatenbank nur erfüllt werden können, wenn die Datenqualität auch den Anforderungen der Werkzeugdatenbank entspricht.

Die Werkzeugdatenbanken lassen sich in zwei Klassen einteilen. Diese sind Werkzeugdatenbanken, die vom CAD/CAM-System abhängig oder unabhängig sind. Die beiden Varianten werden in den folgenden Unterabschnitten separat betrachtet.

2.4.1 Werkzeugdatenbank abhängig vom CAD/CAM-System

Eine Werkzeugdatenbank, welche von einem CAD/CAM-System abhängig ist, kann nur in Verbindung mit einem solchen System betrieben werden. Bereits bei der Beschaffung wird eine Kombination dieser beiden Systeme erworben. Bei dieser Klasse von Werkzeugdatenbanken steht eine optimale Schnittstelle zum CAD/CAM-System im Vordergrund. Dem Shopfloor wird hingegen eine geringe Aufmerksamkeit zuteil. [42] Eine weitere Problematik stellt ein Umstieg auf ein anderes CAD/CAM-System dar. Es muss nicht nur die Integration eines neuen CAD/CAM-Systems beurteilt werden, sondern zusätzlich die Integration der abhängigen Werkzeugdatenbank.

[41] Vgl. Hofmann (2017), S. 62.
[42] Vgl. Hofmann (2017), S. 64.

2.4.2 Werkzeugdatenbank unabhängig vom CAD/CAM-System

Neben der abhängigen Werkzeugdatenbank sind auch solche am Markt verfügbar, die nicht an ein bestimmtes CAD/CAM-System gebunden sind. Diese Datenbanken orientieren sich neben der Schnittstelle zum CAD/CAM-System auch am Shopfloor gleichermaßen. Eine Herausforderung für diese Werkzeugdatenbanken stellen die vielen unterschiedlichen Möglichkeiten an Schnittstellen zu den verfügbaren CAD/CAM-Systemen dar.[43]

2.4.3 Generierung und Zusammenbau von Digitalen Werkzeugen

Grundsätzlich ist zwischen dem Komplettwerkzeug und den Einzelkomponenten zu unterscheiden. Zur Erzeugung von Werkzeugmodellen gibt es unterschiedliche Ansätze. Es wird zwischen der Generierung eines Komplettwerkzeugs mit Hilfe einer automatisierten 3D-Modellerstellung durch festgelegte Geometriedaten und dem Zusammenbau der 3D-Modelle der Einzelkomponenten unterschieden. Bei der Generierung eines Komplettwerkzeugs gibt die Anwenderin oder der Anwender geometrische Parameter in das System ein, um die Geometrie des Werkzeugs zu definieren. Auf Basis dieser Daten wird vom System ein vereinfachtes 3D-Modell des Komplettwerkzeugs erstellt.

Diese Methode bietet den Vorteil, dass kein CAD-Wissen benötigt wird, die Bedienung einfach und für die meisten Simulationen ausreichend ist. Dagegen spricht, dass nur vordefinierte Werkzeuge erstellt werden können. Es kann nicht auf spezifische Eigenschaften spezieller Werkzeuge eingegangen werden. Ein weiterer Nachteil ist die Eingabe der Werkzeuggeometriedaten, die von der Anwenderin oder vom Anwender aus dem Werkzeugdatenblatt abgelesen und anschließend in das System eingegeben werden müssen. Dabei können Fehler entstehen, die zu Folgefehlern in der Fertigung führen können.

Beim Zusammenbau von 3D-Modellen der Einzelkomponenten zum Komplettwerkzeug gibt es eine weitere Unterteilung in zwei verschiedene Möglichkeiten. Entweder werden die 3D-Modelle der Einzelkomponenten vom Werkzeughersteller zur Verfügung gestellt oder sie müssen manuell erstellt werden. Diese 3D-Modelle werden in der Werkzeugdatenbank gespeichert und können bei Bedarf von der Benutzerin oder vom

[43] Vgl. Hofmann (2017), S. 64.

Benutzer für die Zusammensetzung des Komplettwerkzeugs abgerufen werden. Die Vorteile dieser Methode sind, dass beliebige Geometrien dargestellt und die Werkzeuge exakt abgebildet werden können. Der damit einhergehende Nachteil ist der Verwaltungsaufwand der Werkzeugdatenbank sowie die Erstellung der Einzelkomponenten.[44] Ein solches Komplettwerkzeug, bestehend aus dem Halter und dem Werkzeug, ist in Abbildung 9 dargestellt.

Abbildung 9: Digitales Komplettwerkzeug
Quelle: Schmid (2021).

Für den Zusammenbau der Einzelkomponenten zu einem Komplettwerkzeug wird ein sogenannter Connectioncode verwendet. Dieser wird durch das Einfügen von Koordinatensystemen bei allen Einzelteilen gespeichert. Das System erkennt, welche Komponenten aufgrund des Connectioncodes zusammenpassen. Beim Zusammenbau werden die Koordinatensysteme der Einzelteile übereinandergelegt, sodass das Komplettwerkzeug entsteht.[45]

In der DIN 4003-1 wird festgelegt, wie diese Koordinatensysteme zueinander auszurichten sind. Dabei wird neben einem Standardkoordinatensystem Primary Coordinate System (PCS) ein werkstückseitiges Koordinatensystem Coordinate System Workpiece (CSW) und ein maschinenseitiges Koordinatensystem Mounting Coordinate System (MCS) definiert. Bei diesen Koordinatensystemen handelt es sich um rechtshändige, orthogonale kartesische Koordinatensysteme.

[44] Vgl. Kief et al. (2017), S. 654 f.
[45] Vgl. Schmid/Pichler (2020).

Das Standardkoordinatensystem dient der Lageorientierung im Raum. Das werkstückseitige Koordinatensystem wird für die Montage von Komponenten verwendet und ist auch als Sekundärkoordinatensystem bekannt. Das maschinenseitige Koordinatensystem, auch bekannt als Zusammenbau-Koordinatensystem, wird für den Zusammenbau einzelner Werkzeugkomponenten zum Komplettwerkzeug eingesetzt.[46]

2.5 Physisches Werkzeugmanagement

Die Werkzeugdatenbank erzeugt auf Basis der CAM-Programmierung ein Datenblatt, das alle notwendigen Einzelkomponenten für ein Komplettwerkzeug enthält. Dieses Datenblatt wird der Fertigung für den Zusammenbau des Komplettwerkzeugs übermittelt. Hierdurch wird sichergestellt, dass das digitale und das reale Komplettwerkzeug mit den gleichen Einzelkomponenten zusammengebaut werden. Auf dem Datenblatt kann zusätzlich ein Barcode abgebildet sein, der bei einem Werkzeugausgabesystem, welches in ein Netzwerk integriert ist, eingelesen werden kann. Nach dem Einscannen wird der Benutzerin oder dem Benutzer die exakte Position der Werkzeugkomponente im Ausgabesystem am Monitor angezeigt.[47] Ein Beispiel für ein Datenblatt eines Werkzeugs wird in Abbildung 10 dargestellt.

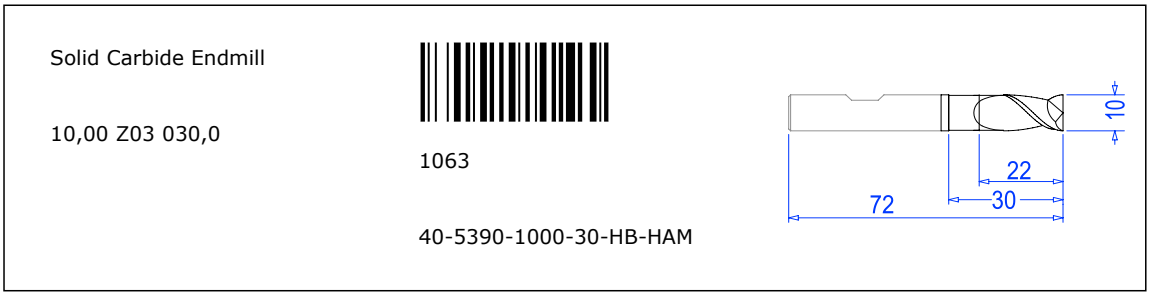


Abbildung 10: Auszug aus einer Komponentenliste
Quelle: Schmid/Pichler (2020).

[46] Vgl. DIN 4003-1 (2017), S. 6 ff.
[47] Vgl. Schmid/Pichler (2020).

Nachdem die Einzelkomponenten zu einem Komplettwerkzeug zusammengebaut wurden, wird es vermessen. Dieser Schritt ist notwendig, bevor es in der Werkzeugmaschine zum Einsatz kommt.[48] Grundsätzlich stehen zur Vermessung eines Komplettwerkzeugs verschiedene Varianten zur Verfügung. Einerseits ist das Vermessen direkt in der Werkzeugmaschine und andererseits extern an einem eigenen Voreinstellgerät möglich.[49] Beide Varianten bieten Vor- und Nachteile. Bei den internen Messsystemen kann zwischen einem Lasersystem und einem Tastsystem unterschieden werden. Beide werden in die Werkzeugmaschine implementiert und mit der Steuerung verknüpft.

Beim Lasersystem wird das Werkzeug durch einen Laserstrahl geführt. Durch einen gewissen Abschattungsgrad des Laserstrahls wird ein Signal erzeugt, welches in der Steuerung die Achspositionen ermitteln lässt. In Verwendung mit einem Referenzwert lassen sich daraus Werkzeuglänge und -radius bestimmen und automatisch in den Werkzeugspeicher eintragen. Die Vorteile eines Lasersystems sind die lokale Nähe zum Fertigungsprozess sowie die berührungslose Vermessung an unzugänglichen Stellen. Einen Nachteil stellt die Zeit der Vermessung dar, während der keine Fertigung stattfinden kann.[50]

Mit einem 3D-Taster kann die Werkzeuglänge und zusätzlich der Werkzeugdurchmesser gemessen werden. Bei der Vermessung wird ein optoelektronisches Signal durch eine interne Lichtschranke generiert, sobald das Antastelement ausgelenkt wird. Hier ergibt sich durch die prozessnahe Messung ein lokaler Vorteil. Einen Nachteil stellen die begrenzten Einsatzmöglichkeiten im Vergleich zum Lasersystem dar.[51] Bei sehr filigranen Werkzeugen ist die Vermessung mittels Taster nicht möglich, da es zum Werkzeugbruch kommen kann. Des Weiteren kann die Vermessung, ebenso wie beim Lasersystem, nicht hauptzeitparallel stattfinden.

[48] Vgl. Vogel-Heuser et al. (2017), S. 183.
[49] Vgl. Mutilba et al. (2018).
[50] Vgl. Scheidegger (2016), S. 334
[51] Vgl. Scheidegger (2016), S. 338

Die Vermessung mittels Voreinstellgerät hat den Vorteil, dass an der Werkzeugmaschine in der Zwischenzeit ein Fertigungsauftrag abgearbeitet werden kann. Der damit einhergehende Nachteil ist, dass die gemessenen Daten des Voreinstellgeräts an die Werkzeugmaschine übertragen werden müssen. In diesem Schritt müssen dem Werkzeug die gemessenen Daten des Voreinstellgeräts eindeutig zugewiesen werden.[52] [53] Zusätzlich wird die Methode, mit der das Werkzeug zu vermessen ist, von der Werkzeugdatenbank an das Voreinstellgerät übergeben.[54]

Die Entscheidung für ein Messsystem wird neben den technischen Aspekten auch von wirtschaftlichen Überlegungen beeinflusst. Das Voreinstellgerät ist im Vergleich zu den beiden internen Messmethoden um ein vielfaches teurer. Dies ist immer für die jeweilige Anwenderin oder den jeweiligen Anwender selbst abzuschätzen und von individuellen Faktoren abhängig. Je nach Art der Produktion kann sich ein Voreinstellgerät durch Verkürzung der Stillstandszeit an der Werkzeugmaschine lohnen.[55]

Zur Übertragung der Messdaten an die Werkzeugmaschine stehen vier Möglichkeiten zur Auswahl:[56]

- Manuelle Dateneingabe
- Datenübertragung mit einem Chip über die Radio Frequency Identification (RFID)-Technologie
- Beschriftung des Komplettwerkzeugs mit einem DataMatrix-Code (DMC) unter Verwendung von Etiketten
- Drahtlose Datenübertragung über ein Netzwerk mittels DNC

Die manuelle Dateneingabe stellt die unsicherste Methode dar. Dabei müssen die Messdaten beim Rüstvorgang der Werkzeugmaschine manuell eingegeben werden.

[52] Vgl. Vieira Junior et al. (2018).
[53] Vgl. Vieira Junior et al. (2011).
[54] Vgl. Reinhart (2017), S. 337.
[55] Vgl. Scheidegger (2016), S. 329
[56] Vgl. Schmid/Pichler (2020).

Durch die Datenübertragung mit einem RFID-Chip, der mit dem Werkzeug fest verbunden ist, kann die Effizienz der gesamten Produktion gesteigert werden. Es handelt sich dabei um eine Technologie zur berührungslosen Identifizierung von Objekten.[57] Dabei kann die Anzahl der Werkzeuge, das Auftreten eines Fehlers an einer Werkzeugmaschine sowie die Stillstandszeit für Vorbereitungen an der Werkzeugmaschine und für den Werkzeugzusammenbau reduziert werden.[58] Die damit verbundenen Investitionskosten dürfen nicht vernachlässigt werden. Neben den Chips an sich müssen eine Einheit zum Beschreiben des Chips am Voreinstellgerät und eine Einheit zum Auslesen der Daten an der Werkzeugmaschine vorhanden sein. Außerdem kann der Chip eine Unwucht bei rotierenden Werkzeugen hervorrufen.[59] In Abbildung 11 wird der Beschreibungsprozess eines RFID-Chips am Voreinstellgerät dargestellt. Der Chip wird in einer Nut an der Greiferrille des Werkzeughalters platziert.

Abbildung 11: Beschreibungsprozess eines RFID-Chips am Voreinstellgerät
Quelle: E. Zoller GmbH & Co. KG (2021a).

[57] Vgl. Kief et al. (2017), S. 729.
[58] Vgl. Emanuele et al. (2015).
[59] Vgl. Hofmann (2017), S. 90 f.

Die Beschriftung einer Etikette mit einem DMC ist eine kostengünstigere Lösung im Vergleich zur RFID-Technologie. Jedoch müssen auch hier Lese- und Schreibgerät am Voreinstellgerät beziehungsweise an der Werkzeugmaschine vorhanden sein. Bei dieser Methode handelt es sich um eine optische Codierung, die durch Umwelteinflüsse in Form von Schmutz oder Schmiermittel beeinträchtigt werden kann.[60] Die beschriftete Etikette wird auf eine Halterung geklebt, welche in der Greiferrille des Halters fixiert ist. In Abbildung 12 wird dies veranschaulicht.

Abbildung 12: DMC basiertes Etikettensystem
Quelle: E. Zoller GmbH & Co. KG (2021b).

Des Weiteren kann eine Übertragung der Messwerte mittels DNC erfolgen. Dabei ist die sichere und korrekte Datenübertragung entscheidend, sodass die Werte dem Werkzeug eindeutig zugeordnet sind. Das Worst-Case-Szenario ist die Zuordnung der gemessenen Daten zu einem anderen Werkzeug, wodurch es bei der Bearbeitung zu einer Kollision kommen kann.[61] Im letzten Schritt wird der NC-Code an der Werkzeugmaschine geladen.

[60] Vgl. Kern (2006), S. 17.
[61] Vgl. Schmid/Pichler (2020).

2.6 Betrachtete CAD/CAM/CNC-Prozesskette

Nachdem die Grundlagen für eine CAD/CAM/CNC-Prozesskette in den Abschnitten zuvor behandelt wurden, wird in diesem Abschnitt auf die CAD/CAM/CNC-Prozesskette der smartfactory@tugraz am Institut für Fertigungstechnik an der Technischen Universität in Graz eingegangen. Die Lernfabrik für agile und datensichere Fertigung dient als Beispiel einer nahtlosen Datenintegration in die CAD/CAM/CNC-Prozesskette und wird in Abbildung 13 veranschaulicht. Anhand dieser werden nachfolgend der Ablauf und die Schnittstellen erläutert, sowie die verwendeten Methoden beschrieben.

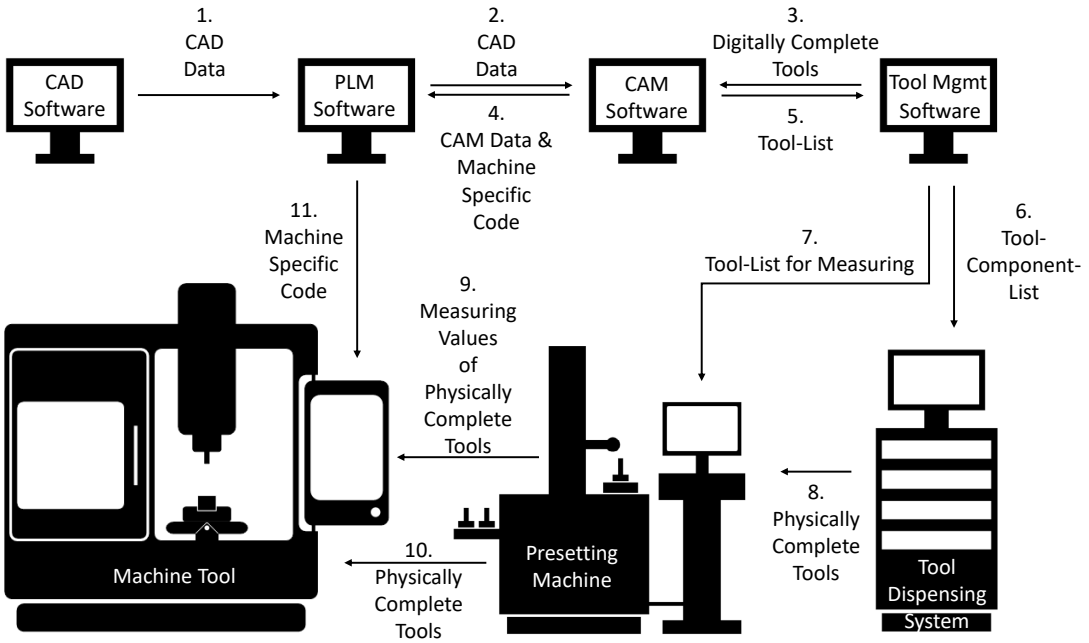


Abbildung 13: CAD/CAM/CNC-Prozesskette der smartfactory@tugraz
Quelle: Schmid/Pichler (2020).

In dieser Prozesskette wird das integrierte CAD/CAM-System Siemens NX eingesetzt. Dadurch ist eine Kommunikation mittels internem Datenformat möglich. Ein Datenverlust aufgrund eines neutralen Datenformats kann ausgeschlossen werden. In Kombination mit dem CAD/CAM-System kommt das PLM-System Siemens Teamcenter zum Einsatz. Product Lifecycle Management (PLM) ist die Verwaltung von Produkten eines Unternehmens über deren gesamten Lebenszyklus auf effizientestem Weg. Angefangen von der Idee bis hin

zum Auslaufen werden alle Daten bezüglich des Produkts in einem PLM-System verwaltet und abgelegt.[62] Dadurch sind alle Dateien vom CAD-Modell über das CAM-Modell bis hin zum NC-Code miteinander verbunden. Für eine CAM-Simulation müssen neben dem 3D-Modell des Werkstücks ebenso Digitale Modelle der Werkzeuge, der Spannmittel und der Werkzeugmaschine zur Verfügung stehen. Außerdem müssen der Postprozessor und gegebenenfalls Steuerungsnachbildungen vorhanden sein. Die Werkzeugmodelle werden in einer vom CAD/CAM-System abhängigen Datenbank mit der Bezeichnung Siemens Manufacturing Resource Library (MRL) gespeichert. Diese ist über eine interne Schnittstelle mit Siemens NX verbunden. Das Anlegen der Werkzeuge in der Werkzeugdatenbank erfolgt in der Regel bereits vor der Konstruktion des Werkstücks.

Wie in Abbildung 13 dargestellt, beginnt der Ablauf der Prozesskette mit der Erstellung eines Bauteils im CAD-System. Dieses wird anschließend im PLM-System abgespeichert und folglich in die CAM-Umgebung geladen. Danach erfolgt die CAM-Programmierung und der maschinenspezifische NC-Code wird durch den Übersetzungsvorgang des Postprozessors generiert. Die generierten Daten werden ebenfalls in der PLM-Software gespeichert. Im Zuge der finalen Generierung des NC-Codes wird dieser auf Fehler in einer CAM-Simulation mit simulierter Steuerungsnachbildung geprüft.

Außerdem wird eine Werkzeugliste von Siemens NX an die Werkzeugdatenbank gesendet. In dieser sind die Komplettwerkzeuge sowie deren Einzelkomponenten enthalten. Danach wird die Werkzeugliste für das Werkzeugausgabesystem im Shopfloor freigegeben. In diesem Fall handelt es sich um das Produkt Toolbase. Das Ausgabesystem ist mit der Werkzeugdatenbank über eine Schnittstelle verbunden, sodass beide Systeme auf dieselben Daten zugreifen können. Die Systeme verfügen über Informationen bezüglich des Werkzeugbestands und der Position der Werkzeuge. In der Werkzeugliste ist ein Barcode enthalten, der vom Werkzeugausgabesystem gescannt werden kann. Infolgedessen wird die exakte Position des jeweiligen Werkzeugs am zugehörigen Monitor angezeigt sowie die Entnahme aus der jeweiligen Schublade ermöglicht.

[62] Vgl. Stark (2015), S. 1.

Nachdem alle Werkzeuge entnommen wurden, werden sie zu einem Komplettwerkzeug zusammengebaut und am Werkzeugvoreinstellgerät vermessen. Das Voreinstellgerät mit der Bezeichnung Zoller »venturion 450« ist mit der Werkzeugdatenbank über eine Datenschnittstelle verbunden. Das Voreinstellgerät verfügt über Modi zur automatisierten Vermessung von Werkzeugen. Die Einstellungen für diese Messvorgänge sind in der Werkzeugdatenbank im Komplettwerkzeug gespeichert und können vom Voreinstellgerät abgerufen werden. Zur Übertragung der Messwerte an die Werkzeugmaschine kommt ein Etikettiersystem von Zoller zum Einsatz. Am Voreinstellgerät wird ein Etikett mit einem DMC erstellt und mit dem Komplettwerkzeug eindeutig verbunden.

Sobald der Rüstvorgang an der Werkzeugmaschine erfolgt ist, kann der NC-Code geladen werden. Dazu muss der NC-Code zuvor durch eine Browseranwendung des PLM-Systems in einem Netzwerkordner gespeichert werden. Die Werkzeugmaschine ist ebenso mit diesem Ordner verbunden. Nach der Freigabe durch das PLM-System erhält sie auf den für sie freigegebenen NC-Code Zugriff. Erst dann ist es der Maschinenbedienerin oder dem Maschinenbediener möglich, den gewünschten NC-Code zu laden und diesen nach Definition der Nullpunkte auszuführen.

3 Experimentelle Durchführung

Nachdem in Kapitel 2 die theoretischen Grundlagen bereitgestellt und die CAD/CAM/CNC-Prozesskette der smartfactory@tugraz beschrieben wurden, wird in diesem Kapitel auf die experimentelle Durchführung der vorliegenden Arbeit eingegangen. Dabei handelt es sich um die automatisierte Generierung digitaler Werkzeugmodelle für Schaftfräser auf Basis eines Werkzeugscans mittels Voreinstellgerät. Hierzu wird zuerst die Eingliederung dieser Arbeit in die betrachtete Prozesskette beschrieben. Folgend werden die benötigte Hard- und Software vorgestellt. Dabei wird vor allem auf die Programmierschnittstelle NX Open näher eingegangen. Direkt im Anschluss wird die Erstellung eines Werkzeugscans am Voreinstellgerät ausführlich erklärt. Die Methodik zur Programmerstellung und die Implementierung des 3D-Modells in die Werkzeugdatenbank werden beschrieben. Abschließend wird eine Evaluierung der Ergebnisse durchgeführt.

3.1 Eingliederung in die betrachtete Prozesskette

In diesem Abschnitt wird die Eingliederung des Prozesses dieser Arbeit in die CAD/CAM/CNC-Prozesskette der smartfactory@tugraz beschrieben. Diese wurde bereits in Abschnitt 2.6 vorgestellt.

Grundsätzlich ist dieser Prozess dem Anlegen von Werkzeugen in der Werkzeugdatenbank unterzuordnen und findet sinngemäß vor der CAM-Programmierung und CAM-Simulation statt. In dieser Arbeit wird davon ausgegangen, dass dieser Prozess unabhängig von der Konstruktion eines Produkts im Vorhinein durchgeführt wird. Er ist demnach beim Durchlaufen der CAD/CAM/CNC-Prozesskette bereits abgeschlossen. Dies wird durch den Schritt 0 in Abbildung 14 veranschaulicht, welcher vom Voreinstellgerät aus zur

Werkzeugdatenbank führt. In der bestehenden Prozesskette erfolgte die Befüllung der Werkzeugdatenbank manuell. Die geometrischen Daten eines Schaftfräsers wurden in die Werkzeugdatenbank eingegeben und anhand dieser das digitale Werkzeug erstellt. Zusätzlich musste ein 3D-Modell des Halters vom Hersteller heruntergeladen und manuell aufbereitet werden. Außerdem mussten die Koordinatensysteme für den Connectioncode in beiden Modellen eingefügt werden.

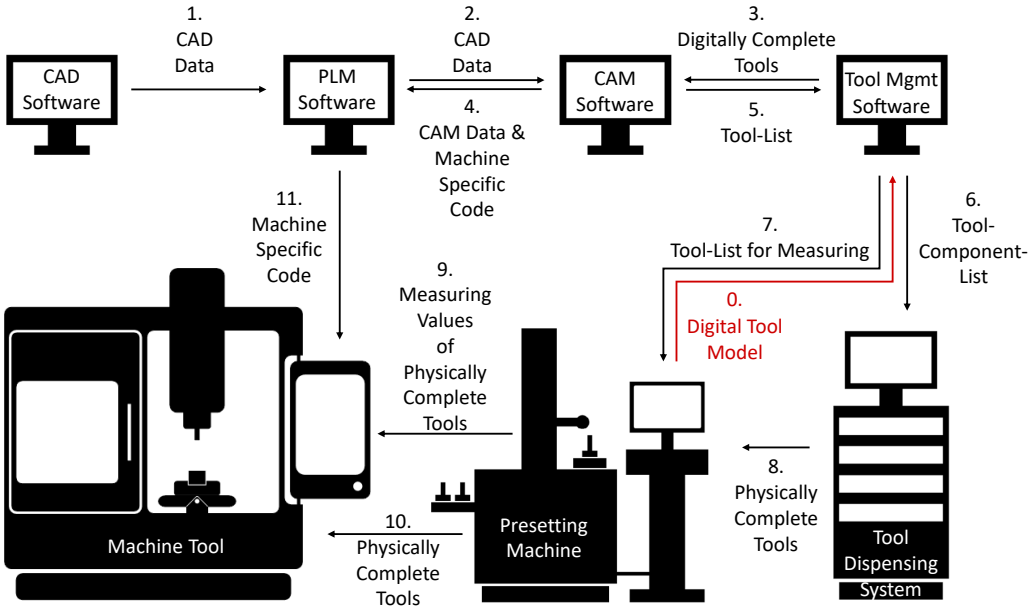


Abbildung 14: CAD/CAM/CNC-Prozesskette der smartfactory@tugraz mit ergänzendem Teilprozess

Quelle: Schmid/Pichler (2020), (leicht modifiziert).

Für die Durchführung des experimentellen Teils wurden sowohl Hardware als auch Software benötigt. Diese standen teilweise bereits in der smartfactory@tugraz zur Verfügung beziehungsweise wurden vom Kooperationspartner für die Durchführung dieser Arbeit bereitgestellt. Um eine bessere Übersicht zu geben folgt in den Abschnitten 3.2 und 3.3 eine Unterscheidung zwischen Hardware und Software.

3.2 Verwendete Hardware

In diesem Abschnitt wird die verwendete Hardware vorgestellt, welche für die Umsetzung der praktischen Durchführung benötigt wird. Dabei handelt es sich um:

- Ein Zoller »venturion 450« Werkzeugvoreinstellgerät
- Einen Vollhartmetall-Schaftfräser
- Einen Steilkegel-Werkzeughalter

Das Werkzeugvoreinstellgerät Zoller »venturion 450« besteht aus dem Grundgerät und der Bedieneinheit »cockpit«.[63] In Abbildung 15 ist ein Bild des Voreinstellgeräts mit Bedieneinheit der smartfactory@tugraz zu sehen.

Abbildung 15: Werkzeugvoreinstellgerät Zoller »venturion 450«
Quelle: Eigene Darstellung.

[63] Vgl. E. Zoller GmbH & Co. KG (2016), S. 46.

Das Voreinstellgerät führt die Vermessung des Werkzeugs vor der Bearbeitung durch und sendet die gemessenen Werte im Anschluss an die Messung an die jeweilige Werkzeugmaschine. Für die Übertragung der Messwerte an die Werkzeugmaschine stehen wie bereits in Abschnitt 2.5 erläutert mehrere Varianten zur Verfügung. Unter anderem besteht die Möglichkeit des Einsatzes von Etiketten mit der Beschriftung eines DMC, welche auch in dieser Arbeit ihre Verwendung findet. Des Weiteren ist das Voreinstellgerät bereits in das Netzwerk der smartfactory@tugraz eingebunden und kann Daten auf einem Netzwerkordner speichern.

Das Angebot verschiedener Messmodi am Voreinstellgerät in Kombination mit der verfügbaren Netzwerkschnittstelle führte zu der Idee, dieses auch für den Scan eines Komplettwerkzeugs einzusetzen. Beruhend auf dem generierten Modell könnte mittels einer automatisierten Methode das zeitaufwändige Ausrichten und Einfügen der Koordinatensysteme in den Einzelkomponenten zur Erstellung von Connectioncodes abgelöst werden. Außerdem liefert es die Grundlage für die Volumenkörpererstellung. Das resultierende Ergebnis ist ein 3D-Modell, welches für eine CAM-Simulation aufbereitet und mit allen notwendigen Eigenschaften zur Integration in eine Werkzeugdatenbank definiert wurde. Durch eine solche Methode und die anschließende Integration in die Werkzeugdatenbank kann nicht nur ein erheblicher Zeitaufwand vermieden, sondern generell eine höhere Prozesssicherheit bei der Befüllung der Werkzeugdatenbank erreicht werden.

Der betrachtete Vollhartmetall-Schaftfräser HAM 40-1281 verfügt über drei Schneiden, wobei eine davon die Zentrumsschneide ist. Der Schaftfräser wird bei der experimentellen Durchführung mit den Durchmessern 6 mm, 8 mm, 10 mm, 12 mm und 16 mm eingesetzt. Aus dem Werkzeugdatenblatt im Anhang A.5 können die genauen Abmessungen zu den jeweiligen Durchmessern entnommen werden.

Der Schaftfräser ist in einer Weldon-Aufnahme eines passenden Steilkegel-Werkzeughalters fixiert. Die Größe wird dem Durchmesser des Schaftfräsers entsprechend angepasst.

Die nachfolgende Erklärung der praktischen Durchführung wird anhand eines Schaftfräsers mit einem Durchmesser von 10 mm und dem dazu passenden Halter demonstriert.

3.3 Verwendete Software

Die verwendete Software lässt sich in zwei Teile gliedern. Einerseits handelt es sich dabei um die Software zur Erstellung des Werkzeugscans an der Bedieneinheit des Voreinstellgeräts und andererseits um die Software zur Erstellung des Programms am PC. Die Bildverarbeitungssoftware »pilot 4.0« des Voreinstellgeräts wurde in Kombination mit der Bedieneinheit »cockpit« ausgeliefert. Für die Durchführung einer Messung des Werkzeugs stehen verschiedene Modi zur Verfügung, wobei jeder einer bestimmten Verwendung zugeordnet ist.[64] Auf der anderen Seite werden am PC das CAD/CAM-System Siemens NX 12.0 mit dessen Programmierschnittstelle NX Open sowie die integrierte Entwicklungsumgebung Microsoft Visual Studio Community 2019, auch als Integrated Development Environment (IDE) bezeichnet, verwendet. Mit Hilfe von Visual Studio wird eine ausführbare Datei in der Programmiersprache Visual Basic (VB) erstellt, mit der ein 3D-Modell des Komplettwerkzeugs in Siemens NX automatisiert erstellt wird.

NX Open ist eine Programmierschnittstelle, die es der Benutzerin oder dem Benutzer ermöglicht, Abläufe in Siemens NX durch das Erstellen von eigenen Programmen zu automatisieren. Diese Programme können mit einer Schaltfläche in die Benutzeroberfläche, Graphical User Interface (GUI), eingebunden oder über einen Menübefehl ausgeführt werden. Dabei bietet NX Open die Möglichkeit, die Programmiersprache nach persönlicher Präferenz auszusuchen. Zur Auswahl stehen C++, Java, Python, C# und die in dieser Arbeit verwendete Programmiersprache VB. Der Vorteil von NX Open liegt in der Automatisierung von sich wiederholenden Tätigkeiten bei der Produktentwicklung in Siemens NX. Darüber hinaus bietet NX Open eine Reihe an hilfreichen Funktionen für den Entwicklungsprozess, welche hier beispielhaft genannt sind:[65]

- Erstellen von CAD- und CAM-Objekten
- Durchlaufen von Objekten eines Bauteils mit dem Ziel Informationen auszulesen oder unterschiedliche Operationen durchzuführen
- Erstellen einer Benutzerschnittstelle

[64] Vgl. E. Zoller GmbH & Co. KG (2012).
[65] Vgl. Siemens PLM Software (2019), S. 1.

Grundsätzlich ist in Siemens NX zwischen mehreren Programmierschnittstellen, welche über die Jahre stetig weiterentwickelt wurden, zu unterscheiden. Die ältere Generation der Schnittstellen sind die „User Functions“, welche auch aktuell noch unterstützt werden. Zunehmend findet jedoch eine Verdrängung durch die neuen Schnittstellen statt. Die Schnittstelle NX Open .NET enthält einerseits den NX.OpenUF-Namensraum und andererseits den NXOpen-Namensraum. Ein Namensraum ist für die Organisation von Objekten zuständig. Außerdem wird er zur Vermeidung von Mehrdeutigkeiten und zur Vereinfachung von Verweisen auf große Sammlungen von Objekten eingesetzt.[66] Der NX.OpenUF-Namensraum stellt Verbindungen zu den veralteten „User Functions“ her. Diese können als Verknüpfungen verstanden werden. Sie verweisen auf deren Funktionen, sodass diese schließlich ausgeführt werden. Der NXOpen-Namensraum stellt die neueren Funktionen der Schnittstelle NX Open .NET dar. Diese greifen direkt auf Siemens NX zu und benötigen keine Verknüpfungen. Die Schnittstelle SNAP kann eigenständig nicht direkt auf Siemens NX zugreifen. Sie benötigt Verknüpfungen zu NX Open .NET.[67] Die Programmierschnittstellen und deren Abhängigkeiten sind in Abbildung 16 visualisiert. Dabei stellen die Punkte Funktionen der jeweiligen API dar. Die Linien geben die Verbindung zwischen den Funktionen zweier Schnittstellen symbolisch wieder. In der vorliegenden Arbeit wird ausschließlich die Schnittstelle NX Open .NET verwendet.

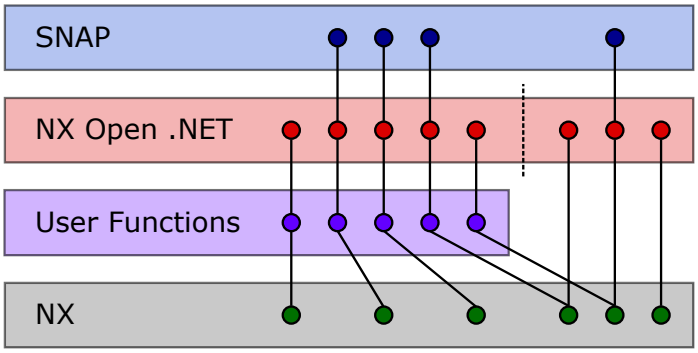


Abbildung 16: Programmierschnittstellen in Siemens NX
Quelle: Siemens PLM Software (2019), S. 43 (leicht modifiziert).

[66] Vgl. Microsoft Corporation (2021a).
[67] Vgl. Siemens PLM Software (2019), S. 43.

3.3.1 Methoden zur Programmerstellung

Dieser Unterabschnitt gibt Auskunft über die Möglichkeiten der Programmerstellung in der vorliegenden Arbeit. Dabei werden zwei Arten unterschieden, welche erst durch deren Kombination den vollen Leistungsumfang bieten. Es handelt sich einerseits um die Journalaufzeichnung und andererseits um das Erstellen von Programmen mittels Microsoft Visual Studio. Die Journalaufzeichnung ist eine interne Funktion von Siemens NX. Bei der Methode mittels Microsoft Visual Studio handelt es sich um eine externe Möglichkeit zum Generieren von Programmen. Beide Varianten werden für die Entwicklung der automatisierten Methode in dieser Arbeit eingesetzt. In den folgenden Unterabschnitten werden die Vor- und Nachteile beider Varianten aufgezeigt und die Grundlagen beschrieben.

3.3.1.1 Journalaufzeichnung

Grundlegende Informationen über die Journalaufzeichnung werden in diesem Unterabschnitt bereitgestellt. Die Anwendung einer Journalaufzeichnung wird in Anhang A.2 näher erläutert.

Ein Journal kann grundsätzlich mit einem Makro verglichen werden. Darin wird eine Abfolge von Funktionen in Siemens NX als Code abgespeichert. Ein Journal kann in den zuvor beschriebenen Programmiersprachen aufgezeichnet und in weiterer Folge wieder abgespielt werden. Alle Funktionen, die durch die Journalaufzeichnung vollständig unterstützt werden, sind während der Journalaufzeichnung mit einem kleinen grünen Quadrat an den Schaltflächen gekennzeichnet. So wird der Benutzerin oder dem Benutzer ein schneller Überblick geboten, ob die gewünschte Funktion mit der Aufzeichnung überhaupt im Code dargestellt werden kann. Während der Journalaufzeichnung werden alle Schritte, die in Siemens NX ausgeführt und von der Journalaufzeichnung unterstützt werden, in Form von Codezeilen in der ausgewählten Programmiersprache im Journal abgespeichert. Hierzu wird im Gegensatz zur Erstellung in Microsoft Visual Studio keine NX Open Author Lizenz benötigt. Dieser im Hintergrund automatisch erstellte Code kann nach Beendigung der Aufzeichnung im Journaleditor nachvollzogen werden. Bei der Aufnahme werden viele Hintergrundprozesse mitaufgezeichnet, sodass eine kurze Abfolge von Tätigkeiten bereits zu einem hohen Umfang an Codezeilen führen kann. Diese Hintergrundprozesse sind zum Beispiel das Setzten von Marken, mit denen zu einem bestimmten Punkt zurückgesprungen

werden kann. Daher ist es notwendig den Code zu durchsuchen, um die relevanten Funktionen zu finden. Zu erwähnen ist, dass Journale nur in den Sprachen C#, Pyhton und VB von Siemens NX ausgeführt werden können. Die Journalaufzeichnung bietet eine gute Möglichkeit herauszufinden, wie bestimmte Funktionen in der Programmierung mit NX Open angesprochen werden können.[68] In Tabelle 4 werden die Vor- und Nachteile der Journalaufzeichnung aufgelistet.

Tabelle 4: Vor- und Nachteile der Journalaufzeichnung
Quelle: Siemens PLM Software (2019), S. 4.

Vorteile	**Nachteile**
Keine Voreinstellungen notwendig	Schlechte Unterstützung für die Programmiererin oder den Programmierer
Ideal für kurze Programme	Einzelne Datei für den gesamten Code
Keine NX Open Author Lizenz benötigt	Beschränkte Funktionen

3.3.1.2 Erstellung mittels Microsoft Visual Studio

Neben der Journalaufzeichnung in Siemens NX gibt es die Möglichkeit, ein Programm mittels Microsoft Visual Studio zu erstellen. Diese IDE bietet der Programmiererin oder dem Programmierer eine Auto-Vervollständigung von Befehlen, die bei Visual Studio den Namen Intellisense trägt. Durch deren Hilfe wird der Programmiererin oder dem Programmierer die Programmerstellung durch das Vorschlagen von möglichen Befehlen wesentlich erleichtert. Bei umfangreichen Programmen wird die Aufteilung auf mehrere einzelne Dateien empfohlen. Grundsätzlich wird zu Beginn eine Projektmappe in Visual Studio erstellt. Die Projektmappe kann mehrere Projekte enthalten und verwalten. In diesem Fall verwaltet die Projektmappe eine Klassenbibliothek vom **.vbproj*-Dateityp. Die **.vbproj* wiederum kann beliebig viele Klassen beinhalten. Durch die Aufteilung des Codes in mehrere einzelne Klassen wird eine bessere Übersicht gewährleistet. Damit in dieser IDE ein Programm erstellt werden kann, müssen gewisse Voreinstellungen getroffen werden und eine NX Open Author Lizenz zur Verfügung stehen. Diese Lizenz ermöglicht den Zugriff auf einen erweiterten Funktionsumfang im Vergleich zur Journalaufzeichnung. Für die notwendigen

[68] Vgl. HBB Engineering GmbH (2019), S. 229.

Voreinstellungen wird auf Anhang A.3 verwiesen. Zusammenfassend werden alle Vor- und Nachteile der Programmerstellung in Microsoft Visual Studio in Tabelle 5 dargestellt.

Tabelle 5: Vor- und Nachteile der Programmerstellung mittels Microsoft Visual Studio
Quelle: Siemens PLM Software (2019), S. 4.

Vorteile	**Nachteile**
Bessere Übersicht	Voreinstellungen notwendig
Autovervollständigung von Befehlen	NX Open Author Lizenz benötigt
Aufteilung auf mehrere Dateien	
Mehr Funktionen	

Damit ein Projekt, welches in Visual Studio erstellt wurde, überhaupt ausgeführt werden kann, muss es zunächst kompiliert werden. Das bedeutet, dass ein ausführbares Programm erstellt werden muss, indem die Textdatei in eine Binärdatei umgewandelt wird. Dabei kann es sich um eine **.dll*-Datei oder um eine **.exe*-Datei handeln. Der Unterschied zwischen diesen beiden Dateien besteht in der Ausführung. Die **.dll* wird intern in einer laufenden NX-Session und die **.exe* außerhalb von Siemens NX ausgeführt.[69] Im weiteren Verlauf dieser Arbeit handelt es sich bei einer ausführbaren Datei immer um eine **.dll*.

3.3.2 Signierung von ausführbaren Dateien

Da die erstellten Programme im Regelfall nicht bei den Entwicklerinnen und Entwicklern sondern bei den Anwenderinnen und Anwendern zum Einsatz kommen, wird die Möglichkeit der Signierung von ausführbaren Dateien geboten. Zur Durchführung einer Signierung muss eine NX Open Author Lizenz zur Verfügung stehen. Die Signierung einer Datei ohne Lizenz ist nicht möglich. Für die Ausführung einer signierten Datei wiederum wird keine NX Open Author Lizenz benötigt. Dadurch können bei der Anwenderin und dem Anwender des Programms Lizenzkosten eingespart werden. Die Lizenz wird folglich nur zum Erstellen von ausführbaren Programmen, zum Signieren von ausführbaren Dateien und zum Ausführen nicht signierter Programme benötigt. Eine Signierung ermöglicht demnach der Anwenderin

[69] Vgl. HBB Engineering GmbH (2019), S. 238.

und dem Anwender die Ausführung einer Datei ohne Lizenz. Für die Durchführung einer Signierung wird auf Anhang A.4 verwiesen.

3.3.3 Debuggen mit NX Open

NX Open bietet die Möglichkeit ein Programm zu debuggen. Dabei können zwei unterschiedliche Varianten verwendet werden.

Zum einen wird die Funktion „An den Prozess anhängen“ in Visual Studio angeboten. Dabei ist ein Breakpoint in der gewünschten Codezeile zu setzen. Dieser ist ein Haltepunkt, an dem das Programm bei der Ausführung stoppt, sodass die Benutzerin oder der Benutzter die Möglichkeit hat, nach einem Fehler zu suchen beziehungsweise den Code ab dieser Stelle Schritt für Schritt zu durchlaufen. Im nächsten Schritt ist unter „Debuggen“ in Visual Studio die Funktion „An den Prozess anhängen“ und der NX-Prozess *ugraf.exe* auszuwählen. Danach kann die **.dll* in Siemens NX ausgeführt werden, das Programm stoppt beim ersten Breakpoint und wechselt zu Visual Studio. [70]

In dieser Arbeit wurde die zweite Möglichkeit, das Hinzufügen einer Codezeile im Programm, verwendet. Der Befehl „System.Diagnostics.Debugger.Launch()“ wird an die gewünschte Position im Code an Stelle eines Breakpoints eingefügt. Beim Ausführen der **.dll* in Siemens NX öffnet sich das Fenster des Just-In-Time-Debuggers. Hier kann ausgewählt werden, welche Instanz für das Debuggen herangezogen werden soll. Sollte das Projekt, mit welchem die **.dll* erzeugt wurde, zeitgleich in Visual Studio geöffnet sein, so ist dieses auszuwählen. Alternativ kann eine neue Instanz in Visual Studio erstellt werden. Dies führt zum Öffnen des Projekts und zum Halten an der gewünschten Stelle. [71]

[70] Vgl. HBB Engineering GmbH (2019), S. 242.
[71] Vgl. Siemens PLM Software (2019), S. 27.

3.4 Generierung eines Werkzeugscans

Die Grundlage für die automatisierte Methode bildet ein Digitales Modell der Kontur des Komplettwerkzeugs. Die Generierung dieses Modells mittels Voreinstellgerät wird in diesem Abschnitt behandelt.

3.4.1 Vorbereitende Maßnahmen

Die Messung am Voreinstellgerät Zoller »venturion 450« hat mit einem Komplettwerkzeug zu erfolgen. Grundsätzlich ist zu Beginn ein Kalibriervorgang durchzuführen. Dazu verfügt das Voreinstellgerät über eine kleine Kalibrierkugel an der Seite der Adapteraufnahme. Wenn das Gerät ordnungsgemäß hochgefahren wurde und betriebsbereit ist, kann mit der Beladung des Komplettwerkzeugs begonnen werden. Dazu muss die Adapteraufnahme in der Werkzeugaufnahme bereits installiert sein. Für die Beladung des Komplettwerkzeugs muss dieses in die Adapteraufnahme gesteckt werden und anschließend die Taste „Clamp“ auf der Folientastatur gedrückt werden. Es entsteht eine Werkzeugspannung im Voreinstellgerät. Bevor nun mit dem eigentlichen Scanvorgang begonnen wird, muss das Komplettwerkzeug und die Kalibrierkugel frei von Verunreinigungen sein, da diese das Resultat des Scans negativ beeinflussen können. Die Reinigung erfolgt mit einer Reinigungsknete. Nach Abschluss der Säuberung ist das Komplettwerkzeug bereit für den Scan. In Abbildung 17 ist ein eingespanntes Komplettwerkzeug (Schaftfräser mit Durchmesser 10 mm) im Voreinstellgerät zu sehen.

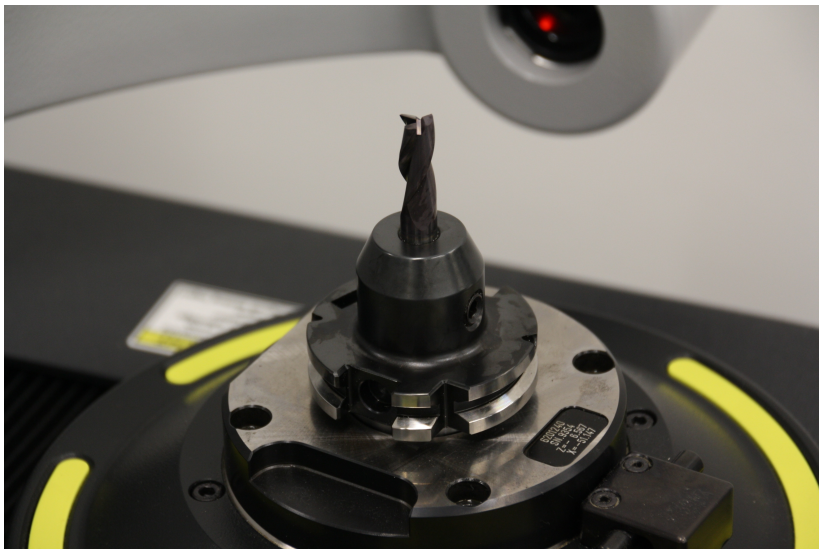


Abbildung 17: Komplettwerkzeug im Voreinstellgerät
Quelle: Eigene Darstellung.

3.4.2 Durchführung

Für die Messung eines Werkzeugs stehen in der Bedieneinheit »cockpit« mehrere Modi zur Verfügung. In dieser Arbeit wird das Modul „Werkzeug messen/einstellen/verwalten“ verwendet. In diesem Messmodul befindet sich die Schaltfläche „3D Grafik anzeigen/erzeugen“, mit welcher der Scan gestartet wird. Wahlweise kann der Einhandbediengriff zur Spitze des Komplettwerkzeugs geführt werden oder die Maschine manövriert selbstständig zu dieser. Die manuelle Variante ist gegenüber der automatischen zeitsparender. Nach dem Start des Scans wird das Werkzeug um die eigene Achse gedreht und der Laser fährt zur Messung an der Kontur des Komplettwerkzeugs entlang. Durch die Überlagerung der Drehbewegung des Werkzeugs mit der Bewegung des Lasers entsteht ein 3D-Modell, welches nach Abschluss des Scans am Bildschirm angezeigt wird. Der Anwenderin beziehungsweise dem Anwender werden zum Export des Scans mehrere Dateiformate zur Verfügung gestellt. In diesem Fall wird der Scan als Vericut-Format(DXF-Layer) im 2D-Format exportiert. Es handelt sich dabei um eine von der Software bearbeitete **.dxf*, welche Linien an Stelle einer Anreihung von Punkten ausgibt. Der Grund für die Auswahl dieser Exportmethode liegt in der Tatsache, dass eine Skizze

in weiterer Folge einfacher im Vergleich zu einem Volumenkörper zu bearbeiten ist. Ein Export der Kontur eines Komplettwerkzeugs mit einem Fräserdurchmesser von 10 mm ist in Abbildung 18 zu sehen.

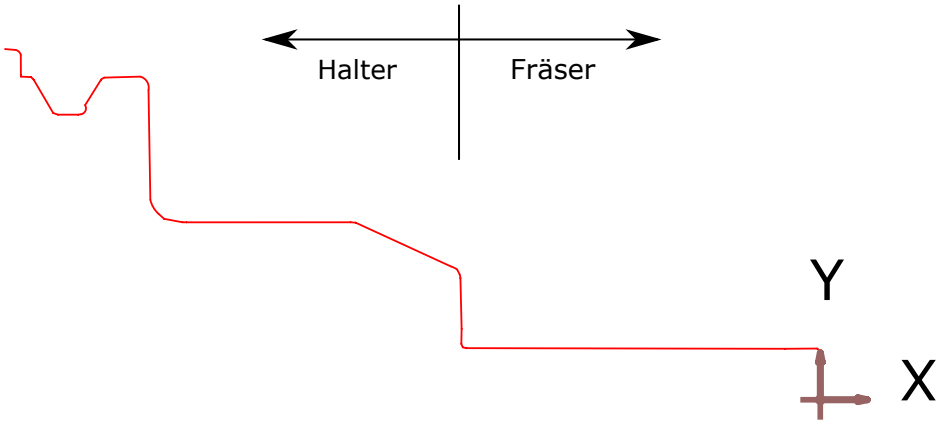


Abbildung 18: Werkzeugscan mit Durchmesser 10 mm
Quelle: Eigene Darstellung.

Der Scan wird auf einem eingerichteten Netzwerkordner abgespeichert und anschließend für den Import in Siemens NX verwendet. Nach der ordnungsgemäßen Durchführung der Messung muss das Komplettwerkzeug wieder entladen werden. Hierzu muss der Einhandbediengriff zur Seite gefahren und die Taste „Clamp“ auf der Folientastatur erneut gedrückt werden. Die Werkzeugspannung im Voreinstellgerät wird gelöst und das Komplettwerkzeug kann entnommen werden.

3.5 Programmerstellung

In dieser Arbeit erfolgt die Programmerstellung basierend auf einem Template in Visual Studio. Für die Methodik der Einrichtung in Microsoft Visual Studio wird auf Anhang A.3 verwiesen. Nachdem alle vorbereitenden Maßnahmen in der Einrichtungsphase abgeschlossen wurden, kann mit der Erstellung des Programms fortgefahren werden. Dazu wurden zu Beginn die Funktionen in Siemens NX erprobt, um ein Verständnis für deren Funktionsweise zu erlangen. Anschließend wurde die Journalaufzeichnung verwendet, um

den Klassenaufbau in NX Open zu verstehen. Ergänzend dazu wird die Dokumentation „NX Open .NET Reference Guide“ herangezogen.[72] In dieser Dokumentation sind alle Klassen der NX Open Bibliothek und deren Funktionen zusammengefasst. Sie ist ein wesentliches Hilfsmittel zum Verständnis der Aufbaustruktur von NX Open.

Das erstellte Programm kann grundsätzlich in drei Teile gegliedert werden. Diese sind die Erstellung der Skizze, die Erstellung und Aufbereitung der Volumenkörper sowie der Export des 3D-Modells.

Das Ziel des ersten Teils ist die Erstellung einer Skizze, welche die importierte **.dxf* hinreichend genau wiedergibt. Dazu wird der Werkzeugscan analysiert und auf Basis der gewonnenen Informationen die neue Skizze erzeugt. Der zweite Abschnitt wird durch die Separierung der Skizze in zwei einzelne und die anschließende Erzeugung von Volumenkörpern bestimmt. Außerdem findet das Anlegen von Expressions und das Einfärben von Bauteilen statt. Zum Schluss werden relevante Daten aus dem 3D-Modell nach Excel in ein Tabellenblatt exportiert. Diese drei Teile können in Siemens NX getrennt voneinander ausgeführt werden, sodass der Benutzerin oder dem Benutzer nach Abschluss jedes Teils eine Möglichkeit zur Überprüfung geboten wird. Bei Erfüllung der Anforderungen kann mit der Durchführung des nächsten Teils begonnen werden.

Beim Aufbau des Codes wird das Prinzip der objektorientierten Programmierung nicht verfolgt, da sie für diesen Anwendungsfall nicht zweckmäßig ist. Vielmehr handelt es sich bei diesem Programm um eine Abarbeitung von Schrittfolgen. Dazu werden Klassen zur Strukturierung erstellt und in diesen Prozeduren und Funktionen angelegt. Dadurch wird eine bessere Übersicht des Programmcodes erzielt. Die Prozeduren und Funktionen erhalten auf spezifische Variablen klassenübergreifenden Zugriff. Die drei Teile des Programms werden je einer Schaltfläche in der Menüleiste von Siemens NX zugewiesen. Durch die Verknüpfung der Programme mit den Schaltflächen wird die Ausführung vereinfacht. Alternativ kann das gewünschte Programm auch über einen Menübefehl ausgewählt werden.

[72] Vgl. Siemens PLM Software (2021c).

3.5.1 Automatisierte Generierung einer Skizze

In diesem Unterabschnitt wird der erste Teil des Programms, welcher sich mit der Erstellung der Skizze beschäftigt, beschrieben. Dabei kommt ausgehend vom importierten Werkzeugscan eine Methode zum Einsatz, welche es ermöglicht, eine eigene Skizze zu erstellen und diese in weiterer Folge für die Erzeugung eines Volumenkörpers zu verwenden. Die nachfolgenden Unterabschnitte lassen sich in drei Phasen unterteilen. Zu Beginn wird der Import des Werkzeugscans behandelt. Anschließend erfolgt beruhend auf einer Analyse des Scans die Erstellung einer Skizze. Zum Abschluss wird die Option zur Überprüfung der erstellten Skizze behandelt.

3.5.1.1 Import des Werkzeugscans

Zu Beginn des Programms müssen Variablen definiert werden. Dabei handelt es sich zum Beispiel um die aktuelle NX-Session oder das Bauteil. Mit Hilfe dieser Definition können die NX-Session und das Bauteil vom System erkannt und infolgedessen Befehle an ihnen ausgeführt werden. Bevor der Import des Werkzeugscans durchgeführt wird, findet im Programm eine Überprüfung statt. Diese prüft, ob beim Anlegen der Bauteilvariable tatsächlich ein Bauteil der aktuellen NX-Session übergeben wurde. Bei Nichterfüllung der Bedingung wird das Programm nicht ausgeführt. Im anderen Fall wird mit der Ausführung des Codes fortgefahren.

Siemens NX verfügt über die Möglichkeit Dateien mit einem anderen Dateiformat zu importieren. Folglich kann die am Voreinstellgerät generierte **.dxf* importiert werden. Für den Import der **.dxf* wird einerseits der Pfad der zu importierenden Datei und andererseits der Pfad des aktuellen Bauteils benötigt. Dazu wurde eine Art Dialog mit der Benutzerin oder dem Benutzer verwendet, welche aufgefordert werden, den vollständigen Dateipfad der zu importierenden **.dxf* einzugeben. Nachdem die Eingabemaske ausgefüllt wurde und die Eingabe bestätigt wird, sucht sich das Programm über den Pfad die Datei und fügt sie in das zuvor definierte Bauteil ein. Dazu muss dem Programm der Pfad des definierten Bauteils bekannt sein, welcher über die Bauteilvariable und die Funktion „FullPath“ ausgelesen werden kann. Die anschließenden Operationen zum Import der **.dxf* wurden mit Hilfe der Journalaufzeichnung aufgezeichnet. Diese ermöglichen Anpassungen bei den Einheiten,

der Skalierung sowie bei den zu verwendenden Layer. Nach Abschluss des Imports wird in Siemens NX der Werkzeugscan, welcher aus Linien und Bögen besteht, angezeigt. In Abbildung 18 in Unterabschnitt 3.4.2 ist dieser veranschaulicht.

3.5.1.2 Analyse und Erstellung

Nachdem die **.dxf* erfolgreich importiert wurde, wird eine leere Skizze erstellt. Diese soll die Basis für die eigene Skizze darstellen. Grundsätzlich wird für die Erstellung einer Skizze eine Achse, ein Ursprung und eine Ebene benötigt. Für die Erzeugung einer Achse wiederum bedarf es eines Punkts und eines Vektors. Die Elemente für die Definition einer Ebene sind ein Punkt und eine Matrix, mit der die Orientierung festgelegt wird. Im Anschluss werden alle Linien mit Hilfe einer For Each-Schleife durchlaufen. Dadurch können mit jeder einzelnen Linie gewisse Operationen durchgeführt werden. Diese Schleife dient zum Finden von Linien, welche für die folgende Skizzenerstellung relevant sind. Dazu werden der Start- und Endpunkt jeder einzelnen ausgelesen. Unter Verwendung der Koordinaten dieser Punkte in Kombination mit einer entwickelten Formel, kann der Winkel zwischen der aktuellen Linie und der gedachten Horizontalen beziehungsweise Vertikalen bestimmt werden. Durch die Berechnung des Winkels kann geprüft werden, ob es sich bei der betrachteten Linie um eine relevante Linie für die weitere Skizzenerstellung handelt. Die Formel ist wie folgt definiert:

$$angle = \arctan\left(\frac{Endpunkt.Y - Startpunkt.Y}{Endpunkt.X - Startpunkt.X}\right) \cdot \frac{180}{\pi} \tag{3.1}$$

Grundsätzlich besteht die Formel aus x- und y-Koordinaten der ausgelesenen Start- und Endpunkte. Durch Anwendung des Arkustangens auf den Quotienten wird der Winkel berechnet. Die Umrechnung von Radiant in Grad erfolgt durch den Zusatz am Ende der Gleichung.

Die erste Linie befindet sich immer an der Werkzeugspitze und stellt die Stirnseite des Werkzeugs dar. Dort liegt auch der Ursprung des Bauteils mit den Koordinaten (0/0/0). Auffallend ist, dass die Koordinaten des Startpunkts der ersten Linie nie mit dem Ursprung übereinstimmen. In Abbildung 19 wird dieses Phänomen veranschaulicht. Die

x- und y-Koordinate der ersten Linie weichen minimal vom Koordinatenursprung ab und sind mit einem roten Rechteck gekennzeichnet.

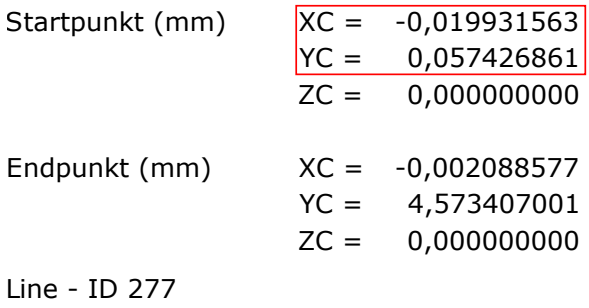


Abbildung 19: Koordinaten der Startlinie des Werkzeugscans
Quelle: Eigene Darstellung.

In diesem Fall führt die positive y-Koordinate zu einer Linie, die den Ursprung nicht erreicht. Im Fall einer negativen y-Koordinate wird der Ursprung durchfahren. Eine mögliche Erklärung für dieses Ereignis ist auf die Eigenschaften des Werkzeugs und die Ausführung des Werkzeugscans zurückzuführen. Der Schaftfräser verfügt über drei Schneiden, wobei eine davon die Zentrumsschneide ist. Diese erstreckt sich bis leicht über den Mittelpunkt des Fräsers. Einerseits kann die Anfangsposition des Werkzeugs den Scan beeinflussen und andererseits die Drehbewegung während des Scans. Durch die Gestalt des Schaftfräsers und die Ausführung der Zentrumsschneide kann es zu dieser leichten Abweichung kommen. Je nach Stellung des Werkzeugs wird die Zentrumsschneide durch den Messvorgang unterschiedlich erkannt. Dabei können zwei Möglichkeiten unterschieden werden. Einmal startet die Skizze mit negativer y-Koordinate und im anderen Fall, wie auch in Abbildung 19 veranschaulicht, mit positiver y-Koordinate. Zur Lösung dieses Problems wird der Startpunkt manuell im Ursprung definiert. Dazu wird beim Durchlaufen der Schleife eine Bedingung eingebaut, welche beim ersten Durchlauf einen Punkt im Ursprung erstellt. Dieser Punkt wird einer Liste zur späteren Verwendung hinzugefügt. Anschließend wird ein Vergleich des berechneten Winkels mit einem empirisch ermittelten Wert durchgeführt. Dabei wird entschieden, ob die Linie überhaupt relevant für die Skizzenerstellung ist. Bei diesem Vergleichswert handelt es sich um einen Winkel von ±1,5 Grad. Dieser hat sich im Laufe der Forschungsarbeiten als geeigneter Wert herausgestellt. Durch diese Gegenüberstellung kann ausgeschlossen werden, dass ungeeignete Linien für weitere Arbeiten in Betracht gezogen

werden. Bei Linien, die diesen Grenzwert unterschreiten, wird der Endpunkt dieser Linie einer Liste zur späteren Verwendung hinzugefügt. Bei Überschreiten der Toleranz wird die Linie zur Entfernung freigegeben. Festzuhalten ist, dass die Linien nie zu 100 Prozent vertikal oder horizontal sind. Eine leichte Winkeldifferenz zur Horizontalen beziehungsweise Vertikalen ist stets vorhanden. Dies ist der Grund für die Einführung des Toleranzbereichs. Bei Erreichen der letzten Linie wird der Endpunkt immer der Liste zur späteren Verwendung hinzugefügt. Dabei wird die y-Koordinate um 3 mm verringert, damit der unerwünschte Überstand der Adapteraufnahme ausgeglichen wird. Dieser ist in Unterabschnitt 3.4.1 in Abbildung 17 in der rechten, unteren Ecke und in Unterabschnitt 3.4.2 in Abbildung 18 am linken Ende der Kontur dargestellt.

Jene Linien, welche die Toleranz überschritten haben und zur Entfernung freigegeben wurden, werden zusammen mit den Bögen in einer Liste zum Löschen gesammelt. Die Bögen werden für das weitere Vorgehen nicht benötigt. Mit Hilfe dieser Liste können die enthaltenen Elemente zu einem definierten Zeitpunkt durch eine Aktualisierung der NX-Session gelöscht werden. Für die Definition dieses Befehls muss eine Marke gesetzt sein. Diese Marke kann an beliebiger Stelle im Code platziert werden und gibt die Stelle an, ab welcher die NX-Session aktualisiert werden soll. In Abbildung 20 sind die Linien, die nach dem Vergleich für relevant gehalten wurden, dargestellt.

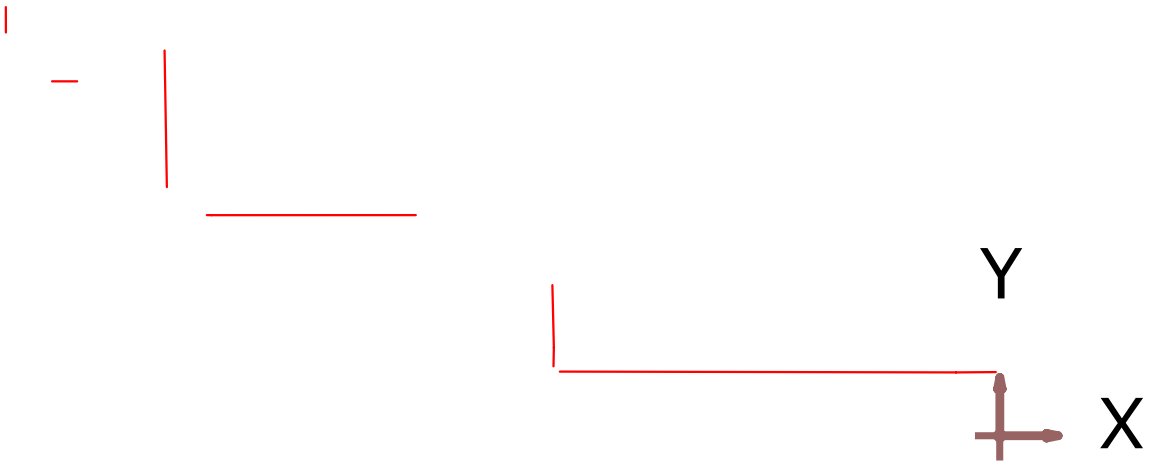


Abbildung 20: Relevante Linien zur weiteren Beurteilung
Quelle: Eigene Darstellung.

Im nächsten Schritt werden die relevanten Punkte, welche in einer separaten Liste gespeichert wurden, beurteilt. Dazu wird die For Each-Schleife erneut eingesetzt und jeder Punkt dieser Liste einzeln durchlaufen. Diese beinhaltet eine Abfrage, welche die Punkte nach ihren Koordinaten bewertet. In weiterer Folge wird entschieden, ob es sich bei einer gedachten Linie zum nächsten Punkt um eine horizontale oder um eine vertikale Linie handelt. Durch diese Information kann herausgefunden werden, ob der gegenwärtige Punkt für die Skizzenerstellung überhaupt benötigt wird. In Abbildung 21 sind die Endpunkte der relevanten Linien dargestellt. Die Linien werden hauptsächlich zur Veranschaulichung der Endpunkte angezeigt. In weitere Folge wird ausschließlich mit diesen Punkten gearbeitet.

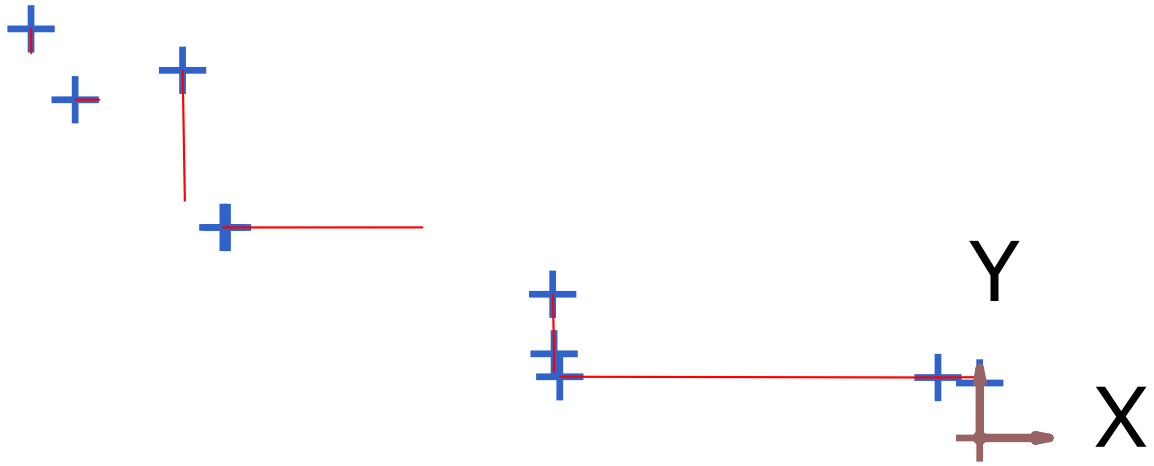


Abbildung 21: Endpunkte der relevanten Linien
Quelle: Eigene Darstellung.

Dabei ist zu erkennen, dass die Horizontalen und Vertikalen der Kontur aus zwei oder mehr aufeinanderfolgenden Linien bestehen können. Zurückzuführen ist dies auf die Messung, welche eine Kontur aus Punkten erzeugt und diese im Anschluss als Vericut-Format(DXF-Layer) ausgibt. Hierbei werden die Punkte in Linien und Bögen zusammengefasst. Somit kann ein leichter Knick entstehen, welcher sich durch zwei Linien mit unterschiedlichen Steigungen darstellen lässt. Festzuhalten ist, dass es sich um minimale Werte handelt, welche grundsätzlich vernachlässigt werden können. Die eindeutige Trennung wird im nächsten Absatz behandelt.

Aufeinanderfolgende Punkte in der gleichen Koordinatenrichtung müssen erkannt werden, da jeweils nur ein Punkt in dieser erforderlich ist. Die Skizzenerstellung beruht auf dem Prinzip, dass die Linie nicht bis zum nächsten Endpunkt führt. Vielmehr wird eine Kombination der Koordinaten des aktuellen Punkts und des nächsten Punkts eingesetzt. Durch die Verwendung der begrenzenden Koordinate des nächsten Punkts kann der neue Punkt, der sich genau im Übergang von der horizontalen zur vertikalen Linie befindet, gefunden werden. Diese Vorgehensweise ist beispielhaft in Abbildung 22 dargestellt. Es wird die y-Koordinate des ersten Punkts mit der x-Koordinate des zweiten Punkts miteinander kombiniert, sodass der gesuchte Punkt entsteht. Demnach wird immer nur ein Punkt der jeweiligen Koordinatenrichtung für die Skizzenerstellung benötigt. Aus diesem Grund werden die Punkte vor dem letzten Punkt der jeweiligen Koordinatenrichtung gelöscht.

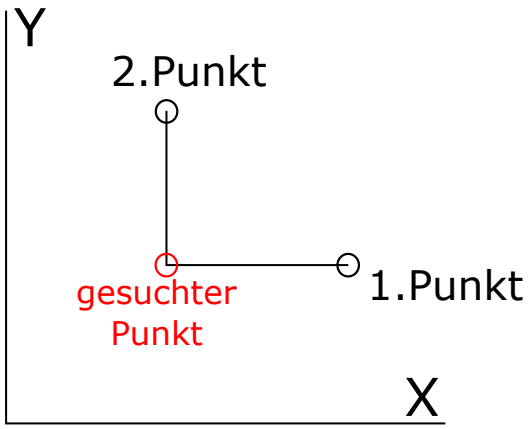


Abbildung 22: Prinzip der Skizzenerstellung
Quelle: Eigene Darstellung.

Zum Finden der nicht benötigten Punkte kommt erneut die bereits erwähnte For Each-Schleife zum Einsatz. Diese führt für jeden Punkt, welcher zuvor in der Liste für relevante Punkte gespeichert wurde, eine Abfrage durch. Dabei wird zu Beginn mit einer if-Anweisung überprüft, ob es sich um den ersten Punkt der Liste handelt. Dies ist jener Punkt, welcher sich im Ursprung befindet und den Ausgangspunkt der Skizze darstellt. Die nächste if-Anweisung überprüft, ob es sich um den letzten Punkt der Liste handelt. Dieser stellt ebenso wie der erste Punkt die Begrenzung der Skizzenkontur dar und ist für das weitere Vorgehen unerlässlich. Bei Erfüllung der Bedingungen der Anweisungen wird mit dem nächsten Punkt in der Liste fortgefahren beziehungsweise im Fall des letzten Punkts die Schleife beendet.

Des Weiteren wird der Punkt, welcher unmittelbar über dem Ursprung liegt, entfernt, da er für die Skizzenerstellung nicht benötigt wird. Durch das zuvor beschriebene Prinzip der Skizzenerstellung kann der Punkt über dem Ursprung mit der x-Koordinate des Ursprungs und mit der y-Koordinate des ersten relevanten Punkts erzeugt werden. Der erste relevante Punkt befindet sich am Ende des Schaftfräsers im Übergang zur Stirnseite des Halters.

Eine weitere if-Anweisung stellt fest, ob es sich beim überprüften Punkt um jenen in der Greiferrille des Halters handelt. Zur Vereinfachung kann dieser vernachlässigt werden, da der Rille bei der Kollisionsüberprüfung im CAM-System eine unbedeutende Rolle zugeteilt wird. Alle diese Abfragen werden beim Durchlaufen der Punkte jeweils zu Beginn durchgeführt.

Im Anschluss daran werden die Differenzen der x- und der y-Koordinaten des aktuellen und des nächsten Punkts berechnet. Diese Berechnung gibt Auskunft darüber, ob es sich bei der gedachten Linie zum nächsten Punkt um eine horizontale oder um eine vertikale Linie handelt, wenn die Differenz innerhalb eines definierten Toleranzbereichs liegt. Mit einer passenden Abfrage wird dies überprüft und in Zusammenhang mit einer speicherbaren Entscheidungsvariable darüber entschieden, ob der Punkt relevant ist oder gelöscht werden kann. Diese Variable speichert den aktuellen Zustand der Linie, horizontal oder vertikal, anhand einer numerischen Klassifizierung ab. Wenn in der Entscheidungsvariable eine horizontale Linie gespeichert ist und die nächste Linie ebenso eine horizontale ist, so wird der aktuelle Punkt der Liste zum späteren Löschen hinzugefügt. Hier handelt es sich um den zuvor beschriebenen Fall, dass zwei horizontale Linien aufeinander folgen und nur der Endpunkt der letzten benötigt wird. Die Speichervariable bleibt dabei unverändert. Stellt sich heraus, dass eine horizontale Linie gespeichert ist und die nächste Linie eine vertikale ist, liegt ein relevanter Punkt vor. Es wird nur die Entscheidungsvariable auf den neuen Status aktualisiert und mit dem nächsten Punkt fortgefahren.

Dabei kann es vorkommen, dass weder eine horizontale noch eine vertikale Linie erkannt wird. Dies ist der Fall, wenn die Differenzen der Koordinaten zu groß sind und der Grenzwert beim Vergleich überschritten wird. Dann werden die if-Anweisungen über die Koordinatendifferenz übersprungen. Implizit ergibt sich daraus, dass es sich um einen Richtungswechsel der Linie handeln muss, da sonst die Koordinatendifferenz unter dem Grenzwert liegen würde. Demnach wird der Punkt für relevant gehalten und nur die Entscheidungsvariable aktualisiert. Der limitierende Wert ist 1,5 mm. Dieser erwies sich im Laufe der Forschungsarbeiten als

passend und wurde empirisch ermittelt. Nach dieser Überprüfung werden ausschließlich die relevanten Punkte in der Liste behalten. Diese Punkte sind in Abbildung 23 dargestellt. Dazu sind zum besseren Verständnis ebenfalls die relevanten Linien abgebildet. Erkennbar ist, dass der Punkt in der Greiferrille bereits entfernt wurde und der letzte Punkt der Skizzenkontur um 3 mm in Richtung der negativen y-Achse verschoben wurde. Des Weiteren ist jener Punkt zu erkennen, welcher die vertikale Linie des Überstands der Adapteraufnahme darstellt. Er befindet sich schräg rechts über dem letzten Punkt und muss im Zuge der weiteren Arbeiten gelöscht werden.

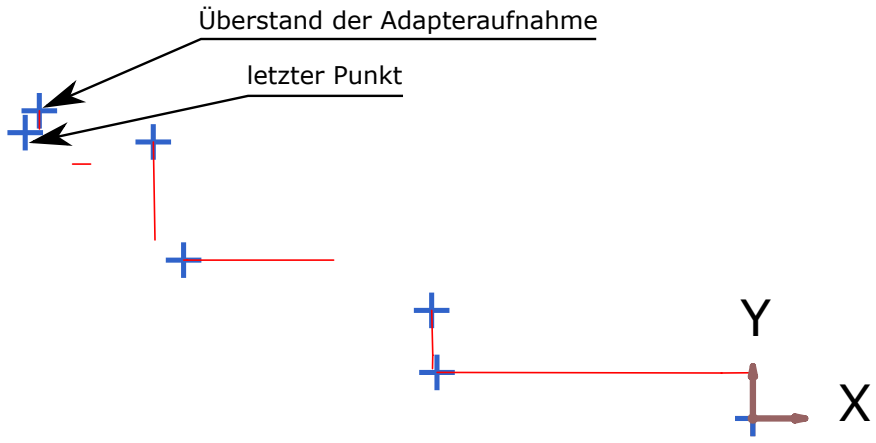


Abbildung 23: Relevante Endpunkte nach der Beurteilung
Quelle: Eigene Darstellung.

Nachdem die Punkte beurteilt wurden, werden alle Linien automatisch gelöscht. Sie werden für das weitere Vorgehen nicht mehr benötigt. Die Linien wurden lediglich zur Einteilung in relevante Linien und zum Auslesen der Endpunkte verwendet. Für die Erstellung der Skizze sind genau sechs Punkte notwendig. Es besteht die Möglichkeit, dass bei der Beurteilung mehr als sechs Punkte für relevant gehalten werden. Damit bei der Skizzenerstellung kein Fehler auftritt und der letzte Punkt, welcher ein elementarer Bestandteil der Skizzenerstellung ist, aus Versehen vernachlässigt wird, erfolgt das Einbinden einer Abfrage. Diese begrenzt die Verwendung der Punkte auf die ersten fünf und den letzten der Liste. Damit wird sichergestellt, dass der letzte Punkt stets für die Skizzenerstellung berücksichtigt wird. Mögliche Probleme können nur kurz vor dem letzten Punkt in der Nähe der Greiferrille entstehen. Hier könnte der Fall eintreten, dass mehr Punkte als notwendig für relevant gehalten werden. Diese Abfrage bietet eine einfache Möglichkeit, dieses Problem zu lösen.

Nachdem die sechs Punkte feststehen, kann mit der Skizzenerstellung fortgefahren werden. Zuvor werden jedoch die Koordinaten der Punkte einer Rundung unterzogen. Alle mit Ausnahme jener des Schaftfräsers werden stets aufgerundet. Somit wird für den Schaftfräser das tatsächliche Maß erreicht und der Halter mit einer kleinen Sicherheit behaftet. Es wird immer darauf geachtet, dass der Halter auf keinen Fall zu klein dargestellt wird. Bei der Kollisionsüberprüfung wird das Digitale Modell des Komplettwerkzeugs zusätzlich immer mit einer Hüllfläche versehen, sodass es bei der Fertigung zu keiner Kollision kommt. Nach Abschluss dieses Vorgangs kann mit der tatsächlichen Skizzenerstellung begonnen werden. Für die Skizzenerstellung werden, wie bereits erwähnt, sechs Punkte benötigt. Diese sind in Abbildung 24 zusammen mit der erzeugten Skizze dargestellt. Anhand dieser Abbildung wird die Vorgehensweise zur Erstellung erläutert.

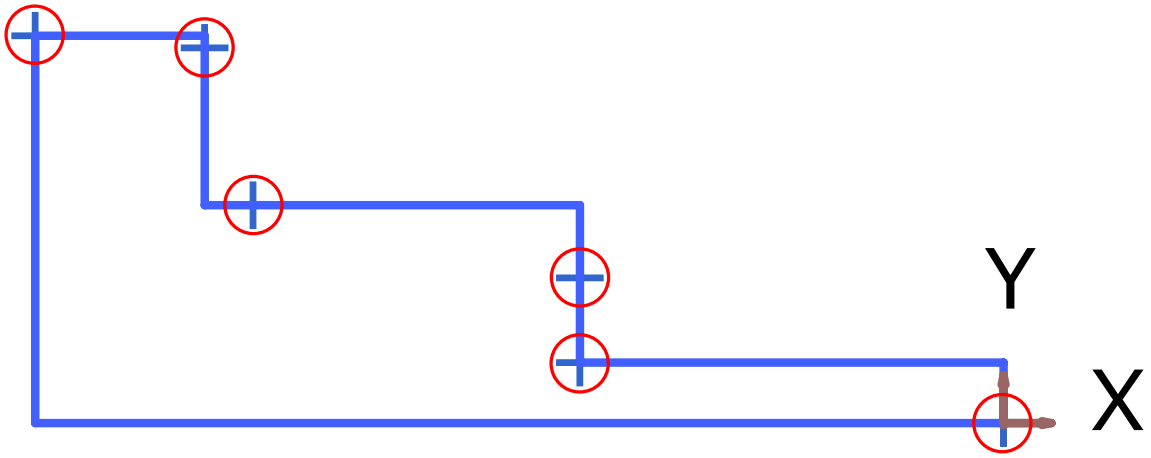


Abbildung 24: Skizze mit den relevanten Punkten zur Erzeugung
Quelle: Eigene Darstellung.

In dieser Abbildung sind die sechs Punkte mit den roten Kreisen gekennzeichnet und die Linien der Skizze in Blau dargestellt. Die Skizzenerstellung beginnt immer im Ursprung. Durch die vorgegebene Kontur muss die erste Linie vom Ursprung aus vertikal in positive y-Richtung verlaufen. Die Stufenform der Skizze entsteht durch abwechselnd vertikale und horizontale Linien. Bei der Skizzenerstellung werden die Punkte derart eingeteilt, dass eine gerade Indexzahl der Punkteliste für eine folgende vertikale Linie und eine ungerade Indexzahl für eine folgende horizontale Linie steht. So kann sichergestellt werden, dass die Linie stets mit der richtigen Koordinatenkombination erstellt wird. Für die Erzeugung einer

Linie werden je ein Start- und ein Endpunkt benötigt. Den Startpunkt bildet zu Beginn der erste Punkt der Liste, also der Punkt im Ursprung, und in weiterer Folge immer der Endpunkt der vorangegangenen Linie. Der Endpunkt der Linie entsteht durch eine Koordinatenkombination. Im Fall einer folgenden vertikalen Linie wird der Endpunkt durch die x-Koordinate des Startpunkts der Linie und die y-Koordinate des nächsten Punkts in der Liste definiert. Bei einer horizontalen Linie wird für die Definition des Endpunkts die x-Koordinate des nächsten Punkts und die y-Koordinate des Startpunkts der Linie herangezogen. Mit diesem Verfahren wird bis zum letzten Punkt vorgegangen.

Sobald der Polygonzug in Stufenform erstellt wurde, wird die Skizze zu einem geschlossenen Linienzug vervollständigt. Für die Drehoperation zum Erzeugen eines Volumenkörpers um eine definierte Achse ist eine geschlossene Skizze notwendig. Hierfür wird mit der x-Koordinate des Endpunkts des Linienzugs und der y-Koordinate des Ursprungs ein Punkt erzeugt, welcher eine Verbindung durch einen rechten Winkel ermöglicht. Im Anschluss wird mit Hilfe des erzeugten Punkts die vertikale Linie mit dem Endpunkt der Stufenform und die horizontale Linie mit dem Startpunkt der Stufenform erstellt. Daraus ergibt sich die fertige Skizze, welche in Abbildung 24 bereits veranschaulicht wurde. Dieser Vorgang wird automatisiert abgewickelt, sodass keine Eingriffe durch die Benutzerin oder den Benutzer notwendig sind.

3.5.1.3 Überprüfung

Damit die Skizze nicht ohne Überprüfung der Anwenderin oder des Anwenders zur Erstellung der Volumenkörper verwendet wird, wird der ursprüngliche Werkzeugscan ein erneutes Mal geladen. Durch diesen Vorgang werden beide Skizzen übereinandergelegt. Somit kann schnell beurteilt werden, ob die Skizze richtig erstellt wurde oder Korrekturen vorzunehmen sind. In Abbildung 25 ist dies veranschaulicht.

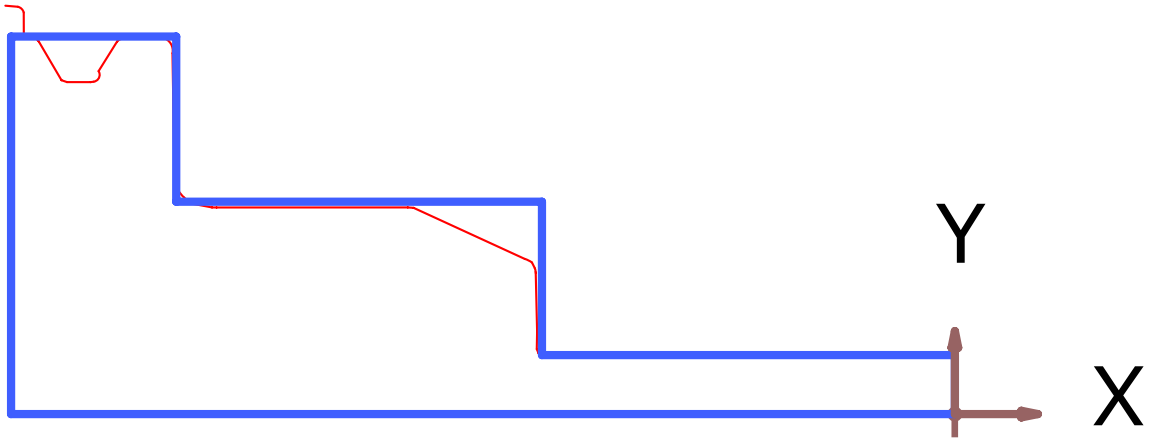


Abbildung 25: Darstellung der übereinandergelegten Skizzen
Quelle: Eigene Darstellung.

Hier ist zu sehen, dass im Bereich des Schaftfräsers die erstellte Skizze den Werkzeugscan exakt abbildet, und dass sich im Bereich des Halters durch das Runden der Koordinaten eine minimale Abweichung ergibt. In diesem Fall erfüllt die erstellte Skizze die Anforderungen vollständig und kann für weitere Schritte verwendet werden. Wenn die Skizze den Anforderungen nicht entspricht, kann sie durch Bearbeitung der Maße korrigiert werden. Entspricht das Ergebnis den Erwartungen, kann mit der Durchführung der zweiten Schaltfläche fortgefahren werden.

3.5.2 Automatisierte Generierung von Volumenkörpern

Nachdem in Unterabschnitt 3.5.1 die Skizzenerstellung beschrieben wurde, erfolgt nachstehend die Beschreibung zur Erzeugung der Volumenkörper. Dieser Teil kann wiederum in zwei Abschnitte gegliedert werden. Dabei werden Maßnahmen behandelt, welche einerseits zur Erzeugung der Volumenkörper und andererseits für kleinere Anpassungen am Modell notwendig sind. Die Vorgehensweise wird in den nächsten zwei Unterabschnitten beschrieben.

3.5.2.1 Erzeugung

Der Inhalt dieses Unterabschnitts befasst sich mit der Erstellung von zwei getrennt voneinander verfügbaren Komponenten in Siemens NX und der Erstellung der Volumenkörper aus den zuvor generierten Skizzen. Dazu ist die einfache Rotation der Skizze um die eigene Achse nicht ausreichend, vielmehr ist eine Separierung der Skizze für den Halter und den Schaftfräser bereits vor der Rotation vorzunehmen. Um eine bessere Übersicht bei der Programmierung zu erhalten, werden zwei Hauptklassen (Werkzeug und Halter) für den Aufbau des Programms erstellt. In diesen sind immer die zugehörigen Funktionen der jeweiligen Komponente enthalten.

Bei der Ausführung des ersten Teilprogramms zur Erstellung der Skizze wurde gegen Ende der Werkzeugscan nochmals importiert. Dieser Schritt wurde zur Überprüfung durch die Anwenderin oder den Anwender eingefügt und wird in diesem Teilprogramm nicht mehr benötigt. Infolgedessen ist es nun die erste Aufgabe, diesen Scan aus dem Bauteil zu entfernen. Die zentrale Frage dabei ist, wie die Linien angesprochen werden können. Sie wurden weder in diesem Teilprogramm erstellt und einer Variablen zugewiesen noch sind sie mit Namen bezeichnet, unter welchen sie gefunden werden können. Dementsprechend musste eine andere Methode gefunden werden. Als Lösung dieses Problems wurde auf das Prinzip der Layer zurückgegriffen. Dieses ermöglicht beim Import das Zuweisen ausgewählter Elemente zu definierten Layer. Die Linien wurden auf einen zusätzlichen Layer gelegt, indem der aktuelle Layer durch Einstellungen beim Import umgangen wird. Dies hat den Vorteil, dass die Linien in dieser Situation mit Hilfe des zusätzlichen Layer angesprochen werden können. Dadurch können die nicht benötigten Linien entfernt werden. Für die Entfernung wird auf die Methode im ersten Teilprogramm zurückgegriffen. Die Linien werden einer Liste hinzugefügt und im Anschluss durch eine Aktualisierung des Modells gelöscht.

Anschließend wird die Skizze verschoben. Der in Unterabschnitt 3.5.1.2 erstellte Punkt zur Schließung der Skizze soll im Ursprung liegen. Dazu wird die vorhandene Skizze als Objekt angesprochen und verschoben. Nachdem dies abgeschlossen ist, wird mit der Separierung der Skizze in zwei einzelne (Werkzeug und Halter) fortgefahren. Hierfür werden im Vorhinein zwei Listen benötigt. Eine Liste repräsentiert die Linien des Werkzeugs und die andere die des Halters. Durch das Wissen, wie die Skizze erstellt wurde, fällt die Separierung der Skizze einfacher aus. Die erste Linie der Skizze muss sich an der Stirnseite des Schaftfräsers

befinden. Folglich muss diese Linie beim Durchlaufen aller Linien des Bauteils an erster Stelle stehen. Damit können die Linien in den jeweiligen Listen abgespeichert werden. Die ersten beiden werden der Liste für das Werkzeug und die folgenden der für den Halter zugewiesen. Die Listen sind in den jeweiligen Klassen (Werkzeug und Halter) angelegt. Aufgrund der Tatsache, dass in weiterer Folge zwei einzelne Komponenten entstehen sollen, müssen der Schaftfräser entsprechend verlängert und der Halter mit einer Bohrung versehen werden. Somit kann der Fräser in der Bohrung des Halters fixiert werden. Die Konstruktion der Bohrung im Halter wird bereits bei der Skizzenerstellung implementiert, sodass diese bei der Rotation der Skizze um eine definierte Achse entsteht. Dazu wird die letzte Linie der Skizze, welche im ersten Teilprogramm zum Schließen der Skizze entlang der Rotationsachse erzeugt wurde, durch die Bohrung ersetzt und kann aus der Liste des Halters entfernt werden. An dieser Stelle wird auf die Erstellung der Bohrung auf einen späteren Zeitpunkt dieses Abschnitts verwiesen. Gegenwärtige Aufgabe war die Trennung der Linien in zwei einzelne Listen zur weiteren Bearbeitung. Zur Veranschaulichung werden die Linien in Abbildung 26 für das Werkzeug mit durchgehender und für den Halter in unterbrochener Linie dargestellt.

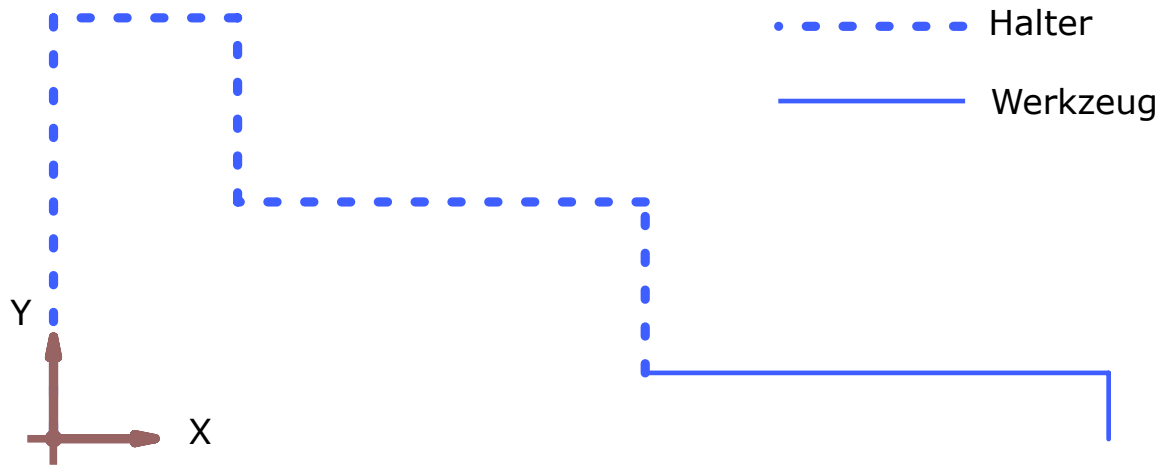


Abbildung 26: Separierung der Linien in die Komponenten
Quelle: Eigene Darstellung.

Siemens NX bietet die Möglichkeit, dem aktuellen Bauteil neue Komponenten hinzuzufügen. Dadurch wird das Bauteil automatisch zu einer Baugruppe. Es können beliebig viele Komponenten hinzugefügt werden, sodass die Baugruppe um die neuen Komponenten erweitert wird. Für das Erstellen neuer Komponenten wurde die Journalaufzeichnung

verwendet. Mit deren Hilfe werden die Komponenten erstellt und notwendige Einstellungen wie beispielsweise das Vergeben eines Namens getroffen. Diese Vorgehensweise wird zuerst für das Werkzeug und anschließend für den Halter durchgeführt. Die Prozeduren sind in den jeweiligen Klassen hinterlegt und werden für das Ausführen aus dem Hauptprogramm aufgerufen. Nachdem die Komponenten erstellt wurden, können weitere Befehle an ihnen durchgeführt werden. Dazu sind sie im Vorhinein zu aktivieren, sodass das Programm nachvollziehen kann, wo genau Änderungen vorzunehmen sind. Der Worst-Case stellt die Ausführung von Operationen an Objekten, welche sich nicht in dieser Komponente befinden, dar. Es entsteht ein Fehler und der Code kann nicht weiter ausgeführt werden. Nachdem das Werkzeug als Komponente definiert wurde, wird mit der Erstellung einer neuen Skizze fortgefahren. Anschließend werden die Linien des Werkzeugs, welche zuvor in der Liste gespeichert wurden, hinzugefügt und um 2 mm in negative x-Richtung verschoben. Der Fräser kann beim Zusammenbau durch verschiedenste Einflüsse nicht an der exakten Stelle im Halter positioniert werden, sodass sich ein Toleranzfeld bei der Fixierung ergibt. Zur Vermeidung einer Kollision an der Werkzeugmaschine wird der Fräser hier um 2 mm für die CAM-Simulation gekürzt. Im Anschluss wird das bereits bekannte Verfahren von Unterabschnitt 3.5.1.2 zur Schließung einer Skizze eingesetzt. Das Ergebnis ist ein Rechteck, welches zur Rotation um eine Achse verwendet werden kann. Des Weiteren müssen die Linien, welche zum Schließen der Skizze erstellt wurden, der Skizze des Werkzeugs zugewiesen werden. In Abbildung 27 ist das Rechteck dargestellt, außerdem ist der Ursprung der Skizze gekennzeichnet, welcher für das weitere Vorgehen benötigt wird.

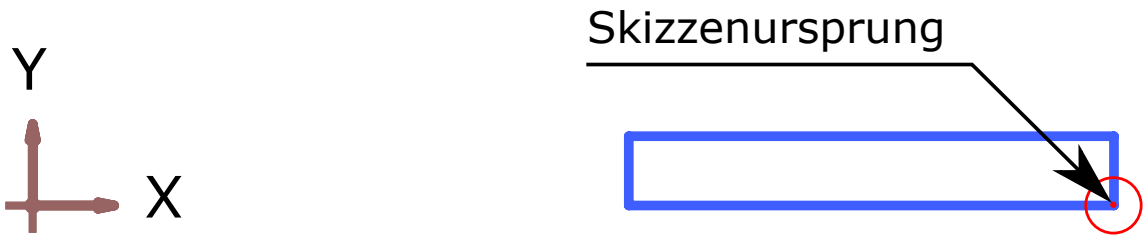


Abbildung 27: Geschlossene Skizze des Werkzeugs
Quelle: Eigene Darstellung.

Zur Parametrisierung der Skizze werden Expressions erstellt. Dazu wird zu Beginn eine Koinzidenz zwischen dem Startpunkt der ersten Linie der Skizze sowie dem Ursprung der Skizze hergestellt. Dies ist in Abbildung 27 veranschaulicht. Außerdem wird neben dieser Zwangsbedingung eine Bemaßung für die vertikale Linie in Form einer Expression angelegt. Diese stellt den Radius des Fräsers dar. Analog dazu wird für die Länge des Schaftfräsers eine Expression mit Hilfe der horizontalen Linie angelegt. Der Fräser wird in der Bohrung des Halters fixiert, folglich muss das Werkzeug verlängert werden, da es nur bis zur Stirnseite des Halters reicht. Dies ist auf den Werkzeugscan des Komplettwerkzeugs zurückzuführen, welcher den Teil des Fräsers im Halter sinngemäß nicht erfassen kann. Dabei ist es von geringer Bedeutung, wie tief der Fräser im Halter fixiert ist, da diese Stelle keinen Einfluss auf die Kollisionsüberprüfung hat. Somit wurde die angelegte Expression, welche die Länge des Fräsers definiert, mit dem Faktor 1,5 multipliziert. Neben der Koinzidenz, der Definition der Länge und des Radius werden weitere Zwangsbedingungen, nämlich gleiche Länge für die horizontalen und vertikalen Linien, definiert. Beim Anlegen von Expressions ist darauf zu achten, dass der Datentyp ein String ist. Wird der Zahlenwert zuerst durch mathematische Operationen berechnet, muss der Datentyp im Anschluss in einen String konvertiert werden. In Abbildung 28 sind die Expressions für die vertikale und die horizontale Linie gekennzeichnet.

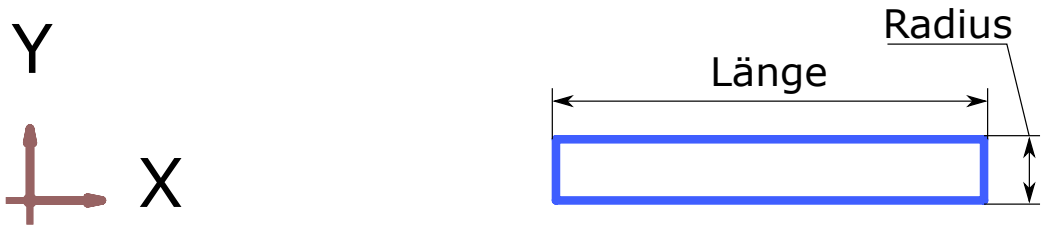


Abbildung 28: Definition der Expressions
Quelle: Eigene Darstellung.

An dieser Stelle ist die Skizze des Werkzeugs für eine Rotation zur Volumenkörpererstellung fertig aufbereitet. Dieser Schritt wird mit Hilfe der Journalaufzeichnung und der beschriebenen Methode im Handbuch „Getting Started with NX Open“ realisiert.[73] Nachdem der Volumenkörper erstellt ist, sind die Bearbeitungen am Fräser abgeschlossen und es kann mit dem Halter fortgefahren werden.

[73] Vgl. Siemens PLM Software (2019).

Dazu muss dieser als aktuelle Komponente aktiviert werden. Grundsätzlich erfolgt die Bearbeitung des Halters analog zum Werkzeug. Am Ablauf ändert sich nichts, jedoch ist bei der Schließung der Skizze in einer anderen Weise vorzugehen, da die Bohrung für den Schaftfräser erzeugt werden muss. Dazu wird die Bohrung tief genug angenommen, sodass der Fräser beim Zusammenbau in jedem Fall Platz findet. Nachdem auch diese Skizze erfolgreich bearbeitet und für die Rotation vorbereitet wurde, kann diese durchgeführt werden. Die fertig geschlossene Skizze ist in Abbildung 29 dargestellt. Hier ist die Ausnehmung für die Bohrung an der x-Achse deutlich sichtbar.

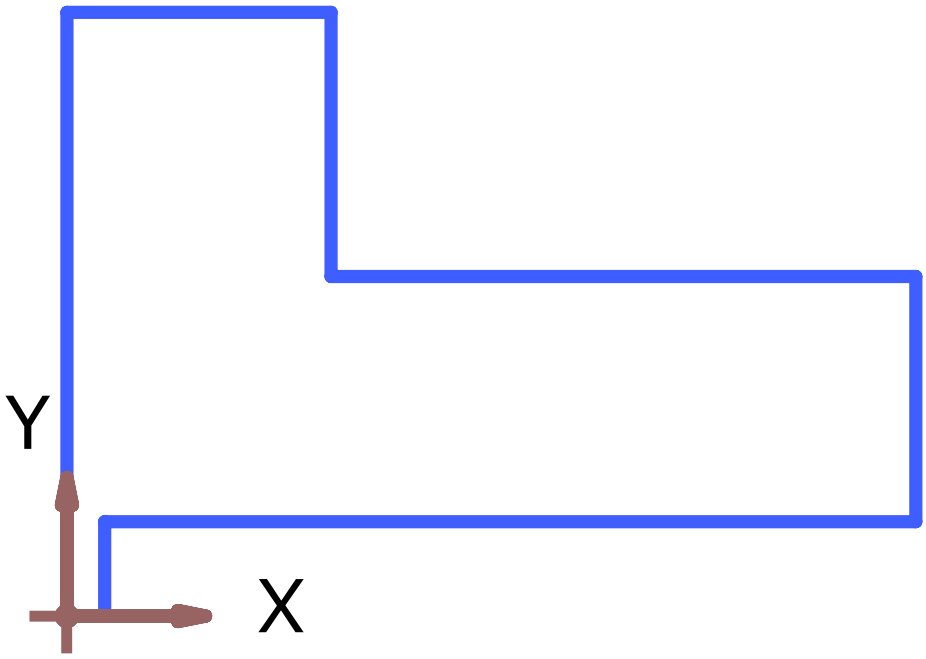


Abbildung 29: Geschlossene Skizze des Halters
Quelle: Eigene Darstellung.

Schließlich sind die beiden Volumenkörper in getrennten Komponenten erstellt. Sie befinden sich in einer gemeinsamen Baugruppe und können nach Belieben gespeichert werden. Das Speichern einer einzelnen Komponente sowie der Baugruppe ist manuell vorzunehmen. In Abbildung 30 sind die erstellten Körper als Baugruppe dargestellt.

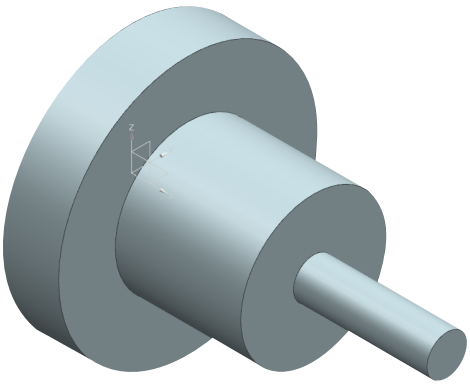


Abbildung 30: Erzeugte Volumenkörper
Quelle: Eigene Darstellung.

3.5.2.2 Modifikationen

Nach der Erstellung der Volumenkörper werden lediglich kleine Anpassungen am Modell vorgenommen. Dabei handelt es sich unter anderem um das Trennen des Schaftfräsers in zwei Teile (schneidend und nicht schneidend), das Einfärben von Teilen, das Einfügen von Koordinatensystemen sowie das Erstellen einer Fase oder Rundung. Diese Operationen werden in Folge beschrieben.

Zunächst wird der Schaftfräser als aktuelles Bauteil aktiviert. Anschließend erfolgt eine Trennung in zwei Teile, welche einer schneidenden und einer nicht schneidenden Komponente entsprechen. Dazu wird die SplitBody-Funktion in Siemens NX verwendet. Diese wurde mit Hilfe der Journalaufzeichnung so weit angepasst, dass sie für diesen Fall anwendbar ist. Zu diesem Zweck wird eine Bezugsebene senkrecht zur Rotationsachse an der Spitze des Fräsers eingefügt. Danach wird anhand dieser Ebene eine Teilung des Fräsers im Abstand des doppelten Durchmessers durchgeführt und dies gleichzeitig mit einer Expression als Länge der schneidenden Komponente angelegt. Dadurch kann im Anschluss eine Anpassung vorgenommen werden, da diese Charakteristik aus dem Scan nicht hervorgeht. Die Art der Berechnung liefert einen ersten Schätzwert, welcher für viele Fräser passend ist. Nachdem die Teilung des Fräsers erfolgt ist, wird die Anwenderin oder der Anwender vom Programm aufgefordert, die schneidende Komponente des Fräsers auszuwählen. Daraufhin wird sie gelb gefärbt, während die nicht schneidende Komponente

unberührt bleibt. Der geteilte Fräser mit der gefärbten schneidenden Komponente ist in Abbildung 31 zu sehen.

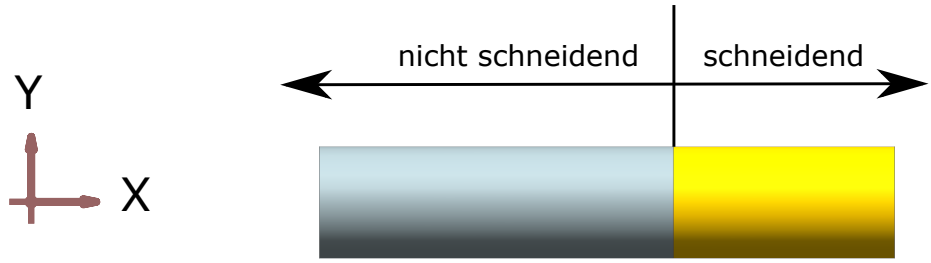


Abbildung 31: Schneidende und nicht schneidende Komponente des Fräsers
Quelle: Eigene Darstellung.

Außerdem werden in den beiden Komponenten jeweils Koordinatensysteme eingefügt, sodass sie in einer Werkzeugdatenbank durch Connectioncodes miteinander verbunden werden können. Im Werkzeug wird ein maschinenseitiges Koordinatensystem platziert. Dieses liegt im Schnittpunkt der Rotationsachse des Fräsers mit der Stirnseite des Halters. An derselben Stelle wird für den Halter ein werkstückseitiges Koordinatensystem erzeugt. Des Weiteren wird beim Halter ein maschinenseitiges Koordinatensystem erstellt, welches sich im Ursprung befindet.

Zusätzlich bietet das Programm der Benutzerin oder dem Benutzer die Möglichkeit, eine Fase beziehungsweise eine Rundung an der Spitze des Schaftfräsers zu erstellen. Dazu wird ein Dialogfenster geöffnet, welches bereits beim Import des Werkzeugscans verwendet wurde, in dem die Benutzerin oder der Benutzer gefragt wird, ob eine Fase erstellt werden soll. Bei der Akzeptierung wird eine Fase mit 45° x 0,1 mm erstellt und das Programm ist abgeschlossen. Des Weiteren werden Expressions zur Definition der Fase angelegt. Im Falle der Ablehnung der Fasenerzeugung besteht die Möglichkeit, eine Rundung auf demselben Prinzip zu erstellen. Die Maße der Fase und der Rundung wurden mit Annahmen erstellt und müssen durch die Benutzerin oder den Benutzer angepasst werden. Vorteil dieser Methode ist, dass die Fase oder Rundung nicht manuell zu erstellen ist, sondern lediglich die Expressions angepasst werden müssen. Werden beide Operationen abgelehnt, verbleibt die Kante an der Spitze des Fräsers.

Bevor das Programm beendet wird, werden an dieser Stelle alle noch notwendigen Expressions für den folgenden Export angelegt. Diese werden im nächsten Unterabschnitt behandelt. Somit ist das Programm durchlaufen und die fertigen Volumenkörper sind in Siemens NX verfügbar. In Abbildung 32 ist das erstellte Modell eines Schaftfräsers mit einem Durchmesser von 10 mm dargestellt.

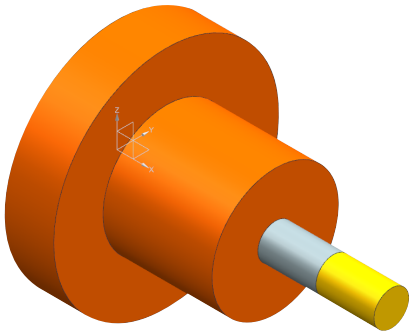


Abbildung 32: Digitales Modell eines Schaftfräsers mit Durchmesser 10 mm
Quelle: Eigene Darstellung.

3.5.3 Export der Expressions

In diesem Unterabschnitt wird der letzte Teil des Programms behandelt. Dieser führt einen Export von Expressions in ein Tabellenblatt nach Excel durch und ist ebenso wie die anderen Teilprogramme mit einer Schaltfläche verknüpft. Dazu wurden bereits in Unterabschnitt 3.5.2.2 die erforderlichen Expressions angelegt. Diese werden in eine vorhandene Excel Datei exportiert, sodass die Daten anschließend in die MRL übertragen und anhand dieser ein Werkzeug anlegt werden kann. Diese Daten werden beispielhaft in Abbildung 33 anhand einer Skizze veranschaulicht. Des Weiteren müssen Parameter für die Anzahl der Schneiden beziehungsweise der Zentrumsschneiden ausgegeben werden. Dazu werden drei Schneiden und eine Zentrumsschneide exportiert, welche im Bedarfsfall angepasst werden müssen. Das Auslesen dieser Charakteristiken aus dem Werkzeugscan ist nicht möglich.

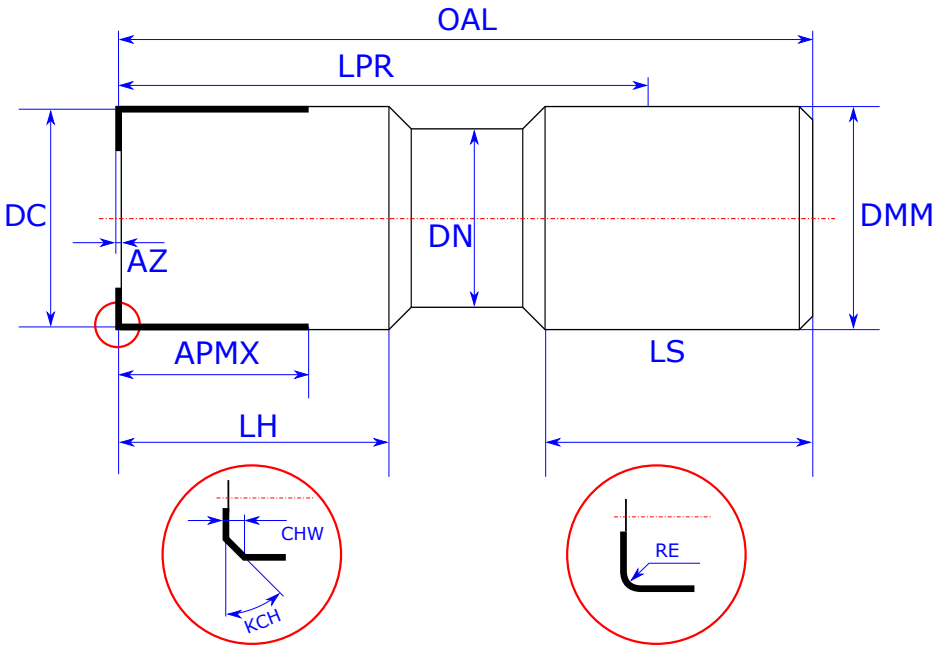


Abbildung 33: Expressions für den Export
Quelle: Siemens PLM Software (2021b), (leicht modifiziert).

3.6 Evaluierung der Ergebnisse

Die Programmerstellung wurde generell anhand eines Schaftfräsers mit einem Durchmesser von 10 mm erläutert. Des Weiteren wurde die beschriebene Methodik auch bei Schaftfräsern mit abweichenden Durchmessern getestet (6 mm, 8 mm, 12 mm, 16 mm). In diesem Abschnitt werden die Ergebnisse der Versuche miteinander verglichen und eine Evaluierung durchgeführt.

Am Voreinstellgerät werden drei Scanvorgänge für jeden Durchmesser durchgeführt. Durch diese Vorgehensweise wird die Wahrscheinlichkeit einer Fehlmessung minimiert und eine Reproduzierbarkeit der Ergebnisse erhöht. Bei fünf Fräsern ergeben sich somit 15 Durchgänge. Dabei wird das Hauptaugenmerk auf die Skizzenerstellung gelegt. Die automatisierte Methode muss in der Lage sein, ausgehend vom Werkzeugscan eine für weitere Zwecke anwendbare Skizze zu erstellen, da in dieser Phase die Fehleranfälligkeit am höchsten ist. Alle weiteren Operationen werden anhand der Skizze durchgeführt und

stellen weniger Fehlermöglichkeiten dar. Die Ergebnisse der jeweiligen Versuche sind in Tabelle 6 visualisiert. Dabei wird ein Haken mit einem erfolgreichen Durchlauf von der Skizzenerstellung bis zum Export assoziiert. Ein Kreuz steht für eine Fehlinterpretation bei der Skizzenerstellung. Alle weiteren Vorgänge werden fehlerfrei vollzogen.

Tabelle 6: Ergebnisse der Programmausführung
Quelle: Eigene Darstellung.

Fräser	**1. Versuch**	**2. Versuch**	**3. Versuch**
6 mm	✓	✓	✓
8 mm	✓	✓	✓
10 mm	✓	✓	✓
12 mm	✓	✓	×
16 mm	×	×	×

Beim dritten Versuch des Schaftfräsers mit Durchmesser 12 mm tritt ein Fehler bei der Skizzenerstellung auf. Das Programm erkennt die erforderliche Kontur des Werkzeugs nicht exakt und erzeugt für die Skizze des Halters ein Rechteck. Grundsätzlich kann auch mit diesem fortgefahren werden, jedoch ist das Ergebnis des Digitalen Modells des Halters im Vergleich zu den anderen einfacher gehalten. Bei einer Kollisionsüberprüfung würde der digitale Halter im Vergleich zum realen Halter früher kollidieren. Dieser Umstand führt zu keinem Prozessrisiko während der Fertigung, jedoch kann auch das Potential nicht vollkommen ausgeschöpft werden. Der Linienzug dieses Versuchs ist in Abbildung 34 dargestellt. Hier ist die Abweichung vom Werkzeugscan an der gekennzeichneten Problemstelle deutlich sichtbar.

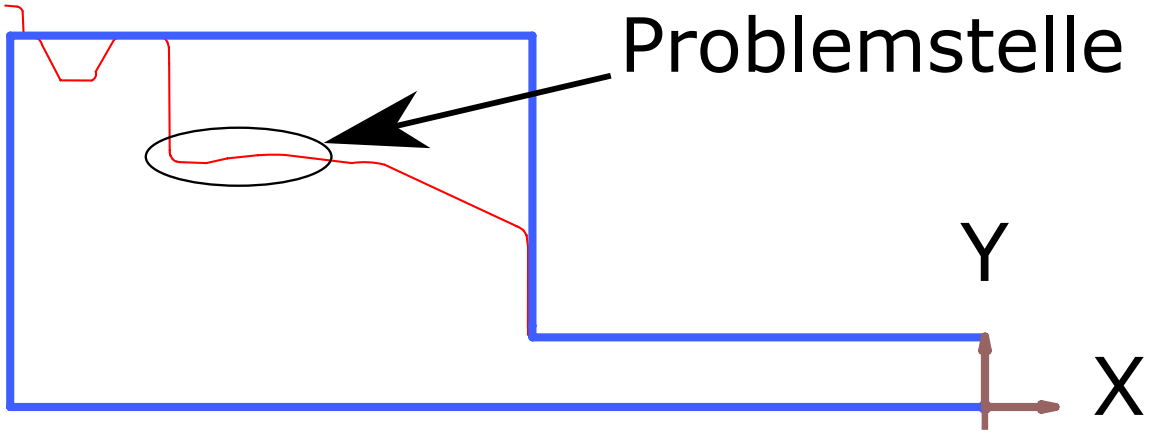


Abbildung 34: Skizze des dritten Versuchs mit Durchmesser 12 mm
Quelle: Eigene Darstellung.

Grundsätzlich kann der Bogen an der gekennzeichneten Problemstelle mit der Wurmschraube der Weldon-Aufnahme in Verbindung gebracht werden. Beim realen Halter ragt die Schraube der Weldon-Aufnahme leicht über die Zylinderkontur und wird vom Voreinstellgerät beim Scan tatsächlich erkannt. Im direkten Vergleich zu den anderen Durchgängen dieses Durchmessers zeigt sich, dass dieser Bogen bei allen Scans vorhanden ist. Er kann demnach keine Auswirkung auf die Erstellung der Skizze haben, vielmehr liegt das Problem an der Linie, welche in Abbildung 34 zusammen mit dem Bogen durch ein Oval gekennzeichnet ist. Im dritten Versuch verfügt die Linie über einen Winkel, welcher der Beurteilung des Programms zufolge außerhalb der Toleranz liegt. Insofern ist die Skizzenerstellung fehlerbehaftet, da die Linie als nicht relevant eingestuft wird. Bei den ersten beiden Versuchen liegt sie innerhalb der Toleranz.

In Tabelle 6 ist außerdem ersichtlich, dass beim Fräser mit Durchmesser 16 mm Komplikationen auftraten. Die drei Durchgänge erfüllen die Anforderungen fast zur Gänze und beinhalten alle den gleichen Fehler. Dieser wird anhand der Skizze des ersten Versuchs erläutert. In Abbildung 35 ist die Skizze dieses Fräsers für den ersten Durchgang dargestellt.

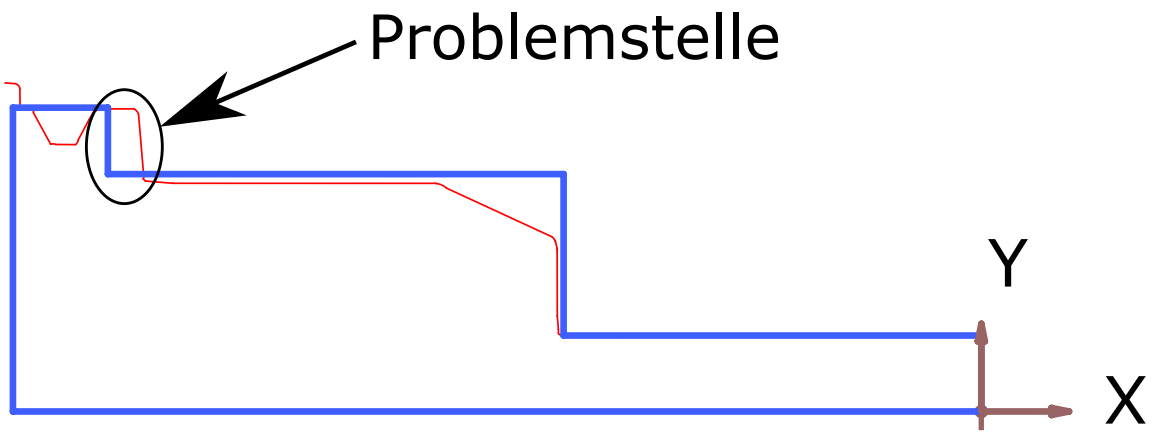


Abbildung 35: Skizze des ersten Versuchs mit Durchmesser 16 mm
Quelle: Eigene Darstellung.

Hier wird an der gekennzeichneten Problemstelle die Linie als nicht vertikal angesehen. Aus diesem Grund kommt es zu einer leichten Abweichung bei der Skizzenerstellung. Gründe hierfür können Verunreinigungen sein, welche auf Späne oder Staub zurückzuführen sind. Festzuhalten ist, dass dies Auswirkungen auf die Kollisionsüberprüfung hat, da das Modell kleiner als das physische Werkzeug ausfällt. Es besteht die Möglichkeit, dass es zu einer Kollision des Halter bei der Fertigung kommt, obwohl bei der Kollisionsüberprüfung keine Fehler festgestellt werden konnten.

4 Zusammenfassung, Fazit und Ausblick

In diesem Kapitel werden die durchgeführten Untersuchungen sowie die daraus resultierenden Ergebnisse zusammengefasst. Dabei wird speziell auf das eingangs formulierte Forschungsziel eingegangen. Des Weiteren werden angewandte Herangehensweisen und ausgewählte Ergebnisse kritisch hinterfragt und mit den theoretischen Grundlagen in Verbindung gebracht. Zum Abschluss wird im Ausblick auf wissenschaftliche Untersuchungen, die weiterführend erfolgen können, hingewiesen.

4.1 Zusammenfassung

Die Ausgangssituation für die vorliegende Arbeit ist die CAD/CAM/CNC-Prozesskette der smartfactory@tugraz, einer Pilotfabrik an der Technischen Universität Graz. In dieser findet die Erstellung von Digitalen Modellen physischer Werkzeuge manuell statt. Im Mittelpunkt dieser Arbeit steht die Optimierung dieses Prozesses, welche die Entwicklung einer automatisierten Methode zur Generierung eines digitalen Werkzeugs darstellt. Diesbezüglich ist die Aufbereitung für eine CAM-Simulation von Relevanz. Mittels vorhandenem Voreinstellgerät wird ein Werkzeugscan für ein Komplettwerkzeug durchgeführt und dessen Resultat als Modell im 2D-Format exportiert. Der Scan besteht aus Linien und Bögen und bildet den Ausgangspunkt für die Entwicklung der automatisierten Methode. Hierfür wurde die Programmierschnittstelle NX Open eingesetzt, welche einen Zugriff auf die Funktionen des verwendeten CAD/CAM-Systems Siemens NX ermöglicht.

Durch Analyse des Scans können Punkte für die Erstellung einer eigenen Skizze gefunden werden. Dazu werden mit Hilfe der Berechnung der Steigung der Linien und der Vorgabe eines Toleranzbereichs die relevanten Linien gefunden und deren Endpunkte ausgelesen. Im Anschluss erfolgt eine Beurteilung, mit welcher die benötigten Punkte für die Skizzenerstellung gefunden werden. Die angewandte Herangehensweise zur Erstellung der Skizze beruht auf der abwechselnden Kombination der Koordinaten der relevanten Punkte zur Erstellung eines Polygonzugs. Das Resultat ist eine eigene Skizze in Stufenform, welche den Scan vereinfacht wiedergibt. Diese stellt die Grundlage für die Separierung des Komplettwerkzeugs in dessen Einzelkomponenten dar. Danach erfolgte die Volumenkörpererstellung durch Rotation der Skizzen um die relevanten Achsen. Zum Abschluss fanden Anpassungen an den Modellen statt und die Durchführung von vorbereitenden Maßnahmen für den Export der relevanten Daten in ein Tabellenblatt wurde forciert.

4.2 Fazit

Die genaue Betrachtung des generierten Modells liefert die Erkenntnis, dass kein automatischer Datenfluss zwischen dem digitalen Objekt und dem physischen Objekt besteht. Durch die eindeutige Definition eines Digitalen Zwillings, Digitalen Schattens und Digitalen Modells nach deren Grad der Datenintegration ist in dieser Arbeit von einem Digitalen Modell auszugehen.

Die Herangehensweise zur Skizzenerstellung liefert die erwarteten Ergebnisse. Mit Hilfe der Kombination der Koordinaten der relevanten Punkte können die Schlüsselstellen der erstellten Skizze hinreichend genau abgebildet werden. Zur Reduktion von Fehlmessungen und zur Reproduzierbarkeit der Ergebnisse wurden die wissenschaftlichen Untersuchungen bei fünf verschiedenen Schaftfräsern zu jeweils drei Versuchen durchgeführt. Alle lieferten grundsätzlich positive Ergebnisse.

Bei einem Durchmesser von 12 mm ergibt sich durch die Schraube der Weldon-Aufnahme, welche über die Zylinderkontur ragt, ein Bogen beim generierten 2D-Modell. Dennoch kann die automatisierte Methode mit diesem Merkmal umgehen. Lediglich beim letzten Versuch dieses Durchmessers konnte ein Problem festgestellt werden. Dabei wird die Linie nach der Weldon-Aufnahme, dessen Winkel außerhalb des vorgegebenen Toleranzbereichs liegt, als nicht relevant erkannt. Dadurch entsteht ein Digitales Modell, welches das Potential einer CAM-Simulation nicht vollständig ausschöpfen kann. Die beiden anderen Durchgänge liefern die erwarteten Ergebnisse. Ähnlich verhält sich der Fräser mit einem Durchmesser von 16 mm. Bei diesem wird die eine Linie durch einen Winkel außerhalb des vorgegebenen Toleranzbereichs nicht erkannt. Dadurch entsteht eine minimal abweichende Skizze, welche durch geringe Nachbesserung zum gewollten Ergebnis führt.

Die durchgeführte Arbeit ist für die Industrie durchaus von Bedeutung. Der Drang nach Produktindividualisierung zwingt Produktionsbetriebe dazu, eine Vielfalt an Varianten anzubieten. Parallel dazu führen diese Maßnahmen zu einer Vielzahl an verschiedenen Werkzeugen. In Zusammenhang mit einer Werkzeugdatenbank kommt der automatisierten Generierung von Digitalen Werkzeugmodellen eine bedeutende Rolle zu. Das Potential wird bei der Anschaffung einer neuen Werkzeugdatenbank vollständig ausgeschöpft. Dabei kann der Zeitraum der Überführung des gesamten Werkzeugbestands in die Werkzeugdatenbank deutlich reduziert werden. Schlussfolgernd zeigt sich, dass der Zeitaufwand mit der in dieser Arbeit entwickelten Methode bei der Generierung von Werkzeugmodellen für eine CAM-Simulation reduziert werden kann. Außerdem wird die Fehleranfälligkeit durch menschliches Versagen reduziert.

4.3 Ausblick

Die in dieser Arbeit entwickelte Methode bezieht sich nur auf die Generierung von Digitalen Modellen eines Schaftfräsers. Zusätzlich können Methoden für weitere Werkzeuge anderer geometrischer Gestalt entwickelt werden. Darunter fallen beispielsweise Kugel- oder Fasfräser, Bohrer oder NC Anbohrer. Dies kann bis zur Abdeckung des gesamten Werkzeugbestands einer Produktionsstätte weitergeführt werden. In der Annahme, dass alle Werkzeugtypen mit dieser Methode automatisiert in Digitale Modelle überführt werden

können, ist mit Zeiteinsparungen und erhöhter Fehlerfreiheit zu rechnen.

Die Integration in die Werkzeugdatenbank ist von deren Anforderungen abhängig. Bei Verwendung einer Werkzeugdatenbank, welche vom CAD/CAM-System unabhängig ist, könnten die erzeugten 3D-Modelle abgespeichert und direkt in die Werkzeugdatenbank geladen werden. Somit entfällt der Export von Expressions. Dieser Vorgang ist von den jeweiligen Anforderungen abhängig und ist mit verschiedenen Methoden durchführbar.

Eine weitere Möglichkeit stellt die Substitution des Voreinstellgeräts zur Erzeugung eines Werkzeugscans dar. Die Beschaffung eines Voreinstellgeräts ist mit hohen Kosten verbunden. Dies kann unter Umständen vermieden werden, indem eine eigene Apparatur gebaut wird. Dadurch kann ein Scan mit Hilfe einer Kamera und anschließender Beurteilung des Fotos erfolgen.

Quellen

Altintas, Yusuf; Tulsyan, Sneha (2015): *Prediction of part machining cycle times via virtual CNC*. CIRP Annals - Manufacturing Technology 64

Bauernhansl, Thomas; Krüger, Jörg; Reinhart, Gunther; Schuh, Günther (2016): *WGP-Standpunkt Industrie 4.0*. Wissenschaftliche Gesellschaft für Produktionstechnik WGP e. V.

Brecher, Christian; Königs, Michael; Lohse, Wolfram (2013): *Ein STEP- und STEP-NC-basiertes PLM-System*. In: Zeitschrift für wirtschaftlichen Fabrikbetrieb, Heft 11/2013, S. 872–877

Brecher, Christian; Lohse, Wolfram; Vitr, Mirco (2011): *CAM-NC Planning with Real and Virtual Process Data*. 17th International Conference on Concurrent Enterprising

Brillinger, Markus; Martinez, Juan; Schmid, Johannes (2019): *Improving CAD/CAM Process Chains in Forging Industries in the Era of Digitalization Based on a Case Study*. 5th World Congress of Mechanical, Chemical, and Material Engineering

Dalmolen, Simon; Cornelisse, Erik; Moonen, Hans; Stoter, Arjan (2012): *Cargo's Digital Shadow: a blueprint to enable a cargo centric information architecture*. Proceedings of the e-Freight Conference, S. 1–16

Dangelmaier, Wilhelm; Gausemeier, Jürgen (2019): *Intelligente Arbeitsvorbereitung auf Basis virtueller Werkzeugmaschinen*. Springer Vieweg

Deng, Changyi; Guo, Ruifeng; Zheng, Pai; Liu, Chao; Xu, Xun; Zhong, Ray (2018): *From Open CNC Systems to Cyber-Physical Machine Tools: A Case Study*. Procedia CIRP 72, S. 1270–1276

Denkena, Berend; Ammermann, Christoph (2010): *Modular Numerical Control Model Including Realistic Motion Planning*. International Conference on Product Lifecycle Management, S. 122–132

Deutsches Institut für Normung e. V., (Hrsg.) (2017): *Konzept für den Aufbau von 3D-Modellen auf Grundlage von Merkmalen nach DIN 4000 - Teil 1: Übersicht und Grundlagen*

Dietrich, Jochen; Richter, Arndt (2020): *Praxis der Zerspantechnik - Verfahren, Prozesse, Werkzeuge*. 13. Auflage, Springer Vieweg

E. Zoller GmbH & Co. KG (2021a): *RFID-Technologie - Der schnelle Weg zur Werkzeugidentifikation und Datenübertragung*. https://www.zoller.info/at/produkte/toolmanagement/datentransfer/rfid-chip [Stand 05.03.2021]

E. Zoller GmbH & Co. KG (2021b): *»zidCode« - Die effiziente Lösung zur Werkzeugidentifikation und Datenübertragung*. https://www.zoller.info/at/produkte/toolmanagement/datentransfer/zidcode [Stand 05.03.2021]

E. Zoller GmbH & Co. KG, (Hrsg.) (2012): *Originalbetriebsanleitung - Bildverarbeitungssoftware »pilot 3.0«*. Ohne Verlagsangaben

E. Zoller GmbH & Co. KG, (Hrsg.) (2016): *Betriebsanleitung - Einstell- und Messgerät »venturion«*. Ohne Verlagsangaben

Emanuele, Dovere; Cavalieri, Sergio; Ierace, Stefano (2015): *An assessment model for the implementation of RFID in tool management*. IFAC-PapersOnLine 48-3, S. 1007–1012

Frochte, Jörg (2021): *Maschinelles Lernen: Grundlagen und Algorithmen in Python*. 3. Auflage, Carl Hanser Verlag, München

Grieves, Michael; Vickers, John (2017): *Digital Twin: Mitigating Unpredictable, Undesirable Emergent Behavior in Complex Systems*. In: Kahlen, Franz-Josef; Flumerfelt, Shannon; Alves, Anabela: *Transdisciplinary Perspectives on Complex Systems - New Findings and Approaches*, S. 85-113

Grieves, Michael (Hrsg.) (2015): *Digital Twin: Manufacturing Excellence through Virtual Factory Replication*. Ohne Verlagsangaben

HBB Engineering GmbH, (Hrsg.) (2019): *NX - Systembetreuer NX12 / NX18xx*. Ohne Verlagsangaben

Hehenberger, Peter (2011): *Computerunterstützte Fertigung - Eine kompakte Einführung*. Springer-Verlag, Berlin

Hofmann, Johann (2017): *Die digitale Fabrik - Auf dem Weg zur digitalen Produktion*. 1. Auflage, Beuth Verlag GmbH

Kern, Christian (2006): *Anwendung von RFID-Systemen*. 2. Auflage, Springer-Verlag, Berlin

Kief, Hans B.; Roschiwal, Helmut A.; Schwarz, Karsten (2017): *CNC - Handbuch*. 30. Auflage, Carl Hanser Verlag, München

Kretzschmann, Ralf (2010): *Entwicklung eines automatisierten CNC-Prozessketten-Generators für spanende Werkzeugmaschinen*. Dissertation, Technische Universität Cottbus, Shaker Verlag, Aachen

Kritzinger, Werner; Karner, Matthias; Traar, Georg; Henjes, Jan; Sihn, Wilfried (2018): *Digital Twin in manufacturing: A categorical literature review and classification*. IFAC-PapersOnLine 51-11, S. 1016–1022

Microsoft Corporation (2021a): *Namespaces in Visual Basic*. https://docs.microsoft.com/de-de/dotnet/visual-basic/programming-guide/program-structure/namespaces [Stand 28.02.2021]

Microsoft Corporation (2021b): *Schnellstart: Ein erster Blick auf die Visual Studio-IDE*. https://docs.microsoft.com/de-de/visualstudio/ide/quickstart-ide-orientation?view=vs-2019 [Stand 10.01.2021]

Mutilba, Unai; Sandá, Alejandro; Vega, Ibon; Gomez-Acedo, Eneko; Bengoetxea, Ion; Yagüe-Fabra, José (2018): *Traceability of on-machine tool measurement: Uncertainty budget assessment on shop floor conditions*. Measurement 135, S. 180–188

Oehler, Dominik (2016): *Definition der CAD/CAM/CNC-Prozesskette auf Basis von Siemens NX10*. GRIN Verlag

Reinhart, Gunther (2017): *Handbuch Industrie 4.0 - Gechäftsmodell, Prozesse, Technik*. Carl Hanser Verlag, München

Scheidegger, Patrick (2016): *CNC-CAM-Techniken - Die Vernetzung von Fertigungs- und CNC-Techniken in der betrieblichen Praxis*. 2. Auflage, Dr.-Ing. Paul Christiani GmbH & Co. KG, Konstanz

Schmid, Johannes (2021): *Digitalisierung im Bereich der Werkzeugmaschinen und des Werkzeugmanagements*. Dissertation, Technische Universität Graz. (Noch nicht veröffentlicht)

Schmid, Johannes; Pichler, Rudolf (2020): *Seamless Data Integration in the CAM-NC Process Chain in a Learning Factory*. Procedia Manufacturing 45, S. 31–36

Schmid, Johannes; Schmid, Alexander; Pichler, Rudolf; Haas, Franz (2020): *Validation of machining operations by a Virtual Numerical Controller Kernel based simulation*. Procedia CIRP 93, S. 1478–1483

Schuh, Günther; Blum, Matthias; Reschke, Jan; Birkmeier, Martin (2016): *Der Digitale Schatten in der Auftragsabwicklung*. In: Zeitschrift für wirtschaftlichen Fabrikbetrieb, Heft 1-2/2016

Siemens Digital Industries Software (2020): *Digitaler Zwilling*. https://www.plm.automation.siemens.com/global/de/our-story/glossary/digital-twin/24465 [Stand 09.12.2020]

Siemens PLM Software (2019): *Getting Started with NX Open*. https://docs.plm.automation.siemens.com/data_services/resources/nx/1872/nx_api/common/en_US/graphics/fileLibrary/nx/nxopen/NXOpen_Getting_Started.pdf

Siemens PLM Software (2021a): *Expressions*. https://docs.plm.automation.siemens.com/tdoc/nx/12.0.2/nx_help/#uid:expressions_exp_ov [Stand 28.02.2021]

Siemens PLM Software (2021b): *Manufacturing Resource Library*. (Version: 2018b) [Software]

Siemens PLM Software (2021c): *NXOpen .NET Reference Guide*. https://docs.plm.automation.siemens.com/data_services/resources/nx/1899/nx_api/custom/en_US/nxopen_net/index.html [Stand 24.03.2021]

Stark, John (2015): *Product Lifecycle Management - Volume 1: 21st Century Paradigm for Product Realisation*. 3. Auflage, Springer Verlag

Tao, Fei; Zhang, Meng; Nee, A. Y. C. (2019): *Digital Twin Driven Smart Manufacturing*. Academic Press

Vieira Junior, M.; Baptista, Elesandro; Araki, Luciana; Smith, Scott; Schmitz, Tony (2018): *The role of tool presetting in milling stability uncertainty*. Procedia Manufacturing 26, S. 164–172

Vieira Junior, M.; Silva, Jose; Correr, Ivan; Coppini, Nivaldo; Baptista, Elesandro (2011): *Losses caused by the presetting of tools by the manual method*. IEEE International Conference on Industrial Engineering and Engineering Management, S. 565–569

Vogel-Heuser, Birgit; Bauernhansl, Thomas; Hompel, Michael ten (2017): *Handbuch Industrie 4.0 Bd.4 - Allgemeine Grundlagen*. 2. Auflage, Springer Vieweg

Westkämper, Engelbert; Löffler, Carina (2016): *Strategien der Produktion - Technologien, Konzepte und Wege in die Praxis*. Springer Vieweg

A Anhang

Damit der Lesefluss dieser Arbeit ungestört bleibt, werden in diesem Kapitel ergänzende Informationen zur Arbeit bereitgestellt. Hierzu werden in Abschnitt A.1 weitere Definitionen zur Beschreibung eines Digitalen Zwillings angeführt. Des Weiteren werden die Durchführung einer Journalaufzeichnung, die Vorlagenerstellung in Microsoft Visual Studio sowie die Durchführung einer Signierung detailliert beschrieben. Anschließend wird das Werkzeugdatenblatt eines Schaftfräsers bereitgestellt. Am Ende ist der gesamte Programmcode vorzufinden.

A.1 Definitionen eines Digitalen Zwillings

In diesem Abschnitt werden Definitionen eines Digitalen Zwillings unterschiedlicher Quellen vorgestellt. Zuerst wird die Definition nach Grieves und Vickers und dann nach Siemens erläutert.

A.1.1 Definition nach Grieves und Vickers

Michael Grieves stellte erstmalig im Jahr 2002 in seiner Präsentation an der Universität Michigan zu Produktlebenszyklusmanagement ein Konzept vor, das dem heutigen Digitalen Zwilling entspricht. Dieses Konzept basiert auf der Idee, eine digitale Informationssammlung für ein physisches Produkt zu erstellen, die eine eigene Einheit darstellt. Diese Informationssammlung stellt den Digitalen Zwilling dar und ist mit dem physischen Objekt verbunden. Ein Digitaler Zwilling setzt ein bestehendes physisches Objekt voraus und verfügt

über einen automatischen Datenfluss zwischen dem physischen und dem virtuellen Objekt in beide Richtungen.[74] Der Digitale Zwilling soll nicht nur eine digitale Abbildung zu einem gewissen Zeitpunkt sein, sondern vielmehr ein durchgängiges, digitales Pendant zum physischen Produkt über die gesamte Lebensdauer darstellen. Dabei ist das digitale Objekt ausgehend von der Idee über die Phase der Produktentwicklung bis hin zum Ausscheiden des Produkts mit dem physischen Objekt verbunden. Der Digitale Zwilling ist somit ein digitales Abbild, welches die gleichen Informationen wie das physische Produkt bereitstellt.

Für den Digitalen Zwilling stehen zwei Typen zur Klassifizierung bereit. Einerseits handelt es sich dabei um den Digitalen Zwilling eines Prototyps und andererseits um den Digitalen Zwilling eines Objekts. Beide befinden sich gleichermaßen in einer Umgebung, welche auf die Verwaltung von Digitalen Zwillingen gerichtet ist. Diese beiden Typen sowie deren Umgebung werden in den nächsten Absätzen behandelt.

Der Digitale Zwilling eines Prototyps beinhaltet alle notwendigen Informationen, um einen physischen Prototyp zu beschreiben und zu fertigen. Diese Informationen sind unter anderem Anforderungen an das Produkt, ein vollständiges 3D-Modell, eine Stückliste, eine Prozessliste, eine Serviceliste sowie eine Entsorgungsstrategie. Der Umfang der Informationen wird nicht begrenzt und kann nach Belieben erweitert werden.

Der Digitale Zwilling eines Objekts beschreibt ein spezifisches, physisches Objekt, mit dem der Digitale Zwilling über die gesamte Produktlebensdauer in Verbindung steht. Dieser Zwilling beinhaltet ein vollständig definiertes 3D-Modell mit Geometrie- und Toleranzangaben, eine Stückliste, eine Prozessliste der Produktion, eine Servicedokumentation, eine Dokumentation über Ergebnisse und Tests sowie eine Datensammlung der Sensoren am physischen Objekt.

In der virtuellen Umgebung agieren die Digitalen Zwillinge für verschiedene Zwecke. Diese können Prognosen für das zukünftige Verhalten und die Leistung eines physischen Produkts liefern. Darüber hinaus können sie den aktuellen und vergangenen Status des physischen Produkts unabhängig vom aktuellen Standort darstellen.[75]

[74] Vgl. Grieves (2015).
[75] Vgl. Grieves/Vickers (2017), S.92ff.

A.1.2 Definition nach Siemens

Laut der Definition nach Siemens handelt es sich beim Digitalen Zwilling um ein Abbild eines physischen Produkts oder Prozesses. Mit Hilfe dieses Zwillings soll eine Vorhersage über die Performance eines Produkts über dessen gesamten Produktlebenszyklus getroffen werden können. Hierzu werden Simulationen, Prognosen und Optimierungen eines Produktionssystems oder Produkts angewandt, damit die physische Realisierung eines Prototyps und die Investition in Ressourcen hinausgeschoben werden kann. Unter dem Einsatz von bestimmten Werkzeugen ist es einem Digitalen Zwilling möglich, Auswirkungen von konstruktiven Änderungen, Umwelteinflüssen oder Nutzenverhalten zu berücksichtigen. Bei diesen Werkzeugen handelt es sich vorwiegend um Instrumente wie Machine Learning, Datenanalysen und Simulationen. Unter Machine Learning versteht man, dass Maschinen ein Verhalten aus einem vorhandenen Datensatz erlernen und auf einen anderen Zustand anwenden.[76] Durch diese Vorgehensweise ist eine erhebliche Zeit- und Kosteneinsparung in der Entwicklung realisierbar, während die Qualität des Produktionsprozesses sowie des Produkts steigen. Sogar das gänzliche Verschwinden eines physischen Prototyps ist denkbar. Der Digitale Zwilling bezieht seine Daten von Sensoren auf dem physischen Objekt zur Simulation des Produkts beziehungsweise der Produktion. Somit ist es ihm möglich, sich ständig mit dem physischen Objekt zu aktualisieren und Änderungen des physischen Objekts virtuell darzustellen. Dementsprechend entsteht ein Kreislauf der Optimierung, der die Produkte und die Produktion kontinuierlich verbessert.

Siemens unterscheidet beim Digitalen Zwilling zwischen drei verschiedenen Typen:

- Digitaler Produktzwilling
- Digitaler Produktionszwilling
- Digitaler Performancezwilling

Der Digitale Produktzwilling zielt auf eine effiziente Konstruktion von Produkten ab und zeigt in der virtuellen Welt, wie sich ein Produkt in der Realität unter verschiedenen Bedingungen verhalten wird.

[76] Vgl. Frochte (2021), S.14

Der Digitale Produktionszwilling bildet einen Fertigungsprozess ab, der simuliert und optimiert werden kann, bevor die Produktion gestartet wurde. Dadurch ist es den Unternehmen möglich, frühzeitig Einflüsse von außen in den Prozess miteinzubinden, um Produktionsstrategien zu erstellen.

Der Digitale Performancezwilling sammelt, analysiert und verarbeitet Daten von physischen Produkten und Produktionsanlagen zur Verbesserung der Systemeffizienz und des Digitalen Zwillings sowie zur Entscheidungshilfe von späteren Produkteigenschaften.[77]

A.2 Durchführung der Journalaufzeichnung

In Unterabschnitt 3.3.1.1 wurden die Grundlagen der Journalaufzeichnung bereitgestellt. In diesem Abschnitt wird die Durchführung erläutert. Wie bereits in Unterabschnitt 3.3.1.1 erwähnt, wird noch einmal explizit darauf hingewiesen, dass die Journalaufzeichnung als Instrument dient, die verschiedenen Funktionen in Siemens NX in einem Journal in Form von Codezeilen abzuspeichern. Somit ist sie ein wesentlicher Bestandteil der Programmerstellung, selbst wenn das Programm in Visual Studio erstellt wird. Die Menüleiste, über welche die Journalaufzeichnung gesteuert wird, ist in Abbildung 36 dargestellt.

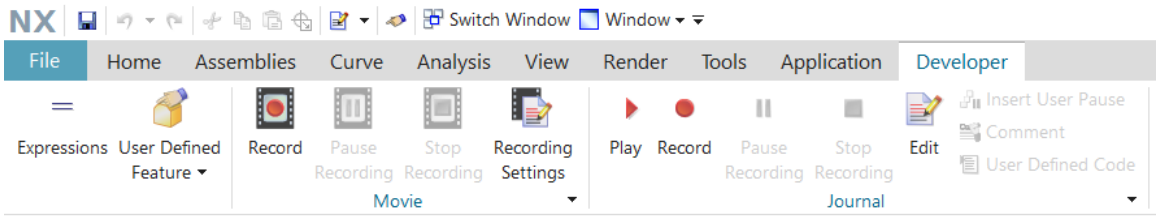


Abbildung 36: Menüleiste „Developer“ in Siemens NX
Quelle: Eigene Darstellung.

[77] Vgl. Siemens Digital Industries Software (2020).

Grundsätzlich ist in der Menüleiste von Siemens NX die Kategorie „Developer“ zu aktivieren. Dies erfolgt durch Rechtsklick auf die Menüleiste und der Auswahl von „Developer“. Nun wurde der Menüreiter „Developer“ in der Menüleiste hinzugefügt und erscheint wie in Abbildung 36 zu sehen ist. Für die Auslösung einer Journalaufzeichnung ist im Abschnitt „Journal“ auf die Schaltfläche „Record“ zu klicken. Anschließend öffnet sich ein Fenster, in welchem der Speicherort des Journals definiert werden kann. Nun kann die Klickfolge in Siemens NX stattfinden. Für das Beenden der Aufzeichnung wird auf die Schaltfläche „Stop Recording“ gedrückt. Das aufgezeichnete Journal ist meist nicht direkt verwendbar, da das Journal mit Funktionen überladen ist, die in weiterer Folge nicht gebraucht werden. Diese müssen entfernt werden, sodass der gesuchte Code gefunden werden kann. [78]

A.3 Einrichtung in Microsoft Visual Studio

Bevor mit der Programmierung in Microsoft Visual Studio begonnen werden kann, muss eine Einrichtung stattfinden. Dadurch wird ein Zugriff auf die Funktionen von NX Open in Microsoft Visual Studio ermöglicht. Es können schließlich alle Funktionen in vollem Umfang verwendet werden. Dieser Vorgang wird in diesem Abschnitt detailliert beschrieben, sodass ein Verständnis für die eigene Einrichtung vermittelt wird. Zur Beschreibung werden Verzeichnisse verwendet, welche nachfolgend vollständig aufgelistet sind:

Verzeichnis 1 *C:\Program Files\Siemens\NX 12.0*

Verzeichnis 2 *UGII_BASE_DIR\UGOPEN\NXOpenExamples\VB\Templates*

Verzeichnis 3 *C:\ Users \ smartfactory \ Documents \ Visual Studio 2019 \ Templates \ ProjectTemplates \ Visual Basic*

Verzeichnis 4 *UGII_BASE_DIR \ NXBIN \ managed*

[78] Vgl. Siemens PLM Software (2019), S.49

Generell ist zu erwähnen, dass die Pfade nur für den Fall dieser Arbeit gültig sind. Je nachdem wie die Installation der Software erfolgt ist, können sich Abweichungen ergeben.

Damit eine für Siemens NX ausführbare Datei in Visual Studio erstellt werden kann, müssen zuvor einige Einstellungen getroffen werden. Dabei ist es von Vorteil, wenn beim Öffnen von Visual Studio eine Vorlage einer NX Open-Anwendung auswählbar ist. Das bedeutet, dass bei einem neuen Projekt alle Einstellungen voreingestellt sind und sofort mit der Programmierung begonnen werden kann. Dabei kann zwischen einem Assistenten und einem Template unterschieden werden. Da der Assistent nicht mit der in dieser Arbeit verwendeten Version von Visual Studio kompatibel ist, muss auf die Variante basierend auf einem Template zurückgegriffen werden. Dieser Umstand hat keine Auswirkungen auf das Ergebnis. Die Programmierung kann mit beiden Methoden gleichermaßen erfolgen. Für die weitere Erläuterung der Einrichtung wird das Installationsverzeichnis von Siemens NX mit der Variable „UGII_BASE_DIR“ abgekürzt. Diese Variable ist als Systemvariable in Windows gespeichert und steht in diesem Fall für Verzeichnis 1.

Für die Implementierung der Vorlage in Visual Studio ist es notwendig, den Ordner mit der Bezeichnung „NXOpenApplication“ aus Verzeichnis 2 zu kopieren und in den Ordner „Visual Studio 2019“, der bei der Installation in Verzeichnis 3 angelegt wurde, einzufügen.[79] Dabei soll der Ordner nur kopiert und nicht entpackt werden. Ergänzend ist festzuhalten, dass es im Verzeichnis 2 einen zweiten Ordner mit der Bezeichnung „NXOpenWinFormApplication“ gibt. Eine „NXOpenWinFormApplication“ dient zum Erstellen von Fenstern, in welchen Funktionen mit Schaltflächen verknüpft sind. Für die Einrichtung einer Vorlage zur Erstellung einer „NXOpenWinFormApplication“ in Visual Studio ist mit diesem Ordner analog zu „NXOpenApplication“ zu verfahren. Diese Arbeit beschränkt sich auf die Erstellung einer gewöhnlichen „NXOpenApplication“. Schließlich kann nach dem Öffnen von Visual Studio ein neues Projekt erstellt und eine NX Open-Anwendung ausgewählt werden. Nachdem man den Speicherpfad und den gewünschten Namen des Projekts definiert hat, wird ein Projekt mit allen Verweisen zu NX Open erstellt. Hiermit ist es Visual Studio möglich, mit Hilfe von NX Open über dessen Objekte und Methoden auf die NX-Funktionen zuzugreifen. Dabei kann es passieren, dass der Pfad zu den Verweisen nicht mit dem tatsächlichen Pfad übereinstimmt. Zur Lösung dieses Problems ist es notwendig, die Projektdatei mit der Endung **.vbproj* in Visual Studio zu öffnen. Diese Datei wird über die mit dem kleinen

[79] Vgl. Siemens PLM Software (2019), S.132.

roten Rechteck umrahmte Schaltfläche in Abbildung 37 geöffnet. In dieser Datei können individuelle Einstellungen in Form von Code getroffen werden. Unter anderem besteht auch die Möglichkeit, die Pfade für die Verweise zu ändern. Die hinterlegten Pfade für die hinzugefügten Verweise sind im roten großen Rechteck in Abbildung 37 dargestellt.

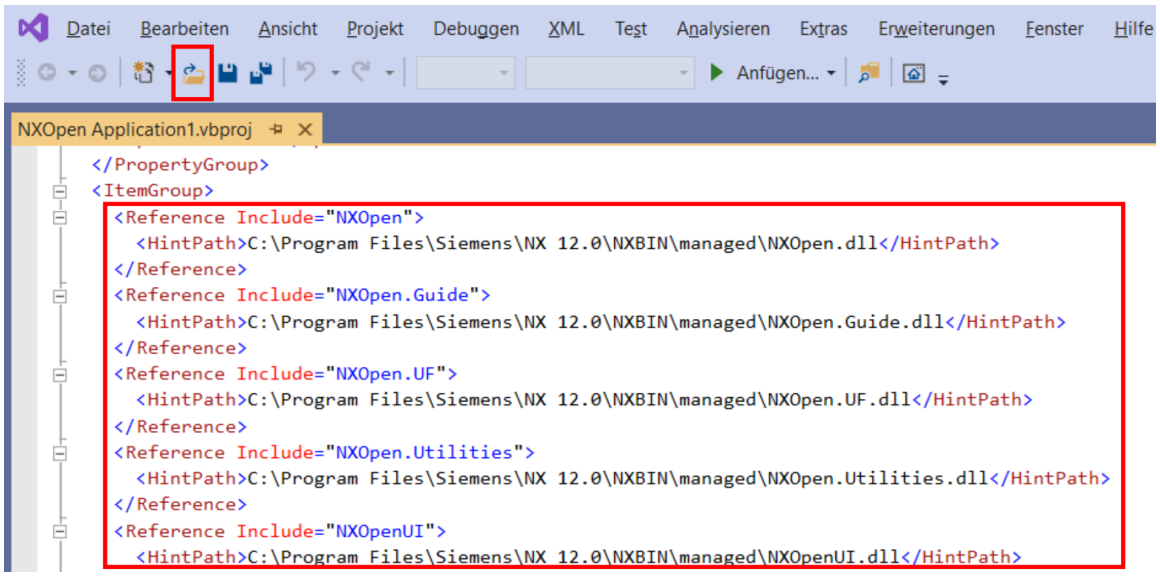


Abbildung 37: Verweise der Projektdatei in Microsoft Visual Studio
Quelle: Eigene Darstellung.

Nachdem das Projekt erfolgreich erstellt wurde und die Verweise zu den korrekten Pfaden hinterlegt sind, wird nun die grafische Oberfläche von Visual Studio angezeigt. Diese ist in Abbildung 38 dargestellt. Die Oberfläche kann in den Editor, den Projektmappen-Explorer, die Menüleiste, das Eigenschaftenfenster, die Fehlerliste und das Ausgabefenster unterteilt werden. Der Editor bildet die Grundlage für die Programmierung. Er stellt den Inhalt der Dateien dar und ermöglicht das Programmieren. Der Projektmappen-Explorer ist für die Verwaltung der Dateien innerhalb eines Projekts zuständig. Hier werden außerdem die Verweise zu anderen Bibliotheken visualisiert. Die Menüleiste fasst Befehle in Kategorien zusammen. Im Eigenschaftenfenster kann der Dateiname schnell geändert werden. Die Fehlerliste und das Ausgabefenster können durch Umschalten der kleinen Schaltflächen angezeigt werden. Die Fehlerliste weist die Programmiererin oder den Programmierer auf Fehler in der Datei hin, sodass umgehend darauf reagiert werden kann. Im Ausgabefenster

werden Meldungen wie zum Beispiel der erfolgreiche Buildvorgang angezeigt. [80]

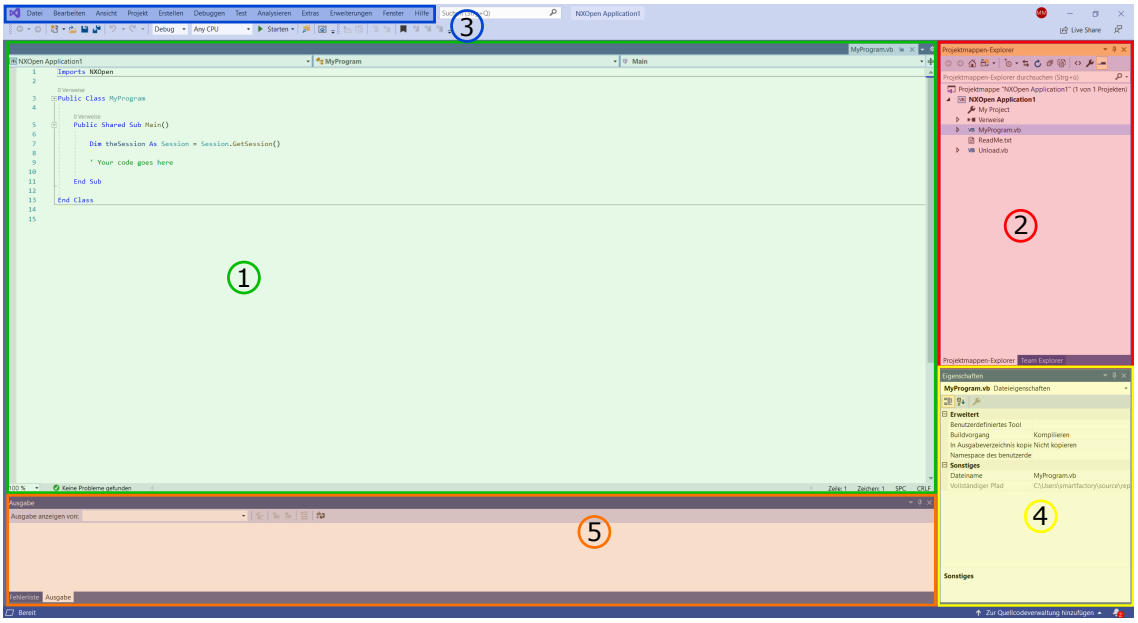


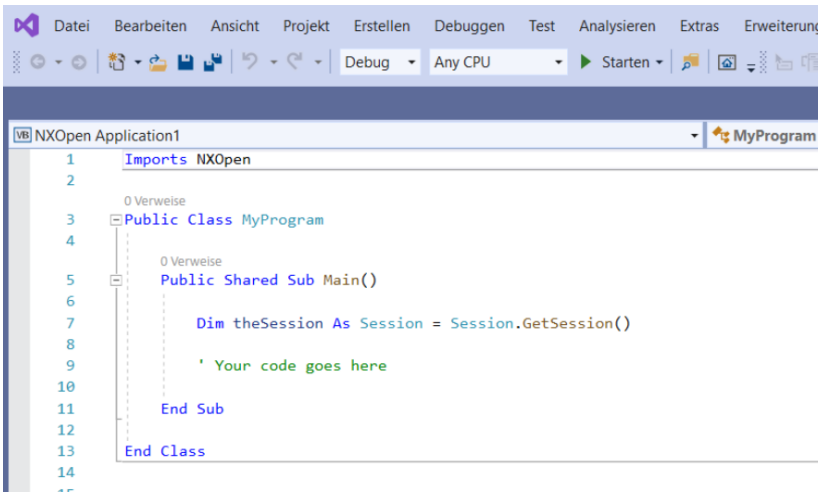
Abbildung 38: Oberfläche von Microsoft Visual Studio bestehend aus Editor (1), Projektmappen-Explorer (2), Menüleiste (3), Eigenschaftenfenster (4) und Fehlerliste und Ausgabefenster (5)
Quelle: Eigene Darstellung.

Mit dem Template werden der Anwenderin oder dem Anwender zwei Klassen bereitgestellt, die zur Programmierung notwendig sind. Dabei handelt es sich um die Hauptklasse *MyProgram.vb* mit der Main-Funktion und um die Klasse *Unload.vb* mit der GetUnloadOption-Funktion. Beide sind im Projektmappen-Explorer zu finden. Deren Bedeutung wird im nächsten Absatz diskutiert.

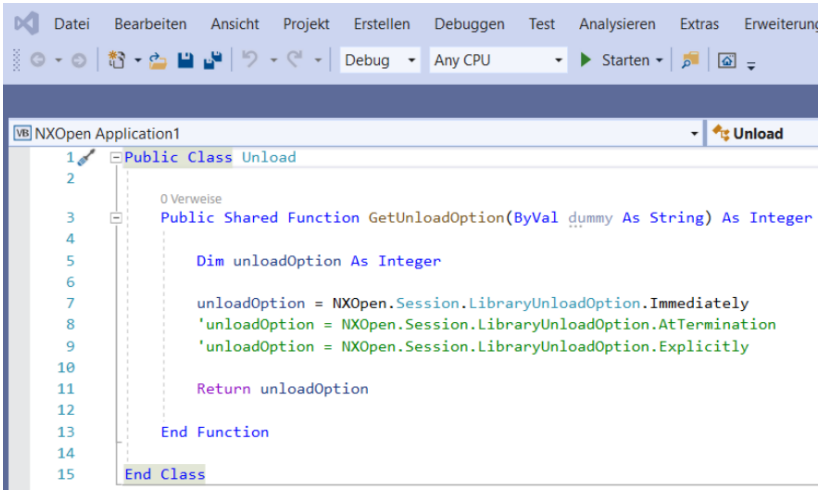
Beim Start eines Programms in Siemens NX wird nach der Main-Funktion gesucht. Diese definiert gleichzeitig den Einstiegspunkt des Programms. Damit die **.dll* in Siemens NX nach der Ausführung auch wieder entladen wird, ist die GetUnloadOption-Funktion zu verwenden. Ansonsten würde ein darauffolgendes Programm nicht ausgeführt werden können, da sich noch die alte **.dll* im Speicher befindet. Die GetUnloadOption-Funktion ist nur

[80] Vgl. Microsoft Corporation (2021b).

für ausführbare Programme relevant und wird bei Journalen nicht verwendet.[81] Innerhalb der Main-Prozedur wird der auszuführende Code platziert. Diese ist in Abbildung 39(a) dargestellt. Die GetUnloadOption-Prozedur ist in Abbildung 39(b) ersichtlich. In dieser stehen drei verschiedene Varianten für das Entladen der Datei zur Verfügung. Dabei handelt es sich einmal um das sofortige Entladen der Datei direkt nach der erfolgreichen Ausführung, einmal um das Entladen beim Schließen von Siemens NX und einmal um das manuelle Entladen der Datei.



(a) MyProgram.vb



(b) Unload.vb

Abbildung 39: Benötigte Klassen einer NX Open-Anwendung
Quelle: Eigene Darstellung.

[81] Vgl. HBB Engineering GmbH (2019), S.238.

Außerdem besteht die Möglichkeit, eine **.dll* ohne Vorlage in Visual Studio zu erstellen. Dazu muss die Vorlage mit dem Namen „Klassenbibliothek (.NET Framework)“ in der Programmiersprache VB ausgewählt werden. Nachdem der Speicherpfad definiert und der Dateiname festgelegt wurden, muss eine Klasse mit der Main-Funktion und eine Klasse mit der GetUnloadOption-Funktion erstellt werden. Anschließend müssen die Verweise zu den NX-Programmierbibliotheken (NXOpen, NXOpen.Guide, NXOpen.UF, NXOpen.Utilities und NXOpenUI) durch Rechtsklick auf Verweise im Projektmappen-Explorer hergestellt werden. Diese Verweise sind je nach Bedarf auszuwählen. Die **.dll* sind im Verzeichnis 4 zu finden.[82] Nun ist es Visual Studio möglich, auf NX Open zuzugreifen.

Beide Varianten führen zum gleichen Ergebnis. Festzuhalten ist, dass die Methode mit der Vorlage für sich wiederholende Arbeiten zeitsparender ist und deshalb Verwendung in dieser Arbeit findet.

A.4 Durchführung der Signierung

Nachfolgend sind die Verzeichnisse aufgelistet, welche zur Beschreibung der Durchführung der Signierung benötigt werden:

Verzeichnis 5 *UGII_BASE_DIR\UGOPEN\NXSigningResource.res*

Verzeichnis 6 *UGII_BASE_DIR\NXBIN*

Wie bereits in Unterabschnitt 3.3.2 erwähnt, ist für die Siegnierung einer **.dll* das Vorhandensein einer NX Open Author Lizenz notwendig. Hierzu ist die Implementierung der Datei *NXSignignResource.res* in das Projekt in Visual Studio notwendig. Die Datei befindet sich in Verzeichnis 5. Nachdem die *NXSignignResource.res* erfolgreich hinzugefügt wurde, erscheint sie im Projektmappen-Explorer unter Ressourcen. Nach der Durchführung des Buildvorgangs entsteht eine **.dll* im Ausgabeverzeichnis „bin\debug“ oder „bin\release“ des Projektordners. Durch Navigation in der Eingabeaufforderung von Windows in das Verzeichnis 6, in welchem sich die *SignDotNet.exe* befindet, wird die **.dll* durch den Befehl *SignDotNet.exe* gefolgt vom Pfad der **.dll* signiert.[83]

[82] Vgl. HBB Engineering GmbH (2019), S.232.
[83] Vgl. HBB Engineering GmbH (2019), S.240.

A.5 Werkzeugdatenblatt

HAM PRÄZISION

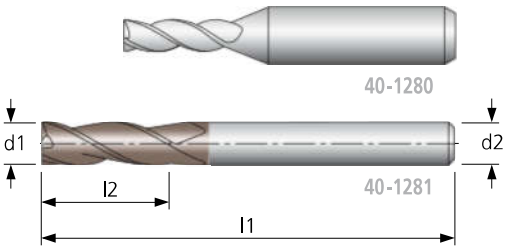
HAM 40-1280
(HAM 401)

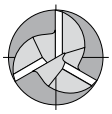
Vollhartmetall-Schaftfräser
solid carbide end mill

VHM | Z 3 | 30° rechts | Werk Norm | Typ N | HA | SHRINK FIT | HB

Konstruktions-Daten
- bis Ø 3,0 mm, zentrumsschneidend
- ab Ø 3,5 mm, eine Schneide über Mitte
- universell einsetzbar

Engineering data
- *up to Ø 3,0 mm, centre cutting*
- *from Ø 3,5 mm, 1 cutting edge over centre*
- *allround end mill*





Eckenfase	d1	b
	≤ Ø 8 = Ø 9 ≥ Ø 10	— 0,05 0,10

Material	Alu Knet-leg.	Alu Guss-leg.	Stahl < 800 N/mm²	Stahl < 1200 N/mm²	Stahl < 1600 N/mm²	Stahl < 55 HRC	Stahl < 60 HRC	Stahl < 66 HRC	INOX < 800 N/mm²	INOX > 800 N/mm²	GG	GGG	hochw. Legie-rungen	Titan	NE-Metalle Cu-Leg.	Graphit & Faser-verbund	MMS	max.	ohne	AIR
40-1280	○	○	●	○	○	○			○	○	●	○	○	○	○		●	●	○	○
40-1281	○	○	●	●	●	○			○	○	●	●	○	○	○		●	●	○	○

Schnittdaten siehe Seite / *cutting data available on page* – 189

● sehr gut geeignet / *very suitable* ○ bedingt geeignet / *limited suitable*

Ø d1 (e8) mm	40-1280	40-1281 TA	l2 mm	l1 mm	Ø d2 (h6) mm
0,6			2	38	3
0,8			3	38	3
1			3	38	3
1,2			4	38	3
1,5			5	38	3
1,6			5	38	3
2			6	38	3
2,5			7	38	3
3			9	38	3
3,5			12	40	3,5
4			12	40	4
4,5			14	50	4,5

Ø d1 (e8) mm	40-1280	40-1281 TA	l2 mm	l1 mm	Ø d2 (h6) mm
5			14	50	5
6			16	50	6
7			20	60	7
8			20	60	8
9			22	70	9
10			22	70	10
11			22	70	11
12			22	70	12
14			25	75	14
16			25	75	16
18			30	100	18
20			30	100	20

Bestellbeispiel / *Order example*: HA-Schaft/*shank* 40-1280-5
HB-Schaft/*shank* 40-1280-6-HB

A.6 Programmcode

In diesem Abschnitt ist der gesamte Code angeführt, welcher sich aus drei Teilen zusammensetzt. Zu Beginn ist der Code der Skizzenerstellung dargestellt. Dieser besteht aus sechs Klassen (MyProgram, Import, SketchClass, Analyse, addToList und Unload). Darauf folgt der zweite Teil, die Erzeugung der Volumenkörper. Dieser setzt sich aus den Klassen (MyProgram, SketchClass, Tool, Holder und Unload) zusammen. Abschließend wird der Code des Exports der Expressions vorgelegt, welche die Klassen (MyProgram und Unload) beinhaltet. Für ein besseres Verständnis sind Kommentare beigefügt, welche in Grün gehalten sind.

...ory\Desktop\code\createSketch\createSketch\MyProgram.vb 1

```vb
Imports NXOpen

Public Class MyProgram
    Public Shared theSession As Session = Session.GetSession()
    Public Shared theWorkPart As Part = theSession.Parts.Work
    Public Shared theUFSession As UF.UFSession = UF.UFSession.GetUFSession()
    Public Shared theUI As UI = UI.GetUI()
    Public Shared theDisplayPart = theSession.Parts.Display

    Public startPoint3dWCS As Point3d
    Public endPoint3dWCS As Point3d

    Public toolSketch As Sketch
    Public toolHolderSketch As Sketch

    Public Shared Sub Main()

        System.Diagnostics.Debugger.Launch()
        If Not theWorkPart Is Nothing Then

            Dim path_input As String
            Dim path_output As String

            path_input = NXOpenUI.NXInputBox.GetInputString("Geben Sie bitte den Pfad zur .dxf-Datei ein: ")
            path_output = theWorkPart.FullPath

            If path_input = Nothing Then
                Exit Sub
            End If

            Import.importDXF(path_input, path_output)

            'Ebene  und Achse zur Skizzenorientierung erstellen
            Dim origin As New Point3d(0, 0, 0)
            Dim originPoint As Point = theWorkPart.Points.CreatePoint(origin)
            Dim vectorX As New Vector3d(1, 0, 0)
            Dim wcsMatrix As Matrix3x3 = theWorkPart.WCS.CoordinateSystem.Orientation.Element
            Dim sketchPlane As Plane = theWorkPart.Planes.CreateFixedPlane(origin, wcsMatrix)
            Dim axisX As Direction = theWorkPart.Directions.CreateDirection(originPoint, vectorX)
            sketchPlane.Blank()
            axisX.Blank()

            'Erstellung der Skizze
            Dim sketch_basis As Sketch = SketchClass.createSketch(sketchPlane, axisX, origin)
            sketch_basis.SetName("Sketch_basis")

            theWorkPart.ModelingViews.WorkView.Orient
```

...ory\Desktop\code\createSketch\createSketch\MyProgram.vb 2

```
                (NXOpen.View.Canned.Top, NXOpen.View.ScaleAdjustment.Fit)
                'Ansicht in Top-View drehen
48              Analyse.Analyse_Lines()    'horizontale und vertikale Linien
                  werden analysiert
49              Analyse.Analyse_Points()   'Punkte werden analysiert
50
51              Dim markid2 As NXOpen.Session.UndoMarkId = Nothing
52              markid2 = theSession.SetUndoMark
                  (NXOpen.Session.MarkVisibility.Visible, "linienloeschen")
53
54              For Each lin In MyProgram.theWorkPart.Lines
55                  MyProgram.theSession.UpdateManager.AddToDeleteList(lin)
56              Next
57              MyProgram.theSession.UpdateManager.DoUpdate(markid2)
58
59              For Each p In addToList.pointDeleteList3d
60                  addToList.pointList3d.Remove(p)
61              Next
62              addToList.pointDeleteList3d.Clear()
63
64
65              'Punkte in der Nähe der Greiferrille werden nicht benötigt
66              If addToList.pointList3d.Count > 6 Then
67                  For Each p In addToList.pointList3d
68                      If addToList.pointList3d.IndexOf(p) > 4 And
                        addToList.pointList3d.IndexOf(p) <>
                        addToList.pointList3d.Count - 1 Then    'nur die punkte
                        löschen die zwischen dem 5ten punkt und dem letzten
                        liegen
69                          addToList.pointDeleteList3d.Add(p)
70                      End If
71                  Next
72              End If
73
74              For Each p In addToList.pointDeleteList3d
75                  addToList.pointList3d.Remove(p)
76              Next
77
78              addToList.roundPoint3d()
79              addToList.point3dToPoint()
80              SketchClass.editSketch()
81              SketchClass.closeSketch()
82              sketch_basis.Activate(Sketch.ViewReorient.False)
83
84              For Each lin In theWorkPart.Lines
85                  sketch_basis.AddGeometry(lin)
86              Next
87
88              sketch_basis.Deactivate(Sketch.ViewReorient.False,
                  Sketch.UpdateLevel.Model)
89              Import.importDXF(path_input, path_output)
90
91          Else
```

...ory\Desktop\code\createSketch\createSketch\MyProgram.vb 3

```
 92
 93              Guide.InfoWriteLine("Es wurde kein Part gefunden!")
 94
 95          End If
 96
 97
 98
 99      End Sub
100  End Class
101
102
```

...actory\Desktop\code\createSketch\createSketch\Import.vb 1

```
1  Imports NXOpen
2  Imports System
3  Public Class Import
4
5      Public Shared Sub importDXF(path_input As String, path_output As String)
6
7
8          Dim dxfdwgImporter As NXOpen.DxfdwgImporter = Nothing
9          dxfdwgImporter = MyProgram.theSession.DexManager.CreateDxfdwgImporter
             ()
10
11         dxfdwgImporter.Units = NXOpen.DxfdwgImporter.UnitsEnum.Metric
12         dxfdwgImporter.ImportTo = DxfdwgImporter.ImportToEnum.Work
13         dxfdwgImporter.ConvModelData = True
14         dxfdwgImporter.ConvLayoutData = True
15         dxfdwgImporter.ImportCurvesType =
             NXOpen.DxfdwgImporter.ImportCurvesAs.Curves
16         dxfdwgImporter.SettingsFile = "C:\Program Files\Siemens\NX 12.0
             \dxfdwg\dxfdwg.def"
17         dxfdwgImporter.InputFile = path_input
18         dxfdwgImporter.Units = NXOpen.DxfdwgImporter.UnitsEnum.English
19         dxfdwgImporter.TemplateFile = "C:\Program Files\Siemens\NX 12.0
             \dxfdwg\dwgnullnx120_mm.prt"
20         dxfdwgImporter.OutputFile = path_output
21         dxfdwgImporter.ConvModelData = True
22         dxfdwgImporter.ConvLayoutData = True
23         dxfdwgImporter.SendLayoutsTo =
             NXOpen.DxfdwgImporter.SendLayoutsAs.ImportedView
24         dxfdwgImporter.ImportDimensionType =
             NXOpen.DxfdwgImporter.ImportDimensionsAs.Group
25         dxfdwgImporter.ModelScaleFactor = 25.4
26         dxfdwgImporter.FileOpenFlag = False
27         dxfdwgImporter.AvoidUsedNXLayers = True
28         dxfdwgImporter.ReadLayerNumFromPrefix = False
29         dxfdwgImporter.TranslateUnselectedLayer = False
30         dxfdwgImporter.DestForUnselectedLayer = 256
31         dxfdwgImporter.ProcessingOrder =
             NXOpen.DxfdwgImporter.ProcessingOrderAs.Alphabetical
32         dxfdwgImporter.SkipEmptyLayer = True
33         dxfdwgImporter.UnSelectedLayers = Nothing
34         dxfdwgImporter.AspectRatioOption =
             NXOpen.DxfdwgImporter.AspectRatioOptions.AutomaticCalculation
35
36         Dim nXObject1 As NXOpen.NXObject = Nothing
37         nXObject1 = dxfdwgImporter.Commit()
38
39         dxfdwgImporter.Destroy()
40
41
42     End Sub
43
44 End Class
45
```

...y\Desktop\code\createSketch\createSketch\SketchClass.vb 1

```vb
Imports NXOpen
Imports System

Public Class SketchClass

    Public Shared endPunktLinie As New Point3d
    Public Shared Function createSketch(sketchPlane As Plane, axis As Direction, origin As Point3d)

        Dim sketchBuilder As SketchInPlaceBuilder = Nothing
        sketchBuilder = MyProgram.theWorkPart.Sketches.CreateSketchInPlaceBuilder2(Nothing)

        sketchBuilder.PlaneReference = sketchPlane
        sketchBuilder.AxisReference = axis
        sketchBuilder.SketchOrigin = MyProgram.theWorkPart.Points.CreatePoint(origin)
        sketchBuilder.PlaneOption = Sketch.PlaneOption.Inferred

        Dim sketchName As Sketch = sketchBuilder.Commit
        sketchBuilder.Destroy()

        Return sketchName

    End Function

    Public Shared Sub editSketch()
        Dim punkte As List(Of Point) = addToList.pointList
        Dim index As Integer
        Dim startPunkt As Point3d
        Dim endPunkt As Point3d

        For Each p In punkte
            index = punkte.IndexOf(p)

            If index = 0 Then
                startPunkt = addToList.roundPointList3d(index)
            ElseIf index = punkte.Count - 1 Then
                startPunkt = endPunktLinie
                endPunkt = addToList.roundPointList3d(index)
                Dim lin As Line = MyProgram.theWorkPart.Curves.CreateLine(startPunkt, endPunkt)
                lin.SetVisibility(SmartObject.VisibilityOption.Visible)
                endPunktLinie = lin.EndPoint
                Exit For
            Else
                startPunkt = endPunktLinie
            End If

            If index Mod 2 = 0 Then ' es handelt sich um eine gerade Zahl --> also um eine vertikale Linie
                endPunkt = New Point3d(startPunkt.X, addToList.roundPointList3d(index + 1).Y, 0)
```

```
...y\Desktop\code\createSketch\createSketch\SketchClass.vb                    2
48                  Dim lin As Line = MyProgram.theWorkPart.Curves.CreateLine
                      (startPunkt, endPunkt)
49                  lin.SetVisibility(SmartObject.VisibilityOption.Visible)
50                  endPunktLinie = lin.EndPoint
51
52              Else    'es handelt sich um eine ungerade Zahl --> also um eine
                  horizontale Linie
53                  endPunkt = New Point3d(addToList.roundPointList3d(index +
                      1).X, startPunkt.Y, 0)
54                  Dim lin As Line = MyProgram.theWorkPart.Curves.CreateLine
                      (startPunkt, endPunkt)
55                  lin.SetVisibility(SmartObject.VisibilityOption.Visible)
56                  endPunktLinie = lin.EndPoint
57              End If
58
59          Next
60      End Sub
61
62      Public Shared Sub closeSketch()
63          Dim origin As New Point3d(addToList.roundPointList3d
              (addToList.roundPointList3d.Count - 1).X,
              addToList.roundPointList3d(0).Y, 0)
64
65          Dim linVer As Line = MyProgram.theWorkPart.Curves.CreateLine
              (addToList.roundPointList3d(addToList.roundPointList3d.Count - 1),
              origin)
66          linVer.SetVisibility(SmartObject.VisibilityOption.Visible)
67
68          Dim linHor As Line = MyProgram.theWorkPart.Curves.CreateLine
              (addToList.roundPointList3d(0), origin)
69          linHor.SetVisibility(SmartObject.VisibilityOption.Visible)
70      End Sub
71
72  End Class
73
```

...ctory\Desktop\code\createSketch\createSketch\Analyse.vb 1

```
1  Imports NXOpen
2  Public Class Analyse
3
4
5      Public Shared limitAngle As Double = 1.5
6      Public Shared limitCoord As Double = 0.5
7
8      Public Shared Sub Analyse_Lines()
9
10         Dim markId1 As Session.UndoMarkId = MyProgram.theSession.SetUndoMark
             (Session.MarkVisibility.Visible, "Analyse")
11         Dim counter As Integer = 0
12         Dim startPoint3d As Point3d
13         Dim endPoint3d As Point3d
14         Dim angle As Double
15
16         For Each lin As Line In MyProgram.theWorkPart.Lines
17
18             startPoint3d = lin.StartPoint
19             endPoint3d = lin.EndPoint
20
21             'Berechnung der Steigung mit Hilfe des Start- und Endpunktes der
                 Linie
22             angle = Math.Atan((endPoint3d.Y - startPoint3d.Y) /
                 (endPoint3d.X - startPoint3d.X)) * 180 / Math.PI
23
24             If counter = 0 Then    'nur für den ersten Startpunkt
25                 Dim startPoint3dfirst As New Point3d(0, 0, 0)
26                 addToList.addToListSub3d(startPoint3dfirst)
27             End If
28
29             'Punkte der relevante Linien der Liste hinzufügen
30             If Math.Abs(angle - 90) < limitAngle Or Math.Abs(angle + 90) <
                 limitAngle Or Math.Abs(angle) < limitAngle Then
31
32                 addToList.addToListSub3d(endPoint3d)
33
34             Else
35                 'nicht relevante Linien der Deletelist hinzufügen
36                 MyProgram.theSession.UpdateManager.AddToDeleteList(lin)
37
38             End If
39
40             counter = counter + 1
41
42         Next
43
44         'letzte Linie der Lines finden
45         Dim arrayLines As Array = MyProgram.theWorkPart.Lines.ToArray
46         Dim index As Integer = arrayLines.GetUpperBound(0)
47         Dim lastLine As Line = arrayLines(index)
48         endPoint3d = lastLine.EndPoint
49         Dim endPoint3dNew As New Point3d(endPoint3d.X, endPoint3d.Y - 3, 0)
```

...ctory\Desktop\code\createSketch\createSketch\Analyse.vb 2

```
            'Korrektur der y-Koordinate um 3 mm
50          addToList.addToListSub3d(endPoint3dNew)
51
52          'Bögen der Deletelist hinzufügen
53          For Each arc As Arc In MyProgram.theWorkPart.Arcs
54              MyProgram.theSession.UpdateManager.AddToDeleteList(arc)
55          Next
56
57          MyProgram.theSession.UpdateManager.DoUpdate(markId1)
58
59      End Sub
60
61      Public Shared Sub Analyse_Points()
62
63          Dim listLength = addToList.pointList3d.Count
64          Dim index As Integer
65          Dim decision As Integer = 0 '1 für horizontale Linien, 2 für
              vertikale Linien
66          Dim xDiff As Double
67          Dim yDiff As Double
68
69          For Each p3d In addToList.pointList3d
70              index = addToList.pointList3d.IndexOf(p3d)
71
72              If index = 0 Then
73                  Continue For
74              End If
75
76              If index = listLength - 1 Then
77                  Exit For
78              End If
79
80              If Math.Abs(p3d.X) < limitCoord Then
81                  addToList.pointDeleteList3d.Add(p3d)
82                  decision = 0
83                  Continue For
84              End If
85
86              If decision = 3 Then
87                  addToList.pointDeleteList3d.Add(p3d)
88                  decision = 0
89                  Continue For
90              End If
91
92              xDiff = p3d.X - addToList.pointList3d(index + 1).X
93              yDiff = p3d.Y - addToList.pointList3d(index + 1).Y
94
95              If Math.Abs(xDiff) < limitCoord And Math.Abs(xDiff) < Math.Abs
                  (yDiff) Then  'es handelt sich um eine vertikale Linie
96
97                  If decision = 1 Then
98                      decision = 2
99                      Continue For
```

...ctory\Desktop\code\createSketch\createSketch\Analyse.vb 3

```
100                 End If
101
102                 decision = 2
103                 addToList.pointDeleteList3d.Add(p3d)
104
105             ElseIf Math.Abs(yDiff) < limitCoord And Math.Abs(yDiff) <
                  Math.Abs(xDiff) Then 'es handelt sich um eine horizontale
                  Linie
106
107                 If decision = 2 Then
108                     decision = 1
109                     Continue For
110                 End If
111
112                 decision = 1
113                 addToList.pointDeleteList3d.Add(p3d)
114
115             Else
116                 If decision = 2 Then
117                     decision = 1
118                 ElseIf decision = 1 Then
119                     decision = 2
120                 Else
121                     decision = 0
122                 End If
123
124
125                 If yDiff > 0 And yDiff > limitCoord Then
126                     decision = 3
127                 End If
128             End If
129
130         Next
131
132
133
134     End Sub
135 End Class
136
```

...ory\Desktop\code\createSketch\createSketch\addToList.vb 1

```
1  Imports NXOpen
2  Imports System
3  Public Class addToList
4      Public Shared pointList3d As New List(Of Point3d)
5      Public Shared pointDeleteList3d As New List(Of Point3d)
6      Public Shared pointList As New List(Of Point)
7      Public Shared roundPointList3d As New List(Of Point3d)
8
9      Public Shared Sub addToListSub3d(p As Point3d)
10         pointList3d.Add(p)
11     End Sub
12
13     Public Shared Sub addToDeleteListSub3d(p As Point3d)
14         pointDeleteList3d.Add(p)
15     End Sub
16
17     Public Shared Sub point3dToPoint()
18         Dim p As Point
19         For Each p3d In roundPointList3d
20             p = MyProgram.theWorkPart.Points.CreatePoint(p3d)
21             pointList.Add(p)
22             p.SetVisibility(SmartObject.VisibilityOption.Invisible)
23         Next
24     End Sub
25
26
27     Public Shared Sub roundPoint3d()
28
29         Dim index As Integer
30
31         For Each p In pointList3d
32
33             index = pointList3d.IndexOf(p)
34             Dim x As Double
35             Dim y As Double
36             Dim z As Double
37
38             If index <= 1 Then   'es handelt sich um die Punkte des
                 Werkzeuges (auf- bzw. abrunden)
39
40                 x = Math.Round(p.X)
41                 y = Math.Round(p.Y)
42                 z = Math.Round(p.Z)
43
44             Else 'es handelt sich um die Punkte des Halters (nur aufrunden)
45
46                 If index = pointList3d.Count - 1 Then
47                     x = p.X
48                     y = Math.Ceiling(p.Y)
49                     z = Math.Ceiling(p.Z)
50
51                 Else
52                     x = Math.Ceiling(p.X)
```

...ory\Desktop\code\createSketch\createSketch\addToList.vb 2

```
53                      y = Math.Ceiling(p.Y)
54                      z = Math.Ceiling(p.Z)
55                  End If
56
57
58
59
60              End If
61
62
63              Dim roundPoint3d As New Point3d(x, y, z)
64              roundPointList3d.Add(roundPoint3d)
65          Next
66      End Sub
67
68  End Class
```

...actory\Desktop\code\createSketch\createSketch\Unload.vb 1

```vb
Public Class Unload

    Public Shared Function GetUnloadOption(ByVal dummy As String) As Integer

        Dim unloadOption As Integer

        unloadOption = NXOpen.Session.LibraryUnloadOption.Immediately
          ' After executing
        'unloadOption = NXOpen.Session.LibraryUnloadOption.AtTermination
          ' When NX session terminates
        'unloadOption = NXOpen.Session.LibraryUnloadOption.Explicitly
          ' Using File-->Unload

        Return unloadOption

    End Function

End Class

```

...factory\Desktop\code\createPart\createPart\MyProgram.vb 1

```
1  Imports NXOpen
2
3  Public Class MyProgram
4      Public Shared theSession As Session = Session.GetSession()
5      Public Shared theWorkPart As Part = theSession.Parts.Work
6      Public Shared theDisplayPart As Part = theSession.Parts.Display
7      Public Shared theUFSession As UF.UFSession = UF.UFSession.GetUFSession()
8      Public Shared theUI As UI = UI.GetUI()
9      Public Shared listLineTool As New List(Of Line)
10     Public Shared listLineHolder As New List(Of Line)
11     Public Shared origin As Point3d
12     Public Shared toolSketch As Sketch
13     Public Shared holderSketch As Sketch
14
15     Public Shared Sub Main()
16
17         'Die importierte Skizze wieder löschen
18         Dim markId1 As NXOpen.Session.UndoMarkId = Nothing
19         markId1 = theSession.SetUndoMark                                    ↵
             (NXOpen.Session.MarkVisibility.Visible, "Import -.dxf")
20         Dim importedLines As NXObject()
21         importedLines = theWorkPart.Layers.GetAllObjectsOnLayer(2)
22
23         For Each impL In importedLines
24             theSession.UpdateManager.AddToDeleteList(impL)
25         Next
26
27         theSession.UpdateManager.DoUpdate(markId1)
28
29         Dim sketch_basis As Sketch()
30         sketch_basis = theWorkPart.Sketches.ToArray
31         SketchClass.moveSketch1(sketch_basis(0))
32
33         'Linien werden in zwei Listen aufgeteilt (für Halter und Werkzeug)
34         Dim i As Integer = 0
35         For Each lin As Line In theWorkPart.Lines
36             If i <= 1 Then
37                 listLineTool.Add(lin)
38             Else
39                 listLineHolder.Add(lin)
40             End If
41             i = i + 1
42         Next
43
44         Dim lastIndex As Integer = listLineHolder.Count - 1
45         origin = listLineHolder(lastIndex).StartPoint
46         listLineHolder(lastIndex).SetVisibility                             ↵
             (SmartObject.VisibilityOption.Invisible)
47         listLineHolder.RemoveAt(lastIndex)
48
49         Tool.createComponent()
50         Holder.createComponent()
51         Tool.makeWorkPart()
```

...factory\Desktop\code\createPart\createPart\MyProgram.vb 2

```
52          Tool.createSketch()
53          Tool.toolSketch.Activate(Sketch.ViewReorient.False)
54
55          For Each lin In theWorkPart.Lines
56              Tool.toolSketch.AddGeometry(lin)
57          Next
58
59          Tool.toolSketch.Deactivate(Sketch.ViewReorient.False,
              Sketch.UpdateLevel.SketchOnly)
60          Tool.moveSketch1(Tool.toolSketch)
61          Tool.closeSketch()
62          Tool.createExpressions()
63          Tool.createRevolve()
64          Holder.makeWorkPart()
65          Holder.createSketch()
66          Holder.holderSketch.Activate(Sketch.ViewReorient.False)
67
68          For Each l In theWorkPart.Lines
69              Holder.holderSketch.AddGeometry(l)
70          Next
71
72          Holder.holderSketch.Deactivate(Sketch.ViewReorient.False,
              Sketch.UpdateLevel.Model)
73          Holder.closeSketch()
74          Holder.createRevolve()
75          Tool.makeWorkPart()
76          Tool.splitBody()
77          Tool.createMCS()
78          Tool.colouring()
79          Tool.createChamfer()
80
81          If Tool.speicher <> 1 Then
82              Tool.createEdgeBlend()
83          End If
84
85          Holder.makeWorkPart()
86          Holder.createCSW()
87          Holder.createMCS()
88          Holder.colouring()
89          Tool.makeWorkPart()
90
91          'Expressions für den Export erzeugen
92          Dim exps As ExpressionCollection = theWorkPart.Expressions
93          'DC
94          Dim dc_pre As String = CType(exps.FindObject("radius").Value * 2,
              String)
95          Dim DC = exps.Create("DC = " & dc_pre)
96
97          'DN
98          Dim DN = exps.Create("DN = " & dc_pre)
99
100         'LPR
101         Dim lpr_pre As String = Tool.lpr
```

...factory\Desktop\code\createPart\createPart\MyProgram.vb 3

```
102         Dim LPR = exps.Create("LPR = " & lpr_pre)
103
104         'OAL
105         Dim OAL = exps.Create("OAL = " & exps.FindObject("laenge").Value)
106
107         'LH
108         Dim lh_pre As String = CType(DC.Value * 2.5, String)
109         Dim LH = exps.Create("LH = " & lh_pre)
110
111         'DMM
112         Dim DMM = exps.Create("DMM = " & DC.Value)
113
114         'LS
115         Dim ls_pre As String = CType(OAL.Value - LH.Value - DC.Value / 2,
              String)
116         Dim LS = exps.Create("LS = " & ls_pre)
117
118         'RE
119         Dim decisionVariable As Integer = 0
120         For Each ex In exps
121             If ex.Name = "RE" Then
122                 decisionVariable = 1
123                 Exit For
124             Else
125
126             End If
127         Next
128
129         If decisionVariable <> 1 Then
130             Dim RE = exps.Create("RE = 0")
131         End If
132
133         'CHW
134         decisionVariable = 0
135
136         For Each ex In exps
137             If ex.Name = "CHW" Then
138                 decisionVariable = 1
139                 Exit For
140             Else
141
142             End If
143         Next
144
145         If decisionVariable <> 1 Then
146             Dim CHW = exps.Create("CHW = 0")
147         End If
148
149         'KCH
150         decisionVariable = 0
151
152         For Each ex In exps
153             If ex.Name = "KCH" Then
```

...factory\Desktop\code\createPart\createPart\MyProgram.vb 4

```
154                 decisionVariable = 1
155                 Exit For
156             Else
157 
158             End If
159         Next
160 
161         If decisionVariable <> 1 Then
162             Dim RE = exps.Create("KCH = 0")
163         End If
164 
165         'AZ
166         Dim AZ = exps.Create("AZ = 0")
167 
168         'ZEFF
169         Dim ZEFF = exps.Create("ZEFF = 1")
170 
171         'ZEFP
172         Dim ZEFP = exps.Create("ZEFP = 3")
173 
174 
175     End Sub
176 
177 End Class
178 
179 
```

...ctory\Desktop\code\createPart\createPart\SketchClass.vb 1

```
1  Imports NXOpen
2  Imports System
3  Public Class SketchClass
4
5
6      Public Shared Function createSketch(sketchPlane As Plane, axis As
         Direction, origin As Point3d)
7
8          Dim sketchBuilder As SketchInPlaceBuilder = Nothing
9          sketchBuilder =
             MyProgram.theWorkPart.Sketches.CreateSketchInPlaceBuilder2(Nothing)
10
11         sketchBuilder.PlaneReference = sketchPlane
12         sketchBuilder.AxisReference = axis
13         sketchBuilder.SketchOrigin = MyProgram.theWorkPart.Points.CreatePoint
             (origin)
14         sketchBuilder.PlaneOption = Sketch.PlaneOption.Inferred
15
16         Dim sketchName As Sketch = sketchBuilder.Commit
17         sketchBuilder.Destroy()
18
19         Return sketchName
20
21     End Function
22
23
24     Public Shared Sub moveSketch1(sketch_basis As Sketch)
25
26         Dim sketchLines As New List(Of Line)
27
28         For Each lin In MyProgram.theWorkPart.Lines
29             sketchLines.Add(lin)
30         Next
31
32         Dim lastLine As Line = sketchLines.Item(sketchLines.Count - 1)
33         Dim endPoint3d As Point3d = lastLine.StartPoint          'Skizze zu
             diesem Punkt verschieben
34         Dim endPoint As Point = MyProgram.theWorkPart.Points.CreatePoint
             (endPoint3d)
35
36         Dim origin3d As New Point3d(lastLine.EndPoint.X, lastLine.EndPoint.Y,
              0)             'Skizze von diesem Punkt verschieben
37         Dim origin As Point = MyProgram.theWorkPart.Points.CreatePoint
             (origin3d)
38
39         Dim moveObjBuilder As Features.MoveObjectBuilder = Nothing
40         moveObjBuilder =
             MyProgram.theWorkPart.BaseFeatures.CreateMoveObjectBuilder(Nothing)
41
42         moveObjBuilder.ObjectToMoveObject.Add(sketch_basis)
43         moveObjBuilder.TransformMotion.Option =
             GeometricUtilities.ModlMotion.Options.PointToPoint
44         moveObjBuilder.TransformMotion.FromPoint = origin
```

...ctory\Desktop\code\createPart\createPart\SketchClass.vb 2

```
45          moveObjBuilder.TransformMotion.ToPoint = endPoint
46
47          Dim nXObject1 As NXObject = Nothing
48          nXObject1 = moveObjBuilder.Commit()
49
50          moveObjBuilder.Destroy()
51
52      End Sub
53
54      Public Shared Sub moveSketch2(sketch_basis As Sketch)
55
56          Dim sketchLines As New List(Of Line)
57
58          For Each lin In MyProgram.theWorkPart.Lines
59              sketchLines.Add(lin)
60          Next
61
62          Dim lastLine As Line = sketchLines.Item(sketchLines.Count - 1)
63          Dim endPoint3d As Point3d = lastLine.StartPoint          'Skizze zu ↵
              diesem Punkt verschieben
64          Dim endPoint As Point = MyProgram.theWorkPart.Points.CreatePoint ↵
              (endPoint3d)
65
66          Dim origin3d As New Point3d(lastLine.EndPoint.X, lastLine.EndPoint.Y, ↵
               0)             'Skizze von diesem Punkt verschieben
67          Dim origin As Point = MyProgram.theWorkPart.Points.CreatePoint ↵
              (origin3d)
68
69          Dim moveObjBuilder As Features.MoveObjectBuilder = Nothing
70          moveObjBuilder = ↵
              MyProgram.theWorkPart.BaseFeatures.CreateMoveObjectBuilder(Nothing)
71          moveObjBuilder.ObjectToMoveObject.Add(sketch_basis)
72          moveObjBuilder.TransformMotion.Option = ↵
              GeometricUtilities.ModlMotion.Options.PointToPoint
73          moveObjBuilder.TransformMotion.FromPoint = origin
74          moveObjBuilder.TransformMotion.ToPoint = endPoint
75
76          Dim nXObject1 As NXObject = Nothing
77          nXObject1 = moveObjBuilder.Commit()
78          moveObjBuilder.Destroy()
79
80      End Sub
81
82  End Class
83
```

...smartfactory\Desktop\code\createPart\createPart\Tool.vb 1

```
1  Imports NXOpen
2  Imports System
3  Public Class Tool
4      Public Shared toolSketch As Sketch
5      Public Shared lengthDiff As Double
6      Public Shared tool_component As Assemblies.Component
7
8      Public Shared Sub createComponent()
9
10         Dim path As String = MyProgram.theWorkPart.FullPath
11         Dim path_only As String
12
13         Dim slash As Integer = InStrRev(path, "\")
14         path_only = Left(path, slash)
15
16         Dim tool As FileNew = Nothing
17         tool = MyProgram.theSession.Parts.FileNew()
18         tool.TemplateFileName = "model-plain-1-mm-template.prt"
19         tool.UseBlankTemplate = False
20         tool.ApplicationName = "ModelTemplate"
21         tool.Units = Part.Units.Millimeters
22         tool.RelationType = ""
23         tool.UsesMasterModel = "No"
24         tool.TemplateType = FileNewTemplateType.Item
25         tool.TemplatePresentationName = "Model"
26         tool.ItemType = ""
27         tool.Specialization = ""
28         tool.SetCanCreateAltrep(False)
29         tool.NewFileName = path_only & "Tool.prt"
30         tool.MasterFileName = "Tool"
31         tool.MakeDisplayedPart = False
32         tool.DisplayPartOption = DisplayPartOption.AllowAdditional
33
34         Dim createNewComponentBuilder1 As
             Assemblies.CreateNewComponentBuilder = Nothing
35         createNewComponentBuilder1 =
             MyProgram.theWorkPart.AssemblyManager.CreateNewComponentBuilder()
36
37         createNewComponentBuilder1.NewComponentName = "Tool"
38         createNewComponentBuilder1.ReferenceSetName = "MODEL"
39         createNewComponentBuilder1.OriginalObjectsDeleted = True
40
41         For Each lin In MyProgram.listLineTool
42             Dim added1 As Boolean = Nothing
43             added1 = createNewComponentBuilder1.ObjectForNewComponent.Add
                 (lin)
44         Next
45
46         createNewComponentBuilder1.NewFile = tool
47         tool_component = createNewComponentBuilder1.Commit()
48         createNewComponentBuilder1.Destroy()
49
50     End Sub
```

...smartfactory\Desktop\code\createPart\createPart\Tool.vb 2

```
51
52      Public Shared Sub createSketch()
53
54          'Ebene und Achse zur Skizzenorientierung erstellen
55          Dim origin As New Point3d(MyProgram.origin.X, 0, 0)
56          Dim originPoint As Point = MyProgram.theWorkPart.Points.CreatePoint
              (origin)
57          Dim vectorX As New Vector3d(1, 0, 0)
58          Dim wcsMatrix As Matrix3x3
59          wcsMatrix.Xx = 1 : wcsMatrix.Yx = 0 : wcsMatrix.Zx = 0
60          wcsMatrix.Xy = 0 : wcsMatrix.Yy = 1 : wcsMatrix.Zy = 0
61          wcsMatrix.Xz = 0 : wcsMatrix.Yz = 0 : wcsMatrix.Zz = 1
62          Dim sketchPlane As Plane =
              MyProgram.theWorkPart.Planes.CreateFixedPlane(origin, wcsMatrix)
63          Dim axisX As Direction =
              MyProgram.theWorkPart.Directions.CreateDirection(originPoint,
              vectorX)
64          sketchPlane.Blank()
65          axisX.Blank()
66
67          toolSketch = SketchClass.createSketch(sketchPlane, axisX, origin)
68          toolSketch.SetName("toolSketch")
69
70
71      End Sub
72      Public Shared Sub moveSketch1(toolSketch As Sketch)
73
74
75          Dim sketchLines As New List(Of Line)
76
77          For Each lin In MyProgram.theWorkPart.Lines
78              sketchLines.Add(lin)
79          Next
80          Dim lastLine As Line = sketchLines.Item(sketchLines.Count - 1)
81          Dim targetPoint3d As New Point3d(toolSketch.Origin.X - 2, 0, 0)
                'Skizze zu diesem Punkt verschieben
82          Dim targetPoint As Point = MyProgram.theWorkPart.Points.CreatePoint
              (targetPoint3d)
83
84          Dim origin3d As Point3d = toolSketch.Origin            'Skizze von
              diesem Punkt verschieben
85          Dim origin As Point = MyProgram.theWorkPart.Points.CreatePoint
              (origin3d)
86
87          Dim moveObjBuilder As Features.MoveObjectBuilder = Nothing
88          moveObjBuilder =
              MyProgram.theWorkPart.BaseFeatures.CreateMoveObjectBuilder
              (Nothing)
89
90          moveObjBuilder.ObjectToMoveObject.Add(toolSketch)
91          moveObjBuilder.TransformMotion.Option =
              GeometricUtilities.ModlMotion.Options.PointToPoint
92          moveObjBuilder.TransformMotion.FromPoint = origin
```

...smartfactory\Desktop\code\createPart\createPart\Tool.vb 3

```
 93          moveObjBuilder.TransformMotion.ToPoint = targetPoint
 94
 95          Dim nXObject1 As NXObject = Nothing
 96          nXObject1 = moveObjBuilder.Commit()
 97          moveObjBuilder.Destroy()
 98
 99      End Sub
100      Public Shared Sub makeWorkPart()
101
102          Dim comp As Assemblies.Component = tool_component
103
104          Dim partLoadStat As PartLoadStatus = Nothing
105          MyProgram.theSession.Parts.SetWorkComponent(comp,
               PartCollection.RefsetOption.Entire,
               PartCollection.WorkComponentOption.Visible, partLoadStat)
106
107          MyProgram.theWorkPart = MyProgram.theSession.Parts.Work ' Tool
108          partLoadStat.Dispose()
109
110      End Sub
111
112      Public Shared lpr As String
113      Public Shared radius As Double
114
115      Public Shared Sub createExpressions()
116
117          Dim lineList As New List(Of Line)
118
119          For Each l In MyProgram.theWorkPart.Lines
120              lineList.Add(l)
121          Next
122
123          toolSketch.Activate(Sketch.ViewReorient.False)
124          Dim geom1 As New Sketch.ConstraintGeometry
125          geom1.Geometry = lineList(0)
126          geom1.PointType = Sketch.ConstraintPointType.StartVertex
127          geom1.SplineDefiningPointIndex = 0
128          Dim geom2 As New Sketch.ConstraintGeometry
129          geom2.Geometry = MyProgram.theWorkPart.Points.CreatePoint
               (toolSketch.Origin)
130          geom2.SplineDefiningPointIndex = 0
131          MyProgram.theSession.ActiveSketch.CreateCoincidentConstraint(geom1,
               geom2)
132
133          'Expression für den Radius
134          Dim geom3 As New Sketch.DimensionGeometry
135          geom3.Geometry = lineList(0)
136          geom3.AssocType = Sketch.AssocType.StartPoint
137          Dim geom4 As New Sketch.DimensionGeometry
138          geom4.Geometry = lineList(0)
139          geom4.AssocType = Sketch.AssocType.EndPoint
140          Dim exps As ExpressionCollection = MyProgram.theWorkPart.Expressions
141          Dim text_ver As String = "radius = " & lineList(0).GetLength
```

...smartfactory\Desktop\code\createPart\createPart\Tool.vb 4

```
142         Dim exp_dmm As Expression = exps.Create(text_ver)
143         radius = lineList(0).GetLength
144         MyProgram.theSession.ActiveSketch.CreateDimension
              (Sketch.ConstraintType.VerticalDim, geom3, geom4,
              toolSketch.Origin, exp_dmm)
145
146         'Expression für die Länge
147         Dim geom5 As New Sketch.DimensionGeometry
148         geom5.Geometry = lineList(1)
149         geom5.AssocType = Sketch.AssocType.StartPoint
150         Dim geom6 As New Sketch.DimensionGeometry
151         geom6.Geometry = lineList(1)
152         geom6.AssocType = Sketch.AssocType.EndPoint
153         Dim text_hor As String = "laenge = " & lineList(1).GetLength
154         Dim exp_ovlen As Expression = exps.Create(text_hor)
155         MyProgram.theSession.ActiveSketch.CreateDimension
              (Sketch.ConstraintType.HorizontalDim, geom5, geom6, lineList
              (1).StartPoint, exp_ovlen)
156
157         lengthDiff = lineList(1).GetLength / 2
158         lpr = CType(lineList(1).GetLength - 2, String)
159         Dim len_new As Integer = CInt(lineList(1).GetLength + lengthDiff)
160         exp_ovlen.RightHandSide = len_new.ToString
161
162         Dim geom7 As New Sketch.ConstraintGeometry
163         geom7.Geometry = lineList(2)
164         Dim geom8 As New Sketch.ConstraintGeometry
165         geom8.Geometry = lineList(0)
166         MyProgram.theSession.ActiveSketch.CreateEqualLengthConstraint(geom7,
              geom8)
167
168         Dim geom9 As New Sketch.ConstraintGeometry
169         geom9.Geometry = lineList(3)
170         Dim geom10 As New Sketch.ConstraintGeometry
171         geom10.Geometry = lineList(1)
172         MyProgram.theSession.ActiveSketch.CreateEqualLengthConstraint(geom9,
              geom10)
173
174         toolSketch.Deactivate(Sketch.ViewReorient.False,
              Sketch.UpdateLevel.SketchOnly)
175
176     End Sub
177     Public Shared tooltip As Point
178     Public Shared Sub closeSketch()
179
180         Dim lineList As New List(Of Line)
181
182         For Each l In MyProgram.theWorkPart.Lines
183             lineList.Add(l)
184         Next
185
186         Dim origin3d As Point3d = toolSketch.Origin
187         Dim lastIndex As Integer = lineList.Count - 1
```

...smartfactory\Desktop\code\createPart\createPart\Tool.vb 5

```
188         Dim lastLine As Line = lineList(lastIndex)
189         Dim lastPoint3d As Point3d = lastLine.EndPoint
190 
191         'Tooltip erstellen
192         tooltip = MyProgram.theWorkPart.Points.CreatePoint(origin3d)
193         tooltip.SetVisibility(SmartObject.VisibilityOption.Visible)
194 
195         Dim startPoint3d As Point3d = lastPoint3d
196         Dim endPoint3d As New Point3d(lastPoint3d.X, origin3d.Y, 0)
197         Dim vertLine As Line = MyProgram.theWorkPart.Curves.CreateLine
              (startPoint3d, endPoint3d)
198 
199         startPoint3d = endPoint3d
200         endPoint3d = origin3d
201         Dim horLine As Line = MyProgram.theWorkPart.Curves.CreateLine
              (startPoint3d, endPoint3d)
202         toolSketch.Activate(Sketch.ViewReorient.False)
203         toolSketch.AddGeometry(vertLine)
204         toolSketch.AddGeometry(horLine)
205         toolSketch.Deactivate(Sketch.ViewReorient.False,
              Sketch.UpdateLevel.SketchOnly)
206         toolSketch.Blank()
207 
208     End Sub
209 
210 
211     Public Shared revolveFeature As Features.Revolve
212 
213     Public Shared Sub createRevolve()
214 
215         Dim i As Integer = 0
216         Dim lineList As New List(Of Line)
217 
218         Dim rul As New List(Of CurveDumbRule)
219 
220         Dim ctol = 0.0095   'chaining tolerance
221         Dim dtol = 0.01     'distance tolerance
222         Dim atol = 0.5      'angle tolerance
223 
224         Dim sect As Section = MyProgram.theWorkPart.Sections.CreateSection
              (ctol, dtol, atol)
225 
226         Dim helpPoint As New Point3d(0, 0, 0)
227         Dim nullObj As NXObject = Nothing
228         Dim noChain As Boolean = False
229         Dim createMode As Section.Mode = Section.Mode.Create
230 
231         For Each linie As Line In MyProgram.theWorkPart.Lines
232             lineList.Add(linie)
233         Next
234 
235         Dim toolBase As BasePart = MyProgram.theWorkPart
236         For Each l In lineList
```

...smartfactory\Desktop\code\createPart\createPart\Tool.vb 6

```
237             rul.Add(toolBase.ScRuleFactory.CreateRuleBaseCurveDumb({l}))
238         Next
239
240         For i = 0 To lineList.Count - 1
241             sect.AddToSection({rul(i)}, lineList(i), nullObj, nullObj,
                  helpPoint, createMode, noChain)
242         Next
243
244         'Drehoperation
245         Dim builder = MyProgram.theWorkPart.Features.CreateRevolveBuilder
              (Nothing)
246
247         builder.Section = sect
248
249         Dim axisPoint3d As New Point3d(0, 0, 0)
250         Dim axisVector As New Vector3d(1, 0, 0)
251         Dim updateOption = SmartObject.UpdateOption.WithinModeling
252
253         Dim direction = MyProgram.theWorkPart.Directions.CreateDirection
              (axisPoint3d, axisVector, updateOption)
254         Dim axisPoint As Point = MyProgram.theWorkPart.Points.CreatePoint
              (axisPoint3d)
255         builder.Axis = MyProgram.theWorkPart.Axes.CreateAxis(axisPoint,
              direction, updateOption)
256
257         builder.Limits.StartExtend.Value.RightHandSide = "0"
258         builder.Limits.EndExtend.Value.RightHandSide = "360"
259
260         revolveFeature = builder.CommitFeature
261         builder.Destroy()
262
263     End Sub
264
265     Public Shared tool_body As Body
266     Public Shared split_body As Features.SplitBody
267     Public Shared apmx_value As Double
268
269     Public Shared Sub splitBody()
270
271         Dim origin As New Point3d(Holder.coordsys_point.X, 0, 0)
272         Dim wcsMatrix As Matrix3x3
273         wcsMatrix.Xx = 0 : wcsMatrix.Yx = 0 : wcsMatrix.Zx = 0
274         wcsMatrix.Xy = 0 : wcsMatrix.Yy = 1 : wcsMatrix.Zy = 0
275         wcsMatrix.Xz = -1 : wcsMatrix.Yz = 0 : wcsMatrix.Zz = 1
276         Dim x As Double = origin.X + (Tool.toolSketch.Origin.X - origin.X)
277         Dim split_point As New Point3d(x, 0, 0)
278         Dim split_plane As DatumPlane =
              MyProgram.theWorkPart.Datums.CreateFixedDatumPlane(split_point,
              wcsMatrix)
279
280         'SplitBody
281         Dim splitBodyBuilder As NXOpen.Features.SplitBodyBuilder = Nothing
282         splitBodyBuilder =
```

...smartfactory\Desktop\code\createPart\createPart\Tool.vb 7

```
            MyProgram.theWorkPart.Features.CreateSplitBodyBuilderUsingCollecto
            r(Nothing)
283
284         Dim origin1 As NXOpen.Point3d = New NXOpen.Point3d(0.0, 0.0, 0.0)
285         Dim normal1 As NXOpen.Vector3d = New NXOpen.Vector3d(0.0, 0.0, 1.0)
286         Dim plane1 As NXOpen.Plane = Nothing
287         plane1 = MyProgram.theWorkPart.Planes.CreatePlane(origin1, normal1,
              NXOpen.SmartObject.UpdateOption.WithinModeling)
288
289         splitBodyBuilder.BooleanTool.FacePlaneTool.ToolPlane = plane1
290         splitBodyBuilder.BooleanTool.ToolOption =
              NXOpen.GeometricUtilities.BooleanToolBuilder.BooleanToolType.NewPl
              ane
291
292         Dim scColl As NXOpen.ScCollector = Nothing
293         scColl = MyProgram.theWorkPart.ScCollectors.CreateCollector()
294
295         For Each bod In MyProgram.theWorkPart.Bodies
296             tool_body = bod
297         Next
298
299         Dim tool_bodies(0) As Body
300         tool_bodies(0) = tool_body
301         Dim bodyDumbRule As BodyDumbRule = Nothing
302         Dim toolBase As BasePart = MyProgram.theWorkPart
303         bodyDumbRule = toolBase.ScRuleFactory.CreateRuleBodyDumb
              (tool_bodies, True)
304
305         Dim rules(0) As SelectionIntentRule
306         rules(0) = bodyDumbRule
307         scColl.ReplaceRules(rules, False)
308
309         splitBodyBuilder.TargetBodyCollector = scColl
310
311         plane1.SetMethod(NXOpen.PlaneTypes.MethodType.Distance)
312
313         Dim geom1(0) As NXOpen.NXObject
314
315         geom1(0) = split_plane
316         plane1.SetGeometry(geom1)
317         plane1.SetFlip(False)
318         plane1.SetReverseSide(True)
319
320         Dim exps As ExpressionCollection = MyProgram.theWorkPart.Expressions
321         Dim APMX As Expression = plane1.Expression
322         APMX.RightHandSide = CType(exps.FindObject("radius").Value * 4,
              String)
323         exps.Rename(APMX, "APMX")
324         plane1.SetAlternate(NXOpen.PlaneTypes.AlternateType.One)
325         plane1.Evaluate()
326
327         Dim nXObject1 As NXOpen.NXObject = Nothing
328         nXObject1 = splitBodyBuilder.Commit()
```

...smartfactory\Desktop\code\createPart\createPart\Tool.vb 8

```
329                                                                          ↵
    splitBodyBuilder.BooleanTool.ExtrudeRevolveTool.ToolSection.CleanMappingData ↵
    ()
330         splitBodyBuilder.Destroy()
331
332
333     End Sub
334
335     Public Shared Sub createMCS()
336
337         Dim datumCoordBuilder As Features.DatumCsysBuilder = Nothing
338         datumCoordBuilder =                                              ↵
              MyProgram.theWorkPart.Features.CreateDatumCsysBuilder(Nothing)
339
340         Dim origin As New Point3d(Holder.coordsys_point.X, 0, 0)
341         Dim matrix As Matrix3x3 = Nothing
342         matrix.Xx = 0 : matrix.Yx = 0 : matrix.Zx = -1
343         matrix.Xy = 0 : matrix.Yy = 1 : matrix.Zy = 0
344         matrix.Xz = 1 : matrix.Yz = 0 : matrix.Zz = 0
345
346         Dim coordSys As CartesianCoordinateSystem = Nothing
347         coordSys =                                                       ↵
              MyProgram.theWorkPart.CoordinateSystems.CreateCoordinateSystem ↵
              (origin, matrix, isTemporary:=True)
348
349         datumCoordBuilder.Csys = coordSys
350         datumCoordBuilder.DisplayScaleFactor = 1.25
351
352         Dim coordMCS As Features.DatumCsys = datumCoordBuilder.Commit()
353         datumCoordBuilder.Destroy()
354
355         'Umbennenen des Koordinatensystems
356         coordMCS.SetName("MCS")
357
358         Dim coords(0) As Features.DatumCsys
359         coords(0) = coordMCS
360
361         Dim featurePropBuilder As FeatureGeneralPropertiesBuilder = Nothing
362         featurePropBuilder =                                             ↵
              MyProgram.theWorkPart.PropertiesManager.CreateFeatureGeneralProper ↵
              tiesBuilder(coords)
363
364         featurePropBuilder.FeatureName = "MCS"
365         featurePropBuilder.Destroy()
366
367     End Sub
368
369
370     Public Shared Sub colouring()
371
372         If selectObject(UF.UFConstants.UF_solid_type,                    ↵
              UF.UFConstants.UF_solid_body_subtype, Nothing, "Selektiere den ↵
              cutting-Teil des Werkstückes!", "Auswahl des cutting-Teils") = ↵
```

...smartfactory\Desktop\code\createPart\createPart\Tool.vb 9

```
            Selection.Response.Ok Then
373
374             Dim displayMod As DisplayModification = Nothing
375             displayMod =
                  MyProgram.theSession.DisplayManager.NewDisplayModification()
376
377             displayMod.ApplyToAllFaces = True
378             displayMod.ApplyToOwningParts = False
379             displayMod.NewColor = 42
380
381             Dim cuttings(0) As Body
382             cuttings(0) = obj
383
384             displayMod.Apply(cuttings)
385             displayMod.Dispose()
386
387         End If
388
389     End Sub
390     Public Shared speicher As Integer
391     Public Shared Sub createChamfer()
392
393         If selectObject(UF.UFConstants.UF_solid_type,
              UF.UFConstants.UF_solid_body_subtype,
              UF.UFConstants.UF_UI_SEL_FEATURE_ANY_EDGE, "Selektiere die Kante
              für die Fase!", "Auswahl der Fase") = Selection.Response.Ok Then
394
395             Dim chamfBuilder As Features.ChamferBuilder = Nothing
396             chamfBuilder =
                  MyProgram.theWorkPart.Features.CreateChamferBuilder(Nothing)
397
398             chamfBuilder.Option =
                  Features.ChamferBuilder.ChamferOption.OffsetAndAngle
399             chamfBuilder.Method =
                  Features.ChamferBuilder.OffsetMethod.EdgesAlongFaces
400
401             Dim scColl As ScCollector = Nothing
402             scColl = MyProgram.theWorkPart.ScCollectors.CreateCollector()
403
404             Dim choosenEdge As Edge = obj
405
406             Dim toolBase As BasePart = MyProgram.theWorkPart
407             Dim edgeRule As EdgeTangentRule = Nothing
408             edgeRule = toolBase.ScRuleFactory.CreateRuleEdgeTangent
                  (choosenEdge, Nothing, False, 0.5, True, False)
409
410             Dim rules1(0) As SelectionIntentRule
411             rules1(0) = edgeRule
412             scColl.ReplaceRules(rules1, False)
413
414             chamfBuilder.SmartCollector = scColl
415
416             Dim ang As Double = 45
```

...smartfactory\Desktop\code\createPart\createPart\Tool.vb 10

```
417             Dim depth As Double = 1
418
419             chamfBuilder.AngleExp.RightHandSide = ang.ToString
420             chamfBuilder.FirstOffsetExp.RightHandSide = depth
421
422             Dim feat As Features.Feature = Nothing
423             feat = chamfBuilder.CommitFeature()
424             chamfBuilder.Destroy()
425
426             'Expression für die Fase erstellen
427             Dim exps As ExpressionCollection =
                  MyProgram.theWorkPart.Expressions
428
429             'CHW
430             Dim exFindCHW = exps.FindObject("p3")
431             exps.Rename(exFindCHW, "CHW")
432             exps.Edit(exFindCHW, "0.1")
433
434             'KCH
435             Dim exFindKCH = exps.FindObject("p5")
436             exps.Rename(exFindKCH, "KCH")
437             speicher = 1
438         End If
439
440     End Sub
441
442     Public Shared Sub createEdgeBlend()
443
444         If selectObject(UF.UFConstants.UF_solid_type,
              UF.UFConstants.UF_solid_body_subtype,
              UF.UFConstants.UF_UI_SEL_FEATURE_ANY_EDGE, "Selektiere die Kante
              für die Rundung!", "Auswahl der Rundung") = Selection.Response.Ok
              Then
445
446             Dim edgeBuilder As Features.EdgeBlendBuilder = Nothing
447             edgeBuilder =
                  MyProgram.theWorkPart.Features.CreateEdgeBlendBuilder(Nothing)
448
449             Dim blendLimitsData1 As GeometricUtilities.BlendLimitsData =
                  Nothing
450             blendLimitsData1 = edgeBuilder.LimitsListData
451
452             Dim scCollector1 As ScCollector = Nothing
453             scCollector1 =
                  MyProgram.theWorkPart.ScCollectors.CreateCollector()
454
455             Dim toolBase As BasePart = MyProgram.theWorkPart
456             Dim choosenEdge As Edge = obj
457             Dim edgeRule As EdgeTangentRule = Nothing
458             edgeRule = toolBase.ScRuleFactory.CreateRuleEdgeTangent
                  (choosenEdge, Nothing, False, 0.5, True, False)
459
460             Dim rules1(0) As SelectionIntentRule
```

...smartfactory\Desktop\code\createPart\createPart\Tool.vb 11

```
461             rules1(0) = edgeRule
462             scCollector1.ReplaceRules(rules1, False)
463 
464             edgeBuilder.Tolerance = 0.01
465             edgeBuilder.AllInstancesOption = False
466             edgeBuilder.RemoveSelfIntersection = True
467             edgeBuilder.PatchComplexGeometryAreas = True
468             edgeBuilder.LimitFailingAreas = True
469             edgeBuilder.ConvexConcaveY = False
470             edgeBuilder.RollOverSmoothEdge = True
471             edgeBuilder.RollOntoEdge = True
472             edgeBuilder.MoveSharpEdge = True
473             edgeBuilder.TrimmingOption = False
474             edgeBuilder.OverlapOption =                                    ↵
                  Features.EdgeBlendBuilder.Overlap.AnyConvexityRollOver
475             edgeBuilder.BlendOrder =                                       ↵
                  Features.EdgeBlendBuilder.OrderOfBlending.ConvexFirst
476             edgeBuilder.SetbackOption =                                    ↵
                  Features.EdgeBlendBuilder.Setback.SeparateFromCorner
477             edgeBuilder.BlendFaceContinuity =                              ↵
                  Features.EdgeBlendBuilder.FaceContinuity.Tangent
478 
479             Dim csIndex1 As Integer = Nothing
480             Dim radiusBlend As Double = 1
481 
482             csIndex1 = edgeBuilder.AddChainset(scCollector1, radiusBlend)
483 
484             Dim feat As Features.Feature = Nothing
485             feat = edgeBuilder.CommitFeature()
486             edgeBuilder.Destroy()
487 
488             'Expression für die Rundung erstellen
489             Dim exps As ExpressionCollection =                             ↵
                  MyProgram.theWorkPart.Expressions
490 
491             'RE
492             Dim reFind = exps.FindObject("p3")
493             exps.Edit(reFind, "0.2")
494 
495         End If
496 
497     End Sub
498 
499     Public Shared obj As NXObject
500     Public Shared antwort As Selection.Response
501     Public Shared Function selectObject(type As Integer, subtype As Integer,↵
          solidbodysubtype As Integer, text As String, titel As String) 'aus   ↵
        dem Buch NX Systembetreuer
502 
503         Dim selman As Selection = UI.GetUI.SelectionManager
504 
505         'Maske für die Selektion (welche Objekte sind auswählbar..)
506         Dim mask(0) As Selection.MaskTriple
```

...smartfactory\Desktop\code\createPart\createPart\Tool.vb 12

```
507         mask(0).Type = type
508         mask(0).Subtype = subtype
509         mask(0).SolidBodySubtype = solidbodysubtype
510 
511         Dim selAction As Selection.SelectionAction =
              Selection.SelectionAction.ClearAndEnableSpecific
512         Dim auswahlbereich As Selection.SelectionScope =
              Selection.SelectionScope.WorkPart
513         Dim auch_feature As Boolean = False
514         Dim hervorherbung_behalten As Boolean = False
515         Dim position As Point3d
516 
517         'Aufruf der Selektion
518         antwort = selman.SelectTaggedObject(text, titel, auswahlbereich,
              selAction, auch_feature, hervorherbung_behalten, mask, obj,
              position)
519 
520         'Auswerten der Antwort
521         If antwort = Selection.Response.ObjectSelected Or antwort =
              Selection.Response.ObjectSelectedByName Then
522             antwort = Selection.Response.Ok
523             Return antwort
524 
525         Else
526             antwort = Selection.Response.Cancel
527             Return antwort
528         End If
529 
530     End Function
531 
532 End Class
533 
```

...artfactory\Desktop\code\createPart\createPart\Holder.vb 1

```
 1  Imports NXOpen
 2
 3  Public Class Holder
 4      Public Shared holderSketch As Sketch
 5      Public Shared coordsys_point As Point3d
 6      Public Shared holder_component As Assemblies.Component
 7      Public Shared Sub createComponent()
 8
 9          Dim path As String = MyProgram.theWorkPart.FullPath
10          Dim path_only As String
11
12          Dim slash As Integer = InStrRev(path, "\")
13          path_only = Left(path, slash)
14
15          Dim holder As FileNew = Nothing
16          holder = MyProgram.theSession.Parts.FileNew()
17          holder.TemplateFileName = "model-plain-1-mm-template.prt"
18          holder.UseBlankTemplate = False
19          holder.ApplicationName = "ModelTemplate"
20          holder.Units = Part.Units.Millimeters
21          holder.RelationType = ""
22          holder.UsesMasterModel = "No"
23          holder.TemplateType = FileNewTemplateType.Item
24          holder.TemplatePresentationName = "Model"
25          holder.ItemType = ""
26          holder.Specialization = ""
27          holder.SetCanCreateAltrep(False)
28          holder.NewFileName = path_only & "Holder.prt"
29          holder.MasterFileName = "Holder"
30          holder.MakeDisplayedPart = False
31          holder.DisplayPartOption = DisplayPartOption.AllowAdditional
32
33          Dim createNewComponentBuilder1 As
              Assemblies.CreateNewComponentBuilder = Nothing
34          createNewComponentBuilder1 =
              MyProgram.theWorkPart.AssemblyManager.CreateNewComponentBuilder()
35
36          createNewComponentBuilder1.NewComponentName = "Holder"
37          createNewComponentBuilder1.ReferenceSetName = "MODEL"
38          createNewComponentBuilder1.OriginalObjectsDeleted = True
39
40          For Each lin In MyProgram.listLineHolder
41              Dim added1 As Boolean = Nothing
42              added1 = createNewComponentBuilder1.ObjectForNewComponent.Add
                  (lin)
43          Next
44
45          createNewComponentBuilder1.NewFile = holder
46          holder_component = createNewComponentBuilder1.Commit()
47          createNewComponentBuilder1.Destroy()
48
49      End Sub
50
```

...artfactory\Desktop\code\createPart\createPart\Holder.vb 2

```
51      Public Shared Sub createSketch()
52
53          'create a plane and a axis to control the orientation of our sketch
54          Dim origin As Point3d = MyProgram.origin
55          Dim originPoint As Point = MyProgram.theWorkPart.Points.CreatePoint
              (origin)
56          Dim vectorX As New Vector3d(1, 0, 0)
57          Dim wcsMatrix As Matrix3x3
58          wcsMatrix.Xx = 1 : wcsMatrix.Yx = 0 : wcsMatrix.Zx = 0
59          wcsMatrix.Xy = 0 : wcsMatrix.Yy = 1 : wcsMatrix.Zy = 0
60          wcsMatrix.Xz = 0 : wcsMatrix.Yz = 0 : wcsMatrix.Zz = 1
61          Dim sketchPlane As Plane =
              MyProgram.theWorkPart.Planes.CreateFixedPlane(origin, wcsMatrix)
62          Dim axisX As Direction =
              MyProgram.theWorkPart.Directions.CreateDirection(originPoint,
              vectorX)
63          sketchPlane.Blank()
64          axisX.Blank()
65
66          holderSketch = SketchClass.createSketch(sketchPlane, axisX, origin)
67          holderSketch.SetName("holderSketch")
68
69      End Sub
70
71      Public Shared Sub makeWorkPart()
72          Dim comp As Assemblies.Component = holder_component
73
74          Dim partLoadStat As PartLoadStatus = Nothing
75          MyProgram.theSession.Parts.SetWorkComponent(comp,
              PartCollection.RefsetOption.Entire,
              PartCollection.WorkComponentOption.Visible, partLoadStat)
76
77          MyProgram.theWorkPart = MyProgram.theSession.Parts.Work ' Holder
78          partLoadStat.Dispose()
79
80      End Sub
81
82      Public Shared Sub closeSketch()
83
84          Dim lineListHolder As New List(Of Line)
85
86          For Each l In MyProgram.theWorkPart.Lines
87              lineListHolder.Add(l)
88          Next
89
90          Dim startPoint3d As Point3d = lineListHolder(0).StartPoint
91          coordsys_point = startPoint3d
92          Dim lastIndex = lineListHolder.Count - 1
93          Dim endPoint3d As Point3d = lineListHolder(lineListHolder.Count -
              1).EndPoint
94
95          Dim startP3d As Point3d = endPoint3d
96          Dim endP3d As New Point3d(2, 0, 0)
```

...artfactory\Desktop\code\createPart\createPart\Holder.vb 3

```
 97          Dim horLine As Line = MyProgram.theWorkPart.Curves.CreateLine
               (startP3d, endP3d)
 98
 99          startP3d = endP3d
100          endP3d = New Point3d(startP3d.X, startPoint3d.Y, 0)
101          Dim vertLine As Line = MyProgram.theWorkPart.Curves.CreateLine
               (startP3d, endP3d)
102
103          startP3d = endP3d
104          endP3d = startPoint3d
105          Dim lastLine As Line = MyProgram.theWorkPart.Curves.CreateLine
               (startP3d, endP3d)
106
107          holderSketch.Activate(Sketch.ViewReorient.False)
108          holderSketch.AddGeometry(horLine)
109          holderSketch.AddGeometry(vertLine)
110          holderSketch.AddGeometry(lastLine)
111          holderSketch.Deactivate(Sketch.ViewReorient.False,
               Sketch.UpdateLevel.SketchOnly)
112          holderSketch.Blank()
113
114
115      End Sub
116
117
118      Public Shared revolveFeature As Features.Revolve
119      Public Shared Sub createRevolve()
120
121          Dim i As Integer = 0
122          Dim lineList As New List(Of Line)
123
124          Dim rul As New List(Of CurveDumbRule)
125
126          Dim ctol = 0.0095   'chaining tolerance
127          Dim dtol = 0.01     'distance tolerance
128          Dim atol = 0.5      'angle tolerance
129
130          Dim sect As Section = MyProgram.theWorkPart.Sections.CreateSection
               (ctol, dtol, atol)
131
132          Dim helpPoint As New Point3d(0, 0, 0)
133          Dim nullObj As NXObject = Nothing
134          Dim noChain As Boolean = False
135          Dim createMode As Section.Mode = Section.Mode.Create
136
137          For Each linie As Line In MyProgram.theWorkPart.Lines
138              lineList.Add(linie)
139          Next
140
141          Dim holderBase As BasePart = MyProgram.theWorkPart
142          For Each l In lineList
143              rul.Add(holderBase.ScRuleFactory.CreateRuleBaseCurveDumb({l}))
144          Next
```

...artfactory\Desktop\code\createPart\createPart\Holder.vb 4

```
145
146         For i = 0 To lineList.Count - 1
147             sect.AddToSection({rul(i)}, lineList(i), nullObj, nullObj,
                  helpPoint, createMode, noChain)
148         Next
149
150         'Drehoperation
151         Dim builder = MyProgram.theWorkPart.Features.CreateRevolveBuilder
              (Nothing)
152
153         builder.Section = sect
154
155         Dim axisPoint3d As New Point3d(0, 0, 0)
156         Dim axisVector As New Vector3d(1, 0, 0)
157         Dim updateOption = SmartObject.UpdateOption.WithinModeling
158
159         Dim direction = MyProgram.theWorkPart.Directions.CreateDirection
              (axisPoint3d, axisVector, updateOption)
160         Dim axisPoint As Point = MyProgram.theWorkPart.Points.CreatePoint
              (axisPoint3d)
161         builder.Axis = MyProgram.theWorkPart.Axes.CreateAxis(axisPoint,
              direction, updateOption)
162
163         builder.Limits.StartExtend.Value.RightHandSide = "0"
164         builder.Limits.EndExtend.Value.RightHandSide = "360"
165
166         revolveFeature = builder.CommitFeature
167         builder.Destroy()
168     End Sub
169
170     Public Shared Sub createCSW()
171
172         Dim datumCoordBuilder As Features.DatumCsysBuilder = Nothing
173         datumCoordBuilder =
              MyProgram.theWorkPart.Features.CreateDatumCsysBuilder(Nothing)
174
175         Dim origin As New Point3d(Holder.coordsys_point.X, 0, 0)
176         Dim matrix As Matrix3x3 = Nothing
177         matrix.Xx = 0 : matrix.Yx = 0 : matrix.Zx = -1
178         matrix.Xy = 0 : matrix.Yy = 1 : matrix.Zy = 0
179         matrix.Xz = 1 : matrix.Yz = 0 : matrix.Zz = 0
180         Dim coordSys As CartesianCoordinateSystem = Nothing
181         coordSys =
              MyProgram.theWorkPart.CoordinateSystems.CreateCoordinateSystem
              (origin, matrix, isTemporary:=True)
182
183         datumCoordBuilder.Csys = coordSys
184         datumCoordBuilder.DisplayScaleFactor = 1.25
185
186         Dim coordCSW As Features.DatumCsys = datumCoordBuilder.Commit()
187         datumCoordBuilder.Destroy()
188
189         'Umbennenen des Koordinatensystems
```

...artfactory\Desktop\code\createPart\createPart\Holder.vb 5

```
190         coordCSW.SetName("CSW")
191
192         Dim coords(0) As Features.DatumCsys
193         coords(0) = coordCSW
194
195         Dim featurePropBuilder As FeatureGeneralPropertiesBuilder = Nothing
196         featurePropBuilder =
              MyProgram.theWorkPart.PropertiesManager.CreateFeatureGeneralProper
              tiesBuilder(coords)
197         featurePropBuilder.FeatureName = "CSW"
198         featurePropBuilder.Destroy()
199
200     End Sub
201
202     Public Shared Sub createMCS()
203
204         Dim datumCoordBuilder As Features.DatumCsysBuilder = Nothing
205         datumCoordBuilder =
              MyProgram.theWorkPart.Features.CreateDatumCsysBuilder(Nothing)
206
207         Dim origin As New Point3d(0, 0, 0)
208         Dim matrix As Matrix3x3 = Nothing
209         matrix.Xx = 0 : matrix.Yx = 0 : matrix.Zx = -1
210         matrix.Xy = 0 : matrix.Yy = 1 : matrix.Zy = 0
211         matrix.Xz = 1 : matrix.Yz = 0 : matrix.Zz = 0
212         Dim coordSys As CartesianCoordinateSystem = Nothing
213         coordSys =
              MyProgram.theWorkPart.CoordinateSystems.CreateCoordinateSystem
              (origin, matrix, isTemporary:=True)
214
215         datumCoordBuilder.Csys = coordSys
216         datumCoordBuilder.DisplayScaleFactor = 1.25
217
218         Dim coordMCS As Features.DatumCsys = datumCoordBuilder.Commit()
219         datumCoordBuilder.Destroy()
220
221         'Umbennenen des Koordinatensystems
222         coordMCS.SetName("MCS")
223
224         Dim coords(0) As Features.DatumCsys
225         coords(0) = coordMCS
226
227         Dim featurePropBuilder As FeatureGeneralPropertiesBuilder = Nothing
228         featurePropBuilder =
              MyProgram.theWorkPart.PropertiesManager.CreateFeatureGeneralProper
              tiesBuilder(coords)
229         featurePropBuilder.FeatureName = "MCS"
230         featurePropBuilder.Destroy()
231
232     End Sub
233
234     Public Shared Sub colouring()
235
```

...artfactory\Desktop\code\createPart\createPart\Holder.vb 6

```
236         Dim displayMod As DisplayModification = Nothing
237         displayMod =
              MyProgram.theSession.DisplayManager.NewDisplayModification()
238
239         displayMod.ApplyToAllFaces = True
240         displayMod.ApplyToOwningParts = False
241         displayMod.NewColor = 114
242
243         Dim bodies() = revolveFeature.GetBodies
244
245         displayMod.Apply(bodies)
246         displayMod.Dispose()
247
248
249     End Sub
250
251 End Class
252
```

...artfactory\Desktop\code\createPart\createPart\Unload.vb 1

```
1  Public Class Unload
2
3      Public Shared Function GetUnloadOption(ByVal dummy As String) As Integer
4
5          Dim unloadOption As Integer
6
7          unloadOption = NXOpen.Session.LibraryUnloadOption.Immediately
             ' After executing
8          'unloadOption = NXOpen.Session.LibraryUnloadOption.AtTermination
             ' When NX session terminates
9          'unloadOption = NXOpen.Session.LibraryUnloadOption.Explicitly
             ' Using File-->Unload
10
11         Return unloadOption
12
13     End Function
14
15 End Class
16
```

...p\code\exportExpressions\exportExpressions\MyProgram.vb 1

```
1  Imports NXOpen
2  Imports System
3
4  Public Class MyProgram
5
6      Public Shared Sub Main()
7
8          Dim theSession As Session = Session.GetSession()
9          Dim theUISession As UI = UI.GetUI
10         Dim theWorkPart As Part = theSession.Parts.Work
11         Dim exps As ExpressionCollection = theWorkPart.Expressions
12         Dim expList As New List(Of Expression)
13
14         Dim markId1 As Session.UndoMarkId
15         markId1 = theSession.SetUndoMark(Session.MarkVisibility.Visible,
             "exportExpressions")
16
17         'Pfad zur Excel-Datei
18         Const excelName As String = "C:\Users\smartfactory\OneDrive
             \Masterarbeit\exportExpressions\Expression.xlsm"
19
20         'objectExcel erstellen
21         Dim objectExcel = CreateObject("Excel.Application")
22         If objectExcel Is Nothing Then
23             Guide.InfoWriteLine("Die Excel-Datei konnte nicht gefunden
                 werden! Der Vorgang wird abgebrochen!")
24             theSession.UndoToMark(markId1, "exportExpressions")
25             Exit Sub
26         End If
27
28         'objectExcel öffnen
29         Dim objectWorkbook = objectExcel.Workbooks.Open(excelName)
30         If objectWorkbook Is Nothing Then
31             Guide.InfoWriteLine("Die Excel-Datei " & excelName & "konnte
                 nicht geöffnet werden!")
32             theSession.UndoToMark(markId1, "exportExpressions")
33             Exit Sub
34         End If
35
36         objectExcel.visible = True
37
38         Dim j As Integer = 1
39         For Each exp1 As Expression In exps
40
41             For Each exp As Expression In exps
42                 If j = 1 And exp.Name = "DC" Then
43                     expList.Add(exp)
44                 ElseIf j = 2 And exp.Name = "DN" Then
45                     expList.Add(exp)
46                 ElseIf j = 3 And exp.Name = "APMX" Then
47                     expList.Add(exp)
48                 ElseIf j = 4 And exp.Name = "LPR" Then
49                     expList.Add(exp)
```

...p\code\exportExpressions\exportExpressions\MyProgram.vb 2

```
50                  ElseIf j = 5 And exp.Name = "OAL" Then
51                      expList.Add(exp)
52                  ElseIf j = 6 And exp.Name = "LH" Then
53                      expList.Add(exp)
54                  ElseIf j = 7 And exp.Name = "DMM" Then
55                      expList.Add(exp)
56                  ElseIf j = 8 And exp.Name = "LS" Then
57                      expList.Add(exp)
58                  ElseIf j = 9 And exp.Name = "RE" Then
59                      expList.Add(exp)
60                  ElseIf j = 10 And exp.Name = "CHW" Then
61                      expList.Add(exp)
62                  ElseIf j = 11 And exp.Name = "KCH" Then
63                      expList.Add(exp)
64                  ElseIf j = 12 And exp.Name = "AZ" Then
65                      expList.Add(exp)
66                  ElseIf j = 13 And exp.Name = "DES" Then
67                      expList.Add(exp)
68                  ElseIf j = 14 And exp.Name = "ZEFF" Then
69                      expList.Add(exp)
70                  ElseIf j = 15 And exp.Name = "ZEFP" Then
71                      expList.Add(exp)
72                  End If
73              Next
74              j = j + 1
75          Next
76
77          Dim i As Double
78
79          For i = 3 To 5
80              objectExcel.Cells(2, i) = "SFD" & exps.FindObject("DC").Value
81          Next
82
83          Dim connectioncode As String
84          connectioncode = "ZYL" & exps.FindObject("DC").Value & "xxxxxx"
85          objectExcel.Cells(2, 6) = connectioncode
86          i = 7
87
88          For Each ex In expList
89              If i = 19 Then
90                  objectExcel.Cells(2, i) = "SFD" & exps.FindObject ↵
                      ("DC").Value
91                  i = i + 1
92              End If
93              objectExcel.Cells(2, i) = ex.Value
94              i = i + 1
95          Next
96
97      End Sub
98
99  End Class
100
101
```

...ktop\code\exportExpressions\exportExpressions\Unload.vb 1

```
1  Public Class Unload
2
3      Public Shared Function GetUnloadOption(ByVal dummy As String) As Integer
4
5          Dim unloadOption As Integer
6
7          unloadOption = NXOpen.Session.LibraryUnloadOption.Immediately
             ' After executing
8          'unloadOption = NXOpen.Session.LibraryUnloadOption.AtTermination
             ' When NX session terminates
9          'unloadOption = NXOpen.Session.LibraryUnloadOption.Explicitly
             ' Using File-->Unload
10
11         Return unloadOption
12
13     End Function
14
15 End Class
16
```