# Embedded Systems

II Forschung 2

Herausgegeben von Prof. Dr.-Ing. Birgit Vogel-Heuser,
Universität Kassel

Andreas David Friedrich

# Anwendbarkeit von Methoden und Werkzeugen des konventionellen Softwareengineering zur Modellierung und Programmierung von Steuerungssystemen

kassel university press

Die vorliegende Arbeit wurde vom Fachbereich Elektrotechnik / Informatik der Universität Kassel als Dissertation zur Erlangung des akademischen Grades eines Doktors der Ingenieurwissenschaften (Dr.-Ing.) angenommen.

| | |
|---|---|
| Erster Gutachter: | Prof. Dr.-Ing. Birgit Vogel-Heuser<br>Universität Kassel |
| Zweiter Gutachter: | Prof. Dr.-Ing. Detlef Zühlke<br>Universität Kaiserslautern |

Tag der mündlichen Prüfung 18. Mai 2009

Bibliografische Information der Deutschen Nationalbibliothek
Die Deutsche Nationalbibliothek verzeichnet diese Publikation in der Deutschen Nationalbibliografie; detaillierte bibliografische Daten sind im Internet über http://dnb.d-nb.de abrufbar

Zugl.: Kassel, Univ., Diss. 2009
ISBN print: 978-3-89958-730-2
ISBN online: 978-3-89958-731-9
Die Publikation mit Anhang finden Sie im Internet unter:
URN: urn:nbn:de:0002-7319



Druck und Verarbeitung: Unidruckerei der Universität Kassel
Printed in Germany

„Wir sind Zwerge auf den Schultern von Riesen“

(frei nach Bernhard von Chartres)

# Inhaltsverzeichnis

# 1 Einleitung

## 1.1 Motivation

In den letzten Jahrzehnten haben sich industrielle Prozesse und Anlagen fortwährend weiterentwickelt. Der Umfang und die Komplexität der ablaufenden Prozesse, und damit auch der Anlagen in denen diese Prozesse ablaufen, haben sich dabei stetig gesteigert. Gleichzeitig findet auch im Bereich der Automatisierungstechnik eine Weiterentwicklung statt. Automatisierungsgeräte werden zunehmend leistungsfähiger und intelligenter. Das hat zur Folge, dass viele Funktionen, die bisher durch den Einsatz entsprechend spezialisierter Hardware realisiert wurden, in die Software von Automatisierungsgeräten verlagert werden. Auch der Einsatz neuer Technologien, wie der Feldbustechnik zur Anbindung von Sensoren und Aktoren oder Ethernet zur durchgängigen Kommunikation über die verschiedenen Ebenen der Automatisierungspyramide, haben Auswirkungen auf heutige Automatisierungssysteme. Diese entwickeln sich zu immer verteilteren und vernetzteren Gebilden von hoher Komplexität.

Die genannten Veränderungen bringen eine Verschiebung der Aufgabenschwerpunkte im Engineering entsprechender Anlagen mit sich. Bestand bis Mitte der 90'er Jahre noch ein Großteil der Arbeit darin die Hardware zu planen, so stellt heute in vielen Fällen die Entwicklung der Software den Großteil der Arbeit in der Planung dar. Auch die stetige Weiterentwicklung von Planungswerkzeugen und die zunehmend verteilte Planung tragen zu einer Veränderung im Engineeringprozess bei. Gleichzeitig führt die fortschreitende Globalisierung zu einem erhöhten Zeit- und Kostendruck und zwingt Anlagenbauer zu einer Anpassung ihrer Prozesse und Strategien, um am Markt bestehen zu können.

Um eine Optimierung des Engineerings zu erreichen, werden heutzutage regelmäßig Maßnahmen, wie die Auslagerung von Aufgaben an spezialisierte Fremd- oder Subunternehmen, genutzt. Durch zusätzliche Kapazitäten oder spezielles Know-how sollen so die Zeit bis zur Inbetriebsetzung einer Anlage reduziert und somit Kosten gesenkt werden. Das Potential solcher Maßnahmen ist jedoch begrenzt und weitestgehend ausgeschöpft. Es stellt sich also die Frage, wie die Effizienz des Engineering weiter gesteigert werden kann.

Einen Ansatzpunkt stellt die verbesserte Unterstützung der Planer durch die Verwendung neuer Methoden und Werkzeuge für das Softwareengineering dar. Im Bereich der konventionellen Softwareentwicklung haben sich objektorientierte Methoden heutzutage zum Stand der Technik entwickelt und werden erfolgreich zur Modellierung und

Programmierung komplexer Systeme eingesetzt. Als Beispiel ist die Unified Modeling Language (UML) zu nennen, die als ein standardisiertes Beschreibungsmittel als Quasi-Standard für die Entwicklung objektorientierter Software anzusehen ist.

Eine vergleichbare Entwicklung hat in der Automatisierungstechnik bisher nicht stattgefunden. Hier bilden Speicherprogrammierbare Steuerungen (SPS) den Stand der Technik, deren Programmierung häufig auf Basis der Norm IEC 61131 immer noch prozedural erfolgt. Auch Methoden zur Modellierung der Automatisierungsaufgaben werden kaum eingesetzt. Die Gründe sind vielfach in der ablehnenden Haltung der Anwender zu sehen. Zwar existiert eine Vielzahl an Beschreibungsmitteln, die für spezielle Anwendungen in der Automatisierungstechnik entwickelt wurde. Ein vergleichbares Beschreibugsmittel, das dem entspricht, was die UML für die konventionelle Softwareentwicklung darstellt, existiert in der Automatisierungstechnik jedoch nicht. Gleichzeitig werden objektorientierte Methoden und Beschreibungsmittel nicht eingesetzt, da vorwiegend die Meinung herrscht, diese seien zu komplex und für Ingenieure oder Techniker, im Gegensatz zu Softwarespezialisten, nicht anwendbar. Diese Einschätzungen sind jedoch zumeist subjektiv und wissenschaftlich nicht bewiesen.

## 1.2 Zielsetzung

Die Zielsetzung dieser Arbeit ist es, eine wissenschaftliche Grundlage für Aussagen über die Anwendbarkeit von Methoden und Werkzeugen des konventionellen Softwareengineering zur Modellierung von Systemen in der Automatisierungstechnik zu schaffen. Dabei soll bewertet werden, in wiefern die existierenden Vorbehalte bestätigt oder widerlegt werden können. Durch eine empirische Evaluation soll analysiert werden, ob die Entwicklung der Steuerungssoftware automatisierungstechnischer Systeme durch den Einsatz einer Modellierung sinnvoll unterstützt werden kann, um die Qualität der Software und die Effizienz dieses Entwicklungsprozesses zu steigern. Darüber hinaus soll ermittelt werden, wie sich die hauptsächlichen Probleme der Steuerungsprogrammierer bei der Anwendung einer Modellierung darstellen. Daraus soll abgeleitet werden, welche Maßnahmen ergriffen werden können, um den Programmierer bei seiner Aufgabe geeignet zu unterstützen und dadurch die Effizienz und Qualität im Softwareengineering automatisierungstechnischer Anlagen zu steigern.

## 1.3 Struktur der Arbeit

In Kapitel 2 erfolgt eine Eingrenzung des Untersuchungsbereiches. Neben der Klärung grundlegender Begrifflichkeiten werden die Eigenschaften des Entwicklungsprozesses von Automatisierungssystemen aufgezeigt. Dabei liegt ein Schwerpunkt auf der Softwareentwicklung, da diese im Fokus der weiteren Arbeit liegt.

Anschließend werden in Kapitel 3 die untersuchten Modellierungsnotationen detailliert vorgestellt. Dabei wird auf einzelne Diagramme und Elemente der Unified Modeling Language (UML) und der Idiomatic Control Language (ICL) eingegangen und gezeigt, dass diese Notationen generell zur Beschreibung steuerungstechnischer Aufgabenstellungen in der Automatisierungstechnik geeignet sind.

Kapitel 4 liefert mit der Darstellung mentaler Modelle in der Softwareentwicklung und weiteren Einflussfaktoren auf die Leistungsfähigkeit von Programmierern die Grundlagen für die empirische Evaluation. Zudem werden ausgewählte Ergebnisse aus relevanten themenverwandten Untersuchungen zusammenfassend dargestellt. Den Abschluss dieses Abschnitts bildet die Herleitung von Fragestellungen, die im Rahmen der experimentellen Evaluation untersucht werden.

Der Entwurf der Aufgabenstellungen sowie die Umsetzung in ein entsprechendes Experiment, das mit 89 Probanden unterschiedlicher Qualifikation durchgeführt wurde, sind Inhalt von Kapitel 5. Neben der Beschreibung des fertigungstechnischen Prozessmodells, auf dessen Basis die Evaluation durchgeführt wurde, wird detailliert auf das Versuchsdesign eingegangen. Dabei werden sowohl die Aufgabenstellung und der Ablauf des Experiments als auch die Qualifikation in den unterschiedlichen Probandengruppen, die für die Durchführung zu Verfügung standen, beschrieben.

In Kapitel 6 werden die Ergebnisse der Experimente ausführlich dargestellt. In einem Unterkapitel wird dabei jeweils ein Aspekt (z.B. Einfluss der Qualifikation auf die Programmierleistung) behandelt und nach Darstellung der Ergebnisse mit einer Diskussion des Themenpunktes abgeschlossen. Das Kapitel fasst zudem weitere, im Vorfeld nicht als These formulierte Erkenntnisse zusammen und beschreibt die Korrelation von Ergebnissen verschiedener Thesen.

Aus den Ergebnissen der Experimente ergeben sich Schlussfolgerungen für das Engineering automatisierungstechnischer Systeme, die in Kapitel 7 diskutiert werden. Darüber hinaus wird die Bedeutung der Evaluation für die Durchführung ähnlicher Evaluationsexperimente in der Automatisierungstechnik aufgezeigt und es werden Vorschläge für weiterführende Untersuchungen auf diesem Gebiet gemacht.

Kapitel 8 schließt die Arbeit mit einer Zusammenfassung der Ergebnisse ab und gibt einen Ausblick auf mögliche weitere Entwicklungen.

# 2 Eingrenzung des Untersuchungsbereiches

In den letzten Jahrzehnten haben sich industrielle Prozesse und Anlagen fortwährend weiterentwickelt. Der Umfang und die Komplexität der ablaufenden Prozesse und damit auch der Anlagen in denen diese Prozesse ablaufen haben sich dabei fortwährend gesteigert. Die Automatisierungstechnik unterstützt diese Entwicklung mit der Entwicklung und Bereitstellung von Konzepten und Technologien, die es ermöglichen Prozesse und Anlagen automatisiert, d.h. *„ganz oder teilweise ohne Mitwirkung des Menschen bestimmungsgemäß“* [DINV19233] ablaufen zu lassen.

Der Entwicklungs- und Entwurfsprozess automatisierungstechnischer Systeme, häufig als Engineering bezeichnet, beschreibt alle Vorgänge von der Idee bis hin zur fertigen Realisierung des Systems. Der Engineeringprozess besteht aus einer Menge von Einzelschritten, deren Existenz und Abfolge im jeweiligen Projekt von einer Vielzahl von Einflussfaktoren abhängig ist.

Durch die ständige Weiterentwicklung steigt besonders die Leistungsfähigkeit von Softwaresystemen. Dadurch findet auch im Bereich der Automatisierungstechnik eine zunehmende Verlagerung von Hardware-Funktionalitäten in die Software statt. Dies hat auch Auswirkungen auf den Entwicklungsprozess entsprechender Systeme. Der Anteil der Softwareentwicklung am gesamten Engineeringaufwand steigt und auch die Aufgaben im Softwareentwicklungsprozess werden vielfältiger. Auf Basis von festgelegten Anforderungen an das zu realisierende System, die beispielsweise das Verhalten oder Sicherheitsaspekte betreffen, sowie Randbedingungen, die z.B. durch die verwendete Automatisierungshardware oder das eingesetzte Zielsystem festgelegt werden, muss in diesem Schritt eine entsprechende Software entwickelt werden.

In der konventionellen Softwareentwicklung hat sich die Modellierung des zu realisierenden Systems für den Entwurf und die Dokumentation der Software bereits fest etabliert. Beschreibungsmittel und dazugehörige Entwicklungswerkzeuge, die auf die entsprechende Zielumgebung bzw. die verwendete Programmiersprache angepasst sind, finden dort eine breite Anwendung. Als Beispiel kann die Unified Modelling Language genannt werden, die häufig für die Entwicklung von objektorientierter Software eingesetzt wird.

Im Bereich der Steuerungsprogrammierung werden entsprechende Beschreibungsmittel und Methoden bisher nur sehr selten eingesetzt. Das liegt einerseits an der Tatsache, dass kein Beschreibungsmittel existiert, das dem entspricht, was die UML für objektorientierte

Software darstellt und explizit für den Entwurf und die Dokumentation bei der Programmierung von Steuerungssystemen vorgesehen ist. Die Idiomatic Control Language (ICL) verfolgt zwar einen derartigen Ansatz, wurde jedoch hauptsächlich für die Anwendung in der Prozessindustrie entwickelt. Zudem existieren für die ICL derzeit nur ein Sprachentwurf und prototypische, nicht voll funktionsfähige Entwicklungswerkzeuge.

Andererseits finden auch vorhandene, für den konventionellen Softwareentwurf etablierte Beschreibungsmittel, in der Automatisierungstechnik nur selten Anwendung. Dies liegt an der weit verbreiteten Meinung, dass vorhandene Modellierungsnotationen für die Automatisierungstechnik als nicht anwendbar bewertet werden. Begründet wird dies damit, dass die Diskrepanz zwischen einer objektorientierten Modellierung auf der Seite der UML und der prozeduralen Programmierung auf der Seite der Speicherprogrammierbaren Steuerung zu groß ist und sich entsprechende Steuerungssysteme mit der UML nicht in geeigneter Weise darstellen lassen. Darüber hinaus besteht häufig die subjektive Einschätzung, dass das Abstraktionsvermögen von Steuerungsprogrammierern für die Modellierungstätigkeit zu gering ist. Es wird erwartet, dass die Kompetenz zur Anwendung abstrakter Softwareentwicklung nur durch erhöhten Schulungsaufwand oder im Extremfall überhaupt nicht erlangt werden kann und dieser erhöhte Zeitaufwand durch eine mögliche Steigerung der Softwarequalität und die Reduktion der Programmierzeit nur unzureichend wieder kompensiert wird.

Die Gründe, die häufig gegen einen Einsatz einer Modellierungsnotation im Rahmen der Steuerungsprogrammierung vorgebracht werden, sind also vielfältig [VFK+05]. Sie basieren jedoch meist auf einer subjektiven Bewertung und sind wissenschaftlich nicht bewiesen. Die Zielsetzung der vorliegenden Arbeit ist es daher zu evaluieren, ob die Entwicklung der Steuerungssoftware automatisierungstechnischer Systeme durch den Einsatz einer Modellierung sinnvoll unterstützt werden kann, um die Qualität der Software und die Effizienz dieses Entwicklungsprozesses zu steigern. Dabei soll auch untersucht werden, in wiefern die oben genannten Vorbehalte begründet sind. Welche dieser Aussagen bestätigen sich oder können widerlegt werden? Darüber hinaus soll ermittelt werden, wie sich die hauptsächlichen Probleme der Steuerungsprogrammierer bei der Anwendung einer Modellierung darstellen und welche Maßnahmen ergriffen werden können, um den Programmierer bei seiner Aufgabe geeignet zu unterstützen.

Zu diesem Zweck wird im vorliegenden Kapitel zu Beginn eine Definition der zentralen Begriffe des Untersuchungsbereiches gegeben, um eine einheitliche Terminologie als Grundlage für die weiteren Betrachtungen zu schaffen. Anschließend werden typische Bestandteile und Abläufe des Engineeringprozesses automatisierungstechnischer Systeme

beschrieben. Dabei ist ein besonderes Augenmerk auf bestehende Defizite zu legen, um somit Ansatzpunkte für eine Steigerung der Qualität und Effizienz durch den Einsatz einer Modellierung zu identifizieren. Abschließend wird detailliert auf die Programmierung Speicherprogrammierbarer Steuerungssysteme eingegangen und beschrieben, welche Unterscheide zu konventionellen Softwaresystemen bei der Modellierung berücksichtigt werden müssen.

## 2.1 Automatisierungstechnik

### 2.1.1 Begriffe und Definitionen

Automatisierungstechnik beschreibt ein fachübergreifendes Querschnittsgebiet, welches sich mit der Automatisierung technischer Systeme befasst. Ein System ist nach DIN 19226 [DIN19226] wie folgt definiert:

> *„Ein System ist eine in einem betrachteten Zusammenhang gegebene Anordnung von Gebilden, die miteinander in Beziehung stehen. Diese Anordnung wird aufgrund bestimmter Vorgaben gegenüber ihrer Umwelt abgegrenzt."*

Als Prozess werden die Vorgänge bezeichnet, die in einem solchen System ablaufen. Die DIN V 19233 definiert den Prozess als *„Gesamtheit von aufeinander einwirkenden Vorgängen in einem System, durch die Materie, Energie oder auch Information umgeformt, transportiert oder auch gespeichert wird."* [DINV19233].

Der Prozessbegriff wird weiter detailliert, indem zwischen dem technischen Prozess und dem Rechenprozess unterschieden wird. Ein technischer Prozess beschreibt dabei die *„…Gesamtheit der Vorgänge in einer technischen Anlage zur Bewältigung einer bestimmten technischen Aufgabenstellung"* [DINV19233] während der Rechenprozess als *„…Gesamtheit der Vorgänge in einem Rechensystem, die an der Ausführung eines Programms oder eines sinnvoll abgegrenzten Programmteils beteiligt sind und von einer Instanz gesteuert werden"* [DINV19233] beschrieben wird. Dieselbe Trennung findet sich auch in der Beschreibung dieser beiden Prozessarten wieder.

Technische Prozesse laufen in technischen Systemen [LaGö99a] oder technischen Anlagen [DINV19233] ab, beide Definitionen sind durchaus als gleichwertig anzusehen. Technische Systeme oder Anlagen bezeichnen die Gesamtheit der technischen Einrichtungen, die zur Realisierung der technischen Aufgabenstellung benötigt werden. Dabei handelt es sich um Apparate, Maschinen, Transporteinrichtungen oder andere Betriebsmittel. Abb. 2.1

verdeutlicht die Zusammenhänge zwischen einem technischen System und dem darin ablaufenden technischen Prozess.

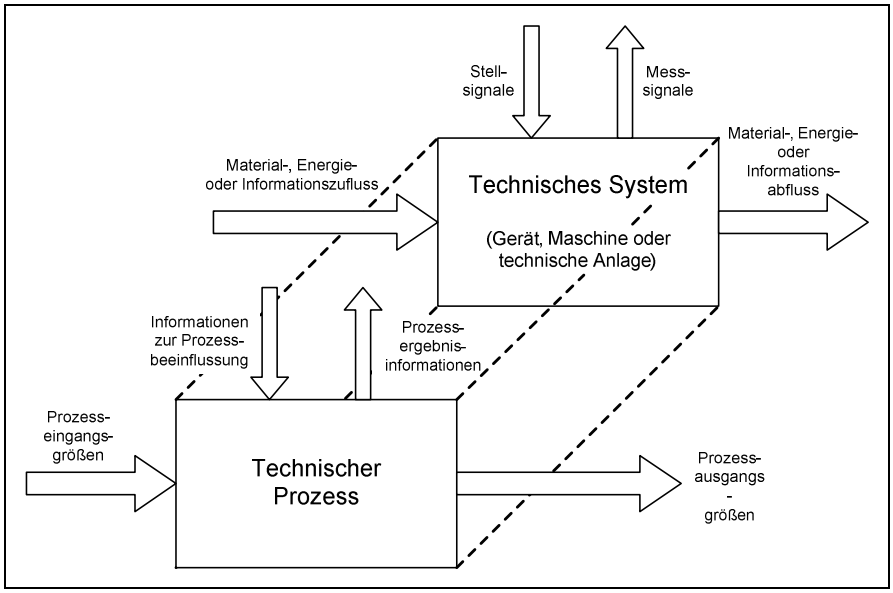



*Abb. 2.1 - Unterscheidung zwischen technischem Prozess und technischen System (nach [LaGö99a])*

Technische Prozesse können aufgrund ihrer Eigenschaften in Prozessklassen zusammengefasst werden. Meist werden Prozesse nach der Art des Materialflusses im Prozess unterteilt, wobei zwischen kontinuierlichen Prozessen, diskreten Prozessen, Chargenprozessen sowie Mischformen daraus, so genannten hybriden Prozessen, unterschieden wird. Auch die Art des Mediums, die Art der Umwandlung oder der Automatisierungsgrad können als Basis der Klassifikation dienen. Abb. 2.2 zeigt eine Übersicht von Klassifikationsmerkmalen nach Langer [Lan02].

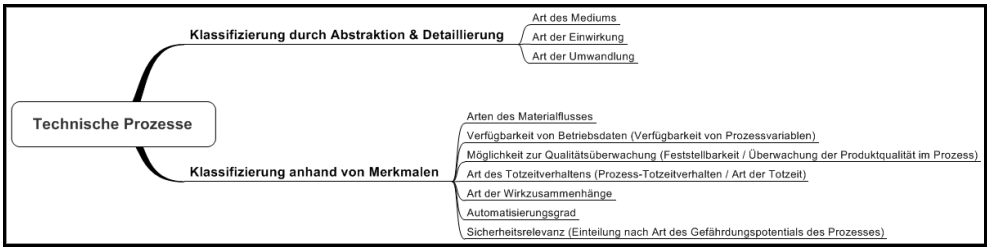



*Abb. 2.2 - Klassifizierungsmöglichkeiten technischer Prozesse nach Langer [Lan02]*

Im Vergleich zum technischen Prozess läuft ein Rechenprozess auf einem Rechensystem ab und wird als Gesamtheit der Vorgänge bezeichnet, die *„… die an der Ausführung eines Programms oder eines sinnvoll abgegrenzten Programmteils beteiligt sind und von einer Instanz gesteuert werden“*. Rechenprozesse sind für die Automatisierungstechnik von sehr hoher Bedeutung. Bereits bei der Aufnahme von Messwerten aus dem Prozess kann es notwendig sein, Signale z.B. durch Mittelwertbildung zu glätten oder sie zu skalieren. Auch die Weiterverarbeitung der Signale oder die Errechnung von Stellgrößen erfolgt wie Erzeugung eines Ausgangssignals meist mit Hilfe von Rechenprozessen. Rechenprozesse können auf verschiedensten Rechensystemen wie PCs oder Industrie-PCs, Speicherprogrammierbaren Steuerungen (SPS) oder Mircrocontrollern ablaufen und in unterschiedlichsten Programmiersprachen realisiert werden.

Aufgrund der Schwierigkeit, die komplexen Zusammenhänge in Systemen vollständig zu beschreiben, werden in der Automatisierungstechnik häufig Modelle eines realen Systems gebildet. Der Modellbegriff wird in der Fachliteratur vielfach definiert und diskutiert ([Bal01], [LaGö99b], [Pol94]), unterscheidet sich jedoch nur unwesentlich. Stellvertretend sei hier der Modellbegriff nach Ahrens und Polke [AhPo89] genannt:

> *„Modelle sind vereinfachende Darstellungen (Bilder) von Realitäten unter einer bestimmten Sicht. Im Allgemeinen werden abstrakte Modelle verwendet. Hier entstehen die Modelle durch einen Abstraktionsvorgang, unter Verwendung eines Modellierungskonzeptes, d.h. einer Idee über die Art der Modellbildung.“*

Ein Modell stellt also ein abstraktes Abbild des realen Systems unter der Berücksichtigung festgelegter Einschränkungen und Randbedingungen dar. Durch geeignete Wahl der Einschränkungen und Randbedingungen können die für die Problemlösung irrelevanten Teile des Systems bei der Modellbildung ausgeblendet und dadurch vernachlässigt werden. Dadurch wird die Komplexität des zu betrachtenden Problembereichs für den Anwender reduziert.

Modelle können Systeme aus verschiedenen Sichten beschreiben, um somit für verschiedene Nutzer oder unterschiedliche Anwendungsfälle die entsprechenden relevanten Eigenschaften darzustellen. In Abhängigkeit der zu beschreibenden Aspekte sind unterschiedliche Beschreibungsmittel notwendig [Sch03]. So beschränkt sich die Beschreibung eines Systems aus Sicht eines Mechanikers zum Beispiel auf mechanische Aspekte wie Abmessungen, Material oder Kräfteberechnungen, während für einen Automatisierungstechniker Eigenschaften wie Anzahl und Eigenschaften von Sensoren oder Aktoren sowie das Verhalten von Interesse sind. Schnittbereiche sind dabei durchaus

denkbar, denn sowohl der Mechaniker als auch der Automatisierungstechniker benötigt die Information, an welcher Stelle sich ein Sensor oder Aktor reell befindet.

Der Kontext stellt also einen wichtigen Aspekt dar, um ein Modell richtig interpretieren zu können. Die Grundlagen für eine Beschreibung werden jedoch in der Entwicklungsphase geschaffen. Hier wird festgelegt, welche Mittel man für die Entwicklung einsetzt. Im Systemsengineering beschreibt Schnieder [Sch93] diese Mittel mit dem BMW-Prinzip und unterteilt diese in Beschreibungsmittel, Methode und Werkzeuge. Ein Beschreibungsmittel wird verwendet, um einen Sachverhalt in einer definierten Art und Weise zu formulieren. Häufig findet auch der Begriff der Notation Verwendung, beide Begriffe sind bedeutungsgleich anzusehen. Eine Methode beschreibt ein geplantes Verfahren, um ein Ergebnis zu erreichen und ist mit dem Begriff der Vorgehensweise gleichzusetzen. Als Werkzeug werden Hilfsmittel bezeichnet, die für eine Lösung des Problems, hier also die Beschreibung des Modells, benötigt werden. Werkzeuge werden auch als Realisierungsmittel bezeichnet.

### 2.1.2 Aufgaben in der Automatisierungstechnik

Automatisierungstechnik bezeichnet die Disziplin, die sich mit der Überwachung und Steuerung technischer Systeme befasst [Lun03]. Ziel ist es, das zu automatisierende technische System durch den Einsatz geeigneter Automatisierungseinrichtungen, welche gemeinsam das Automatisierungssystem bilden, so zu steuern, dass sie eine gewünschte Funktionalität bieten und gleichzeitig den geltenden Sicherheitsanforderungen entsprechen.

Mögliche Bestandteile eines Automatisierungssystems sind in Abb. 2.3 zusammenfassend dargestellt. Nicht jedes Automatisierungssystem muss alle Komponenten enthalten, je nach Art des technischen Systems und den an die Realisierung gestellten Anforderungen können einzelne Komponenten entfallen.

Der Automatisierungsrechner nimmt im Automatisierungssystem eine zentrale Rolle ein. Er bildet die Hardwareplattform, auf welcher die Automatisierungssoftware ausgeführt wird. Automatisierungsrechner können als Mikrocontroller, Industrie PC (IPC), Speicherprogrammierbare Steuerung (SPS) oder Prozessleitsystem (PLS) ausgeführt sein. Während in der Fertigungstechnik (z.B. bei Werkzeugmaschinen) häufig Speicherprogrammierbare Steuerungen (SPS) zum Einsatz kommen, werden in der Produktautomatisierung (z.B. bei Haushaltsgeräten) Mikrocontroller eingesetzt. Prozessleitsysteme finden hingegen meist in der Verfahrenstechnik und der Energietechnik Einsatz.

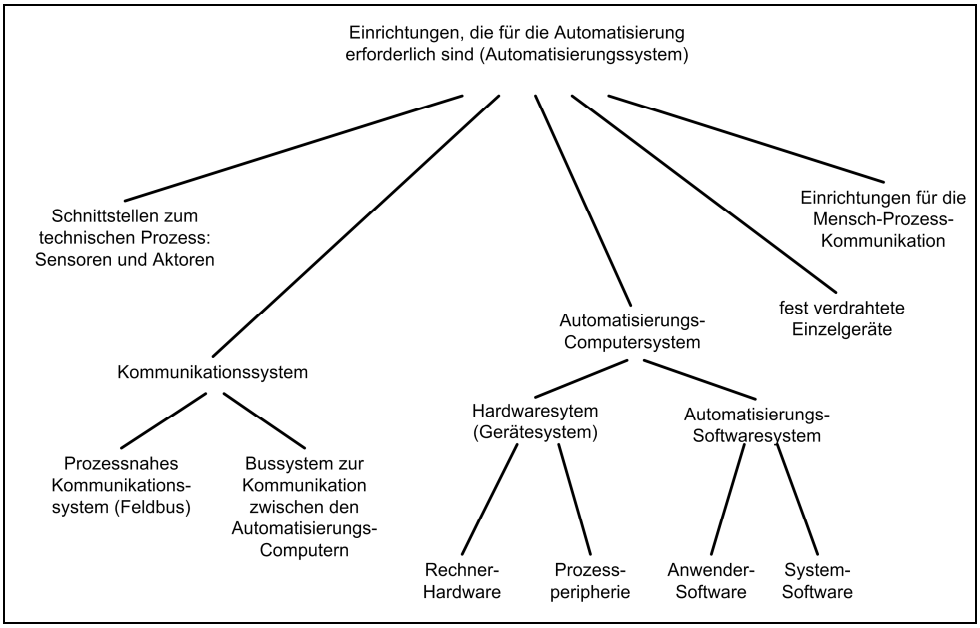



*Abb. 2.3 - Bestandteile eines Automatisierungssystems (in Anlehnung an [LaGö99a])*

Sensoren und Aktoren bilden die Schnittstelle zwischen dem technischen System und dem Automatisierungssystem. Sensoren (z.B. Temperaturfühler, Durchflussmesser oder Druckmesser) erfassen die benötigten physikalischen Prozessgrößen und wandeln diese in elektrische Signale um. Die Sensoren sind entweder direkt mit einem Automatisierungsrechner verbunden oder ihre Signale werden über ein Feldbussystem mit dem Automatisierungsrechner ausgetauscht. Eine auf dem Automatisierungsrechner ausgeführte Automatisierungssoftware errechnet auf Basis der eingehenden Prozessgrößen und gespeicherten Werten die Stellgrößen für die Beeinflussung des Prozesses. Die entsprechenden Signale gelangen wiederum direkt oder über Feldbussysteme zu den Aktoren (z.B. Motoren oder Ventile), welche so die eigentlichen physikalischen Größen im Prozess beeinflussen.

Die Anbindung der Sensoren und Aktoren geschieht heute bis hinunter in die Feldebene meist mit Hilfe von Feldbussystemen. Es handelt sich dabei um industrielle Kommunikationssysteme, die eine Vielzahl von Feldgeräten wie Messfühler (Sensoren), Stellglieder und Antriebe (Aktoren) mit einem Steuerungsgerät verbinden. Die Feldbustechnik wurde in den 80er Jahren entwickelt, um die bis dahin übliche Parallelverdrahtung binärer Signale sowie die analoge Signalübertragung durch digitale Übertragungstechnik zu ersetzen. Heute sind unterschiedliche Feldbussysteme am Markt etabliert, die unterschiedliche Eigenschaften aufweisen und für unterschiedliche

Einsatzzwecke geeignet sind. Beispiele für Feldbussysteme sind Profibus, Fieldbus Foundation, INTERBUS oder CAN-Bus. Seit 1999 werden Feldbusse in den Normenreihen IEC 61158 sowie IEC 61784 weltweit standardisiert.

Die Automatisierungstechnik verfolgt neben der Erfüllung der eigentlichen Funktionalität häufig auch übergeordnete Ziele an den Betrieb eines Prozesses oder einer Anlage [LaGö99a], welche bei der Realisierung der Automatisierungsfunktionen besondere Berücksichtigung finden. Unter anderem sind zu nennen:

- Zuverlässigkeit und Sicherheit (z.B. Vermeidung von Produktionsausfällen oder Gefahren)
- Gleichmäßigkeit (z.B. im Bezug auf die Qualität des Endproduktes)
- Wirtschaftlichkeit
- Ökologische Vereinbarkeit.

Das breite Aufgabenspektrum macht deutlich, dass diese Ziele eine Integration verschiedener Fachdisziplinen erfordern. Neben der reinen Elektrotechnik und ihren Teilaufgaben wie Messtechnik, Antriebstechnik, Steuer- und Regelungstechnik oder Prozessvisualisierung, findet man häufig interdisziplinäre Schnittstellen zur Verfahrenstechnik, Sicherheitstechnik, Informationstechnik oder auch Informatik [Fri02]. Diese Interdisziplinarität bringt jedoch auch viele Herausforderungen hinsichtlich der Kommunikation und Abstimmung mit sich. So sind in den unterschiedlichen Disziplinen beispielsweise häufig abweichende Fachtermini definiert, was zu falscher Interpretation von textlichen Dokumentationen oder zu Missverständnissen in Diskussionen führen kann. Es ist deshalb notwendig eine gemeinsame Basis zu schaffen, um eine fehlerfreie und möglichst effiziente interdisziplinäre Planung automatisierungstechnischer Systeme zu ermöglichen.

### 2.1.3 Speicherprogrammierbare Steuerungen

Heutzutage werden zur Realisierung von Automatisierungsaufgaben im industriellen Umfeld häufig Speicherprogrammierbare Steuerungen (SPS) eingesetzt [ErMa04]. Die DIN EN 61131-3 definiert eine Speicherprogrammierbare Steuerung wie folgt:

> *„… ein digital arbeitendes elektronisches System, entworfen für den Einsatz in industrieller Umgebung. Es verwendet einen programmierbaren Speicher zur internen Speicherung der anwenderorientierten Anweisungen, um Funktionen wie Logik, Ablauf, Zeit, Zählen und Arithmetik auszuführen und*

*durch digitale oder analoge Eingänge und Ausgänge verschiedene Typen von Maschinen oder Prozessen zu steuern. Die SPS und auch ihre zugehörigen Peripheriegeräte sind so entworfen, dass sie leicht in ein industrielles System integriert und für alle bestimmungsgemäßen Funktionen verwendet werden können." [DIN61131-1].*

Aufgrund der Stärken in der binären Verknüpfungs- und Ablaufsteuerung werden SPSen besonders häufig in der Fertigungstechnik eingesetzt, aber auch in der Verfahrenstechnik werden SPSen mittlerweile verstärkt eingesetzt. Durch ihre robuste Bauweise in Bezug auf mechanische Beanspruchung, Umgebungstemperatur sowie EMV, sind sie für die dort auftretenden extremen Bedingungen gut geeignet. SPSen werden dabei in verschiedenen Ausführungen von der Kompakt-SPS für kleine Einzelanwendungen bis hin zu modularen, frei konfigurierbaren SPSen für dezentrale verteilte Anwendungen angeboten. Auch Softwarerealisierungen die z.B. auf einem Industrie-PC ablaufen können, so genannte Soft-SPSen, haben in den letzten Jahren deutlich an Bedeutung gewonnen.

*Tab. 2.1 - Übersicht IEC 61131*

| Teil | Titel | Inhalt |
|---|---|---|
| 1 | Allgemeine Informationen | - technisches Umfeld der Norm<br>- Definition von Begrifflichkeiten<br>- Beziehungen zu anderen relevanten Standards |
| 2 | Ausrüstung und Testanforderungen | - Hardwaremodell einer SPS<br>- mechanische und elektrische Anforderungen<br>- notwendige Testmethodiken |
| 3 | Programmiersprachen | - Softwaremodell einer SPS<br>- Textuelle und grafische Programmiersprachen |
| 4 | Anwenderrichtlinien | - Diskussion von Fragen zu: Systemauswahl, Prüfung, Inbetriebnahme und Wartung von Speicherprogrammierbaren Steuerungen |
| 5 | Kommunikation | - Beschreibung des Kommunikationsmodells<br>- Funktionalitäten zur Kommunikation zwischen Speicherprogrammierbaren Steuerungen |
| 7 | Fuzzy Control Programmierung | - Diskussion der Fuzzy-Regelung auf Speicherprogrammierbaren Steuerungen<br>- Definition einer Sprache zur Realisierung |
| 8 | Richtlinien zur Implementierung von Programmiersprachen | - Definition von Richtlinien für die Implementierung und Anwendung der Programmiersprachen aus Teil 3 |

Um die Kompatibilität und Interoperabilität von Speicherprogrammierbaren Steuerungen zu gewährleisten wurde von der International Electrotechnical Commission (IEC) der

Standard IEC 61131 „Programmable Controllers“ erarbeitet. Dieser Standard legt neben einem einheitlichen Hard- und Softwaremodell auch die Sprachen fest, welche zur Programmierung von SPSen verwendet werden sollen. Die deutsche Entsprechung bildet die DIN EN 61131 „Speicherprogrammierbare Steuerungen“ (Tab. 2.1).

Die Anforderungen an die Hardware einer Speicherprogrammierbaren Steuerung wird in der DIN EN 61131-2 in Form eines Hardwaremodells definiert [DIN61131-2]. Kern ist eine Signalverarbeitungs-Funktion in welcher ein Anwenderprogramm abgearbeitet wird und so Eingangssignale und interne Speicher zu Ausgangssignalen oder neuen internen Speichern verknüpft werden(Abb. 2.4). Ein- und Ausgangssignale werden dabei mit Hilfe der Interface-Funktion aus dem Prozess gelesen oder in den Prozess geschrieben. Für den Datenaustausch von SPSen untereinander oder zu anderen externen Geräten existieren Kommunikationsfunktionen, der Anschluss der SPS an das Stromnetz erfolgt mit Hilfe der Stromversorgungsfunktion. Aufbauend auf diesem Hardwaremodell besteht die Realisierung einer Speicherprogrammierbaren Steuerung im Wesentlichen aus einer Stromversorgung, einem Prozessor mit zugehörigem Speicher, den Ein-/Ausgabebaugruppen sowie optionalen Prozessoren für spezielle Aufgabenstellungen (z.B. Wegerfassung oder Kommunikation).

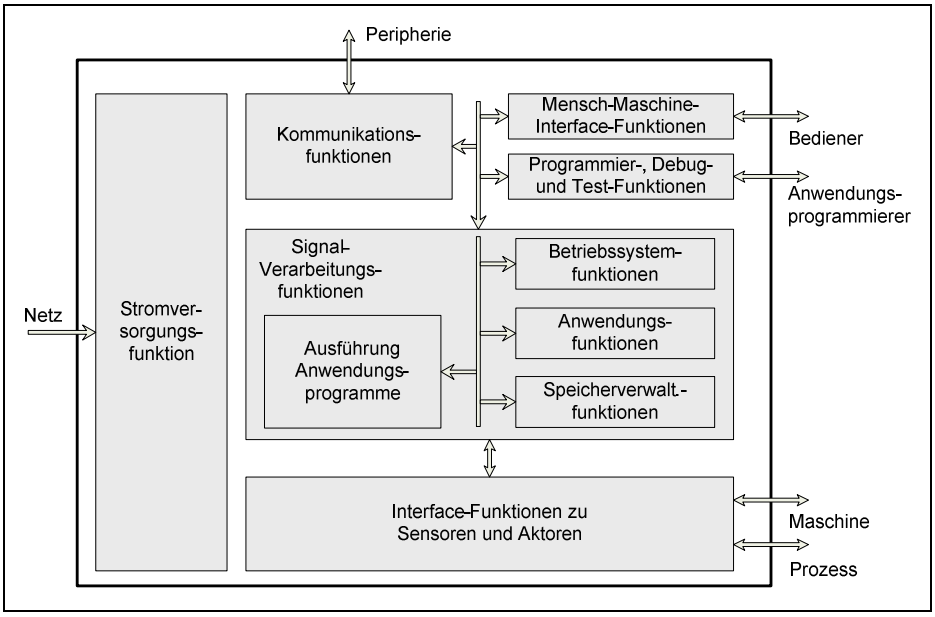



*Abb. 2.4 - Hardwaremodell der IEC 61131 [DIN61131-2]*

## 2.2 Entwicklung von Automatisierungssystemen

Das Ziel von Automatisierungssystemen ist es, einen technischen Prozess weitestgehend ohne den Einfluss eines menschlichen Bedieners selbstständig ablaufen zu lassen. Dies stellt eine Reihe von Anforderungen an das Automatisierungssystem, die von den Eigenschaften des technischen Prozesses, des technischen Systems, Sicherheits- und Umweltfaktoren oder auch gesetzlichen Bestimmungen abhängen.

Der Entwurf und die Realisierung von Automatisierungssystemen erfordern folglich einen hohen Entwicklungsaufwand von Ingenieuren. Lauber/Göhner definieren in diesem Zusammenhang den Begriff des Automatisierungsprojektes als „*ein Vorhaben zur Lösung einer Automatisierungsaufgabe durch Ingenieure*“ [LaGö99b]. In Anlehnung an DIN 69901 [DIN69901] wird ein Projekt durch Ziele und Aufgabenstellungen, zeitliche Begrenzungen, finanzielle und personelle Begrenzungen, seine Infrastruktur, Richtlinien, Standards und gesetzliche Vorschriften sowie Erfahrungen aus früheren Projekten gekennzeichnet. Abb. 2.5 zeigt die Einflüsse auf ein Automatisierungsprojekt.

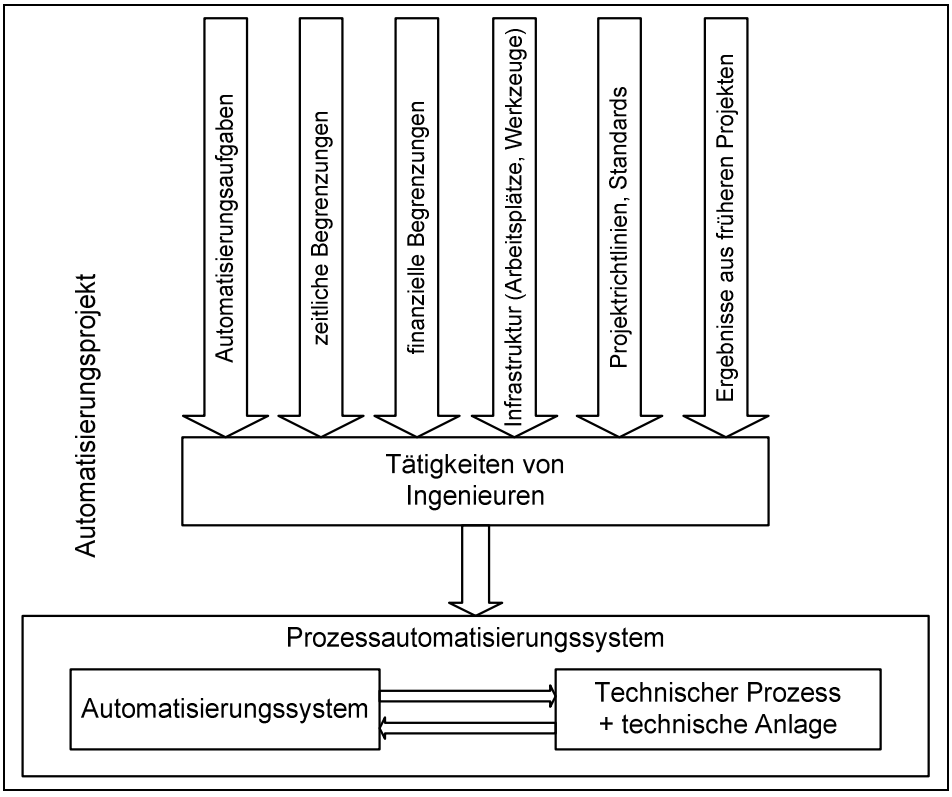



*Abb. 2.5 - Kennzeichen eines Automatisierungsprojekts [LaGö99b]*

Der Entwicklungsprozess von Automatisierungssystemen wird dabei nach Lauber/Göhner in zwei Arten unterschieden [LaGö99b]:

a) Produkt-Entwicklungsprojekt:
   Bei der Entwicklung von Produkten läuft der technische Prozess auf einem Gerät oder einer Maschine ab. Die Automatisierungsaufgaben sind meist relativ klein. Als Beispiel kann ein Kaffee-Automat genannt werden. Der Umfang eines solchen Projektes hängt nicht nur von der Größe der eigentlichen Automatisierungsaufgabe sondern auch stark von der zu produzierenden Stückzahl ab.

b) Anlagen-Entwicklungsprojekt:
   Bei Anlagen-Entwicklungsprojekten läuft der technische Prozess meist in großen technischen Anlagen ab. Beispiele dafür sind Anlagen der chemischen Industrie oder des Maschinen- und Anlagenbaus. Aus diesem Grund wird die automatisierungstechnische Lösung meist speziell an die Anforderungen des Prozesses angepasst. Ausschlaggebend für den Umfang und die Laufzeit solcher Projekte sind die Eigenschaften der technischen Anlage.

Beide Arten von Entwicklungsprojekten unterscheiden sich im Hinblick auf den Umfang, die Aufteilung in einzelne Phasen sowie die Aufgaben und den zeitlichen Ablauf des Projektes voneinander. Auch im Bezug auf die beteiligten Disziplinen werden Unterschiede deutlich. So existiert z.B. bei Anlagen-Entwicklungsprojekten die Schnittstelle zum Bauingenieurwesen, welche bei Produkt-Entwicklungsprojekten kaum eine Rolle spielt. Weiterhin kommen hinsichtlich der eingesetzten Automatisierungshardware aufgrund der jeweiligen Anforderungen unterschiedliche Zielsysteme zum Einsatz. Während bei Produkten häufig kleine zentrale Steuerungseinheiten (z.B. Mikrokontroller) verwendet werden, werden Anlagen heutzutage meist dezentral gesteuert, wobei häufig Speicherprogrammierbare Steuerungen oder Industrie-PCs zum Einsatz kommen. Die vorliegende Arbeit beschränkt sich im W6eitern auf die Betrachtung von Anlagen-Entwicklungsprojekten.

### 2.2.1 Aufgaben und Werkzeuge im Engineering Lifecycle

Der Engineering Lifecycle beschreibt automatisierungstechnische Systeme über alle Phasen von der ersten Idee bis zu Stilllegung und Recycling [Fri02]. Eine grobe Einteilung kann in folgende Phasen vorgenommen:

- Vorplanung
- Detailplanung / Basic-Engineering
- Ausführungsplanung / Detail-Engineering
- Installation und Inbetriebnahme
- Betrieb, Wartung und Modifikation
- Stilllegung und Recycling.

Die einzelnen Phasen können dabei weiter detailliert werden.

In den verschiedenen Phasen arbeitet eine Vielzahl von Ingenieuren und Technikern verschiedenster Fachdisziplinen daran, die Konfiguration des Systems an die Anforderungen des Prozesses anzupassen. Wie Untersuchungen des Workflow beim Engineering der Automatisierungstechnik im Maschinen- und Anlagenbau exemplarisch gezeigt haben, weist der Workflow dabei keinesfalls geordnete, rein sequentielle Strukturen auf, sondern ist vielmehr durch parallele und iterative Arbeitsabläufe sowie Rücksprünge in vorhergehende Planungsphasen gekennzeichnet [FrVo02]. Dabei existieren gegenseitige Abhängigkeiten zwischen einzelnen Phasen aber auch zwischen den verschiedenen Disziplinen, die bei der Planung berücksichtigt werden müssen. Zudem werden die Systeme, auch aufgrund zunehmender Globalisierung, zunehmend verteilt geplant und produziert. Häufig werden Großanlagen in Zusammenarbeit mehrerer Anlagenbauer und Subunternehmer geplant und realisiert, um spezielles Know-how einzubinden, Termine halten zu können oder Anforderungen an die Leistungserbringung im Zielland zu erfüllen [Fay05].

Das Engineering automatisierungstechnischer Systeme wird durch eine Vielzahl von Softwaretools, sogenannten ECAD-Werkzeugen, unterstützt. Diese sind häufig auf einen bestimmten Aspekt der Planung spezialisiert und unterstützen die entsprechenden Tätigkeiten höchst optimiert. Als Beispiele sind Werkzeuge für die Planung von R&I-Fließbildern, die Planung der mechanischen (z.B. CAD-Zeichnungen) oder der gerätetechnischen Ausführung (z.B. Schaltplanunterlagen), die Planung der Software oder der Prozessvisualisierung zu nennen [Fri02]. Unterschiedliche Unternehmen aber auch Abteilungen eines Unternehmens nutzen dabei unterschiedliche Engineeringwerkzeuge unterschiedlicher Hersteller aber auch Eigenentwicklungen. Die Fähigkeit zur Interoperabilität der verschiedenen Systeme ist bisher jedoch unzureichend realisiert. Zwar besitzen die meisten Tools Schnittstellen, mit denen sie Daten zu vor- oder nachgelagerten Systemen austauschen können. Die Schnittstellenformate sind jedoch meist proprietär und

der Datenaustausch läuft nur unidirektional. In den letzten Jahren gibt es deshalb verstärkt Bestrebungen, standardisierte Schnittstellen und Schnittstellenformate zu definieren. Ein Beispiel stellt das auf XML basierende Format CAEX dar, mit dessen Hilfe hierarchische Anlagen- und Systemdaten beschrieben und ausgetauscht werden können [DrFe04a], [DrFe04b].

Eine der Schwierigkeiten in diesem verteilten, hoch komplexen Engineeringprozess liegt darin, die einzelnen Teiltätigkeiten und die heterogenen Werkzeuge aufeinander abzustimmen, um so die Qualität der Projektierungsergebnisse und die Kosten ihrer Umsetzung sicherstellen und kontrollieren zu können.

### 2.2.2 Vorgehensmodelle

Vorgehensmodelle gliedern Prozesse in verschiedene, strukturierte Phasen, denen entsprechende Methoden und Techniken der Organisation zugeordnet sind [LaGö99b]. Dadurch ist es möglich, die notwendigen Aufgabenstellungen und Aktivitäten festzulegen und in ihrer logischen Ordnung darzustellen. Die Anwendung standardisierter Vorgehensmodelle stellt somit eine Möglichkeit dar, den Anforderungen, wie sie in Kapitel 2.2.1 beschrieben wurden, zu begegnen und kann dazu beitragen, Qualität und Effizienz im Engineering zu steigern.

Für den Systementwurf existieren zahlreiche Vorgehensmodelle, die in allgemeingültige und anwendungsspezifische Modelle unterteilt werden können [Vog03]. Allgemeingültige Modelle sind in ihrer Anwendung nicht eingeschränkt und können für die Entwicklung von Systemen jeglicher Art, für Software und sicherheitstechnische Systeme angewendet werden. Als Beispiele sind das Wasserfallmodell, das Spiralmodell oder das V-Modell zu nennen. Anwendungsspezifische Modelle sind auf die Anforderungen aus dem entsprechenden Anwendungsbereich angepasst und teilweise sehr spezialisiert oder auch firmenspezifisch. Dadurch ist eine organisationsübergreifende Abstimmung von Abläufen deutlich erschwert, wenn nicht sogar unmöglich. Aus diesem Grund wurden im Rahmen von diversen Normungs- und Standardisierungsmaßnahmen verschiedene Vorgehensmodelle einheitlich beschrieben.

Eines der bekanntesten Vorgehensmodelle in der Softwareentwicklung stellt das V-Modell dar. Ursprünglich für den militärischen Bereich entwickelt, wird das V-Modell seit Anfang der 90’er Jahre auch von den meisten Bundesbehörden dazu eingesetzt, Softwareprojekte zu planen und durchzuführen. Der Einfluss neuer Ansätze in der Softwareentwicklung (z.B. die Objektorientierung) machte eine Weiterentwicklung des Modells notwendig, die sich im V-Modell ’97 ausprägte [DHM98].

Die aktuellste Weiterentwicklung stellt das V-Modell XT dar. Es ist durch einen sehr hohen Grad der Anpassbarkeit an die jeweiligen Bedürfnisse (XT = Extreme Tailoring) gekennzeichnet und ist im Gegensatz zum ursprünglichen V-Modell nun explizit auch dazu geeignet, Hardware-Aspekte des Projektes mit zu berücksichtigen [RaBr06]. Eine weitere Änderung stellt die Trennung der Sichten von Auftraggeber und Auftragnehmer da. Das V-Modell XT besteht aus verschiedenen Vorgehensbausteinen, für deren zeitliche Abfolge es keine Vorschriften gibt. Somit eignet es sich auch sehr gut agile und inkrementelle Entwicklungsansätze zu beschreiben.

Neben den Vorgehensmodellen, die sich auf die reine Softwareentwicklung beschränken, haben sich in der Automatisierungstechnik verschiedene Modelle etabliert, die den Entwicklungsprozess des gesamten Automatisierungssystems im Fokus haben. Ein Beispiel stellt das NAMUR-Vorgehensmodell dar [NA35]. Der Lebenszyklus einer Anlage wird hierin mit dem Fokus auf die Prozessleittechnik von der Projektierung bis zur Inbetriebnahme und dem Abschluss des Projektes für die Teilbereiche Projektierung, Projektmanagement und Qualitätssicherung in einem top-down-Ansatz beschrieben. Der laufende Betrieb sowie Wartung und Instandhaltung werden nicht berücksichtigt. Die Beschreibung findet ausschließlich auf Dokumentenebene statt, einzelne Datenelemente werden nicht betrachtet. Das beteiligte Fachpersonal sowie die eingesetzten Werkzeuge werden nur grob betrachtet und nicht weiter spezifiziert. Bedingt durch den top-down-Ansatz beschreibt das NAMUR-Vorgehensmodell den Workflow als sequentiellen Ablauf und berücksichtigt keine iterativen oder parallelen Vorgänge.

Einen weiteren Ansatz zur Integration der Aktivitäten und Daten entlang des Lebenszyklus stellt die ISO 10303 - Standard for the Exchange of Product Model Data (STEP) dar [AnTr00]. Mittels der STEP-Normenreihe wurde der Austausch von Produktdaten zwischen verschiedenen CAD/CAM-Systemen und Produkt-Daten-Management-Systemen (PDM-Systemen) standardisiert. Die Normen sind in Gruppen von Modellen zur Beschreibung von Produktdaten (Integrated Resources, Anwendungsprotokolle), Beschreibungsmethoden (Description Methods), Implementierungsmethoden (Implementation Methods) und Methoden zum Konformitätstest (Conformance Testing Methodology and Framework) zusammengefasst. Spezifische Datenmodelle sind in den Applikationsprotokollen jeweils unter einem spezifischen Anwendungsaspekt festgelegt.

Im Rahmen der STEP-Entwicklung wurde von der PI-STEP-Initiative [PISTEP94] das Process Plant Engineering Activity Model entworfen, das für die Prozessindustrie eine Unterstützung bei der Erstellung von Applikationsprotokollen bieten soll. Es handelt sich dabei genau wie beim NAMUR-Modell um eine top-down Modellierung der Abläufe, die

in die Teilgebiete Managementaktivitäten, Prozess Design, Engineering Design und Materialverwaltung gegliedert ist. Über den gesamten Lebenszyklus von der Prozessentwicklung bis zur Stilllegung und dem Abriss der Anlage sind die Aktivitäten und die ausgetauschten Dokumente der einzelnen Teilphasen modelliert. Die Beschreibung beschränkt sich auf die Dokumentenebene, einzelne Daten, Fachpersonal oder Werkzeuge werden nicht beschrieben.

## 2.3 Softwareentwicklung in der Automatisierungstechnik

Durch die zunehmende Verlagerung von Hardware-Funktionalitäten in die Software ist der Anteil der Softwareentwicklung am gesamten Engineeringaufwand stetig gestiegen. Die Entwicklung der Steuerungssoftware stellt somit einen wesentlichen Teil des Automatisierungsprojektes dar. Dabei sind sowohl die Vorgehensweisen als auch die verwendeten Programmiersprachen historisch gewachsen. In den Anfängen bedeutete die Erstellung einer Steuerungssoftware die Entwicklung von Datenstrukturen und Algorithmen, die eine spezifische Anforderung erfüllen sollten. Dabei konnten die meisten Probleme durch eine Reihe von Schritten oder Prozeduren gelöst werden. Dieser Ansatz, meist als imperative oder prozedurale Programmierung bezeichnet, ist jedoch nur bedingt für kleine, wenig komplexe Probleme effektiv anwendbar. Mit steigender Komplexität der zu automatisierenden Systeme ist jedoch auch der Aufwand zur Erstellung der Software sowie die Schwierigkeit die Software zu dokumentieren, zu warten oder zu erweitern, gestiegen.

Mit dem Ansatz der objektorientierten Programmierung, der seit Mitte der achtziger Jahre verfolgt wurde, wurde ein neues Paradigma eingeführt. Im Gegensatz zur prozeduralen Programmierung, in deren Zentrum die Modellierung von Programmablaufplänen stand, sieht das objektorientierte Konzept eine Abstraktion durch die Modellierung von Objekten vor [For07]. Ein Objekt kann als abgeschlossenes Element betrachtet werden, welches gewisse Strukturen, Verhalten oder Zustände, Schnittstellen sowie Programmcode besitzt und diese kapselt. Neben der Kapselung sind Vererbung und Polymorphie zwei weitere Grundgedanken der Objektorientierung. Dabei erlaubt die Vererbung Objekte zu verändern ohne das ursprüngliche Objekt zu verändern. Polymorphie dient dazu, scheinbar gleichartige Objekte in unterschiedlichem Kontext mit abweichendem Verhalten auszustatten. Gemeinsam bilden diese drei Eigenschaften die Grundsäulen der Objektorientierung und fördern somit die Wiederverwendung von Softwarebausteinen. Unterstützt wird die Softwareentwicklung durch objektorientierte Designtechniken, in denen mit Hilfe von objektorientierter Analyse (OOA) ein Domänenmodell erstellt und weiterentwickelt und darauf aufbauend ein Systementwurf erstellt wird.

Eine große Herausforderung stellt die Sicherstellung der Qualität und Wartbarkeit der Software dar, wobei dadurch die Kosten für die Erstellung nicht steigen sollen. Da der Anteil der Kosten für Wartung von Software abhängig von der Anwendungsdomäne im Regelfall mindestens 50% und in Extremfällen bis zu 90 % der Gesamtkosten betragen kann [Jon98], wird das Einsparungspotential sofort deutlich. Deshalb ist es sinnvoll die Wartbarkeit der Software durch geeignete Maßnahmen in der Entwicklungsphase zu erhöhen.

Zudem besteht immer häufiger die Anforderung Softwarelösungen zielsystemunabhängig in unterschiedlichen Umgebungen mehrfach nutzen zu können, da es zu kostspielig ist, einmal erarbeitete Lösungen ohne Wiederverwendung nur einmal einzusetzen. Ein bestehender Lösungsansatz für eine effektive Wiederverwendung von Software stellt die Verwendung von Modulen dar. Module in der Softwareentwicklung sind Programmbausteine, welche eine oder mehrere bestimmte Funktionalitäten erfüllen. Dabei ist der Zugriff auf diese Funktionalität nicht beliebig sondern nur über definierte Schnittstellen möglich. Diese „Verhüllung / Verbergen“ von inneren Vorgängen und Zusammenhängen schafft eine neue Abstraktionsebene und reduziert somit die Komplexität für den Nutzer. Dieser sieht nun nicht mehr das eventuell komplexe Innere eines Moduls, sondern nur noch dessen Funktionalität, welche er in vielen Fällen durch Parameter beeinflussen kann. Doch nicht nur die genannten Zugriffsmöglichkeiten auf ein Modul, sondern die gesamte Interaktion des Moduls mit dessen Umgebung, findet über Schnittstellen statt. Sie entscheiden in einem erheblichen Maße darüber, wie gut es sich an anderer Stelle wieder verwenden lässt oder wie leicht es mit anderen Modulen zu größeren Modulen zu kombinieren ist.

Durch Anwendung von Modulen können Programme aus bereits bestehenden Modulen durch Parametrierung und eine Verschaltung der Schnittstellen erstellt werden [MBW95]. Der Zeitaufwand bei der Programmierung kann dadurch deutlich gesenkt werden. Darüber hinaus steigt die Qualität der Software, da die Module einzeln getestet werden können und die Verschaltung über Schnittstellendefinitionen geregelt werden kann. Einen weiteren, nicht zu unterschätzenden Faktor stellt die bessere Wartbarkeit der Software durch Wiederverwendung bekannter Module dar.

Objektorientierte Notationen und Programmiersprachen liefern im Vergleich zu prozeduralen Sprachen erheblich bessere Möglichkeiten der Kapselung von Code und Daten. Die Forderungen nach Modularität sind damit folglich deutlich einfacher umzusetzen. Dennoch sind noch heute besonders im Bereich der

Steuerungsprogrammierung prozedurale und funktionsorientierte Programmiersprachen Stand der Technik.

### 2.3.1 SPS-Programmierung mit IEC 61131-3

Speicherprogrammierbare Steuerungen besitzen eine zentrale Rolle bei der Automatisierung technischer Systeme. Durch die steigende Komplexität der Systeme wird auch die Komplexität der Software von SPS-Projekten gesteigert. Gleichzeitig werden Automatisierungssysteme durch die heute verfügbare Hardware mit zum Teil stark unterschiedlichen Programmiersystemen immer heterogener. Um Automatisierungssysteme auch weiterhin mit hoher Qualität und unter angemessenem Einsatz von Ressourcen entwickeln zu können, wurde eine Standardisierung der Programmiersprachen eingeführt.

In Kapitel 2.1.3 wurde der Standard IEC 61131 „Programmable Controllers“ bereits kurz vorgestellt. Den für die Programmierung solcher Systeme und damit den für die vorliegende Arbeit relevanten Teil der Norm stellt die „IEC 61131-3 - Programmiersprachen“ dar. In diesem Teil der Norm werden fünf Programmiersprachen definiert, die zur Programmierung von Speicherprogrammierbaren Steuerungen eingesetzt werden können [DIN61131-3]. Dabei wird zwischen textuellen und grafischen Sprachen unterschieden. Bevor eine Übersicht der Programmiersprachen gegeben wird, soll kurz auf das zugrunde liegende Konzept der IEC 61131-3 eingegangen werden.

Steuerungsprogramme sind keine monolithisch aufgebauten, unstrukturierten Folgen von Befehlen sondern bestehen aus einzelnen, funktionell voneinander unabhängigen Bausteinen, so genannten Programm-Organisationsbausteinen (POE), die getrennt voneinander erstellt und bearbeitet werden können. Die IEC 61131-3 definiert drei Klassen von Programm-Organisationseinheiten, die verschiedene Eigenschaften aufweisen: Funktionen, Funktionsbausteine und Programme.

Eine Funktion gemäß IEC 61131-3 ist eine POE, die beliebig viele Eingangsparameter hat und genau ein Ergebnis-Element zurückliefert. Ein Beispiel stellt die Funktion sin(x) dar. Für Funktionen gelten bestimmte Einschränkungen. So dürfen Funktionen keine interne Speicherfähigkeit besitzen, keine globalen Variablen lesen oder schreiben und aus Funktionen heraus dürfen keine Funktionsbausteine aufgerufen werden. Nur so kann sichergestellt werden, dass die Forderung, bei gleichem Satz von Eingangsvariablen stets dasselbe Ergebnis zu liefern, erfüllt wird.

Funktionsbausteine und Programme unterscheiden sich nur geringfügig voneinander. Beide POEs können beliebig viele Ein- und Ausgangsparameter haben und interne Daten besitzen, die nach einem Aufruf erhalten bleiben und beim nächsten Aufruf wieder genutzt

werden können. Die Ausgangswerte eines Funktionsbausteins oder eines Programms können somit nicht nur von den Eingangsparametern, sondern auch von dem internen Zustand der POE abhängen. Diese Eigenschaft, ein Gedächtnis besitzen zu können, sind ein wesentliches Unterscheidungsmerkmal zu Funktionen, wodurch Funktionalitäten, wie Zähler- oder Timer-Bausteine, realisiert werden können. Funktionsbausteine und Programme können mehrfach instanziiert werden. Jede Instanz besitzt einen eigenen Bezeichner und eine eigene Datenstruktur. Der wesentliche Unterschied zwischen diesen beiden POEs ist die die Eigenschaft, dass Programme weder von anderen Programmen noch von Funktionsbausteinen aufgerufen werden dürfen. Programme werden ausschließlich von Ressourcen aufgerufen, können explizit einer Task zugeordnet werden oder laufen implizit in der Hintergrundtask.

Ein wesentlicher Unterschied zwischen dem Programmablauf in der konventionellen Datenverarbeitung und bei SPS-Systemen ist die Art der Abarbeitung des Programmcodes. Während herkömmliche Softwaresysteme meist ereignisgesteuert arbeiten und bei Auftreten eines Ereignisses z.B. eine Methode ausführen, werden SPS-Programme zyklisch abgearbeitet (Abb. 2.6). Zu Beginn eines Zyklus werden die Zustände der Eingänge in das Prozessabbild der Eingänge kopiert. Anschließend wird das Programm unter Verwendung des Prozessabbildes abgearbeitet. Ausgänge werden dabei nicht direkt gesetzt, sondern ebenfalls in das Prozessabbild geschrieben. Am Ende des Zyklus wird das Prozessabbild der Ausgänge auf die physikalischen Ausgänge übertragen.

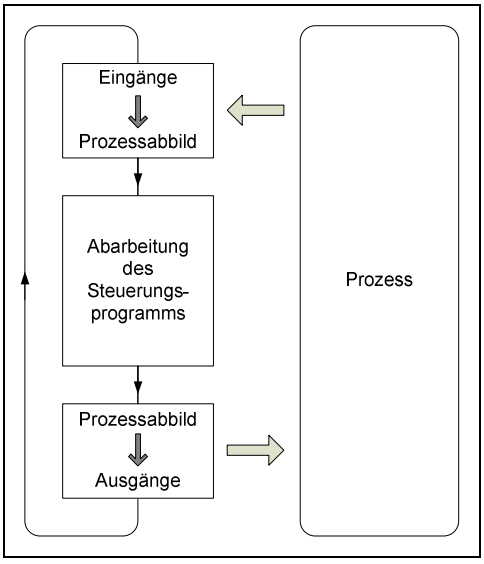



*Abb. 2.6 - Zyklische Abarbeitung eines SPS-Programms (in Anlehnung an[Grö04])*

Aus dieser Art der Programmabarbeitung ergeben sich verschiedene Randbedingungen, die bei der Programmierung der SPS besonders beachtet werden müssen. So ist zu berücksichtigen, dass die Reaktion auf ein Ereignis im ungünstigsten Fall dem doppelten der Zykluszeit entspricht, was besonders bei zeitkritischen Anwendungen berücksichtigt werden muss [Grö04]. Darüber hinaus dürfen SPS-Programme keine Rekursionen enthalten, da sonst eine Berechnung des maximalen Speicherbedarfs nicht mehr möglich ist [JoTi00].

### 2.3.2 Programmiersprachen der IEC 61131-3

Die IEC 61131-3 unterscheidet zwischen textuellen Sprachen, wie der Anweisungsliste (AWL) und dem Strukturierten Text (ST) und grafischen Sprachen, wie der Ablaufsprache (AS), der Funktionsbausteinsprache (FBS) und dem Kontaktplan (KOP). Die Sprachen können wie folgt charakterisiert werden:

- Die Anweisungsliste (AWL) ist eine Assembler-ähnliche Sprache, bei der die einzelnen Anweisungen, ähnlich einer Maschinensprache, listenartig untereinander aufgeführt werden.
- Strukturierter Text (ST) stellt im Gegensatz zur AWL eine Pascal-ähnliche Hochsprache dar. Leistungsfähige Schleifenprogrammierung, mathematische Funktionen, Iteration sowie konditionierte Befehle sind die herausragenden Bestandteile dieser Sprache. Sie ist vielmehr als Ergänzung denn als Ersatz für AWL zu sehen.
- Die Ablaufsprache (AS) definiert eine POE als eine Reihe von Schritten und Transitionen (Übergangsbedingungen), die durch gerichtete Verbindungen miteinander verbunden sind. Dadurch ist die Ablaufsprache eher zur Realisierung übergeordneter Abläufe geeignet.
- Die Funktionsbausteinsprache (FBS) ist eine grafische Sprache, die den Grundgedanken funktionsorientierter logischer Ablaufketten widerspiegelt. Eine POE in der Funktionsbausteinsprache besteht aus Netzwerken logisch verknüpfter Bausteine.
- Der Kontaktplan (KOP) ist eine grafische Programmiersprache, die sich am Stromlaufplan elektrischer Schaltungen orientiert. Zwischen zwei vertikal verlaufenden Stromschienen werden die einzelnen Teile einer Schaltung (hauptsächlich Schalter, Spulen und Funktionsbausteine) als Netzwerk dargestellt.

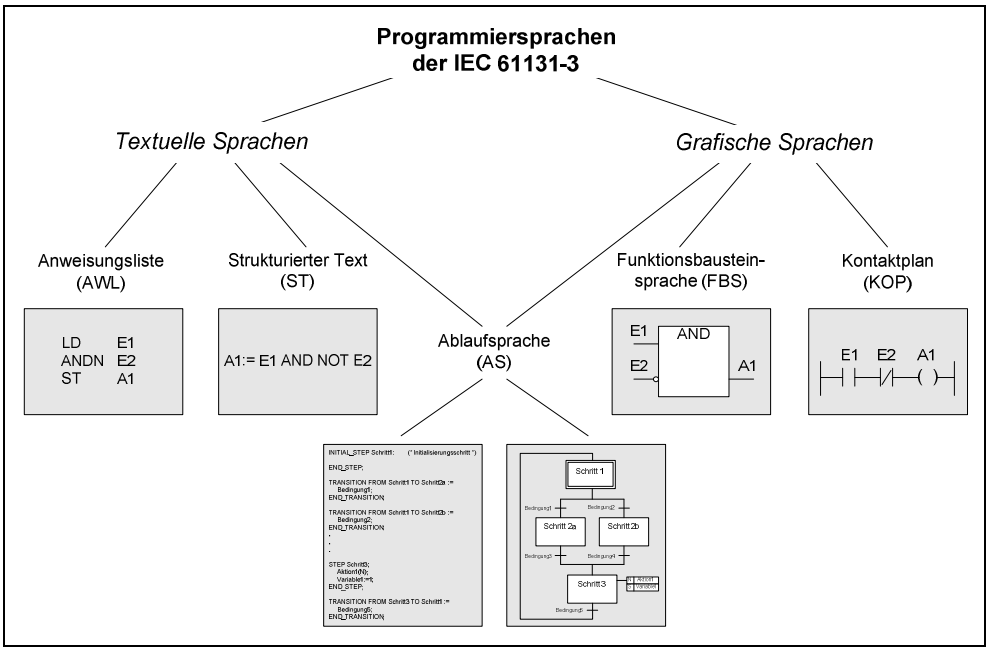



*Abb. 2.7 - Programmiersprachen der IEC 61131-3 im Überblick*

Abb. 2.7 gibt einen Überblick über die Programmiersprachen der IEC 61131-3. Auf Einzelheiten zu den Sprachen soll an dieser Stelle nicht weiter eingegangen werden, sondern auf die ausreichend verfügbare Fachliteratur zu diesem Thema verweisen werden.

### 2.3.3 Objektorientierte Ansätze in der Steuerungsprogrammierung

In der konventionellen Softwareentwicklung gab es in den letzten Jahren starke Weiterentwicklungen in Bezug auf die Verwendung objektorientierter Ansätze und Sprachen. Objektorientierung wird als das Mittel der Wahl gesehen Modularität und Wiederverwendung effizient anzuwenden. Programme in der Prozessautomatisierung sind hingegen nahezu unverändert geblieben und orientieren sich nach wie vor am Standard der IEC 61131-3. Diese Kontinuität ist auf der einen Seite gewünscht und ermöglicht so auch den Einsatz bestehender Softwaremodule. Auf der anderen Seite wird aber die Fortentwicklung durch die Anpassung an heutige Anforderungen gehemmt. Wie in den vorangegangenen Abschnitten dargestellt, handelt es sich bei den Programmiersprachen der IEC 61131-3 um prozedurale Programmiersprachen. Hier werden Funktionen und Funktionsbausteine dazu genutzt um Modularität und Wiederverwendung bis zu einem gewissen Grad zu realisieren.

Objektorientierte Ansätze werden zwar seit längerem diskutiert, haben sich bisher jedoch nicht derart durchgesetzt und wurden bisher nicht in marktreife Programmierumgebungen umgesetzt.

Katzke et al. [KVW05] verfolgen den Ansatz objektorientiert Modelle in die Standard-IEC-Sprachen zu transformieren. Hier wird eine Kombination von Funktionsbausteinen und Variablenstrukturen zur Abbildung von Klassen genutzt. Dieser Ansatz wurde prototypisch in Form eines Codegenerators umgesetzt, der ein objektorientiertes Modell zur automatischen Erstellung von Steuerungscode nutzt. Einen ähnlichen Ansatz nutzen auch Bonfe et al. [BFS05].

Durch die Programmierumgebung CoDeSys V3 [Hes05] wurde der objektorientierte Ansatz nun erstmals als Erweiterung der bestehenden IEC 61131-3 Sprachen implementiert und somit die Nutzung objektorientierter Sprachmittel im Umfeld der Speicherprogrammierbaren Steuerungen ermöglicht. Zu diesem Zweck werden die in der IEC 61131-3 enthaltenen Funktionsbausteine zu Klassen erweitert und können damit Interfaces implementieren, von anderen Funktionsblöcken erben und untergeordnete Methoden enthalten. Realisiert wird die objektorientierte Erweiterung durch die Einführung von sechs neuen Schlüsselworten [Sch08]:

- METHOD: Bezeichner für eine Methode, d.h. einer Funktion die einer Funktionsbausteindefinition zugeordnet ist.
- PROPERTY: Bezeichner für Eigenschaften, die den externen Zugriff auf Daten des Funktionsbausteins erlauben, als wären es Attribute des Bausteins.
- THIS: Bezeichner, um auf eine implizite Instanz eines Funktionsbausteins zu zeigen, die automatisch verfügbar ist.
- EXTENDS: Bezeichner für eine Vererbung von Methoden und Attributen von einem Funktionsbaustein (Vaterklasse) zu einem anderen (Kindklasse).
- INTERFACE: Bezeichner für die Realisierung von Schnittstellen, also einem Satz von Methoden, der mit gleichen Parametern aber unterschiedlicher Ausführung von verschiedenen Funktionsbausteinen realisiert werden kann.
- IMPLEMENTS: Bezeichner für die Instanz einer Schnittstelle innerhalb eines Funktionsbausteins

Diese objektorientierten Mechanismen stellen eine reine Option dar, die die Wiederverwendung von Softwarebausteinen deutlich verbessern und Entwicklungszeiten reduzieren soll. Die herkömmliche, prozedurale Programmierung kann dabei frei mit

objektorientierter Programmierung kombiniert werden. Einen ersten Anwendungsleitfaden für die Lösung automatisierungstechnischer Problemstellungen mit Hilfe objektorientierter Methoden in CoDeSys V3 stellt [VoWa08] dar. Da CoDeSys V3 jedoch erst kurz vor Veröffentlichung dieser Arbeit auf den Markt gebracht wurde, stehen bezüglich der Anwendung noch wenig Erfahrungswerte zur Verfügung.

# 3 Modellierungsnotationen für die Spezifikation von Steuerungssoftware

In der konventionellen Softwareentwicklung hat sich die Modellierung des zu realisierenden Systems für den Entwurf und die Dokumentation der Software bereits fest etabliert. Beschreibungsmittel und dazugehörige Entwicklungswerkzeuge, die auf die entsprechende Zielumgebung bzw. die verwendete Programmiersprache angepasst sind, finden dort eine breite Anwendung.

Auch in der Automatisierungstechnik wird die Modellierung für verschiedenste Aufgabenstellungen angewendet. Das Spektrum reicht dabei von der Anforderungsformulierung über die Spezifikation von Strukturen oder Verhalten, Sprachen zur Implementierung bis zur Produktdokumentation [FrKo03]. Dafür steht eine Reihe von Beschreibungsmitteln zur Verfügung, die den entsprechenden Sachverhalt mit Hilfe grafischer, textueller oder symbolischer Notationen beschreiben. Zur Klassifizierung und Bewertung verschiedener Beschreibungsmittel in der Automatisierungstechnik stellt die VDI/VDE 3681 ein Bewertungsschema zur Verfügung, mit dessen Hilfe die Auswahl geeigneter Beschreibungsmittel unterstützt wird [VDE3681]. Zudem gibt die Richtlinie für eine Auswahl an Beschreibungsmitteln eine Einordnung in das Schema. Die Bewertung macht dabei deutlich, dass die meisten Beschreibungsmittel nur einen Teil des Engineering Lifecycle abdecken oder nur eine bestimmte Sicht auf das System darstellen. Folglich werden entsprechende Systeme meist nur in bestimmten Phasen des Lebenszyklus oder nur zur Beschreibung bestimmter Aspekte eines Systems eingesetzt.

Die Unified Modeling Language (UML) stellt ein Beschreibungsmittel dar, das nach der in der VDI/VDE 3681 durchgeführten Bewertung „*… prinzipiell eine Unterstützung über sämtliche Phasen des Systemlebenszyklus hinweg …*“ bietet [VDE3681]. Dennoch wird die UML derzeit nur zur Unterstützung der frühen Phasen (Anforderungsanalyse, Grobdesign) eingesetzt. Als eine Schwäche nennt die Norm die hohe Komplexität. Ein Beleg dafür wird zwar nicht geliefert, die Einschätzung deckt sich jedoch mit den in Kapitel 1 dargestellten Vorbehalten gegen die Anwendung der UML in der Automatisierungstechnik.

Aus diesem Grund wurde die UML für eine vergleichende Evaluierung zur Modellierung von Prozessen und Programmierung von Steuerungen ausgewählt. Als Referenz dient die Idiomatic Control Language (ICL). Beide Beschreibungsmittel werden im Folgenden beschrieben und im Anschluss ihre Eignung für die Beschreibung von steuerungstechnischen Aufgaben gezeigt.

## 3.1 Unified Modeling Language (UML)

Die Unified Modeling Language ist eine Sprache zur objektorientierten Softwaremodellierung. Sie entstand 1995 aus der Vereinigung der Ansätze von Booch, Rumbaugh (OMT) und Jacobsen (OOSE) zu einer einheitlichen Notation und wurde 1997 in der Version 1.1 von der OMG als Standard anerkannt [WeOe04]. Seit dem wird die UML unter Aufsicht der OMG weiterentwickelt. Der gegenwärtige OMG-Standard ist die Version 2.1.2. Eine Chronologie der Entwicklung wird in Abb. 3.1 gegeben. Mit der ISO/IEC 19501 [IEC19501] besteht auch eine internationale Standardisierung der UML in der Version 1.4, die unabhängig von den Herstellerinteressen der OMG-Mitglieder ist. Die vorliegende Arbeit basiert auf diesem Standard.

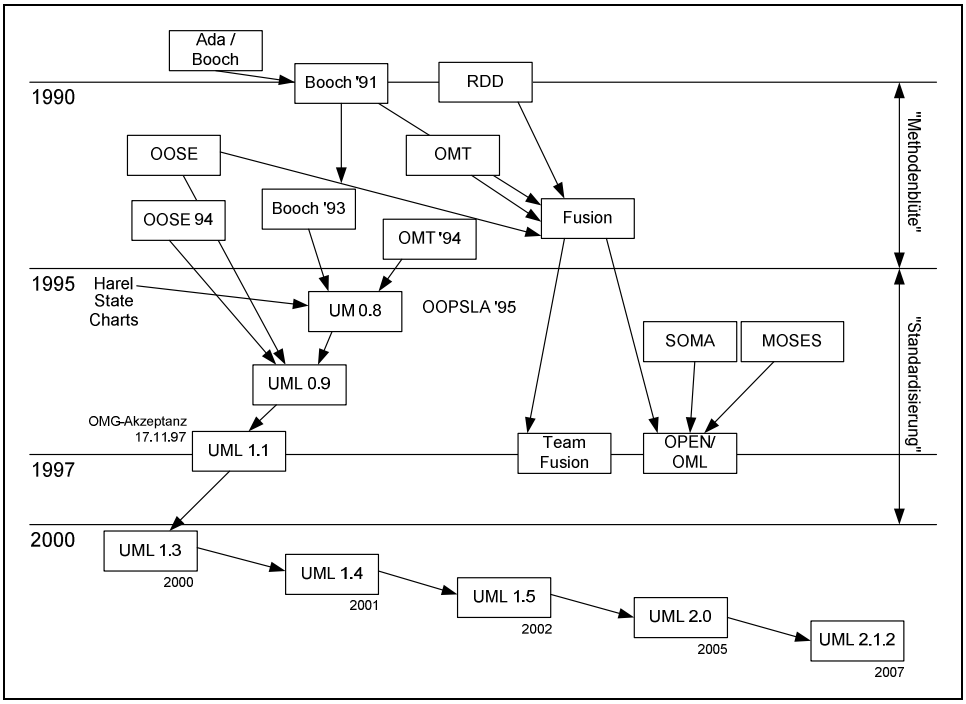



*Abb. 3.1 - Historie der Entwicklung der Unified Modeling Language (in Anlehnung an [Oes01])*

Die UML unterstützt die Spezifikation, Visualisierung und Dokumentation von komplexen Systemen [Pen03], [IEC19501]. Sie basiert, gemeinsam mit der Meta-Object Facility (MOF), auf einem Meta-Metamodell, welches die Basis für die Entwicklung objektorientierter Modellierungssprachen bildet. Die UML definiert sowohl die Notation als auch die Semantik der enthaltenen Modellelemente in einem Meta-Modell. Die Definition des Metamodells ist semiformal und nicht vollständig. Die Definition der UML

verzichtet in einigen Aspekten (z. B. imperative Sprachelemente) explizit auf eine vollständige Festlegung der UML und gibt damit Werkzeugherstellern und Modellierern einen Freiraum für eigene, zusätzliche Spezifikationen.

Die UML stellt Diagramme zur Verfügung, mit deren Hilfe der Nutzer die grafische Modellierung des Systems durchführen kann [BRJ99]. Die unterschiedlichen Diagrammarten dienen dazu, das System aus verschiedenen Blickwinkeln in unterschiedlicher Detaillierung zu beschreiben. Die Diagramme lassen sich in verschiedene Klassen einteilen und beschreiben dabei die Anforderungen an das System, die Struktur, das Verhalten oder die Implementierung des Systems. Tab. 3.1 zeigt die Unterscheidung der Diagrammarten, die im Folgenden näher beschrieben werden.

*Tab. 3.1 - Klassifizierung der Diagrammarten in der UML 1.4*

<table>
<tr><th>Sichtweise</th><th colspan="2">Diagrammtyp</th></tr>
<tr><td>Anforderungen</td><td colspan="2">- Anwendungsfalldiagramm</td></tr>
<tr><td>Struktur</td><td colspan="2">- Klassendiagramm<br>- Objektdiagramm</td></tr>
<tr><td rowspan="2">Verhalten</td><td colspan="2">- Zustandsdiagramm<br>- Aktivitätsdiagramm</td></tr>
<tr><td>Interaktion</td><td>- Sequenzdiagramm<br>- Kollaborationsdiagramm</td></tr>
<tr><td>Implementierung</td><td colspan="2">- Komponentendiagramm<br>- Verteilungsdiagramm</td></tr>
</table>

### 3.1.1 Anwendungsfalldiagramm

Das Anwendungsfalldiagramm (engl. *use case diagram*) dient der Spezifikation der Anforderungen an das System. In einem Anwendungsfalldiagramm werden Anwendungsfälle (use cases) und Akteure mit ihren Beziehungen dargestellt. Es handelt sich dabei nicht um die Beschreibung eines zeitlichen Ablaufs, vielmehr werden funktionale Zusammenhänge in der Art dargestellt, dass gegenseitige Abhängigkeiten sichtbar werden und Teilfunktionen sowie Spezialisierungen vorhandener Funktionen unterscheidbar sind. Die Gesamtheit der in einem UML Modell aufgeführten Anwendungsfälle beschreibt die funktionalen Anforderungen an das System.

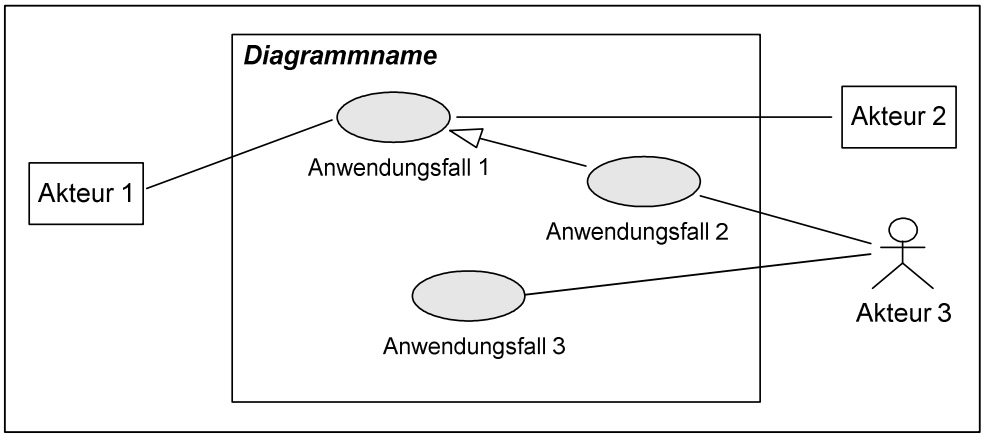



*Abb. 3.2 - Anwendungsfalldiagramm (Use-Case-Diagramm) in der UML [Oes01]*

Anwendungsfalldiagramme können hierarchisch angeordnet sein, wodurch es möglich ist, Anwendungsfälle durch ein weiteres Diagramm zu verfeinern.

### 3.1.2 Klassendiagramm

Die statische Struktur eines Systems wird durch das Klassendiagramm dargestellt. Die Struktur von Elementen des Systems wird durch Klassen und Objekte repräsentiert. Klassen stellen selbst noch keine Elemente eines Systems dar, sondern sie kennzeichnen die Typen solcher Elemente. Klassen bilden eine Zusammenstellung aus Attributen zur Beschreibung des Zustands und Operationen zur Beschreibung des Verhaltens. Klassen werden durch ihre Namen identifiziert, die dann als Bezeichner für Objekttypen des Systems zur Verfügung stehen.

Auf der Basis von definierten Klassen bilden Objekte die tatsächlichen Elemente eines Systems. Objekte werden als Instanzen von Klassen gebildet. Ein Objekt, dem keine Klasse als Typ zugeordnet ist, ist eigenschafts- und funktionslos. In einem System können beliebig viele Objekte den gleichen Typ haben. Entsprechend können in einem UML-Modell eines Systems beliebig viele Objekte als Instanzen einer Klasse gebildet werden. Das Klassendiagramm kann Klassen und Objekte beinhalten.

Das Klassendiagramm beschreibt zudem die Beziehungen (Relationen) zwischen den Klassen. Die UML definiert u. a. folgende Beziehungen:

- Vererbung
  Vererbung bezeichnet das Konzept eine neue Klasse durch die Ableitung aus einer Oberklasse zu erzeugen. Dabei erbt die Unterklasse alle Attribute und Operationen

ohne diese neu definieren zu müssen. Die Unterklasse kann durch weitere Attribute und Operationen erweitert werden. Es ist auch möglich, vorhandenes Verhalten einzelner Operationen neu zu definieren. Mit Hilfe der Vererbung können Generalisierung und Spezialisierung realisiert werden. Die Vererbung ist geeignet, um Varianten ähnlicher Systemelemente in ihren Gemeinsamkeiten und Unterschieden zu beschreiben.

- Assoziation
  Eine Assoziation ist reflexiv oder sie besteht zwischen mehreren Klassen. Ist sie binär, besteht sie immer zwischen zwei Klassen. Sie gibt ohne Festlegung der Richtung an, dass sich Objekte beider Klassen kennen und mit einander in Interaktion treten können. Mit Hilfe von Rollen kann festgehalten werden, über welche Objektnamen sich die Klasseninstanzen in der Relation jeweils Identifizieren können. Um festzulegen welche Anzahl von Instanzen einer Klasse A mit welcher Anzahl von Instanzen der Klassen B verbunden sind, werden Kardinalitäten verwendet. Auch Assoziationen einer Klasse auf sich selbst sind möglich, diese geben an, wie viele Instanzen einer Klasse miteinander in Verbindung stehen.

- Aggregation
  Eine Aggregation ist eine besondere Form der binären Assoziation, bei der die beteiligten Klassen keine gleichwertige Beziehung führen, sondern eine „Ganzes-Teile“-Hierarchie darstellen. In Abhängigkeit des Blickwinkels wird sie als „ist-Teil-von“- oder als „besteht-aus“-Beziehung gedeutet. Dabei kann das Teil auch ohne das Ganze bestehen.

- Komposition
  Im Gegensatz zur Aggregation beschreibt die Komposition eine „Ganzes-Teile“-Hierarchie, bei der das Einzelteil vom Ganzen existenzabhängig ist, so dass die Existenz seiner Instanzen mit der Existenz des jeweiligen übergeordneten Ganzen endet.

### 3.1.3 Objektdiagramm

Während das Klassendiagramm Ressourcen modelliert, modelliert das Objektdiagramm Beispiele [OMG02]. Zu einem bestimmten Zeitpunkt werden die Instanzen der Klasse des Klassendiagramms so wie ihre Beziehungen beschrieben. Das Objektdiagramm kann also als Diagramm instanziierter Klassen zu einem bestimmten Zeitpunkt gedeutet werden. Ein Klassendiagramm, welches nur Objekte und keine Klassen enthält, ist ein Objektdiagramm.

### 3.1.4 Sequenzdiagramm

Mit Sequenzdiagrammen ist es möglich ausgewählte Szenarien detailliert zu beschreiben. Das Sequenzdiagramm ist ein Interaktionsdiagramm. Jedes Sequenzdiagramm beschreibt eine mögliche, begrenzte Folge von Nachrichten, die zwischen einer Auswahl von Objekten ausgetauscht werden. Dazu werden die betrachteten Objekte in horizontaler Achse aufgetragen. Auf vertikaler Achse, die den Fortschritt in der Folge der Nachrichten beschreibt, wird auf der Lebenslinie dargestellt, wann Objekte erzeugt oder zerstört werden und wann sie aktiv sind. Nachrichten und Antworten werden mit Hilfe von horizontalen Pfeilen, die vom Sender zum Empfänger führen, realisiert. Die vertikale Anordnung der Botschaften verdeutlicht ihre Reihenfolge. Die vertikale Anordnung von Nachrichten kann gemeinsam mit der Ausdehnung von aktiven und inaktiven Abschnitten der Lebenslinien auch als zeitlicher Verlauf interpretiert werden. Eine Skala zur Kennzeichnung feststehender Zeitpunkte und Zeitspannen existiert nicht.

### 3.1.5 Zustandsdiagramm

Das Zustandsdiagramm stellt alle diskreten Zustände dar, die ein Objekt oder eine Operation im Laufe seiner Laufzeit einnehmen kann. Zudem beschreibt es, wie externe Stimuli Änderungen des Objektzustandes hervorrufen. Ein Zustandsdiagramm setzt sich aus Zuständen, Transitionen und Ereignissen zusammen. Sowohl Zustände, als auch Ereignisse können um Aktionen ergänzt werden.

In dem diskreten Verhaltensmodell der UML kann jedes Objekt eine endliche Menge von möglichen Zuständen einnehmen, jedoch ist für ein Objekt genau ein Zustand zu einem Zeitpunkt gültig. Zustände sind eindeutig und besitzen einen Namen. Ein Zustandsdiagramm besitzt genau einen Anfangszustand, kann jedoch mehrere Endzustände enthalten. Zustandsdiagramme können hierarchisch erweitert werden, indem ein Zustand durch ein detailliertes Zustandsdiagramm weiter beschrieben wird. Jeder Zustand kann Aktionen und eine Aktivität enthalten.

Eine Aktion ist eine atomare, nicht unterbrechbare Handlung, die unabhängig von äußeren Einflüssen vollständig ausgeführt wird. Aktionen können beim Eintritt in oder Austritt aus einem Zustand ausgeführt werden und sind mit dem Schlüsselwort *entry* oder *exit* gekennzeichnet.

Im Gegensatz zur Aktion ist eine Aktivität eine unterbrechbare Handlung, die durchgeführt wird während der Zustand der Aktivität gültig ist. Eine Aktivität beginnt mit der Aktivierung eines Zustandes und endet mit seiner Deaktivierung, falls sie nicht bereits vorher terminiert.

Eine Transition beschreibt den Übergang zwischen zwei Zuständen. Sie wird durch Ereignisse ausgelöst und ist nicht unterbrechbar. Eine Transition kann um eine, durch einen logischen Ausdruck formulierte, Bedingung ergänzt werden. Der Zustandsübergang bei eintreffen eines Ereignisses findet nur dann statt, wenn die zugehörige Bedingung erfüllt ist. Nicht immer führt eine Transition zu einem Zustandswechsel, es gibt Transitionen, deren Ausgangs- und Folgezustand identisch sind. Eine Transition kann ebenfalls eine Aktion auslösen. Sie wird unmittelbar nach Verlassen des Ausgangszustandes ausgeführt.

Ereignisse sind Begebenheiten, die im Zusammenhang mit einem Zustand einen Zustandsübergang auslösen können. Ereignisse können z.B. eintreffende Nachrichten oder der Aufruf einer Operation der übergeordneten Klasse sein.

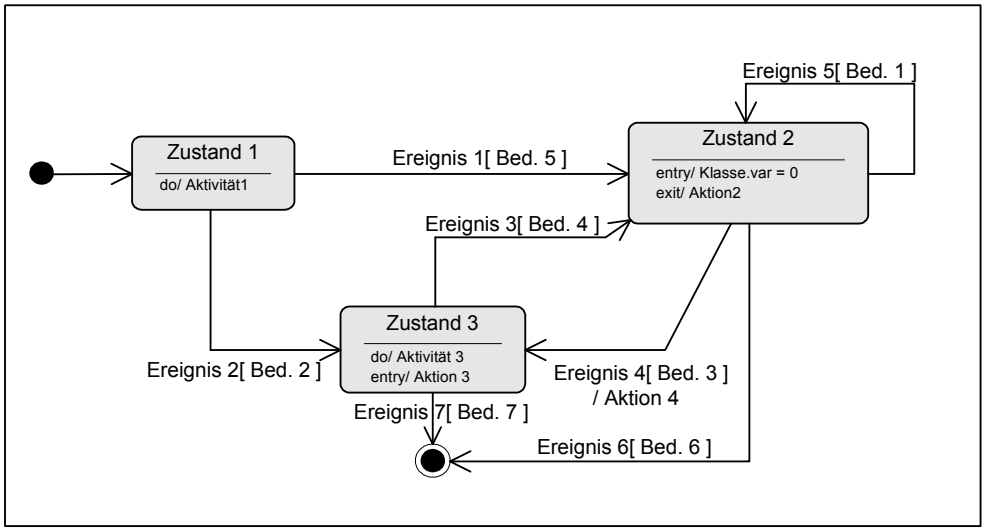



*Abb. 3.3 - Beispiel für ein Zustandsdiagramm in der UML*

Aufgrund ihrer Eigenschaften sind Zustandsdiagramme sehr gut dazu geeignet, das vollständige Verhalten von Objekten darzustellen. Insbesondere können sie

- zeigen, wie ein Objekt auf äußere Einflüsse reagiert,
- darstellen, auf welche Ereignisse ein Objekt in einem bestimmten Zustand reagiert und auf welche nicht und
- analysieren, welche Daten zur Beschreibung eines Zustandes notwendig sind.

### 3.1.6 Aktivitätsdiagramm

Ein Aktivitätsdiagramm der UML 1.4 ist eine spezielle Form des Zustandsdiagramms, das überwiegend Zustände mit Aktionen enthält. Das Ende einer Aktion bildet implizit das Ereignis für Transitionen zu Folgezuständen und –aktionen. Das Diagramm beschreibt den Kontrollfluss zwischen den Aktivitäten und entspricht in seinem Aufbau einem Flussdiagramm. Durch Splitting- und Synchronisationselemente besteht die Möglichkeit auch parallele Abläufe dazustellen. Eine Unterteilung in Verantwortlichkeitsbereiche (so genannte Swimlanes) ermöglicht die Zuordnung einzelner Aktivitäten zu bestimmten Strukturen.

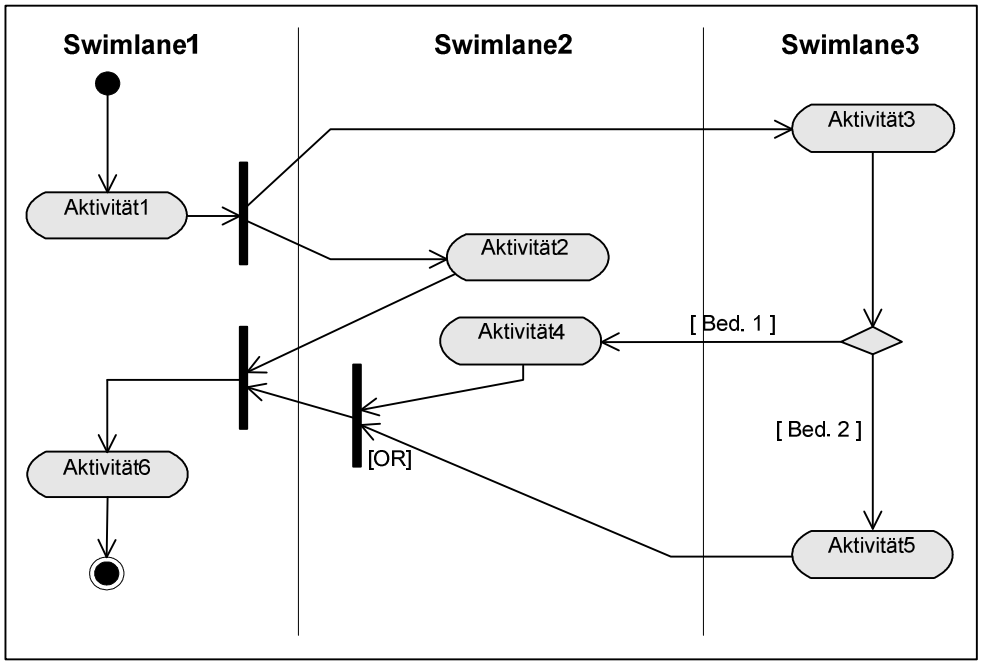



*Abb. 3.4 - Beispiel für ein Aktivitätsdiagramm in der UML*

### 3.1.7 Kollaborationsdiagramm

Ein Kollaborationsdiagramm zeigt die Interaktionen und Beziehungen zwischen einer Menge ausgewählter Objekte in einem bestimmten Kontext. Inhaltlich kann das Kollaborationsdiagramm weitgehend die gleichen Inhalte wie das Sequenzdiagramm beschreiben, es werden dabei jedoch die Beziehungen zwischen den Objekten und ihre Topographie in den Vordergrund gestellt. Sender und Empfänger müssen eine Assoziation miteinander besitzen, damit sie Nachrichten austauschen können. Der zeitliche

Zusammenhang zwischen den ausgetauschten Nachrichten wird durch Nummerierung gekennzeichnet.

### 3.1.8 Komponentendiagramm

Das Komponentendiagramm zeigt die Organisation und die Beziehungen von Komponenten. Während Klassen die logische Organisation und Absicht des Softwaredesigns darstellen, bieten Komponenten gekapselte Bündel von Klassen und Artefakten. Sie werden in Komponentendiagrammen mit ihren Kommunikationsbeziehungen dargestellt und bieten so eine weitere Sicht auf das System. Zur weiteren Strukturierung können Komponenten zu einem Paket zusammengefasst werden und das System so in kleine überschaubare Teile gegliedert werden.

### 3.1.9 Verteilungsdiagramm

Mit dem Verteilungsdiagramm wird die Hardwarearchitektur des Systems mit Hilfe von Knoten, Komponenteninstanzen, mit darin enthaltenen Objekten, und Beziehungen beschrieben.

Knoten beschreiben physikalische Objekte wie z.B. Prozessoren oder Geräte, die zur Laufzeit für die Verarbeitung von Softwarekomponenten zur Verfügung stehen. Knoten haben Kommunikationsbeziehungen untereinander, durch die physikalische Verbindungen wie Kabel oder Netzwerke beschrieben werden. Die Verbindungen zwischen Komponenteninstanzen in Verteilungsdiagrammen weisen auf logische Kommunikationsbeziehungen.

### 3.1.10 Erweiterungsmechanismen und Profile

Die UML bildet eine Vereinheitlichung früherer objektorientierter Modellierungsansätze. Sie erhebt nicht den Anspruch auf Vollständigkeit. Stattdessen werden Mechanismen angeboten, mit der die UML und der Umgang mit ihr in der Modellierung für spezielle Aufgaben angepasst werden können. Das Modell der UML bildet ein Metamodell für die Anwendungsentwicklung. Jedes Element dieses Metamodells kann beschränkt und erweitert werden. Beschränkungen werden in der Object Constraint Language (OCL) beschrieben. Die OCL ist Teil der UML Spezifikation. Ebenso kann jedes Modellelement durch Tag Definitions erweitert werden. Sie bilden Attribute des Metamodells, die in der Anwendungsentwicklung mit Tagged Values belegt werden müssen. Beschränkungen und Erweiterungen werden zu Stereotypen zusammengefasst. Ein Stereotyp definiert ein neues Element des Metamodells auf der Basis vorhandener Elemente. Beispielsweise ist eine Schnittstellenklasse eine spezielle Klasse, die durch einen Stereotypen

<<Schnittstellenklasse>> gekennzeichnet werden kann. Stereotypen beziehen sich auf ein Element des UML Metamodells. Um viele Änderungen und Ergänzungen zusammenzufassen, werden Profile gebildet.

Ein UML Profil ist eine Zusammenfassung von Stereotypen mit dem Ziel, die UML zur Modellierung spezieller Aufgaben geeignet zu machen. Es existieren zahlreiche UML Profile für spezielle Modellierungsaufgaben, auf die hier nicht gesondert eingegangen wird. Auch für die Modellierung von Automatisierungsaufgaben oder Automatisierungssoftware liegen unterschiedliche Konzepte vor. Beispiele sind das Profil der UML-RT [SeRu98] oder das UML Profile for Schedulability, Performance and Time [OMG05].

Mit der SysML und der UML-PA sind im Zusammenhang mit dem hier diskutierten Themenbereich zudem zwei Profile von besonderem Interesse, die erst während oder nach Durchführung der Evaluationsexperimente veröffentlicht wurden.

Die Systems Modeling Language (SysML) [OMG07] basiert auf der UML 2.1 und ist für die Spezifikation, das Design und die Verifikation von komplexen Systemen geeignet. Die SysML ordnet die verfügbaren Strukturdiagramme völlig neu. Klassendiagramm, Objektdiagramm, Kompositionsstrukturdiagramm, Kommunikationsdiagramm, Komponentendiagramm und Verteilungsdiagramm werden durch das Blockdefinitionsdiagramm und das Interne Blockdiagramm ersetzt [Wei06]. Darüber hinaus werden mit dem Anforderungsdiagramm und dem Parameterdiagramm zwei neue Diagrammtypen eingeführt. Die Diagramme der SysML und deren Zusammenhänge sind in Abb. 3.5 dargestellt.

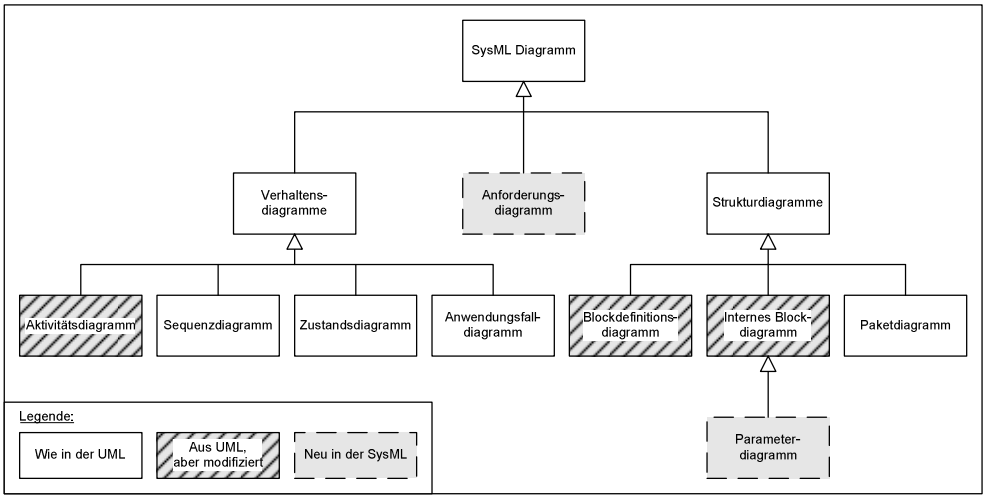



*Abb. 3.5 - Übersicht der Diagramme der SysML*

Die UML-PA [Kat08] stellt eine domänenspezifischen Anpassung der UML 2.0 speziell für die Automatisierungstechnik dar. Im Rahmen des DFG Schwerpunktprogramms 1064 „Integration von Techniken der Softwarespezifikation für ingenieurswissenschaftliche Anwendungen“ wurden im Rahmen des Projekts DisPA die verschiedenen Anforderungen in der Automatisierungstechnik aufgenommen und versucht deren Lösungen in einer angepassten UML-Variante, der so genannten UML-PA (UML für die Prozess-Automatisierung) verfügbar zu machen. Die UML-PA versucht durch die Zusammenführung und/oder Eliminierung verschiedener, teilweise redundanter Diagrammtypen, das Hinzufügen des Timingsequenzdiagramms und die Erweiterung des Sprachumfangs die Anwendbarkeit der UML für die Beschreibung automatisierungstechnischer Problemstellungen zu steigern. Die Diagramme der UML-PA und deren Zusammenhänge sind in Abb. 3.6 dargestellt.

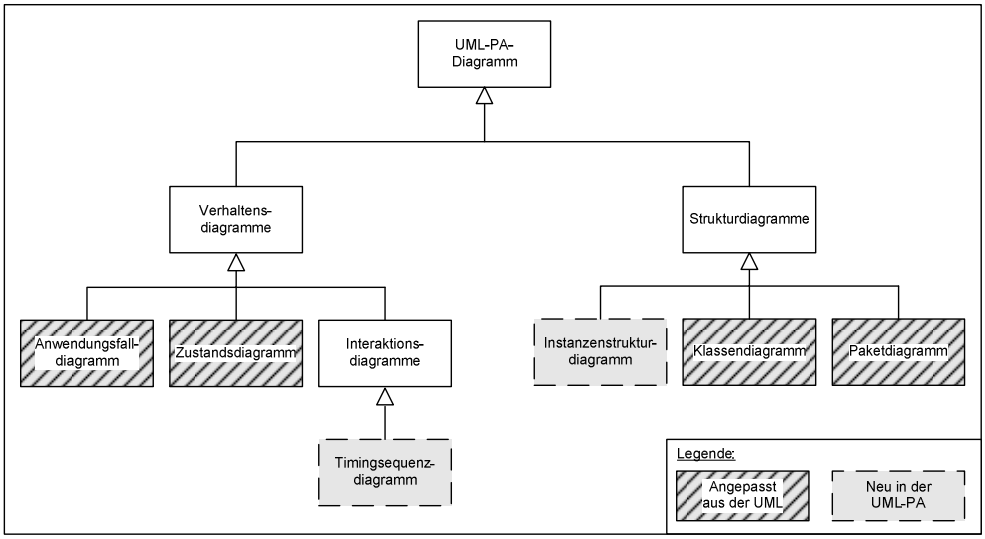



*Abb. 3.6 - Übersicht der Diagramme der UML-PA*

### 3.1.11 Vorgehensmodelle und Werkzeugunterstützung

Die UML ist nicht auf die Modellierung von Systemen innerhalb spezieller Domänen beschränkt. Entsprechend unterscheidet sich das Vorgehen im Entwurf eines plattformunabhängigen Softwaresystems von dem Vorgehen zur Modellierung eines eingebetteten Systems [Kat08]. Für den ersten Fall wird innerhalb der UML Spezifikation beispielhaft das UML Profile for Software Development Processes vorgeschlagen. Dieses Vorgehensmodell orientiert sich an dem Konzept des Rational Unified Process, einem von Rational gemeinsam mit dem UML Werkzeug Rational Rose vertriebenen Entwurf zur

Software Modellierung. Das Prinzip beider Ansätze ist die schrittweise Verfeinerung des Softwaremodells. In einem ersten Schritt wird ein Modell ausschließlich durch seine Anforderungen in einem Use Case Model beschrieben. In einem Analysemodell werden dann Softwareobjekte erfasst und in ihren Beziehungen beschrieben. Der Softwareentwurf und die detaillierte Verhaltensspezifikation erfolgt in dem Designmodell. Erst dann wird in dem Implementierungsmodell der Bezug zur Hardware hergestellt.

Für die Modellierung eingebetteter Systeme ist dieses Vorgehen problematisch, da Verteiltheit und Heterogenität der Hardware und auch das Zeitverhalten der Kommunikation wesentliche Aspekte sind, die nicht erst spät einbezogen werden dürfen. In dem Konzept der agilen Softwareentwicklung [HrRu02] wird angeregt, solche Aspekte zu berücksichtigen und den Modellierungsprozess daran anzupassen. Speziell für die Modellierung eingebetteter Systeme werden dort Prinzipien der strukturierten Analyse aufgegriffen, nach denen zunächst der Kontext eines Systems erfasst wird. Die Anforderungen werden dann aus den ein- und ausgehenden Daten- und Kontrollflüssen zwischen dem System und seiner Umgebung bestimmt.

Die UML gibt von sich aus kein Vorgehensmodell vor. Es existieren naturgemäß jedoch gewisse Abhängigkeiten zwischen den Modellelementen, durch die auch gewisse Abläufe vorgegeben sind. Darüber hinaus hat der Anwender jedoch die Möglichkeit die Modellierung im Bezug auf den zeitlichen Ablauf (wann wird welches Diagramm erstellt) frei zu wählen. Auch im Bezug auf die Modellierung mit Hilfe der einzelnen Diagrammtypen gibt es meist nur syntaktische Einschränkungen, denen sich der Anwender fügen muss. Diese Freiheit ermöglicht eine weite Anwendung der UML zu vielen verschiedenen Zwecken. Darüber hinaus enthält die UML durch ihre Erweiterungsmechanismen und Unschärfen viele Freiheitsgrade, die von den Herstellern der Modellierungswerkzeuge unterschiedlich interpretiert werden. Je nach dem Fokus eines Werkzeugs sind die Vorgaben der UML unterschiedlich stark umgesetzt. Ähnlich verhält es sich mit den vorgeschlagenen Vorgehensweisen dieser Konzepte. Sie orientieren sich an den Möglichkeiten der jeweils angebotenen Werkzeuge.

## 3.2 Idiomatic Control Language (ICL)

Die Idiomatic Control Language (ICL) ([Bri94], [Bri00]) ist eine nicht standardisierte Notation zur Beschreibung von Steuer- und Regelsystemen, die von E. Bristol auf Basis seiner langjährigen Erfahrung in der Verfahrentechnik entworfen wurde. Ziel ist es, eine Notation bereit zu stellen, mit der eine integrierte Modellierung und Konfigurierung von

Steuer- und Regelungssystemen ermöglicht wird. Die Notation soll dabei für Ingenieure und Techniker gleichermaßen leicht verständlich und anwendbar sein:

> *„Idiomatic Control Language is designed for the integrated modeling and configuration of control systems, to emphasize ease of use and system integration.“ [Bri06]*

Die ICL verfolgt einen idiomatischen Ansatz. Dabei werden in der ICL verstärkt Begriffe der natürlichen Sprache verwendet, die ihren Ursprung im Zielbereich, also der Verfahrens- und Prozessleittechnik sowie der Regelungstechnik, haben. Zudem wird durch Reduktion der Komplexität einzelner Elemente versucht die Funktionalität in den Vordergrund zu stellen und Realisierungsaspekte in einer unterlagerten Ebene zu beschreiben.

Das Basiselement einer ICL-Modellierung ist eine Operation, die einen Ausschnitt des Prozesses beschreibt. Eine Operation kann untergeordnete Operationen (Sub-Operationen) enthalten und wird mit Hilfe von Pages detailliert beschrieben.

### 3.2.1 Pages

Die *pages* sind das Strukturbeschreibungselement der ICL und bilden das Grundgerüst einer jeden Modellierung. Zur Strukturierung der Beschreibung werden unterschiedliche Arten von *pages* angeboten, die jeweils unterschiedliche Aufgaben erfüllen:

- operation page:
  Die operation page ist ein geschlossenes Element, welches alle Teile einer Steuerung oder Regelung eines Anlageteils zusammenfasst. Neben den Zuständen der Operation (System States und User States) werden alle Tasks und Suboperationen beschrieben.

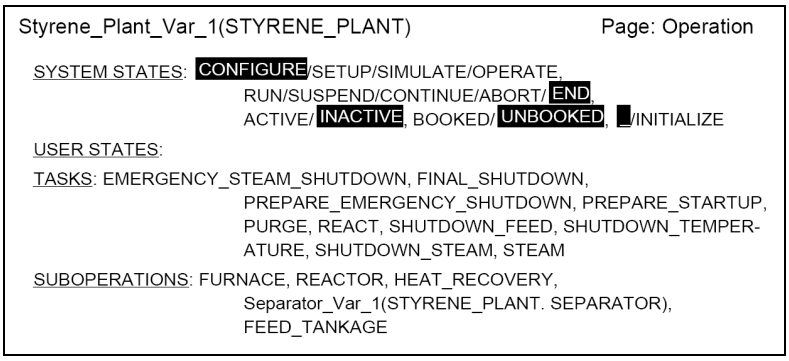



*Abb. 3.7 - Beispiel einer operation page [Bri06]*

- definition page:
  Die definition page ist einer operation page untergeordnet. Sie bietet die Möglichkeit, alle Variablen sowie Prozesseingänge und –ausgänge zu definieren. In Form einer tabellarischen Übersicht werden die Größen gruppiert dargestellt. Zudem können Grenzwerte oder charakteristische Zustände einzelner Prozessgrößen definiert und mit natürlichspachlichen Ausdrücken verknüpft werden.

- procedure page:
  Eine procedure page ist einer operation page untergeordnet und beschreibt den Hauptablauf der Operation (erste Operation, unbenannt) sowie den Ablauf aller ihr zugeordneten Tasks (weitere Operationen, benannt). Zur Beschreibung der Abläufe können Sequential Function Charts (Kap. 3.2.2) und Idiome (Kap. 3.2.4) verwendet werden.

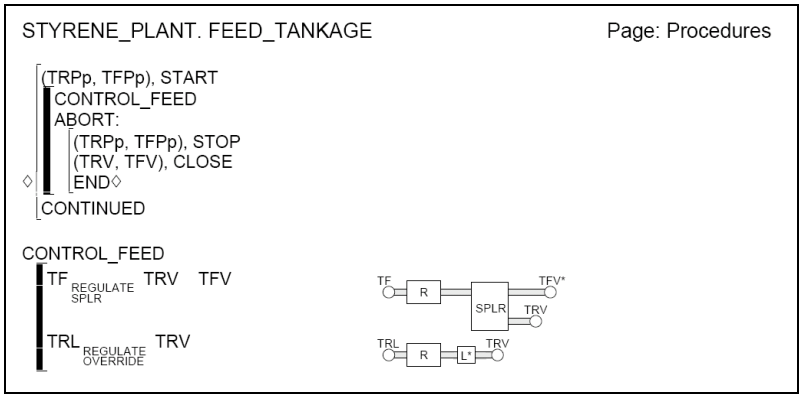



*Abb. 3.8 - Beispiel einer procedure page [Bri06]*

- details page:
  Eine details page dient der separaten Darstellung von Berechnungen, die mit anderen pages verknüpft sind. Damit sind z.B. arithmetische Berechnungen realisierbar. Auch Entscheidungstabellen (Kap. 3.2.3) und Regler (Kap. 3.2.4) können innerhalb der details page beschrieben werden.

- parameter page:
  Eine parameter page ermöglicht die Festlegung spezifischer Parameter für Prozeduren oder Regler. Bei einer Rezeptsteuerung ist es beispielsweise möglich, dass eingesetzte Prozeduren oder Regler, bei unterschiedlichen Rezepten, durchaus gleich sind. Es unterscheiden sich dabei jedoch die spezifischen Charakteristiken

der einzelnen Rezepte (z.B. Grenzwerte, spezifische Temperaturen, Reaktionszeiten oder Reglerparameter), die mit Hilfe der parameter page festgelegt werden können.

- comments page:
  Zur weiterführenden Dokumentation dienen comment pages, auf denen Kommentare und Erläuterungen in textueller Form hinterlegt werden können. Zudem unterstützt die ICL Fußnoten, mit deren Hilfe Kommentare zu verschiedenen Elementen einer page unterschieden werden können.

Die verschiedenen Pages eines Systemmodells bilden eine hierarchische Struktur (Abb. 3.9). Ausgangspunkt ist die Operation mit der zugehörigen Operation Page, der die anderen Pages untergeordnet sind. Pages werden mit einem eindeutigen Namen gekennzeichnet. Die hierarchische Strukturierung und die entsprechende Namensgebung erfolgt mit Hilfe der Punktnotation. Dadurch sind die Identifikation und der Zugriff auf die einzelnen Elemente des Hierarchiebaums möglich.

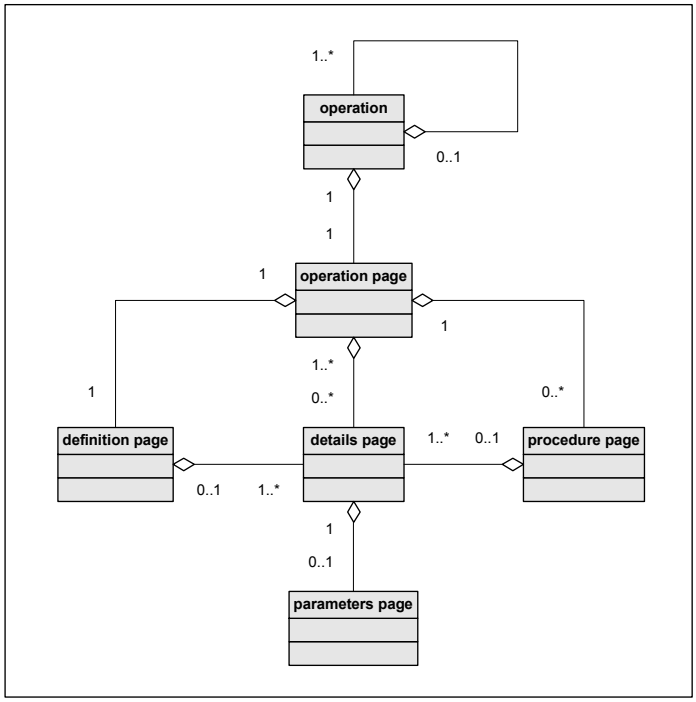



*Abb. 3.9 - Metamodell der Idiomatic Control Language (Darstellung mittels Klassendiagramm)*

### 3.2.2 Sequential Function Chart

Dynamisches Verhalten kann innerhalb der procedure page oder details page mit Hilfe von Sequential Function Charts (SFC) beschrieben werden. Neben einer grafischen Darstellung, welche der Ablaufsprache in der IEC 61131-3 ähnlich ist, wird auch eine textbasierte Form der Sequential Function Charts unterstützt.

Der grundsätzliche Aufbau des SFC ist konform zur Definition in der IEC 61131-3. Der Ablauf wird durch eine Abfolge von Schritten und Transitionen spezifiziert. Im Gegensatz zur IEC sieht die ICL per Definition zunächst keine Transitionsbedingungen vor. Die fehlende Standardisierung lässt jedoch den Freiraum, die Notation an die Bedürfnisse des Anwenders anzupassen und somit auch Bedingungen zur Steuerung der Übergänge zwischen den Schritten zu verwenden. Zur Ablaufsteuerung können normenkonform verzweigte oder parallele Teilabläufe beschrieben werden. In der ICL existieren keine Aktionsbestimmungszeichen, alternativ können jedoch spezielle Konstrukte wie die Loop, Continuous oder State Driven Acitivties für die Realisierung dieser Funktionalität verwendet werden (vergl. Abb. 3.10). Zur Detaillierung einzelner Schritte können untergeordnete details pages erstellt werden.

In der Klammerdarstellung werden textbasierte Aufrufe der procedure oder details pages mit Hilfe grafisch unterschiedlich gestalteter Klammern, wie in Abb. 3.10 dargestellt, strukturiert. Beide Formen des Sequential Function Chart in der ICL sind äquivalent und können, ähnlich wie die grafische und die textuelle Form der Ablaufsprache in der IEC 61131-3, ineinander überführt werden.

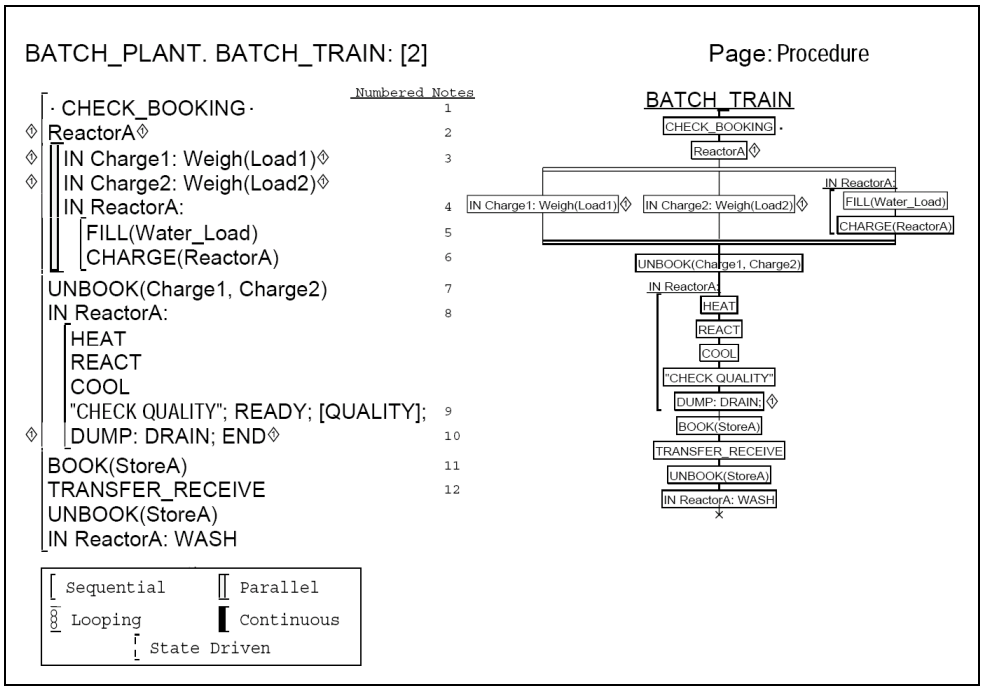
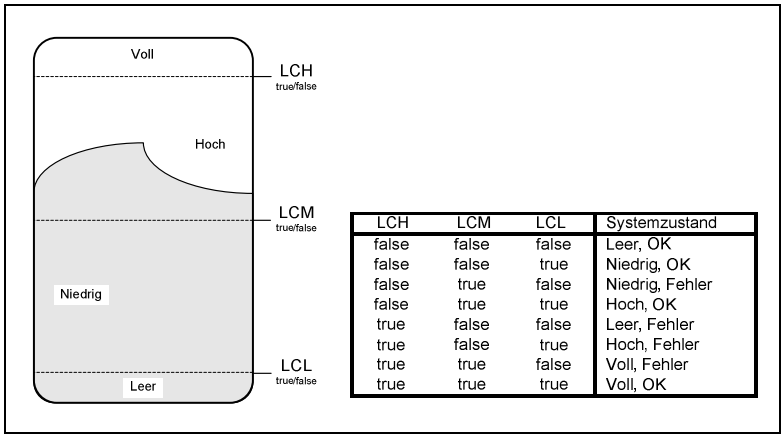



*Abb. 3.10 - Gegenüberstellung der Ablaufbeschreibung mit SFC in textueller und grafischer Form [Bri95]*

### 3.2.3 Truth Tables

Zur Beschreibung einfacher diskreter Verknüpfungen bis hin zu komplexen Logiken stellt die ICL Truth Tables, eine Form von Entscheidungstabellen, bereit. Mit Hilfe der Truth Tables können einzelne Regeln oder ganze Regelwerke definiert werden, die vorhandene Zusammenhänge oder gewünschtes Verhalten beschreiben. Aus der logischen Kombination von binären Variablen (Bedingungen) können so eine oder mehrere Aktionen abgeleitet werden. Als Aktion kann dabei sowohl das Setzen von Variablen auf einen vorgegebenen Wert wie auch der Aufruf bestimmter procedure pages realisiert werden. ICL erlaubt „Don't-Care-Einträge" in Form von leeren Bedingungen, wodurch Regeln zusammengefasst und die Truth Tables konsolidiert werden können. Truth Tables werden innerhalb einer details page beschrieben und können somit einer operation page oder einer definition page zugeordnet sein. Abb. 3.11 zeigt das Beispiel eines Tanks, dessen Füllstand mit Hilfe mehrerer binärer Füllstandssensoren beschrieben wird.




| LCH | LCM | LCL | Systemzustand |
|---|---|---|---|
| false | false | false | Leer, OK |
| false | false | true | Niedrig, OK |
| false | true | false | Niedrig, Fehler |
| false | true | true | Hoch, OK |
| true | false | false | Leer, Fehler |
| true | false | true | Hoch, Fehler |
| true | true | false | Voll, Fehler |
| true | true | true | Voll, OK |

*Abb. 3.11 - Füllstandsberechnung eines Tanks mit Hilfe eines Truth Tables*

### 3.2.4 Idiome

Regelkreise eines Systems werden in der ICL mit Hilfe von Idiomen beschrieben. Idiome fokussieren nicht auf die mathematische Beschreibung von Reglern oder anderen Bausteinen, sondern auf eine höhere, die Funktionalität des Elements betreffenden Abstraktionsebene. Die logische Zielsetzung des Bausteins steht im Vordergrund, was besonders in den frühen Phasen des Engineering nützlich ist, wenn Funktionen bereits beschrieben werden sollen, jedoch die genaue Realisierung oder Parametrierung eines Bausteins noch nicht festgelegt ist. In der Realisierungsphase werden dann die die spezifischen Eigenschaften eines Idioms genau beschrieben. Idiome können sowohl textuell als auch graphisch repräsentiert werden. Beide Darstellungen sind äquivalent.

Syntaktisch setzen sich Idiome aus einer oder mehrerer Eingangsvariablen, dem Verarbeitungsblock und einer oder mehrerer Ausgangsvariablen zusammen. Abb. 3.12 zeigt das Beispiel eines stetigen kontinuierlichen Reglers REGULATE sowohl in der grafischen als auch in der textuellen Form. Der Vergleich mit dem zugehörigen Blockschaltbild verdeutlicht die Reduktion auf die Funktionalität des Idioms.

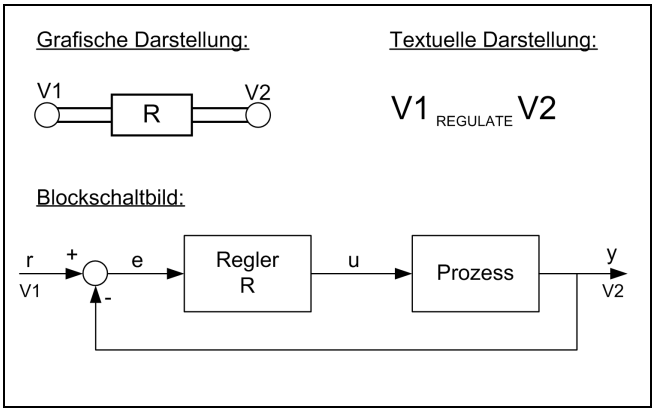



*Abb. 3.12 - Vergleich des Blockschaltbildes eines Reglers mit der grafischen und textuellen Form des Idioms*

Eine Auswahl von Bausteinen ist bereits direkt als Idiom in die ICL implementiert und ist in verschiedene Klassen gegliedert. Basic Idioms beschreiben grundlegende Basisfunktionalitäten. Ihnen können Support Idioms vor- oder nachgeschaltet werden, um die Funktionalität zu erweitern. Komplexere Funktionen werden mit Coordinating Idioms oder Multivariable Idioms realisiert [Bri05].

Das Metamodell der Idiomatic Control Language (Abb. 3.9) lässt sich durch die einzelnen Beschreibungselemente, wie in Abb. 3.13 dargestellt, erweitern und ist in dieser Form die Grundlage einer Systemmodellierung mit Hilfe der ICL. Obwohl die ICL ursprünglich für die Beschreibung verfahrenstechnischer Prozesse entwickelt wurde, ist sie aufgrund ihrer Eigenschaften nicht auf die Anwendung auf kontinuierliche Prozesse beschränkt, sondern eignet sich gleichwohl für die Beschreibung diskreter und hybrider Prozesse. Der bisherige Entwurfscharakter und die fehlende Unterstützung durch bestehende Softwarewerkzeuge erlauben es zudem fehlende Elemente oder Funktionalität per Definition festzulegen und die Notation dadurch zu erweitern.

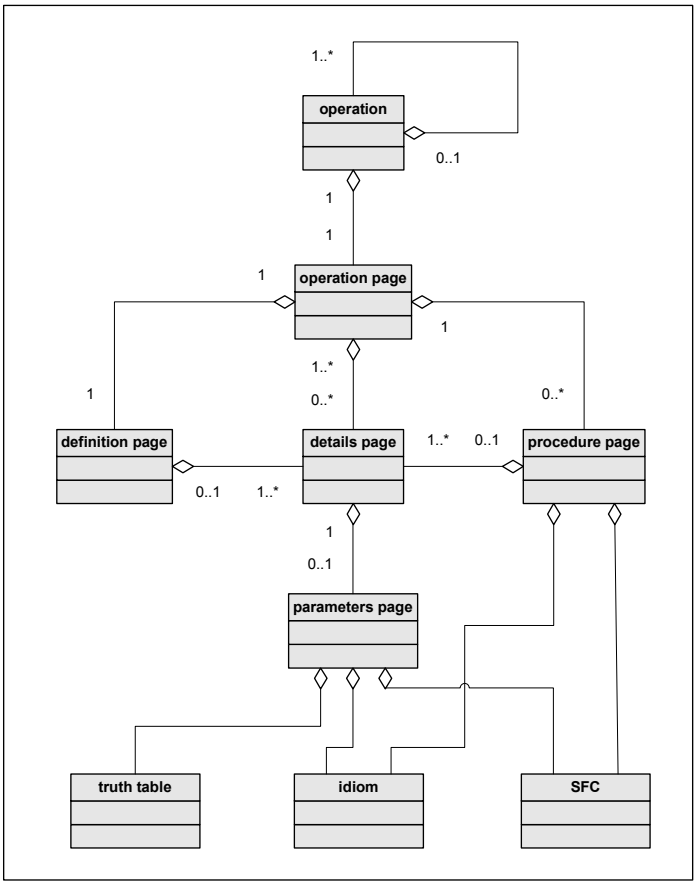



*Abb. 3.13 - Erweitertes Metamodell der Idiomatic Control Language*

## 3.3 Bewertung der Anwendbarkeit der Modellierungsnotationen für den Problembereich

Eine Grundvoraussetzung für die Bewertung des Nutzens einer Modellierungsnotation ist, dass die Notation die notwendigen Mittel, die für die Beschreibung der relevanten Systemeigenschaften benötigt werden, zur Verfügung stellt. Folglich muss vor der Evaluation der Beschreibungsmittel sichergestellt sein, dass sie eine generelle Eignung zur Beschreibung der notwendigen Aspekte besitzen.

Die Bewertung erfolgt im Bezug auf die zu beschreibenden Aspekte des Prozesses sowie die implementierungsrelevanten Aspekte des Automatisierungssystems. Neben der Beschreibung der Struktur und des dynamischen Verhaltens eines Prozesses, ist es erforderlich, die zu Grunde liegende Datenstruktur, Verriegelungen und Aufrufbedingungen sowie Regler und Regelkreise beschreiben zu können. Zudem müssen

die Modellierungsnotationen einen modularen Entwurf unterstützen. Tab. 3.2 stellt den zu beschreibenden Prozesseigenschaften die Realisierung der IEC 61131-3 sowie den in dieser Arbeit untersuchten Modellierungsnotationen gegenüber.

*Tab. 3.2 - Gegenüberstellung der Prozesseigenschaften mit IEC 61131-3, UML und ICL [FrVo06]*

| **Prozess-eigenschaften** | **Realisierung in IEC 61131-3** | **Realisierung in UML** | **Realisierung in ICL** |
|---|---|---|---|
| Struktur | Engineeringtool | Klassendiagramm, Objektdiagramm | Seitenkonzept |
| Variablen, Merker | Symboltabellen | Attribute | Variablendeklaration |
| Verhalten, Sequentielle Abläufe | Programmcode | Zustandsdiagramm, Aktivitätsdiagramm, Sequenzdiagramm | Sequential Function Chart (SFC) |
| Verriegelungen, Bedingungen | Programmcode | nicht festgelegt, Realisierung in Transitionen | Entscheidungstabellen |
| Regler, Regelkreise | Funktionsblöcke | Stereotypen | Idiome |
| Modularer Entwurf | Funktionen, Funktionsblöcke | Objektorientierung | Operationen, Seitenkonzept |

Beide Modellierungsnotationen besitzen die notwendigen Mittel, die zur Beschreibung der programmierungsrelevanten Prozesseigenschaften benötigt werden und sind somit für die zu untersuchende Aufgabenstellung generell geeignet.

# 4 Grundlagen für die Evaluation

Zentrale Frage der vorliegenden Arbeit ist es, wie der Engineeringprozess automatisierungstechnischer Systeme speziell für den Bereich der Steuerungsprogrammierung sinnvoll unterstützt werden kann. Der Mensch nimmt im Engineeringprozess eine zentrale Rolle ein, deshalb werden in diesem Kapitel zuerst die psychologischen Grundlagen der Informationsverarbeitung und der Wissensrepräsentation beim Menschen diskutiert. Im Anschluss daran wird ein Überblick über relevante empirische Untersuchungen im Bereich der Softwareentwicklung gegeben. Anschließend werden auf Basis dieser Ausführungen Fragestellungen abgeleitet, die im Rahmen der Evaluation untersucht werden, um den Engineeringprozess automatisierungstechnischer Systeme sinnvoll zu unterstützen.

## 4.1 Psychologische Grundlagen

### 4.1.1 Mentale Modelle

Seit jeher versucht die Psychologie das Denken und Handeln des Menschen zu erklären. Mit der so genannten „kognitiven Wende“ hat sich in den sechziger Jahren im psychologischen Umfeld ein wichtiger Paradigmenwechsel vollzogen [Sca87]. Bis zu diesem Zeitpunkt herrschte die Sichtweise des Behaviorismus mit dem Schwerpunkt auf offenes beobachtbares Verhalten vor. Innere, nicht beobachtbare Vorgänge wurden dabei vernachlässigt und meist als nicht wissenschaftlich abgelehnt. Mit der kognitiven Wende fand die Kognitionspsychologie Einzug, die den Menschen als „*… ein System auffasst, das aktiv Informationen aus der Umwelt aufnimmt, speichert, manipuliert und zum Teil zielgerichtet wieder verwendet*“ [Dut94]. Damit rückten auch nicht direkt beobachtbare Verhaltensweisen des Menschen in den Fokus der Forschung. Es wurde nun auch verstärkt untersucht, wie Informationen im menschlichen Gehirn gespeichert oder verarbeitet werden.

Teil dieser Theorie sind so genannte mentale Modelle, welche als hypothetische Konstrukte der Erklärung menschlicher Informationsverarbeitung dienen. Mentale Modelle sind die Grundlage für die Wahrnehmung und das Verständnis komplexer Vorgänge oder Zusammenhänge beim Menschen. Die vom Menschen aufgenommenen Reize werden, zusammen mit seinen Erfahrungen, Erinnerungen sowie individuellen Denkprozessen, dazu genutzt, um im Gehirn eine Repräsentation der für die Beschreibung der Realität relevanten Aspekte und ihrer Wechselwirkungen zu schaffen. Mentale Modelle sind von den

Zielvorstellungen und kognitiven Voraussetzungen des jeweiligen Individuums und von der zu bewältigenden Aufgabe abhängig und deshalb als subjektive Gebilde zu verstehen, die das Verhalten des Individuums determinieren. Mentale Modelle sind nicht statisch sondern hoch dynamisch und somit einer ständigen Adaption unterlegen [Nor83]. Weidenmann [Wei86] beschreibt diese Dynamik mit vier Kernannahmen:

- Transformationsannahme:
  Die Reize aus der Umwelt werden bei der Wahrnehmung in eine interne Repräsentation (mentale Codes) gewandelt ohne die eine Informationsverarbeitung nicht möglich ist.

- Elaborationsannahme:
  Wahrgenommenes wird nicht nur additiv hinzugefügt, sondern durch vorhandene Modellinformationen interpretiert und in das bestehende Modell integriert. Das Modell wird somit erweitert.

- Konstruktionsannahme:
  Die interne Repräsentation stellt eine aktive Rekonstruktion der wahrgenommenen Strukturen dar und wird wesentlich durch die Ziele und somit von der zu lösenden Aufgabe des Individuums beeinflusst.

- Systemannahme:
  Es besteht eine zyklische Rückkopplung zwischen der Wahrnehmung, dem Gedächtnis und der Informationssuche.

Diese vier Annahmen zeigen, dass der Mensch durch ständige Anpassung und Rückkopplung versucht, die Qualität seiner mentalen Modelle zu steigern. Da mentale Modelle jedoch auch einem natürlichen Alterungsprozess unterliegen, kann bereits vorhandenes Wissen, durch den Prozess des Vergessens, teilweise wieder verloren gehen, wodurch die Qualität des Modells sinkt.

Mentale Modelle dienen in der Psychologie und ihren Anwendungsfeldern zur Erklärung von Verhaltensweisen. Mentale Modelle sind schwer zu verbalisieren und somit schwer zu messen, was jedoch eine zwingende Voraussetzung darstellt, um durch entsprechende Interventionen (z.B. durch Präsentationsform) das Lernen zu verbessern. Dennoch werden mentale Modelle insbesondere in der Arbeitspsychologie zur Erklärung menschlicher Lern- und Arbeitsleistung verwendet. So wird versucht die Arbeitsleistung durch didaktisch-methodische Maßnahmen in der Aus- und Weiterbildung oder das zur Verfügung stellen geeigneter Hilfsmittel zu steigern [Dut94]. Dadurch soll eine höhere Arbeitssicherheit erreicht oder die Belastung bei der Bewältigung einer Aufgabe reduziert werden.

In der Kognitionspsychologie unterscheidet man zwischen verständnisorientierten und handlungsorientierten Modellen [Dut94]. Verständnisorientierte Modelle beschreiben hauptsächlich die Struktur oder den Aufbau eines Systems. Sie beschränken sich darauf, Faktenwissen zu beschreiben oder zu vermitteln und eignen sich nur bedingt oder mit erhöhtem Aufwand dazu, Handlungswissen für die Durchführung einer bestimmten Aufgabe abzuleiten. Handlungsorientierte Modelle hingegen beschreiben explizit bestimmte Ablaufsequenzen, welche für die Erledigung einer Aufgabe durchlaufen werden müssen. Sie bestehen beispielsweise aus Ablaufdiagrammen oder Wenn-Dann-Beziehungen. Verständnisorientierte und handlungsorientierte Modelle können miteinander kombiniert werden, um einander bei der Durchführung einer Arbeitsaufgabe zu ergänzen.

Darüber hinaus dienen mentale Modelle dazu Sachverhalte vor dem „inneren Auge“ dynamisch zu simulieren [Wei02]. Durch gedankliches Durchspielen von Handlungen und resultierenden Ereignisfolgen können somit auch neue, bislang unbekannte Systemzustände und Zusammenhänge entwickelt werden. Das Ergebnis einer Simulation und damit auch die Struktur und die Eigenschaften des zugrunde liegenden mentalen Modells haben dadurch direkten Einfluss auf das Handeln eines Menschen. Je genauer das Modell die Realität und die Zusammenhänge beschreibt und je genauer die Eingaben und die Randbedingungen für eine Simulation definiert werden, desto exakter wird das Ergebnis der Simulation die Ereignisse der Realität wiedergeben.

Mentale Modelle werden beim Menschen im Gedächtnis gespeichert. Dabei wird zwischen dem Arbeitsgedächtnis (früher auch als Kurzzeitgedächtnis bezeichnet) und dem Langzeitgedächtnis unterschieden. Das Arbeitsgedächtnis stellt einen Speicher dar, in dem eine kleine Menge von Informationen derart vorgehalten wird, dass auf sie jederzeit zugegriffen werden kann. Das Arbeitsgedächtnis wird genutzt, um die uns unmittelbar umgebende Umwelt zu verstehen und eine mentale Repräsentation dieser herzustellen. Nach Miller [Mil56] verfügt es über eine begrenzte Kapazität von 7 +/- 2 Informationseinheiten, so genannte „chunks“, die nur für eine begrenzte Zeit im Arbeitsgedächtnis vorgehalten werden und dort für eine Weiterverarbeitung zur Verfügung stehen. Für eine längere Speicherung müssen Informationen in das Langzeitgedächtnis transferiert und dort gespeichert werden, da sie ansonsten, durch das Eintreffen neuer Informationen, ersetzt und somit wieder vergessen werden. Im Gegensatz zum Arbeitsgedächtnis ist das Langzeitgedächtnis in seiner Kapazität nicht begrenzt, jedoch müssen die Informationen vor ihrer Weiterverarbeitung wieder zurück in das Arbeitsgedächtnis transferiert werden. Die generelle Fähigkeit des Rücktransfers sowie die dabei erzielte Geschwindigkeit variiert in Abhängigkeit verschiedener Parameter und wird auch als kognitive Verfügbarkeit von Informationen bezeichnet [Nit08]. Aktualität,

Anschaulichkeit erhöhen die Verfügbarkeit genauso wie eine höhere Frequenz des Abrufens. Der Rücktransfer ist bei Automatismen relativ schnell, bei weniger häufig benutzten Informationen langsam. Auch das Langzeitgedächtnis unterliegt dem Prozess des Vergessens, der hier, jedoch nicht wie beim Arbeitsgedächtnis durch die Größe des Speichers, sondern vielmehr durch die Frequenz der Benutzung und die Bewertung der persönlichen Relevanz der Information beeinflusst ist.

Der Aufbau und die Weiterentwicklung mentaler Modelle erfolgt in einem Lernprozess, der auf zwei unterschiedliche Arten erfolgen kann. Dieser Lernprozess kann dabei entweder extern oder intern gesteuert sein [Dut94].

Als extern gesteuertes Lernen wird ein von außen gesteuerter Wissensaneignungsprozess bezeichnet. Bekanntestes Beispiel für solche Prozesse findet man in Verbindung mit dem schulischen Lernen [GrPa78], wo der Lehrende die Art und die Darstellungsform der Information sowie den zeitlichen Ablauf des Lernprozesses bestimmt. Das Lernverhalten ist dabei sehr stark vom Inhalt und der Darstellungsform der extern bereitgestellten Information abhängig.

Intern gesteuertes Lernen ist durch ein hohes Maß an Selbststeuerung gekennzeichnet. Durch Analyse des zu betrachtenden Ausschnitts der Realität und den Vergleich mit bereits Bekanntem, erarbeitet sich das Individuum selbständig ein Modell. Anschließend erfolgt, durch den Vergleich des Ergebnisses der kognitiven Simulation mit dem Verhalten des Originalsystems, eine Validierung des mentalen Modells, welche die Basis für eine Erweiterung oder Modifizierung des Modells ist. Dieses Vorgehen wird als explorierendes Handeln oder explorierendes Lernen bezeichnet.

Unabhängig davon, ob der Aufbau eines mentalen Modells durch externes oder internes Lernen gesteuert wird, hat die Darstellung der zur Verfügung stehenden Information einen Einfluss auf die Aufnahmefähigkeit und die Erinnerungsfähigkeit. Nach Paivio [Pai86] werden eingehende Informationen dual kodiert. Dabei werden grafische und verbale Informationen auf unterschiedlichen Kanälen bearbeitet und bilden im Arbeitsgedächtnis unterschiedliche Repräsentationen der Information. Die verbale Darstellung konzentriert sich dabei auf konzeptionelle Informationen, wie die Zugehörigkeit zu Kategorien, die grafische Darstellung konzentriert sich auf physikalische Attribute, wie die Farbe, Form oder Anordnung von Objekten. Beide Repräsentationen werden anschließend dazu verwendet die Information als Wissen im Langzeitgedächtnis zu organisieren. Bei diesem Prozess werden bidirektionale Verbindungen zwischen der verbalen und der grafischen Repräsentation gebildet. Im Bezug auf die Speicherung von grafischen und verbalen Informationen konnte Paivio belegen, dass die Präsentation von Bild und zugehörigem

Wort, insbesondere bei kurzen Präsentationszeiten, zu besseren Behaltensleistungen führen als nur die Bild- oder gar nur die Wort-Präsentation. Darüber hinaus konnte Paivio nachweisen, dass der Mensch versucht, sich zu grafisch repräsentierten Informationen eine verbale Repräsentation zu schaffen, was bei der Präsentation eines Bildes dann zur dualen Kodierung und somit zu besseren Behaltensleistungen führt. Voraussetzung dafür ist jedoch, dass die zur Verfügung stehende Zeit nicht zu kurz ist und die zugehörige verbale Beschreibung bereits bekannt ist.

Der Ansatz von Paivio wird durch die kognitive Theorie des multimedialen Lernens [May01], die durch eine Vielzahl von Untersuchungen überprüft wurde, bestätigt und elaboriert. Neben dem Prinzip der dualen Codierung beinhaltet diese Theorie das Prinzip der räumlichen Nähe, die besagt, dass die räumlich benachbarte Darstellung textueller und bildlicher Informationen den Wissenserwerb mehr fördert, als eine getrennte Präsentation von Texten und Bildern. Daraus folgt, dass zusammengehörende Worte und Grafiken nahe beieinander platziert werden sollten.

Mit dem Einfluss der Motivation betrachten Astleitner et al. [APW06] einen in der Theorie des multimedialen Lernens bislang nur wenig untersuchten Aspekt. Aufbauend auf dem ARCS-Modell (Attention, Relevance, Confidence and Satisfaction) leiteten Astleitner et al. theoretisch ab, dass die Fähigkeit der Informationsverarbeitung im Arbeitsgedächtnis mit der Verbesserung motivationsgerichteter Aspekte wie Aufmerksamkeit, Erfolgszuversicht oder Zufriedenheit ansteigt. Eine experimentelle Evaluation dieser theoretisch abgeleiteten Zusammenhänge steht noch aus.

### 4.1.2 Mentale Modelle in der Softwareentwicklung

Auch im Bereich der Softwareentwicklung bilden mentale Modelle die Grundlage für die Entwicklung von Softwaresystemen. Sie werden dabei als eine geistige Repräsentation und das Verständnis eines physikalischen Hardwaresystems, eines Softwaresystems oder eines hybriden Systems verstanden [All97]. Das mentale Modell beschreibt also das Verständnis, welche Teilsysteme oder Elemente das System beinhaltet, wie es funktioniert und warum es auf diese Art funktioniert. Es wird vom Anwender oder Programmierer einerseits dazu verwendet, das System zu analysieren, sein Verhalten nachzuvollziehen und es zu erklären. Andererseits dient es auch der mentalen Simulation, um zukünftiges Verhalten auf Basis gedanklich vorgenommener Eingriffe in das System vorhersagen zu können.

Bezogen auf das hier diskutierte Aufgabengebiet, die Erstellung von Steuerungsprogrammen, wird in verschiedenen Schritten auf unterschiedlichen Ebenen auf mentale Modelle zurückgegriffen, die das zu realisierende System aus verschiedenen

Sichten und unter Berücksichtigung verschiedener Aspekte beschreiben. Carroll [Car88] beschreibt die notwendigen Sichten auf das System wie folgt:

- Aufgabenstellung (task-knowledge):
  Die Aufgabenstellung beschreibt das Ziel des zu realisierenden Steuerungsprogramms. Unabhängig von der späteren Realisierung werden hier die Anforderungen an das Steuerungsprogramm auf einem sehr abstrakten Level beschrieben.

- Systemarchitektur (system architecture-knowledge):
  Die Systemarchitektur beschreibt den Aufbau und die Funktionsweise des Systems. Dabei liegt der Fokus, neben der Erklärung der inneren Struktur, auch auf inneren kausalen Zusammenhängen, also der Beschreibung von wechselseitigen Beziehungen und Ereignisabfolgen.

- Schnittstellen (system interface-knowledge):
  Die Schnittstellen beschreiben, welche Ein- und Ausgangssignale oder Nachrichten zur Verfügung stehen, um mit dem System interagieren zu können. Dabei ist neben der Festlegung der Bedeutung eines Signals oder einer Nachricht, auch die Definition des Datenformats notwendig.

Neben seinem Wissen über den Prozess benötigt der Programmierer auch Wissen über die eingesetzte Zielumgebung. Da sich die verschiedenen Zielumgebungen in ihren Eigenschaften durchaus entscheidend voneinander unterscheiden können, muss dies bereits vor der Implementierung, in der Designphase des Programms, berücksichtigt werden. Eine Speicherprogrammierbare Steuerung mit zugehöriger IEC-61131-3-konformer Programmierumgebung hat, verglichen mit einem Mikrocontroller und einer objektorientierten Programmiersprache, besondere abweichende Eigenschaften (z.B. zyklische Abarbeitung der Tasks) und Restriktionen, die dem Programmierer bewusst sein müssen. Bei der eigentlichen Programmierung müssen die Informationen aus beiden mentalen Modellen, d.h. dem Modell der Zielumgebung und dem Modell der zu erstellenden Software, miteinander verknüpft und anschließend für die Implementierung verwendet werden.

Dieser Einfluss wird häufig mit dem Begriff der kognitiven Ergonomie bezeichnet [Lon89]. Kognitive Ergonomie bezeichnet die Beziehungen zwischen einem Stimulus und darauf folgenden Reaktionen im Zusammenhang mit einer eher auf mentalen als auf physikalischen Tätigkeiten beruhenden Aufgabe. Ein solcher Stimulus kann beispielsweise ein Software-Werkzeug, eine zur Verfügung gestellte Information, eine Dokumentation

aber auch zugehörige Hardware, wie ein Benutzerterminal, sein. Im Bereich der Softwareentwicklung findet der Ansatz der kognitiven Ergonomie vor allem bei der Untersuchung von geeigneten Unterstützungen in der Phase der Anforderungsanalyse und des Designs, der Bewertung von Spezifikationsformaten sowie der Bewertung von Programmiersprachen seine Anwendung.

Wie in Kapitel 4.1.1 dargestellt, können mentale Modelle individuell stark variieren. Daraus resultieren verschiedene Probleme im Bezug auf kooperatives Arbeiten, die bei der Entwicklung von Steuerungssoftware besonderen Einfluss haben. Häufig müssen Softwareentwickler auf Informationen zurückgreifen, die aus anderen Fachdisziplinen stammen. In einer Domäne herrschen häufig einheitliche Gedankenmodelle vor, zwischen den einzelnen Gewerken können sich diese jedoch stark unterscheiden. Dadurch kann es zu Fehlinterpretation der fachfremden Information kommen, die im weiteren Verlauf zu Fehlern in der erstellten Software führen. Es muss also sichergestellt werden, dass die mentalen Modelle von Beteiligten unterschiedlicher Domänen angeglichen werden, um so Fehler zu reduzieren und die Qualität der Software zu steigern.

Eine weitere Problematik ergibt sich aus der Tatsache, dass sowohl der Entwickler als auch der Nutzer einer Software ein mentales Modell des Systems und der Aufgabe gebildet haben. Die Modelle von Entwickler und Nutzer können sich dabei deutlich unterscheiden [GaCh87]. Daraus resultieren Schwierigkeiten bei der Anwendung solcher Tools. Durch eine Angleichung der Modelle von Entwickler und Nutzer bzw. der Berücksichtigung des anwenderseitigen mentalen Modells bei der Entwicklung von Softwaretools besteht die Möglichkeit solche Probleme zu minimieren und die Usability für den Anwender zu erhöhen.

### 4.1.3 Einflussfaktoren für die Leistungsfähigkeit von Programmierern

Seit den fünfziger Jahren studieren Psychologen verhaltensbezogene Einflüsse bei der Erstellung von Computerprogrammen. Eine große Anzahl von Untersuchungen und Experimenten in diesem Umfeld wurden durchgeführt, um den Beweis zu erbringen, dass bestimmte Aspekte das Design und die Erstellung von Computerprogrammen positiv oder negativ beeinflussen können. Die Arbeiten konzentrierten sich dabei anfangs auf prozedurale und funktionale Programmierung. Später wurde auch der Einfluss im Bezug auf objektorientierte Programmieransätze untersucht. Das Studium relevanter Literatur macht auch deutlich, dass bis Anfang der 90er Jahre sehr viel Aufwand getrieben wurde, um diese Aspekte zu erforschen. Seit Mitte der 90er Jahre zeigt sich auf diesem Gebiet deutlich weniger Aktivität.

Eine der ersten Untersuchungen fand bei IBM statt, wo versucht wurde, einen Zusammenhang zwischen Leistungstests von Programmierern und ihrer Performance im Beruf zu finden. Die Korrelation zwischen beiden Parametern war jedoch sehr gering [RHP+66]. In der Nachbetrachtung sieht Curtis [Cur88] als Grund dafür nicht etwa die Tatsache, dass diese Theorien im Grundsatz falsch waren, sondern dass die Untersuchungen den Einfluss weiterer psychologischer Faktoren vernachlässigten. Des Weiteren wurde häufig versucht, einen Zusammenhang zwischen Aspekten unterschiedlicher Ebenen herzustellen. So wurde beispielsweise versucht, Tests, welche lediglich allgemeine Programmierkenntnisse ermittelten, zu nutzen, um gezielt Programmierfähigkeiten, wie z.B. die Fähigkeit zu debuggen, vorher zu sagen. Gründe wie diese führten zu einer detaillierten Betrachtung der Einflüsse und resultierten in einem Modell, welches die Einflüsse auf die Programmierleistung beschreibt. Es wurden fünf Klassen identifiziert, in welche die beeinflussenden Faktoren unterteilt werden können: Individuelle Charakteristiken des Programmierers, Gruppenverhalten, Organisationsstruktur des Progammierprojekts, kognitive Ergonomie sowie kognitionswissenschaftliche Einflüsse.

Der Einfluss individueller Charakteristiken auf die Leistungsfähigkeit eines Software-Entwicklers wurde in vielen Studien untersucht [Cur88]. Als wichtige Einflussfaktoren, die bei der Untersuchung der Leistungsfähigkeit von Programmierern berücksichtigt werden sollten, fasst Curtis wie in Abb. 4.1 dargestellt sechs Faktoren zusammen. Es handelt sich hierbei um die intellektuelle Begabung, die individuelle Wissensbasis, die kognitiven Fähigkeiten, die Motivation, die Persönlichkeitsmerkmale und die Verhaltensmerkmale des Programmierers.

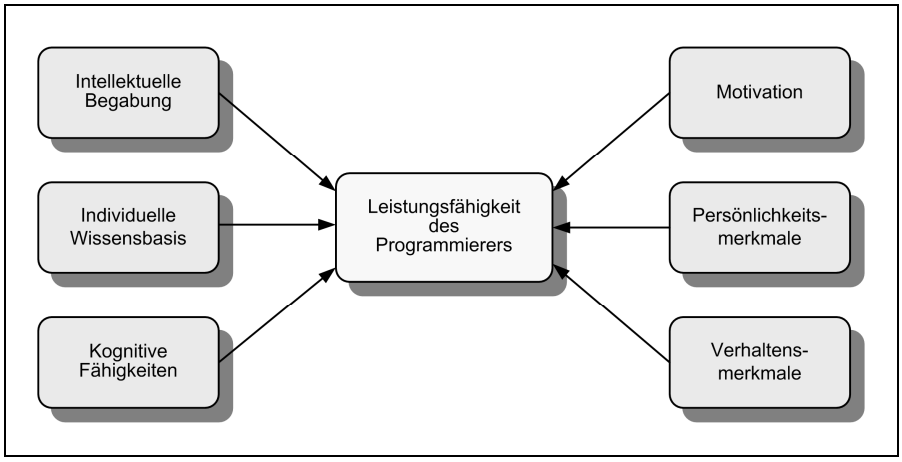



*Abb. 4.1 - Einfluss individueller Charakteristiken auf die Leistungsfähigkeit eines Programmierers in Anlehnung an Curtis [Cur88]*

Über den Grad, zu welchem einer dieser Faktoren die Leistung des Programmierers beeinflusst, macht Curtis keine Aussagen, auch in der weiterführenden Fachliteratur sind kaum empirisch belegte Bewertungsschemata zu finden. Zwar exstieren Ansätze, die dazu gedacht sind, die Leistung von Softwareentwicklern durch die Messung und Bewertung bestimmter Kriterien vorher zu sagen [ArHo74], der Erfolg solcher Methoden ist jedoch nicht nachgewiesen. Auch der Grad der gegenseitigen Abhängigkeit von individuellen Charakteristiken wird nicht bewertet. Dies ist in der Tatsache begründet, dass meist nur eine begrenzte Anzahl der individuellen Einflussfaktoren in der gleichen Studie untersucht werden konnten.

Neben den persönlichen Eigenschaften der Programmierer wird häufig auch der Einfluss von Qualifikation und Erfahrung des Programmierers als Einflussgröße auf die Leistungsfähigkeit bei der Erstellung von Software genannt. Diese sind jedoch nur schwer von den bereits genannten Faktoren zu trennen, was am Beispiel der Fähigkeit des objektorientierten Softwareentwurfs gezeigt werden kann: Bei der Objektorientierung handelt es sich um ein Modellierungs- oder Programmierparadigma, welches den kognitiven Fähigkeiten zugeordnet werden kann. Die Fähigkeit objektorientiert zu denken oder zu modellieren, wird jedoch maßgeblich auch davon beeinflusst, welche Programmiersprachen ein Programmierer beherrscht. Programmierkenntnisse werden dabei häufig der Qualifikation und der Erfahrung zugeordnet. In dem Modell nach Curtis können diese beiden Punkte unter dem Aspekt der individuellen Wissensbasis eingeordnet werden und beeinflussen somit die Leistungsfähigkeit. Durch die Weiterqualifizierung sowie Erfahrungen aus bereits durchgeführten Softwareprojekten kann die Wissensbasis somit vergrößert werden. Damit wird auch beeinflusst, wie ein Programmierer eine Aufgabe löst, welche Fehler er dabei macht und welchen Aufwand er dafür tätigen muss.

Von den dargestellten Einflussfaktoren sind drei im Rahmen dieser Arbeit besonders hervor zu heben, da sie durch die Verwendung einer geeigneten Systemmodellierung positiv beeinflussbar scheinen: die individuelle Wissensbasis, die kognitiven Fähigkeiten und die Motivation.

Unter der individuellen Wissensbasis wird sowohl das Wissen über das zu automatisierende System als auch Wissen über die zur Verfügung stehende Zielumgebung und die geplante Realisierung zusammengefasst. Durch die Verwendung einer geeigneten Modellierungsnotation, die bei der Analyse des zu automatisierenden Systems benutzt wird, wird der Programmierer beim Aufbau seiner Wissensbasis unterstützt. Dabei hat die Notation in dieser Hinsicht besonders die Aufgabe, die große Menge an Informationen, die der Programmierer in diesem Stadium der Arbeitsaufgabe aufnehmen und speichern muss,

so festzuhalten, zu organisieren und zu dokumentieren, dass sie später bei der Programmierung des Systems möglichst schnell und fehlerfrei abgerufen und auf die Erstellung des Steuerungsprogramms angewendet werden kann.

Der Einfluss der kognitiven Fähigkeiten hat in den letzten Jahren deutlich an Aufmerksamkeit im Bereich der Forschung erfahren. Bestimmte kognitive Faktoren wie die Problemlösungsfähigkeit, abstraktes und logisches Denken, die Fähigkeit Wissen im Problembereich umzusetzen und anzuwenden, sowie der kognitive Stil werden als mögliche Einflussfaktoren auf die Programmierleistung vermutet. Ihr Einfluss konnte bisher jedoch nur bedingt empirisch nachgewiesen werden [BeRe05b]. Dennoch scheint es möglich zu sein, den Anwender durch die Verwendung eines geeigneten Beschreibungsmittels bei kognitiven Prozessen zu unterstützen. So kann der Vorgang der Modellierung bei der Problemanalyse und Problemlösung helfen. Die Anwendung eines abstrakten Beschreibungsmittels kann das abstrakte Betrachten des Prozesses fördern. Auch die Übertragung des Modells auf die Zielumgebung kann unterstützt werden, sofern die Analogien zwischen dem Modell und dem Zielsystem erkennbar sind. In der Addition dieser die kognitiven Vorgänge unterstützenden Aspekte, wird somit eine Steigerung der Leistungsfähigkeit bei der Programmierung erwartet.

In der Psychologie wird der Begriff der Motivation als Sammelbezeichnung von Prozessen und Effekten gebraucht. Ihr gemeinsamer Kern besteht darin, dass eine Person ihr Verhalten auf Basis der zu erwartenden Folgen auswählt und es hinsichtlich Richtung und Energieaufwand steuert. Im Hinblick auf die Leistungsfähigkeit bei der Programmerstellung bedeutet dies, dass ein Programmierer seine Vorgehensweise so auswählt, dass eine bestmögliche Programmqualität, in möglichst kurzer Zeit und mit möglichst geringem Aufwand, erzielt werden kann.

Im Allgemeinen wird zwischen intrinsischer und extrinsischer Motivation unterschieden. Intrinsische Motivation bezeichnet dabei das Handeln aus eigenem Antrieb, d.h. aufgrund eigener Ziele, wie die Aussicht auf eigene Befriedigung oder Genugtuung. Extrinsische Motivation wird hingegen durch externe Einflüsse wie die Aussicht auf eine Gegenleistung (z.B. Bezahlung) oder die Vermeidung von Nachteilen (z.B. schlechte Bewertung) hervorgerufen. So konnten Bergin und Reilly [BeRe05a] belegen, dass intrinsische Motivation deutlich mehr dazu beiträgt die Programmierleistung zu erhöhen, als extrinsische Motivation. Angewendet auf die vorliegende Arbeit bedeutet das, dass die Modellierung des Prozesses für den Anwender nicht als zusätzliche Arbeit oder Belastung empfunden werden sollte. Ziel muss es vielmehr sein, dass ihm der Nutzen, den er durch

die Verwendung einer geeigneten Modellierungsnotation im Engineering erzielt, verdeutlicht wird.

Es wird deshalb der Ansatz verfolgt, durch die Verwendung einer geeigneten Modellierungsnotation die wahrgenommene Komplexität der Aufgabe zu reduzieren und dadurch die intrinsische Motivation im laufenden Programmerstellungsprozess und somit die Leistungsfähigkeit zu steigern. Neben der kurzfristigen Steigerung der Motivation im Projekt muss es jedoch auch ein Ziel sein, die Bereitschaft der Anwendung einer derartigen Vorgehensweise nachhaltig zu steigern. Durch einen Erfolg bei der erstmaligen Anwendung der Modellierungsnotation steigt auch die Motivation, bei einer ähnlichen Aufgabe erneut die Modellierung des Zielsystems vorzunehmen.

Die bisher genannten Einflussgrößen beschreiben ausschließlich Eigenschaften und Fähigkeiten einzelner Individuen. In der Fachliteratur wird darüber hinaus auch der Einfluss von Beziehungen zwischen den einzelnen Programmierern unter dem Begriff des „Gruppenverhaltens“ beschrieben. Das Ergebnis eines Programmierprojekts unterscheidet sich abhängig davon, ob eine Einzelperson oder eine Gruppe von Programmierern an dem Projekt arbeitet. Nach Shaw [Sha76] erarbeiten Gruppen eine höhere Anzahl sowie qualitativ bessere Lösungen für dasselbe Problem. Auch das Lernverhalten von Gruppen ist besser einzustufen. Allerdings wird in Gruppen mehr Zeit für die Lösung eines Problems benötigt als bei Einzelpersonen. Dies wird durch den erhöhten Koordinierungsaufwand innerhalb der Gruppe erklärt.

Auch auf einer höheren Hierarchieebene existieren Einflüsse auf das Ergebnis eines Programmierprojektes. Unter dem Begriff der „Organisationsstruktur des Programmierprojektes“ werden diese Aspekte zusammengefasst. Dazu zählen unter anderem die Einbettung in überlagerte Organisationsstrukturen, die örtliche Trennung der am Projekt beteiligten Einheiten oder der Grad der Interaktion mit dem Endkunden. Ansätze einen solchen Zusammenhang zu beschreiben, basieren jedoch meist mehr auf persönlichen Erfahrungen als auf empirischen Studien [Cur88]. Sie werden im Rahmen dieser Arbeit nicht weiter untersucht.

## 4.2 Themenverwandte empirische Untersuchungen im Softwareengineering

Mit der steigenden industriellen Nutzung von Software ist die Frage, wie man die Softwareerstellung sinnvoll unterstützen kann, immer weiter in den Fokus von wissenschaftlichen Untersuchungen gerückt. Die Themenschwerpunkte der Forschung haben sich dabei, auch aufgrund der Weiterentwicklung der Rechnertechnik- und

Leistungsfähigkeit sowie der Einführung neuer Programmiersprachen, ständig verändert. Dabei konzentrierten sich die meisten Arbeiten auf die Entwicklung konventioneller Software, aus dem Bereich der Softwareentwicklung in der Automatisierungstechnik sind kaum empirische Untersuchungen bekannt. Die Übertragung der Ergebnisse auf den Bereich der Steuerungstechnik ist deshalb nur bedingt möglich. Dennoch bieten die genannten Untersuchungen Hinweise darauf, welche Aspekte bei der Evaluation der Anwendbarkeit von Modellierungsnotationen und deren Einfluss auf die Qualität der Steuerungsprogrammierung gezielt untersucht werden sollten.

Viele Untersuchungen aus den 70'er oder 80'er Jahren des zwanzigsten Jahrhunderts beschäftigten sich mit der Eignung verschiedener Darstellungsformate im Rahmen der Softwareentwicklung. Carol, Thomas und Malhotra [CTM80] konnten nachweisen, dass es für Menschen schwieriger ist ein Problem mit zeitlichen Relationen (zeitliches Problem) zu lösen als eines mit räumlichen Relationen (Layout-Problem). Das Layout-Problem impliziert von seinen Eigenschaften her bereits eine grafische Repräsentation zur Lösung und wurde als leichter lösbar eingestuft. Im Vergleich dazu wurde bei dem zeitlichen Problem meist eine verbale Beschreibung gewählt, einige Probanden konnten das Problem gar nicht lösen. Wurde hingegen eine grafische Unterstützung zur Lösung angeboten, wurde das Problem als leichter lösbar eingestuft.

Häufig wurde der Einsatz von Flussdiagrammen zur Informationsbereitstellung untersucht. So verglich Blaiwes [Bla74] die Eignung von Flussdiagrammen zur Repräsentation von Anweisungen für die Bedienung einer Kommunikationskonsole mit Anweisungen, die in kurzen Sätzen geschrieben waren. Im Bezug auf die Fehlerrate bei Eingaben konnte er belegen, dass zwischen diesen Formaten bei einfachen Problemen kein Unterschied bestand. Bei komplexeren Problemen hingegen wurde eine niedrigere Fehlerrate erzielt, wenn ein Flowchart verwendet wurde. Die Ergebnisse konnten durch Mayer [May76] bestätigt werden.

Einen ähnlichen Zusammenhang konnten Cunniff und Taylor im Bezug auf die Nachvollziehbarkeit von Programmen nachweisen. Sie untersuchten den Einfluss, den eine grafische Repräsentation einer Programmiersprache auf die Nachvollziehbarkeit des Programms ausübt [CuTa87]. Sie konnten, besonders bei unerfahrenen Programmierern, einen deutlichen Unterschied im Vergleich zu einer textuellen Darstellung belegen. Programmierer konnten Fragen zu vorgegebenen Programmsegmenten schneller und mit weniger Fehlern beantworten, wenn diese grafisch dargestellt wurden. Einen ähnlichen Effekt konnten Curtis et al. für erfahrene Programmierer nacheisen [CSK+89]. Im Vergleich von natürlichsprachlichem Text, Constrained Language und Ideogrammen

konnte ein Text die Teilnehmer am wenigsten dabei unterstützen, ein Programm nachzuvollziehen oder selbstständig Programmcode aus einer im entsprechenden Format gegebenen Spezifikation zu erstellen.

Bei allen genannten Untersuchungen ist anzumerken, dass sie sich, wie die meisten anderen Arbeiten auch, lediglich mit der Interpretation von Informationen in den entsprechenden Darstellungsformaten befassen und die Erstellung entsprechender Beschreibungen außer Acht lassen.

Mit der Akzeptanz der der Unified Modeling Language (UML) durch die OMG und die folgende Etablierung der UML als standardisiertes Beschreibungsmittel für konventionelle Softwareprojekte, rückte auch die UML-basierte Beschreibung in den Fokus der Untersuchungen.

Hahn und Kim verglichen verschiedene grafische Darstellungsformen für die Anwendung bei Integrationsaufgaben [HaKi99]. Zwar wurde in diesem Fall nicht die gesamte UML untersucht, jedoch sind mit Sequenzdiagramm, Aktivitätsdiagramm und Kollaborationsdiagramm drei der untersuchten Diagramme der UML zuzuordnen. Darüber hinaus wurden Aktivitäts-Fluss-Diagramme untersucht. In einem Experiment hatten Probanden die Aufgabe, verschiedene Teilprozesse zu integrieren. Die Information über die einzelnen Prozesse wurde dabei in verschiedenen genannten Diagrammen zur Verfügung gestellt. Dabei konnte für diese Aufgabe eine unterschiedliche Eignung der Diagrammtypen nachgewiesen werden. So nutzen die Probanden die Möglichkeit, Elemente in den Diagrammen räumlich zu gruppieren (z.B. durch Swimlanes in Aktivitätsdiagrammen), dazu, zuerst notwendige Komponenten zu identifizieren und erst im Anschluss daran den Prozess zu beschreiben. Dies führte in der Folge zu einer geringeren Fehlerrate bei der Integration.

In einer Reihe von Arbeiten wurde versucht, die Verständlichkeit einzelner Diagrammtypen der UML zu bewerten. Kutar et al. untersuchten Unterschiede in der Verständlichkeit von Sequenzdiagrammen und Kollaborationsdiagrammen [KBB02], konnten jedoch keine signifikanten Unterschiede nachweisen. Einen ähnlichen Ansatz für die Bewertung einzelner Diagrammtypen wählten Otero und Dolado [OtDo02]. Sie verglichen Sequenzdiagramme, Kollaborationsdiagramme und Zustandsdiagramme und konnten im Gegensatz zur erstgenannten Untersuchung signifikante, jedoch abhängig von der in den Diagrammen beschriebenen Applikation, unterschiedlich starke Unterschiede nachweisen.

Im Bezug auf die Qualität eines durch den Anwender erstellten UML-Modells führten Lange und Chaudron eine Fallstudie in industriellen Unternehmen durch [LaCh04]. Sie

konnten dabei zeigen, dass die Qualität der erstellten Modelle, gemessen an den aufgestellten Bewertungskriterien Vollständigkeit, Konsistenz und Wohlgeformtheit, individuell sehr stark variiert. Lange und Chaudron untersuchten auch die Erkennbarkeit von Defekten in UML-Modellen und die Konsequenzen, die aus diesen Defekten resultieren können [LaCH06]. Es konnte gezeigt werden, dass die Erkennbarkeit von der Art des Fehlers abhängt. Als Folge solcher Fehler konnten Fehlinterpretationen identifiziert werden, die ebenso von der Art des Fehlers abhängen. Lange und Chaudron schließen daraus, dass der Vermeidung von Defekten in UML-Modellen besondere Aufmerksamkeit gewidmet werden muss.

Der Nutzen einer UML-basierten Dokumentation, als Hilfestellung bei der Analyse von Programmen, wurde von Tilley und Huang untersucht [TiHi03]. Sie konnten qualitativ den Einfluss verschiedener Eigenschaften auf das Verstehen des Programms nachweisen. Neben der Syntax und Semantik des Programms sowie der räumlichen Anordnung der Elemente eines Diagramms, beeinflusst auch das Wissen über die Zieldomäne selbst in welchem Maße ein Nutzer Informationen aus dem Modell herauslesen kann. Fehlendes Domänenwissen führte dazu, dass die Repräsentation des Systems nur unvollständig verstanden und wiedergegeben werden konnte.

Arisholm et al. untersuchten den Einfluss einer UML-basierten Dokumentation für die Aufgabe der Änderung von bestehender Software [ABH+05]. Im Rahmen von zwei Experimenten bestand die Aufgabe der Probanden darin, das Programm eines Geldautomaten und das Programm eines Softdrink-Automaten zu erweitern. Ein Teil der Probanden benutzte für diese Aufgabe eine UML-basierte Softwaredokumentation, der andere Teil musste die Aufgabe ohne Unterstützung durchführen. Im Bezug auf die für die Durchführung der Änderungen konnte zwar eine Zeitersparnis für die Programmierung nachgewiesen werden, diese wurde jedoch durch die Modifikation der zur Verfügung gestellten Dokumentation wieder kompensiert. Im Bezug auf die Genauigkeit der Änderungen konnte durch Anwendung der UML-basierten Dokumentation eine Verringerung der Fehlerrate erreicht werden. Dieser positive Effekt zeigte sich jedoch nur für Aufgaben mit einer ausreichenden Komplexität.

## 4.3 Fragestellungen zum Einfluss einer Modellierung auf die Entwicklung von Steuerungsprogrammen

Die Analyse der bisher aufgeführten theoretischen Grundlagen zeigt, dass die Bildung eines mentalen Modells des zu realisierenden Systems eine der Haupttätigkeiten bei der Erstellung von Steuerungssoftware darstellt. Der in der vorliegenden Arbeit verfolgte

Ansatz geht deshalb davon aus, dass eine Unterstützung bei der Bildung des mentalen Modells in der Analysephase sich positiv auf die Programmierleistung auswirkt. Eine Modellierung unterstützt den Anwender, weil dabei das System systematisch analysiert und die Anforderungen und Eigenschaften festgelegt werden. Durch die Verwendung einer grafischen Modellierungsnotation werden zudem die relevanten Systemaspekte sowie die Zusammenhänge grafisch darstellt und gleichzeitig dokumentiert. Durch die verschiedenen Diagramme, die in den einzelnen Modellierungsnotationen zur Verfügung stehen, hat der Anwender die Möglichkeit, verschiedene Sichten auf das System zu beschreiben. Durch die Trennung der Sichten wird einerseits die Komplexität reduziert, da nur die jeweils relevanten Aspekte beschrieben werden. Andererseits bleibt der Gesamtzusammenhang dabei jedoch erhalten und kann durch die Integration der verschiedenen Sichten wieder hergestellt werden. Durch die Verwendung einer einheitlich festgelegten Notation kann das Modell zudem als eine gemeinsame Informationsbasis und Diskussionsgrundlage bei der Arbeit in einem Projektteam dienen.

Im Bezug auf die Implementierung des Steuerungsprogramms werden bei der Modellierung bereits Designentscheidungen getroffen. Durch den hierarchisch gliedernden bzw. objektorientierten Ansatz der Modellierungsnotationen wird dabei die Modularisierung des Systems unterstützt und gleichzeitig die Komplexität der Aufgabe reduziert. Das System wird in der Modellierungsphase gleichzeitig systematisch dokumentiert. Dadurch steigt die kognitive Verfügbarkeit der relevanten Informationen. In der Modellierungsphase führt dies zu einem höheren Prozesswissen, da sich besonders bei komplexen Systemen bereits in der Modellierungsphase die Wahrscheinlichkeit verringert, dass relevante Aspekte vergessen oder vernachlässigt werden. Bei der Programmierung führt die Nutzung des Modells ebenfalls zu einer höheren kognitiven Verfügbarkeit der Informationen, wodurch ein positiver Einfluss im Bezug auf die benötigte Zeit und die auftretenden Fehler zu erwarten ist.

Die genannten Aspekte führen zu der Annahme, dass sich eine Unterstützung des Anwenders, durch die Bereitstellung eines geeigneten Beschreibungsmittels für die Analyse und zur Modellierung des automatisierungstechnischen Systems für die anschließende Programmierung einer Steuerung, positiv auf das Ergebnis auswirkt. Durch die Verwendung des dokumentierten Modells zur Programmierung der Steuerung ist deshalb zu erwarten, dass der Zeitaufwand für diese Arbeitsaufgabe und die Fehlerquote signifikant sinken. Die im Folgenden beschriebenen Aspekte sollen in diesem Zusammenhang im Rahmen einer Evaluation genauer untersucht werden.

### 4.3.1 Qualität der modellierten Systembeschreibung

Die Qualität des Modells ist, unabhängig davon, ob es vom Programmierer selbst oder von einem anderen Anwender erstellt wurde, mit ausschlaggebend dafür, wie gut es für die Programmieraufgabe geeignet ist und wie es das Ergebnis der Programmierung beeinflusst.

Ein zu berücksichtigender Punkt ist die Frage, welche Aspekte des Systems in dem Modell beschrieben werden. Wie in Kapitel 4.1.2 beschrieben, muss das Modell die drei Sichten Aufgabenstellung, Systemarchitektur und Schnittstellen beinhalten, um dem Programmierer ein möglichst komplettes Bild des Systems zu vermitteln. Fehlt eine dieser Sichten, ist das Modell unvollständig. Dadurch besteht die Gefahr, dass wichtige, nicht dargestellte Aspekte beim Design des Systems nicht berücksichtigt werden und das System die entsprechenden Anforderungen nicht erfüllt oder falsch realisiert.

Darüber hinaus muss auch die Qualität der einzelnen Teilmodelle berücksichtigt werden. Auch innerhalb der einzelnen Sichten muss die Beschreibung konsistent, vollständig und entsprechend der gestellten Aufgabe ausreichend detailliert sein, um für den Programmierer die optimale Unterstützung zu gewährleisten. Inkonsistenz, Unvollständigkeit und fehlende Detaillierung können ansonsten dazu führen, dass die Software die geforderten Aufgaben fehlerhaft ausführt und das System somit in unerwünschte, gegebenenfalls unsichere Zustände geführt wird.

Inkonsistenzen im Modell können dazu führen, dass das System in einem entsprechenden Zustand aufgrund widersprüchlicher Auslegung falsch reagiert oder entsprechende Anweisungen sich blockieren. Unvollständigkeit des Modells kann zur Folge haben, dass das realisierte System zwar in Teilen entsprechend der Spezifikation arbeitet, aber in bestimmten Situationen oder auf bestimmte Ereignisse nicht reagiert. Ein Modell kann ein System in der angewendeten Detaillierungstiefe durchaus konsistent und vollständig beschreiben. Im Bezug auf die durch die Modellierung zu erfüllenden Aufgabe kann die Detaillierungstiefe jedoch zu gering sein, wodurch die Anwendung des Modells erschwert oder verhindert wird. In diesem Zusammenhang ist festzuhalten, dass fehlende Detaillierung auch als eine Form der Unvollständigkeit angesehen werden kann. Im Rahmen der vorliegenden Arbeit wird jedoch bewusst zwischen beiden Punkten unterschieden, um den Einfluss beider Aspekte im Rahmen der Auswertung evaluieren zu können.

In der Bewertung der Qualität von Software haben sich Metriken als Bewertungsmaßstab bewährt. So wird z.B. eine Sofwaremetrik als Funktion definiert, die eine Software-Einheit in einen Zahlenwert abbildet. „*Dieser berechnete Wert ist interpretierbar als der Erfüllungsgrad einer Qualitätseigenschaft der Software-Einheit*“ [IEEE1061]. Diese

Definition kann analog auf die Betrachtung von Modellen übertragen werden. Mit Hilfe von Metriken besteht hier die Möglichkeit, die Qualität eines Modells für verschiedene Aspekte quantitativ zu erfassen. Damit wird eine Grundlage geschaffen, Modelle im Bezug auf ihre Qualität miteinander zu vergleichen. Für UML-basierte Modelle existiert bereits eine Reihe von Metriken, für die ICL existiert ein solches Bewertungsschema noch nicht.

### 4.3.2 Arbeitsaufgabe

In Kap 2.2 wurde anhand verschiedener Vorgehensmodelle gezeigt, dass entlang des gesamten Lebenszyklus automatisierungstechnischer Systeme in den verschiedenen Phasen von unterschiedlichen Mitarbeitern dazu beigetragen wird die Steuerungssoftware zu spezifizieren, zu entwickeln, zu implementieren, zu warten oder auch zu optimieren. Ein Modell des Systems kann folglich in jeder dieser Phasen erstellt oder zur Realisierung entsprechender Aufgaben angewendet werden. Bei einer Untersuchung der Anwendbarkeit von Modellierungsnotationen ist es deshalb wichtig, die Arbeitsaufgabe des Anwenders in die Betrachtungen zu integrieren. Das bedeutet, es muss differenziert werden, auf welche Art eine Modellierungsnotation oder ein erstelltes Modell für die Realisierung welcher Aufgabe eingesetzt wird.

Zu diesem Zweck werden die beiden Teilaufgaben Modellierung und Programmierung unterschieden und für diese die Ausprägungen „Erstellung“ bzw. „Nutzung“ des Modells sowie „Erstellung“ bzw. „Modifizierung“ des Programms festgelegt. Ein drittes Unterscheidungskriterium ist durch den Anwender selber gegeben, der entweder beide Teilaufgaben selber durchführen oder bei der Programmierung auf ein Modell eines anderen Anwenders zurückgreifen kann. Dadurch ergeben sich sinnvoller Weise folgende Szenarien:

1. Ein Programmierer nutzt eine Modellierungsnotation während der Analyse des zu automatisierenden Systems sowie zur Dokumentation der analysierten Informationen und Zusammenhänge. Das erstellte Modell wird anschließend von demselben Anwender dazu verwendet, das Steuerungsprogramm zu erstellen.
2. Für die Programmierung der Steuerung nutzt der Programmierer ein Modell des Systems, welches von einem anderen Anwender erstellt wurde, ohne das System dabei selber analysiert oder modelliert zu haben.
3. Zur Modifizierung der Steuerungssoftware nutzt ein Programmierer ein bereits verfügbares Modell des ursprünglichen Systems. Dies wird von ihm gemäß den neuen

Anforderungen angepasst und anschließend zur Programmierung der Steuerung verwendet.

4. Zur Modifizierung der Steuerungssoftware erstellt der Programmierer zuerst ein Modell des ursprünglichen Systems. Dies wird von ihm gemäß den neuen Anforderungen angepasst und anschließend zur Programmierung der Steuerung verwendet.

Unterschiedliche Arbeitsaufgaben können dazu führen, dass sich der Nutzen der Anwendung einer Modellierung für diese Aufgaben unterscheidet. Auch können die einzelnen Notationen in den unterschiedlichen Szenarien unterschiedlich starke Auswirkungen zur Folge haben. Aus diesem Grund sind die Arbeitsaufgaben getrennt voneinander für jede der Notationen zu untersuchen.

### 4.3.3 Gruppenarbeit

Die Entwicklung eines Automatisierungssystems erfolgt zumeist nicht durch einzelne Personen, sondern durch ein Projektteam. Innerhalb eines solchen Teams besitzt jedes Mitglied zumeist einen fest zugeordneten Aufgabenbereich. Im Laufe der Entwicklung des Systems besteht also die Notwendigkeit, dass die einzelnen Mitarbeiter untereinander kommunizieren und Informationen miteinander austauschen.

Die Art, wie Informationen weiter gegeben werden, und die Art der Beschreibung von Information sind dabei ein Einflussfaktor für die Qualität und Effektivität des Informationsaustausches. Werden Informationen beispielweise in einem Gespräch ausgetauscht, erhöht sich die Gefahr, dass einzelne Aspekte von der informationsgebenden Person vergessen werden. Darüber hinaus kann die individuelle Bewertung auf der informationsempfangenden Seite dazu führen, dass einzelne Angaben nur kurzzeitig aufgenommen werden, später jedoch nicht erinnert werden können. Durch beide Effekte kommt es zu einer unvollständigen Übertragung der Information. Ein Austausch dokumentierter Information trägt zur Vermeidung dieser Effekte bei und steigert dadurch die Qualität und Effektivität des Informationsaustausches.

Individuelle Unterschiede in den mentalen Modellen der einzelnen Mitarbeiter sind ein weiterer Aspekt, der im Bezug auf die Gruppenarbeit zu berücksichtigen ist. Wie in Kapitel 4.1.1 dargestellt, ist das mentale Modell eines Menschen entscheidend dafür, wie er aufgenommene Informationen interpretiert und daraus Schlussfolgerungen für sein Handeln ableitet. Unterschiedliche Modelle können so, besonders unter Verwendung der natürlichen Sprache, zu verschiedenen Interpretationen derselben Information bei verschiedenen

Personen führen, was wiederum die Qualität des Informationsaustausches negativ beeinflusst. Dieser Effekt wird an den Schnittstellen zwischen den einzelnen Gewerken noch verstärkt, wenn Mitarbeiter unterschiedlicher Domänen zusammen arbeiten.

Die Anwendung eines einheitlichen Beschreibungsmittels, zur Dokumentation der Information, soll der unvollständigen Informationsübertragung entgegenwirken. Darüber hinaus soll durch die auch über die Grenzen der Gewerke einheitliche Bedeutung der Beschreibungselemente die Wahrscheinlichkeit der Fehlinterpretation der übertragenen Information reduziert werden. In wiefern sich die vorgestellten Modellierungsnotationen in diesem Zusammenhang eignen und unterscheiden muss untersucht werden.

### 4.3.4 Anwenderqualifikation

Die Qualifikation sowie das Vorwissen des Anwenders beeinflusst maßgeblich seine Leistungsfähigkeit bei dem Einsatz einer Modellierungsnotation zur Analyse und Beschreibung des Systems. Zum einen ist dabei das eigentliche Wissen über die Syntax und Semantik der anzuwendenden Notation notwendig. Dies hat zum einen Einfluss auf die Auswahl geeigneter Diagrammarten für die zu beschreibenden Aspekte, zum anderen beeinflusst es auch mit Hilfe welcher Notationselemente die analysierten Eigenschaften und Zusammenhänge dargestellt werden.

Das Wissen über die verwendete Notation allein führt jedoch nicht zu einer qualitativ hochwertigen Modellierung des Systems. Die Analyse des zu erstellen Systems und die anschließende Erstellung einer Modellierung mit Hilfe adäquater Beschreibungsmittel stellt einen Prozess dar, der ein gewisses Abstraktionsvermögen erfordert. So muss der Anwender beispielsweise die Fähigkeit besitzen gleichartige Strukturen zu erkennen, um modular modellieren und in geeigneter Weise Wiederverwendung betreiben zu können. Außerdem sind Fähigkeiten in entsprechenden (z.B. objektorientierten) Paradigmen notwendig, um eine entsprechende Vorgehensweise anwenden zu können.

Letztendlich beeinflusst auch die Qualifikation, die ein Anwender hinsichtlich der Programmierung von Speicherprogrammierbaren Steuerungen besitzt, sowohl das Ergebnis der Modellierung, als auch der Programmierung. Wie in Kapitel 2.3.2 dargestellt unterscheiden sich IEC 61131-basierte Steuerungssysteme in einigen Punkten grundlegend von herkömmlicher PC-basierter Software. Als Beispiel hierfür sind die zyklische Programmabarbeitung oder die Arbeit mit einem Prozessabbild zu nennen. Diese Eigenschaften müssen bereits bei der Modellerstellung berücksichtigt werden. Im Bezug auf die Modellierung ist anzunehmen, dass unerfahrene SPS-Programmierer größere Schwierigkeiten bei der Modellierung zeigen, da das mentale Modell der Arbeitsweise von

Speicherprogrammierbaren Steuerungen nicht so ausgeprägt ist. Anwender mit guten Programmierkenntnissen besitzen hingegen ein elaboriertes mentales Modell der Arbeitsweise einer Speicherprogrammierbaren Steuerung. Es ist anzunehmen, dass dies dazu führt, dass zielgerichtet die Aspekte analysiert und modelliert werden, welche für die spätere Realisierung relevant sind. Die Qualität des Modells wird folglich bei erfahrenen SPS-Programmierern höher sein, als bei unerfahrenen Anwendern.

Letztendlich beeinflusst die Anwenderqualifikation auch direkt die Programmierleistung. Es ist zu erwarten, dass eine größere Erfahrung in der Programmierung von Speicherprogrammierbaren Steuerungen dazu führt, dass die Programmierleistung höher ist.

### 4.3.5 Subjektive Bewertung der Anwendbarkeit der Modellierungsnotationen

Die Motivation von Programmierern stellt einen weiteren wichtigen Einflussfaktor bei der Evaluation von Modellierungsnotationen für die Steuerungsprogrammierung dar. Wie in Kapitel 4.1.1 erläutert wurde, ist anzunehmen, dass sich sowohl ein hohes Maß an Selbstbewusstsein als auch an intrinsischer Motivation positiv auf die Leistung bei der Programmierung auswirken. Es wird vermutet, dass durch die Modellierung des Systems und die Verwendung des Modells für die Programmierung die durch den Anwender wahrgenommene Komplexität der Arbeitsaufgabe sinkt. Dadurch steigen das Selbstbewusstsein des Programmierers und seine Motivation und folglich auch seine Leistung bei der Programmierung der Steuerung.

Neben den Auswirkungen auf die eigentliche Arbeitsausgabe ist ein weiterer Aspekt von hohem Interesse. Um die für die Anwendung einer abstrakten Modellierung notwendige Änderung in der Vorgehensweise beim Engineering erwirken zu können, muss die Akzeptanz der neuen Methode beim Anwender vorhanden sein. Dafür muss untersucht werden, ob die Einschätzung der Anwender durch die praktische Anwendung einer Modellierung positiver gestaltet werden kann, oder welche weiteren Maßnahmen zur Steigerung der Akzeptanz ergriffen werden können.

# 5 Entwurf und Realisierung des Versuchsdesign

In Kapitel 1 erfolgte die Herleitung von Fragestellungen, die bezüglich der Anwendbarkeit von Modellierungsnotationen und deren Einfluss auf die Qualität der Steuerungsprogrammierung zu untersuchen sind. Für die Evaluation von theoretisch abgeleiteten Thesen stehen verschiedene Methoden zur Verfügung, die in beschreibende und experimentelle Methoden unterteilt werden können [Chr98].

Beschreibende Methoden zeichnen sich durch das Ziel aus, eine bestimmte Situation oder ein Phänomen möglichst genau zu beschreiben. Der Schwerpunkt liegt darin die Variablen zu identifizieren, die eine gegebene Situation beschreiben, und die Beziehungen zwischen den Variablen zu identifizieren. Die Beziehung zwischen Ursache und Wirkung steht dabei jedoch nicht im Vordergrund. Beispiele für beschreibende Methoden sind Fallstudien, Korrelationsstudien oder Stichprobenerhebungen. Die auszuwertenden Merkmale werden dabei z.B. durch Beobachtung, Interviews oder Fragebögen erhoben.

Experimentelle Methoden untersuchen kausale Zusammenhänge, d.h. die Beziehung zwischen Ursache und Wirkung, mit Hilfe von Experimenten [Sch97]. Zu diesem Zweck werden die zu untersuchenden Eigenschaften identifiziert und festgelegt, welche messbaren Größen (Variablen) diese Merkmale im Experiment beschreiben. Dabei werden die Größen in unabhängige und abhängige Variablen unterteilt. Im Experiment werden die unabhängigen Variablen (Einflussgrößen) kontrolliert variiert und die Auswirkungen auf die abhängigen Variablen (Ziel- oder Antwortgrößen) gemessen.

In der vorgestellten Arbeit erfolgte die Untersuchung der Fragestellungen mit Hilfe eines gemischten Ansatzes, bei dem die Daten zur Evaluation sowohl im Rahmen eines Experimentes als auch durch Beobachtung durch den Versuchsleiter und die Beantwortung von Fragebögen durch die Probanden erhoben wurden. Dieser Ansatz wird im Folgenden beschrieben.

## 5.1 Der fertigungstechnische Beispielprozess

Für die Durchführung der Evaluationsexperimente wurde das Labormodell eines fertigungstechnischen Prozesses (Abb. 5.1) gewählt. Die Grundaufgabe des Prozesses besteht darin, Werkstücke, die aus einem Materiallager bereitgestellt werden, in Abhängigkeit ihrer Eigenschaften einem Stempelmodul zuzuführen und zu bearbeiten oder diese direkt in eine Endposition zu transportieren.

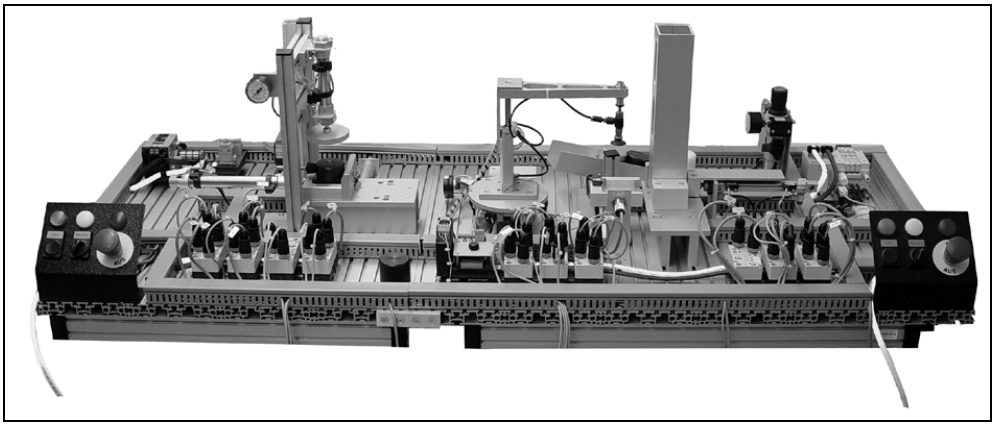


*Abb. 5.1 - Labormodell des fertigungstechnischen Prozesses (ohne Steuerungshardware)*

### 5.1.1 Mechanischer Aufbau des Modells

Das Modell besteht aus drei Modulen: einem Materiallager mit dazugehörigem Drehkran, einem Stempelmodul und einer Ausschussrutsche, welche die Endposition darstellt. Die Bedienung des Prozesses erfolgt über zwei Bedienpulte, welche identisch aufgebaut und mit einem Start-Taster, einem Hand-Automatik-Umschalter, einem Not-Aus-Schlagschalter und diversen Meldeleuchten ausgestattet sind. Abb. 5.2 zeigt eine schematische Darstellung des Prozesses, in welcher zur Vereinfachung auf die Darstellung der Sensorik / Aktorik verzichtet wird.

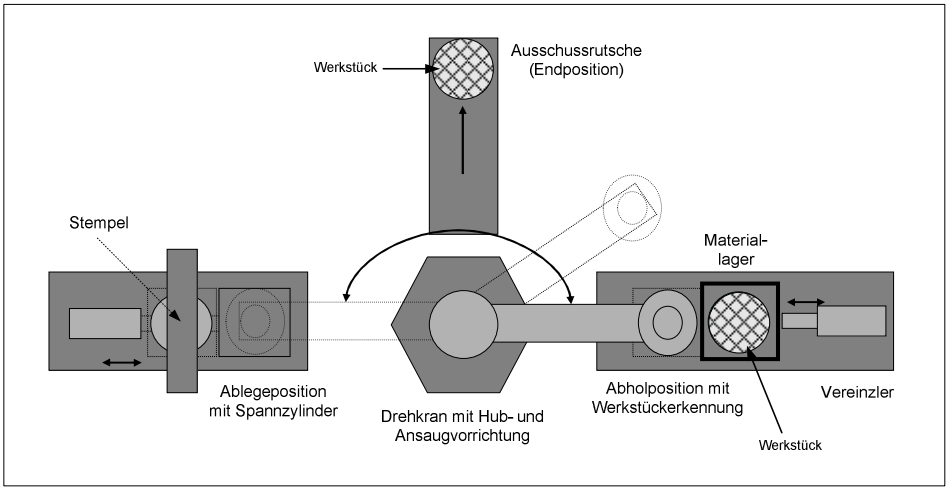



*Abb. 5.2 - Schematische Darstellung des fertigungstechnischen Prozesses*

Im Sortiermodul (Abb. 5.1 und Abb. 5.2 jeweils rechts) befinden sich in dem Materiallager Werkstücke aus unterschiedlichen Materialen und in unterschiedlicher Farbe. Die Werkstücke sind übereinander aufgestapelt und können mit Hilfe eines Vereinzlers einzeln aus dem Lager in die Abholposition geschoben werden. An dieser Position befindet sich neben einem induktiven und einem optischen Sensor zur Erkennung der Materialeigenschaften auch ein Mikroschalter, durch den ein aus dem Lager geschobenes Werkstück erkannt werden kann.

Zentrales Element des Modells bildet der zum Sortiermodul gehörende Drehkran, welcher für den Transport der Werkstücke zwischen den einzelnen Prozessteilen verantwortlich ist. Der Kran kann pneumatisch gesenkt und wieder angehoben werden und ist mit einer Ansaugvorrichtung ausgestattet, um Werkstücke aufnehmen und transportieren zu können. Die Stellung des Krans wird einerseits über Mikroschalter detektiert, welche die drei definierten Positionen „Stapelmagazin“, „Stempel“ und Rutsche“ registrieren. Darüber hinaus kann der Drehwinkel auch mit Hilfe eines Drehpotentiometers gemessen werden.

Das Stempelmodul (Abb. 5.1 und Abb. 5.2 jeweils links) setzt sich aus dem Spannzylinder und dem eigentlichen Stempel zusammen. Ein in der Werkstückaufnahme des Spannzylinders abgesetztes Werkstück wird zuerst unter dem Stempel arretiert. Der Stempel simuliert eine Bearbeitung des Werkstücks. Er wird über ein Proportionalventil angesteuert und kann mit einem kontinuierlich Druck von 0…6 bar Arbeitsdruck beaufschlagt werden.

Die Endposition der Werkstücke bildet eine Aluminium-Rutsche. Die Werkstücke werden dort abgesetzt und gleiten bis an das Ende der Rutsche. Von dieser Endposition müssen die Werkstücke vom Bediener per Hand wieder in das Magazin befördert werden. Die Rutsche stellt ein rein passives Element dar und besitzt keine Sensoren oder Aktoren.

### 5.1.2 Automatisierungstechnischer Aufbau des Modells

Das Modell ist mit einer Reihe binärer und analoger Sensoren und Aktoren ausgestattet, die zur Ermittlung des aktuellen Prozesszustandes sowie zur Steuerung des Prozesses dienen. Tab. 5.1 zeigt eine modulbezogene Übersicht.

*Tab. 5.1 - Modulbezogene Übersicht der Sensorik und Aktorik des Beispielprozesses*

| Modul | Sub-modul | Bestand-teile | Analoge Eingänge | Analoge Ausgänge | Digitale Eingänge | Digitale Ausgänge |
|---|---|---|---|---|---|---|
| Stapel | Vereinzler | - Vereinzlungszylinder<br>- Positionssensoren für Zylinder | - | - | 2 | 1 |
| | Werkstück-erkennung | - Optischer Sensor<br>- Induktiver Sensor<br>- Mikroschalter | - | - | 3 | - |
| | Kran | - Hubzylinder<br>- Positionssensoren für Zylinder<br>- Motor zur Drehung<br>- Endschalter für definierte Positionen<br>- Sauggreifer zum Ansaugen | 2 | 1 | 6 | 5 |
| | Bedieneinheit | - Anzeigeleuchten<br>- Starttaster<br>- Umschalter Hand-/Automatikbetrieb<br>- Not-Aus-Schlagschalter | - | - | 3 | 2 |
| Stempel | Spannzylinder | - Spannzylinder für Werkstücke<br>- Positionssensoren für Zylinder | - | - | 2 | 2 |
| | Stempelzylinder | - Proportionalventil für Stempeldruck<br>- Zylinder zum Stempeln<br>- Positionssensoren für Zylinder | 1 | 1 | 2 | 1 |
| | Bedieneinheit | - Anzeigeleuchten<br>- Starttaster<br>- Umschalter Hand-/Automatikbetrieb<br>- Not-Aus-Schlagschalter | - | - | 3 | 2 |
| | | Summe | 3 | 2 | 21 | 13 |

Die Steuerung des Prozesses erfolgt mit Hilfe einer Speicherprogrammierbaren Steuerung, die mit den Sensoren und Aktoren mittels AS-Interface verbunden ist.

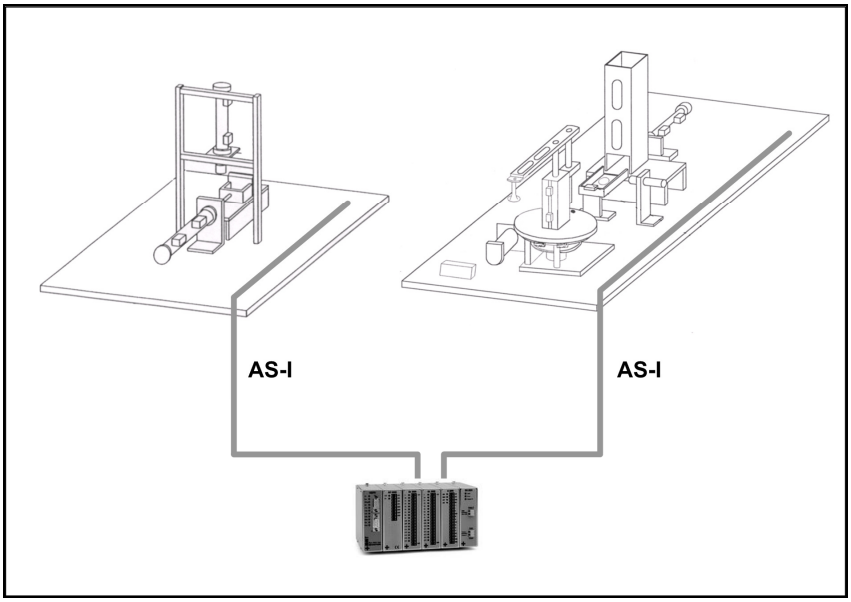



*Abb. 5.3 - Zentrales Steuerungskonzept*

Zur Steuerung des Prozesses stehen zwei unterschiedliche Steuerungsplattformen zur Verfügung. Die erste Konfiguration ermöglicht es den Prozess mit einer Speicherprogrammierbaren Steuerung der Firma Siemens, Typ S7-300, zu steuern. In diesem Aufbau erfolgt die Programmierung mit Hilfe der Softwareumgebung SIMATIC Step7 welche die Sprachen Anweisungsliste, Kontaktplan und Funktionsbausteinsprache zur Verfügung stellt. Teilweise weicht die Umsetzung der Programmiersprachen aus Kompatibilitätsgründen zur Vorgängerversion (Step5) jedoch von der IEC-Norm ab. Die IEC-Sprachen Strukturierter Text und Ablaufsprache stehen für die verwendete Hardware nicht zur Verfügung. Trotz dieser Einschränkungen ist diese Konfiguration aufgrund der weiten Verbreitung der Hardware am Markt als Standardlösung für derartige Automatisierungsaufgaben zu bewerten und stellt somit eine für die Untersuchung geeignete Versuchsumgebung dar.

In der zweiten Konfiguration erfolgt die Steuerung des Prozesses mit Hilfe eines Industrie PCs und der Soft-SPS TwinCat der Firma Beckhoff. Die Programmierung erfolgt hierbei in der Softwareumgebung TwinCat PLCcontrol, welche alle fünf Sprachen der IEC 61131-3 Normkonform unterstützt. Beide Programmierumgebungen stellen den gleichen Sprachumfang zur Programmierung der SPS bereit und stellen somit im Bezug auf den Versuch eine äquivalente Versuchsumgebung dar. Im Rahmen der Versuchsvorbereitung wurden die Probanden in der jeweils genutzten Umgebung geschult.

### 5.1.3 Ablauf des fertigungstechnischen Prozesses

Mit dem beschriebenen Prozessmodell können verschiedene Prozessabläufe realisiert werden. Zwei dieser Abläufe werden im Folgenden kurz beschrieben:

Nachdem die Starttaster beider Module gedrückt wurden, schiebt der Vereinzler ein Werkstück in die Abholposition. Das Werkstück wird analysiert, vom Kran aufgenommen und in Abhängigkeit der Materialeigenschaften zum Stempel oder direkt zur Endposition transportiert. Werkstücke aus schwarzem Kunststoff werden aus der Abholposition direkt zur Endposition befördert. Anschließend dreht sich der Kran wieder in seine Ausgangsposition am Magazin. Werkstücke aus weißem Kunststoff und Werkstücke aus Aluminium werden zum Stempelmodul transportiert und dort in der Werkstückaufnahme des Spannzylinders abgesetzt. Der Spannzylinder arretiert das Werkstück unter dem Stempel. Anschließend werden weiße Kunststoffwerkstück für 10 Sekunden mit 2 bar gestempelt, Aluminiumwerkstücke werden mit 6 bar für 20 Sekunden gestempelt. Nachdem der Spannzylinder das Werkstück wieder unter den Kran geschoben und aufgenommen hat, wird es zur Endposition transportiert und dort abgelegt. Abschließend dreht sich der Kran wieder in seine Ausgangsposition am Magazin. Befinden sich noch Werkstücke im Magazin startet ein neuer Zyklus, anderenfalls wird der Prozess beendet.

Alternativ zu diesem rein sequentiellen Ablauf kann auch ein teilweise paralleler Ablauf realisiert werden. Während ein Werkstück vom Stempel bearbeitet wird, schiebt der Vereinzler bereits das nächste Werkstück zur Analyse in die Abholposition. Während eines aktiven Stempelvorgang kann der Kran so gegebenenfalls ein oder mehrere schwarze Werkstücke vom Magazin zur Rutsche transportieren. Dadurch kann der gesamte Prozess zeitlich optimiert werden.

Für die Evaluationsexperimente wurde der zweite Ablauf gewählt, da er aufgrund parallel ablaufender Bearbeitungsschritte eine erhöhte Komplexität aufweist. Die Programmierung wurde dabei auf das Stapelmodul beschränkt, da das Stempelmodul kaum neue Aspekt für die Programmierung erbracht, den zeitlichen Aufwand jedoch deutlich erhöht hätte.

### 5.1.4 Bewertung des Prozessmodells für die experimentelle Evaluation

Für die Erzielung verwertbarer Ergebnisse ist zuvor die Eignung des zuvor beschriebenen Prozessmodells für die vorliegenden Fragestellungen sowie die experimentelle Aufgabenstellung zu prüfen. Zu diesem Zweck wird im Folgenden die Komplexität des Prozesses bewertet. Zudem erfolgt auch die Betrachtung der Eigenschaften der Modellierungsnotationen in Bezug auf die modellierungsrelevanten und programmierungsbezogenen Eigenschaften des Prozessmodells.

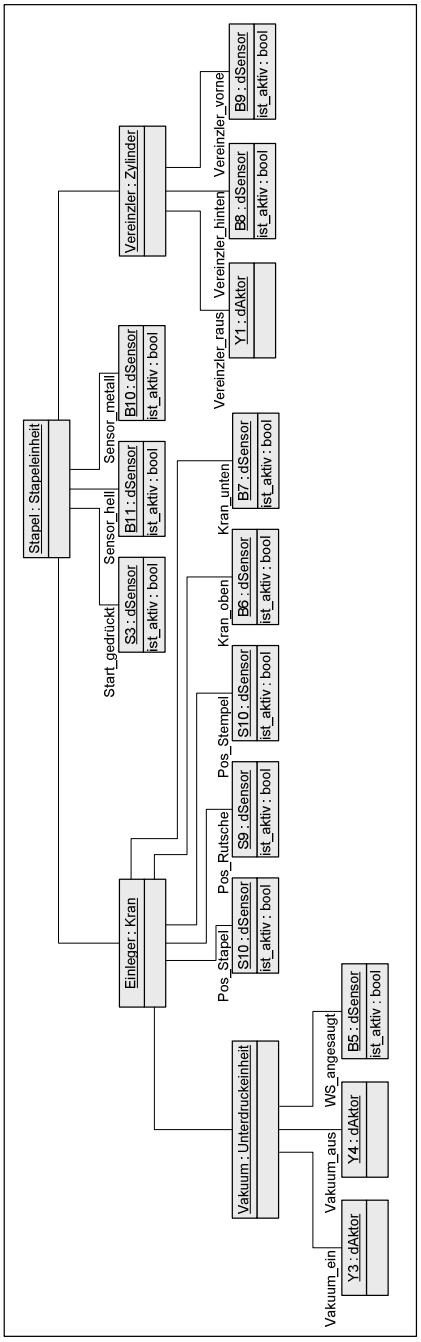



*Abb. 5.4 - Objektdiagramm für den Stempelteil des Prozessmodells (Ausschnitt)*

Die Komplexität des Prozesses muss sowohl in Bezug auf die strukturellen als auch auf die dynamischen Eigenschaften bewertet werden. Wie in den Kapiteln 5.1.1 und 5.1.2 dargestellt, kann der gesamte Prozess modular gegliedert werden. Der aus sieben, teilweise gleichartigen Modulen (vergleiche Tab. 5.1 - Submodule) aufgebaute Prozess besitzt wieder verwendbare Elemente, die eine modularisierte und objektorientierte Betrachtung und Beschreibung erlauben. Auf diese Module verteilen sich insgesamt 34 digitale sowie 5 analoge Ein-/Ausgangssignale. Abb. 5.4 zeigt ausschnittsweise die Strukturmodellierung der Mechanik sowie der Sensorik und Aktorik mittels Objektdiagramm.

Die dynamischen Eigenschaften des Prozesses können ähnlich modular beschrieben werden, indem gemäß des objektorientierten Modellierungsansatzes der UML jeder Klasse ein eigenes Verhalten beispielsweise in Form eines Zustandsdiagramms zugeordnet wird. Die einzelnen Verhaltensmerkmale einer Klasse können dabei recht einfach beschrieben werden. Die wechselseitige Koordination des Verhaltens der einzelnen Elemente und somit das Verhalten des gesamten Prozesses ist jedoch vergleichsweise komplex. Abb. 5.5 verdeutlicht dies durch die Darstellung des Systemverhaltens in einem integrierten Zustandsdiagramm.

Die Komplexität des Prozesses ist für die Durchführung der Evaluationsexperimente angemessen. Einerseits ist der Prozess realistisch und als nicht zu trivial einzustufen, um die Ergebnisse auf die betrachtete Anwendungsdomäne übertragen zu können. Andererseits sind Umfang und Komplexität der Arbeitsaufgabe nicht zu hoch, sodass die Probanden diese in der zur Verfügung stehenden Zeit erfüllen können.

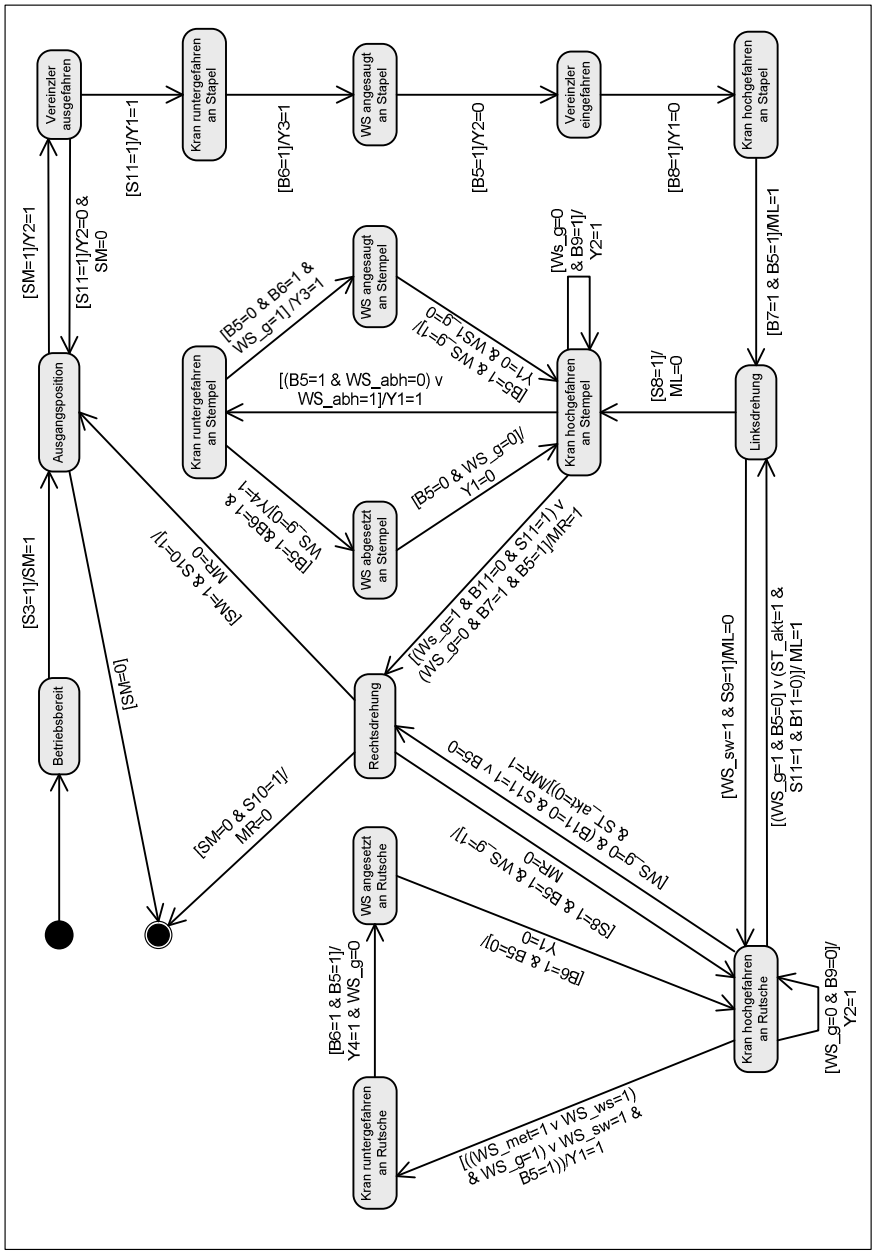



*Abb. 5.5 - Systemverhalten des Prozesses als integriertes Zustandsdiagramm*

## 5.2 Versuchsaufbau

Die Entwicklung des Versuchsdesigns ist in drei Phasen unterteilt. Im ersten Schritt wird die auf Basis der in Kapitel 4.3 zusammengestellten Fragestellungen entwickelte Aufgabenstellung beschrieben, mit der es möglich ist, die für eine Evaluation notwendigen Parameter empirisch zu erheben. Dazu erfolgt eine Zweiteilung der Aufgabe für die Ermittlung modellierungsbezogener Parameter und programmierungsbezogener Parameter. In einem zweiten Schritt wird, aufbauend auf dieser Aufgabenbeschreibung, festgelegt, welche Art von Vorbereitung die am Experiment teilnehmenden Probanden benötigen, bevor im dritten Teil die Durchführung beschrieben wird.

### 5.2.1 Aufgabenbeschreibung

Den verschiedenen Anwendungsszenarien des Einsatzes einer Modellierung im Verlauf des Engineeringprozesses von Automatisierungssystemen muss bei dem Entwurf der Experimente Rechnung getragen werden. Aus diesem Grund wurden zwei unterschiedliche Aufgabenstellungen entwickelt und in getrennten Versuchsdesigns angewendet. Die zwei Versuchsdesigns unterschieden sich in dem zur Verfügung gestellten Stimulusmaterial sowie der durchzuführenden Aufgabe und ermöglichen somit eine Überprüfung der Thesen für den jeweiligen Anwendungsfall.

Das erste Szenario beschreibt eine Situation in der ein Programmierer den Prozess zunächst selbstständig analysiert und ein Modell erstellt. Dies entspricht Variante 1 der in Kapitel 4.3.2 beschriebenen Szenarien. Dabei wird das entsprechende Beschreibungsmittel dazu verwendet, das analysierte System und somit das erarbeitete Wissen zu dokumentieren. Anschließend benutzt dieselbe Person dieses Modell dazu, ein Programm für eine Speicherprogrammierbare Steuerung zu implementieren. In der realen Anwendung entspricht dies der Aufgabe eine Neuanlage zu programmieren. Dieses erste Szenario wurde in Versuchsdesign 1 umgesetzt (Abb. 5.6). Um die Vergleichbarkeit zu einer Vorgehensweise ohne vorherige Modellierung zu gewährleisten, wurde der Versuch ebenfalls mit einer Kontrollgruppe durchgeführt, welche die Aufgabe ohne die Anwendung einer Modellierung realisierte.

Als Ausgangsmaterial für die Erstellung des Modells stand den Probanden in Szenario 1 eine grobe textuelle Beschreibung des zu realisierenden Systemverhaltens zur Verfügung [Fri09]. Dieser Text beschrieb in grundlegenden Anweisungen, wie mit den einzelnen, sich unterscheidenden Werkstücken verfahren werden soll. Zum anderen enthielt er funktionale Beschränkungen, die durch das System vorgegeben sind. So war beispielsweise zu beachten, dass ein Werkstück zuerst angesaugt werden muss, bevor der Spannzylinder

zurückgefahren werden darf oder der Spannzylinder zurückgefahren werden muss, bevor der Kran ein Werkstück anheben kann. Des Weiteren wurde den Probanden eine Übersicht der zur Verfügung stehenden Sensor- und Aktorsignale zur Verfügung gestellt [Fri09]. Es wurde vermieden, eine Zuordnung der Sensor und Aktorsignale zu einzelnen Modulen zu geben, um den Einfluss auf eine mögliche modulare Modellierung oder Programmierung auszuschließen.

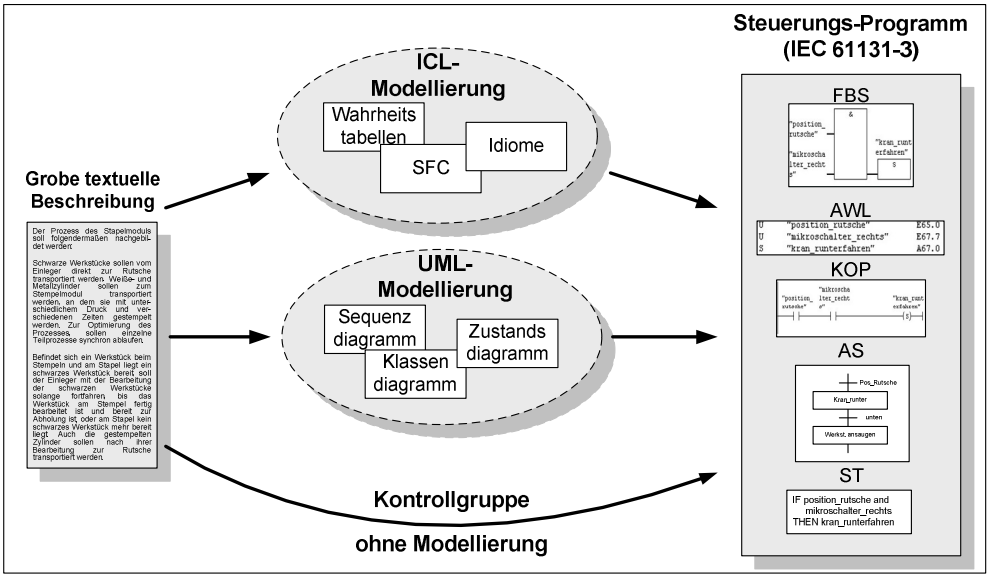



*Abb. 5.6 - Versuchsdesign 1 - Modellerstellung und Programmierung durch den Probanden*

Das zweite Szenario beschreibt eine Situation in der die Modellierung und die Implementierung von zwei unterschiedlichen Entwicklern bzw. einem Projekteur und einem Programmierer durchgeführt werden. In einem ersten Schritt modelliert ein Entwickler bzw. Projekteur den Prozess und dokumentiert diesen mit Hilfe eines Beschreibungsmittels. Das erstellte Modell wird anschließend an einen zweiten Entwickler bzw. Anwendungsentwickler weitergegeben, der darauf basierend das Steuerungsprogramm entwirft. Dieses Szenario entspricht Variante 2 der in Kapitel 4.3.2 beschriebenen Möglichkeiten. Im Gegensatz zum ersten Szenario analysiert der Programmierer also nicht selber das zu automatisierende System, sondern stützt sich einzig und allein auf die Informationen aus dem Modell. Dieses Szenario wurde in Versuchsdesign 2 realisiert (Abb. 5.7). Dabei wurde sich aus zwei Gründen dafür entscheiden, das Modell des Systems als Stimulusmaterial durch den Versuchsleiter vorzugeben und nicht durch eine zweite Versuchsperson erstellen zu lassen. Erstens wurde dadurch erreicht, dass alle Gruppen ein vergleichbares Modell für die Programmierung des Systems zu Grunde legten, wodurch

eine Vergleichbarkeit der Ergebnisse gewährleistet wurde. Zweitens konnten so alle zur Verfügung stehenden Probanden des zweiten Versuchsdesigns zur Auswertung des Versuchs herangezogen werden, da der Modellierungsaspekt bereits in Versuchsdesign 1 untersucht wurde.

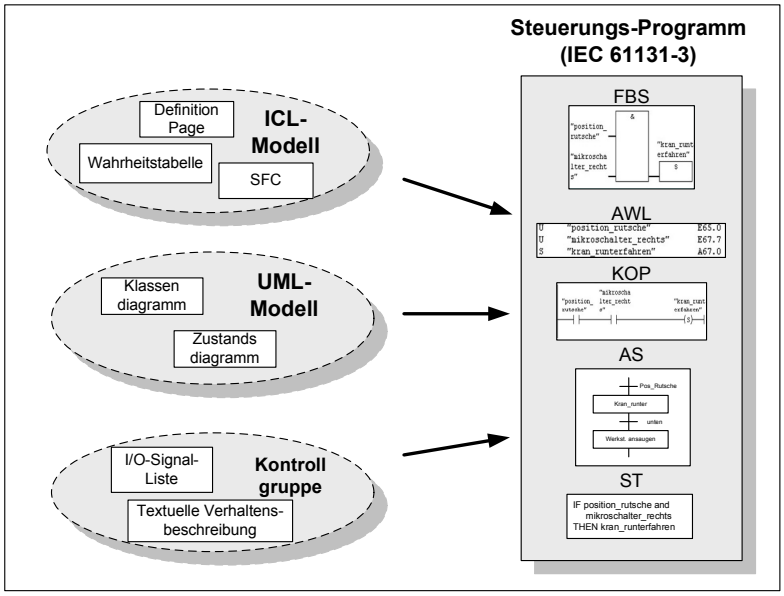



*Abb. 5.7 - Versuchsdesign 2 - Programmierung durch den Probanden nach vorgegebenem Modell*

Das zur Verfügung gestellte Material unterschied sich in Versuchsdesign 2 von dem zuvor beschriebenen ersten Versuchsdesign. Alle drei Versuchsgruppen erhielten eine vollständige, für die Programmierung der Steuerung geeignete Beschreibung des Systems, die sowohl strukturelle Eigenschaften des Systems als auch das zu realisierende Systemverhalten beschrieb.

Das UML-Modell bestand aus einem Objektdiagramm und einem Zustandsdiagramm [Fri09]. Das Klassendiagramm beschrieb den strukturellen Aufbau als Klassen mit ihren Assoziationen und die zur Verfügung stehenden Sensor- und Aktorsignale als zugeordnete Attribute der Klassen. Das Verhalten des Prozesses wurde mit einem Zustandsdiagramm beschrieben.

Das ICL-Modell bestand aus einer Definitionsseite (definition page), einer Wahrheitstabelle (truth table) und einem Sequential Function Chart [Fri09]. Die Definitionsseite gab eine Übersicht über die zur Verfügung stehenden Sensor- und Aktorsignale, die Wahrheitstabelle zählte Verriegelungsbedingungen für ausgewählte Situationen auf und das

Sequential Function Chart beschrieb das Systemverhalten als sequentielle Folge von Aktionen.

Die Kontrollgruppe bekam für die Erstellung des Steuerungsprogramms eine tabellarische Übersicht der Sensor- und Aktorsignale sowie eine detaillierte textuelle Beschreibung des Systemaufbaus und des Systemverhaltens zur Verfügung gestellt [Fri09]. Bei der Erstellung des Stimulusmaterials musste darauf geachtet werden, dass identische Informationen zur Verfügung gestellt werden, die sich nur in der Darstellungsform unterscheiden. Um dies sicher zu stellen, wurde das Stimulusmaterial im Vorfeld durch Experten auf Analogie geprüft.

Im Rahmen der Evaluationsexperimente wurde der Fokus auf die Neuerstellung von Steuerungsprogrammen und somit auf die in Kapitel 4.3.2 beschriebenen Varianten 1 und 2 gelegt. Die beschriebenen Varianten 3 und 4 wurden nicht ausführlich untersucht. Hierfür liegen keine verwertbaren Ergebnisse vor, diese müssen bei Bedarf in weiteren Arbeiten erhoben werden.

Es ist anzumerken, dass sich das zur Verfügung gestellte Material in beiden Versuchsdesigns von der in der industriellen Praxis verwendeten Dokumentation (z.B. Lastenheft oder Funktionsbeschreibung), welche Softwareentwicklern zur Verfügung gestellt wird, unterschied. Im Fokus des Versuchs stand jedoch die Modellierung mit den untersuchten Beschreibungsmitteln und nicht die Bewertung entsprechender Spezifikationsdokumente, weshalb dieser Unterschied im Bezug auf die spätere Auswertung der Ergebnisse als nicht ausschlaggebend eingestuft wird.

Im zweiten Teil des Experimentes bestand die Aufgabe für die Probanden darin, die aus der Modellierung bzw. der Modellanalyse gewonnenen Informationen in ein Steuerungsprogramm auf einer Speicherprogrammierbaren Steuerung umsetzen. Der Aufbau des Steuerungsprogramms sowie die Wahl der einzusetzenden Programmiersprache wurde den Probanden überlassen. Zum Zweck der Programmierung wurde den Probanden lediglich eine IEC61131-3-konforme Programmierumgebung und ein in Vorbereitung auf die Experimente erstelltes Musterprojekt (Rahmenwerk) für diese Umgebung zur Verfügung gestellt. Da die Projektierung der Steuerungsumgebung und der Automatisierungshardware nicht im Fokus der Evaluierung stand, enthielt das Projekt alle notwendigen Konfigurationen sowie die Verknüpfungen zu der realen Steuerungshardware. Zudem war eine Haupttask und ein Programmbaustein (Main-Baustein) definiert. Die Task wurde zyklisch aktiviert und rief jeweils den Main-Baustein auf. Nach Abschluss der Programmierung mussten in Abhängigkeit der erstellten Programmstruktur entsprechende Programmaufrufe eingefügt werden, um eine lauffähige und am Modell testbare

Programmierung zu erhalten. Abb. 5.8 stellt den Aufbau des Musterprojekts dar und kennzeichnet die vom Probanden im Rahmen der Experimente zu erstellenden Teile des Projektes.

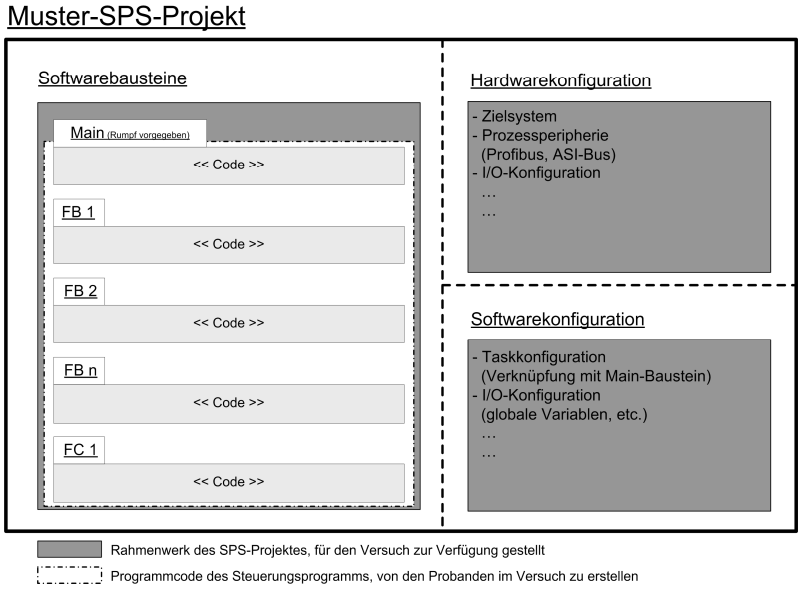



*Abb. 5.8 - Musterprojekt für die Durchführung der Experimente*

### 5.2.2 Vorbereitung und Training der Probanden

Die im vorangegangenen Kapitel beschriebene Aufgabenstellung für das Experiment erfordert von den Probanden sowohl Kenntnisse über die anzuwendenden Modellierungsnotationen als auch über die Programmierung von Speicherprogrammierbaren Steuerungen. Um die Abhängigkeit der Resultate von den individuellen Vorkenntnissen der Probanden zu verringern, erhielten alle Probanden vor der Durchführung der Experimente eine Schulung in den für die Aufgabenfüllung notwendigen Themenbereiche. Dadurch wurden den Teilnehmern nicht bekannte Sachverhalte neu vermittelt oder das Wissen über bereits Bekanntes aufgefrischt. Diese Vorbereitung bestand aus Schulungseinheiten zu den Themenbereichen „Objektorientierung und Unified Modeling Language“, „Idiomatic Control Language“ und „SPS-Programmierung nach dem Standard IEC 61131-3“. Die Schulungseinheiten waren jeweils zweigeteilt und bestanden aus einem Vorlesungs- und einem Übungsteil. Die Vorlesung hatte das Ziel, das für die Durchführung des Versuchs notwendige Wissen zu vermitteln, welches in der anschließenden Übung anhand eines Beispiels praktisch angewendet wurde. Pro

Schulungseinheit standen für die Vorlesung 90 Minuten und für die Übung 45 Minuten zur Verfügung.

Die Schulungseinheit „Objektorientierung und Unified Modeling Language“ fasste zu Beginn die Grundlagen der Klassenbildung sowie die Bedeutung von Attributen und Operationen zusammen und erklärte den Zusammenhang zwischen Klassen und Objekten. Darüber hinaus wurden die verschiedenen Arten von Relationen sowie die Bildung einer Klassenhierarchie erklärt. Im Anschluss an diese Einführung bekamen die Teilnehmer eine Einführung in die verschiedenen Diagrammtypen der UML. Zum einen wurde dargestellt, welche Aspekte eines Systems durch welches Diagramm dargestellt werden können, zum anderen wurde, jeweils unterstützt durch Beispiele, auf die Symbolik der Elemente der einzelnen Diagrammtypen eingegangen [Fri09].

Im Bereich der Idiomatic Control Language wurden zu Beginn die Pages als Strukturierungselemente der Systembeschreibung erläutert. Im Anschluss daran wurde die Bedeutung von Truth Tables, Sequential Function Charts und Idioms zur Beschreibung verschiedener Systemaspekte erklärt und, analog zur Schulung der UML, anhand von Beispielen auf die Symbolik der Elemente der einzelnen Beschreibungsformen eingegangen. Die Erklärung der Idiome wurde dabei bewusst auf die Grundlagen beschränkt, da die im Experiment gestellte Aufgabe ausschließlich Steuerungs- und keine Regelungsaspekte beinhaltete [Fri09].

In der Schulungseinheit zum Thema „IEC 61131-3“ wurde zu Beginn auf die Arbeitsweise einer Speicherprogrammierbaren Steuerung eingegangen. Die speziellen Unterschiede der Programmabarbeitung im Vergleich zu konventionellen Softwaresystemen wurden besonders hervorgehoben. Anschließend wurden die Programmiersprachen im Detail dargestellt und mögliche Anwendungsfälle für die einzelnen Sprachen gezeigt [Fri09].

Nach Abschluss der Schulungseinheiten erhielten die Probanden einen Fragebogen, der ihre Vorkenntnisse in den Bereichen der Modellierung von Systemen und der Programmierung von Speicherprogrammierbaren Steuerungen ermittelte [Fri09]. Neben der Frage nach bereits durchgeführten Projekten, die eine Modellierung oder Programmierung zur Aufgabe hatten, sollten die Probanden auch ihre Fähigkeiten in diesen Bereichen selber bewerten. Neben den Vorkenntnissen gehörten auch Fragen zur Verständlichkeit und Nachvollziehbarkeit der Schulungseinheiten zum Inhalt des Fragebogens.

Auf Grundlage der Ergebnisse der Befragung konnte anschließend die Aufteilung der Probanden auf die verschiedenen Gruppen (UML, ICL und Kontrollgruppe) erfolgen.

Dabei wurde soweit möglich auf die Ausgeglichenheit der Gruppen im Bezug auf die Vorkenntnisse geachtet, um eine Vergleichbarkeit der Ergebnisse gewährleisten zu können.

### 5.2.3 Versuchsdurchführung und Randbedingungen

Der eigentliche Evaluationsversuch war in vier Abschnitte eingeteilt (Abb. 5.9) und bestand aus dem Eingangsfragebogen, dem Modellierungsteil, dem Programmierteil sowie einem Abschlussfragebogen. Die zeitlichen Rahmenbedingungen wurden im Vorfeld festgelegt und mit Hilfe von Vorversuchen evaluiert [FVB03]. Für die Beantwortung der Fragebögen standen jeweils 15 Minuten zur Verfügung, der Modellierungs- und der Programmierteil dauerten in beiden Versuchsdesigns je 90 Minuten. Es wurden keine Pausen zwischen den einzelnen Abschnitten eingelegt, sodass der Versuch eine Gesamtdauer von 210 Minuten hatte.

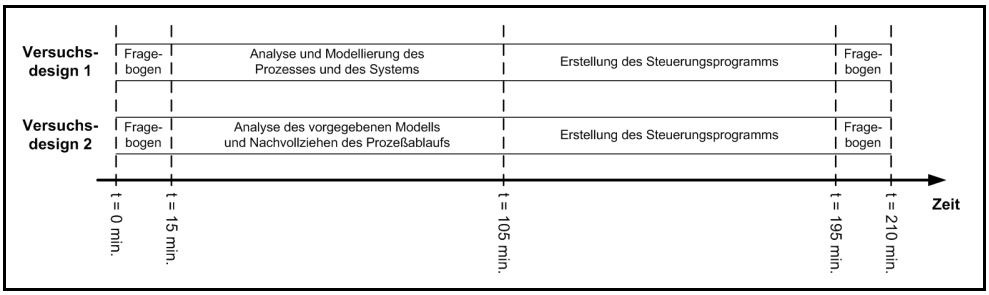



*Abb. 5.9 - Zeitlicher Ablauf der Experimente*

Für die Erstellung der Modellierung wurde den Probanden im ersten Versuchsdesign eine Tafel, ein Flipchart sowie Papier und Stifte zur Verfügung gestellt. Es wurde davon abgesehen die Modellierung mit Hilfe eines Softwarewerkzeugs durchzuführen. Die Gründe dafür liegen zum einen in der Tatsache, dass für die Modellierung mit der Idiomatic Control Language im Gegensatz zur Unified Modeling Language kein marktreifes Tool existiert, mit dem die Modellierung durchgeführt werden kann. Durch die daraus resultierenden Abweichungen im Versuchsaufbau hätten sich folglich Probleme in der Vergleichbarkeit dieser beiden Gruppen ergeben. Zudem bieten die am Markt befindlichen Tools für die Modellierung mit der UML teilweise starke Unterschiede im Bezug auf die zur Verfügung gestellte Funktionalität und nutzen die Anpassbarkeit der UML auf unterschiedliche Weise aus. Nicht zu vernachlässigen ist auch der Einarbeitungsaufwand in entsprechende Modellierungstools, der sich mit steigender Komplexität der Werkzeuge ebenfalls erhöht. Die Wahl des Werkzeugs hätte somit auch einen Einfluss auf das Modellierungsergebnis gehabt. Um diesen Einfluss zu minimieren, wurde auf die Verwendung eines Softwarewerkzeugs verzichtet.

Während der Versuchdurchführung wurden die Probanden von einem Versuchsleiter beobachtet. Dieser nahm eine überwiegend passive Rolle ein und dokumentierte besondere Auffälligkeiten während der Experimente. Da ein Teil der Experimente aus Zeitgründen parallel durchgeführt werden musste, wurden für die Experimente zwei Versuchsleiter eingesetzt. Vor den Versuchen stimmten sich die Versuchsleiter ab, indem Regeln zur Protokollführung aufgestellt wurden. Anhand mehrerer Beispiele wurde im Rahmen der Vorversuche sichergestellt, dass beide Versuchsleiter die Regeln gleich umsetzen. Dies wurde anhand eines Expertenratings der geschriebenen Protokolle validiert.

Darüber hinaus stand der Versuchsleiter den Probanden für Fragen zur Verfügung. Diese wurden ebenfalls dokumentiert, jedoch nur beantwortet, wenn die Antwort keine offensichtliche Beeinflussung des Versuchsergebnisses im Bezug auf die zu untersuchenden Fragestellungen darstellte. Ein Beispiel für unkritische Fragestellungen sind beispielsweise Fragen zur Bedienung der Programmierumgebung. Auch hierzu fand vor dem Versuch eine Abstimmung zwischen den Versuchsleitern statt. Für jeden Teilnehmer wurde ein Versuchsprotokoll angelegt, in dem weitestgehend formlos Uhrzeit und Inhalt für die oben genannten Punkte dokumentiert wurde.

Nach Durchführung des Versuches mussten alle Teilnehmer einen Abschlussfragebogen beantworten [Fri09]. Dieser Fragebogen diente dazu, die subjektiven Wahrnehmungen der Teilnehmer zu bestimmten Aspekten zu ermitteln. Getrennt nach Modellierungs- und Programmierteil mussten die Probenden verschiedene Aussagen zur Anwendung der Modelierung nach ihrer persönlichen Einschätzung auf einer vorgegebenen Skala bewerten. Darüber hinaus konnte zusätzlich zu jeder Aussage ein freier Text formuliert werden um Besonderheiten zu nennen.

## 5.3 Operationalisierung

Für eine Überprüfung der in Kapitel 4.3 abgeleiteten Fragestellungen ist zunächst eine Operationalisierung notwendig [Chr98]. Dabei wird festgelegt, wie die verbal gefassten Aussagen mit Hilfe von messbaren Größen in den Experimenten erfasst werden können, um die Zusammenhänge evaluieren zu können. Im Folgenden wird eine Übersicht über die zur Verwendung vorgesehenen unabhängigen und abhängigen Variablen gegeben und die Art der Erhebung sowie die Skalierung der Variablen beschrieben. Es werden nominalskalierte Variablen (z.B. ungeordnete Klassifizierung nach Studiengang oder Ausbildung) und ordinalskalierte Variablen (z.B. geordnete Bewertung einer Aussage von „sehr gut“ bis „sehr schlecht“) unterschieden [Wot88].

### 5.3.1 Unabhängige Variablen

Die unabhängige Variable ist diejenige Größe, die in einer Untersuchung variiert wird, um deren Auswirkungen auf die abhängige Variable zu erfassen. Die unabhängige Variable stellt also eine Einflussgröße bzw. eine Ursache in einem vermuteten Zusammenhang dar. Zur Untersuchung wurden die folgenden unabhängigen Variablen genutzt:

***Qualifikation des Probanden:***

Die Qualifikation der Probanden wurde vor der Durchführung des Versuches mit Hilfe von Fragebögen erhoben. Ziel war es durch Variation dieser Variablen einen möglichen Zusammenhang zu der erbrachten Leistung bei der Modellierung und Programmierung zu evaluieren. Neben der Art der beruflichen Ausbildung (nominalskaliert) wurden auch Kenntnisse im Hinblick auf die Anwendung einer Modellierung sowie Programmierkenntnisse als Indikator für die Qualifikation angenommen. Diese beiden Größen (ordinalskaliert) wurden mit Hilfe einer Selbsteinschätzung durch den Probanden in einem Fragebogen ermittelt.

Die Qualifikation konnte als unabhängige Variable nur bedingt variiert werden, da sie maßgeblich von den für die Evaluation zur Verfügung stehenden Probanden abhängt. Die Qualifikation der Probandengruppen ist bezüglich der Vorerfahrungen und der Modellierungs- bzw. Programmierkenntnisse zwar unterschiedlich einzustufen. Eine Aussage über Zuordnung der verschiedenen Probandengruppen zu qualitativen oder gar quantitative Qualitätsleveln stand nicht im Fokus der Untersuchung und ist mit den erhobenen Daten nicht möglich. Vielmehr soll die Qualifikation hier als eine Größe verstanden werden, welche sich nach der Ausbildung der Probanden unterscheidet und eher klassifizierenden als quantifizierenden Charakter hat.

***Arbeitsaufgabe:***

Die durchzuführende Arbeitsaufgabe stellt den Teil der unabhängigen Variablen dar, die am besten beeinflusst werden konnten. Folgende Aspekte sollen im Hinblick auf die Arbeitsaufgabe untersucht werden:

- Einzelarbeit ⇔ Gruppenarbeit
- Mit Unterstützung ⇔ Ohne Unterstützung durch eine Modellierung
- Unterscheidung nach Art der Modellierungsnotation UML ⇔ ICL
- Selbst durchgeführte Modellierung ⇔ Verwendung vorgegebener Modelle

Alle genannten Variablen sind nominalskaliert und wurden durch ein entsprechendes Design des Versuchsaufbaus gezielt verändert.

### 5.3.2 Abhängige Variablen

Die abhängige Variable ist diejenige Größe, deren Veränderung infolge der unabhängigen Variablen gemessen wird. Es ist sinnvoll, die abhängigen Variablen in folgende drei Gruppen gemäß den Auswirkungen auf das Modellierungsergebnis, auf das Programmierergebnis sowie die subjektive Bewertung zu unterscheiden.

***Auswirkungen auf das Modellierungsergebnis:***

Die Bewertung der Modellierung des Prozesses erfolgte durch Analyse des Modells im Hinblick auf verschiedene Aspekte. Zum einen wurde bewertet, ob und in welchem Unfang das Modell die Struktur und das Verhalten des Prozesses fehlerfrei wieder gibt. Zum anderen wurde der Grad der Modularität der Modellierung überprüft. Dadurch ergaben sich folgende Kennwerte, die zur Bewertung der Modellierungsleistung verwendet wurden:

- Gesamtzahl der modellierten Schritte / Zustände $Z_M$ und Anzahl der richtig modellierten Schritte / Zustände $Z_{M,R}$
- Gesamtzahl der modellierten Transitionen / Zustandsübergänge TM und Anzahl der richtig modellierten Transitionen / Zustandsübergänge $T_{M,R}$
- Anzahl der Modulebenen in der Modellierung $N_{M,M}$:
- Die Kenngröße $N_{M,M}$ ermöglicht die Beurteilung, ob die Modellierung modular vorgenommen wurde. $N_{M,M}=1$ bezeichnet eine monolithische, nicht modulare Modellierung. Je größer der Wert für $N_{M,M}$ ist, desto mehr Hierarchieebenen enthält die Modellierung.

***Objektive Bewertung des Programmierergebnisses:***

Die Bewertung der Programmierleistung erfolgte durch Codeinspektion der erstellten Programme. Zu diesem Zweck wurde im Vorfeld des Versuchs die Musterlösung für das zu realisierende Systemverhalten in einem UML-Zustandsdiagramm modelliert [Fri09]. Dieses Modell wurde von verschiedenen Experten evaluiert und als angemessen für die Auswertung der in den Experimenten erstellten Programme bewertet. Dieses Zustandsdiagramm diente als Vorlage für die Analyse und ermöglichte es, die minimal notwendigen Schritte des Programms und die notwendigen Zustandsübergänge mitsamt der zu berücksichtigenden Bedingungen herauszukristallisieren und zu beschreiben. Die

Auswertung der Musterlösung ergab insgesamt 32 Schritte, die das Programm realisieren muss.

Die von Experten durchgeführte Codeinspektion der erstellten Programme auf Basis der Musterlösung lieferte anschließend drei Kennwerte, die zur Bewertung der Programmierleistung herangezogen werden:

- Anzahl der programmierten Schritte $S_b$:
  $S_B$ bezeichnet die Anzahl der bearbeiteten Schritte und beschreibt, welche Zustandswechsel der Vorlage im Steuerungsprogramm realisiert wurden. Dabei wird lediglich geprüft, ob Quell- und Zielzustand richtig identifiziert wurden, nicht berücksichtigt wird, ob die Bedingungen für den Zustandwechsel korrekt sind.

- Anzahl der richtig programmierten Schritte $S_{b,r}$:
  $S_{B,R}$ bezeichnet die Anzahl der richtig bearbeiteten Schritte und beschreibt, welche Zustandswechsel der Vorlage im Steuerungsprogramm sowohl im Bezug auf Quell- und Zielzustand als auch auf die erforderlichen Bedingungen zum Zustandswechsel korrekt sind. Diese Kenngröße wird nicht direkt zur Auswertung der Experimente herangezogen, sondern nur indirekt für die Ermittlung der Fehlerrate verwendet.

- Fehlerrate:
  Die Fehlerrate beschreibt den Anteil der fehlerhaften Schritte gemessen an der Gesamtzahl der bearbeiteten Schritte und berechnet sich nach:

$$f_r = \frac{S_E - S_{B,r}}{S_E}$$

Die Kenngrößen $S_B$ und $S_{B,R}$ stellen aufgrund der identischen zeitlichen Vorgaben für alle Experimente ein Maß für die Geschwindigkeit bei der Programmierung dar und ermöglichen so einen Vergleich der Programmierleistung der einzelnen Teilnehmer.

Im Bezug auf Modularität wurde analysiert, ob bei der Erstellung der Programme ein monolithischer oder modularer Ansatz gewählt wurde. Dazu wurde die Anzahl der Module als Kenngröße verwendet:

- Anzahl der Module in der Programmierung $N_{M,P}$:
  Die Kenngröße $N_{M,P}$ ermöglicht die Beurteilung, ob das Programm modular erstellt wurde. $N_{M,P}$=1 bezeichnet eine monolithische, nicht modulare Programmierung.

***Subjektive Bewertung durch die Probanden***

Die subjektive Bewertung wurde mit Hilfe eines Fragebogens erhoben. Die Probanden müssen vorgegebene Aussagen auf einer Ordinalskala bewerten. Die Fragen beziehen sich dabei auf die folgenden Aspekte:

- Wahrgenommener Nutzen der Modellierung für die Aufgabe:
  Ziel der Fragen zu diesem Thema war es zu ermitteln, ob die durchgeführte Modellierung als eine nutzbringende Unterstützung für die anschließende Modellierung empfunden wurde. Dabei wurde die Bewertung für die struktur- und verhaltensbezogene Modellierung getrennt aufgenommen. Fragen dieses Themas wurden nur den Probanden gestellt, die eine Modellierung durchführen mussten oder ein vorgegebnes Modell zur Programmierung verwendeten.

- Wahrgenommene Schwierigkeit der gestellten Aufgabe:
  Die Fragen zu diesem Bereich wurden allen Probenden gestellt und ermöglichten durch einen Vergleich der Modellierungsgruppen mit der Kontrollgruppe die Bewertung, ob eine Modellierung wahrnehmbar zu einer Reduzierung der Komplexität der gestellten Aufgabe führt.

- Erwarteter Nutzen der Modellierung für die Aufgabe:
  Die Fragen zu diesem Thema wurden nur von Probanden der Kontrollgruppen beantwortet, die zwar wie alle anderen in den Modellierungsnotationen geschult wurden, jedoch die Programmierung ohne jegliche Unterstützung durchführen mussten.

### 5.3.3 Ermittlung der Größen im Experiment

Die unabhängigen Variablen wurden durch gezielte Variation beeinflusst und wurden während des Experiments größtenteils nicht gesondert ermittelt. Eine Ausnahme bildeten die Größen zur Qualifikation und zu den Vorkenntnissen. Diese Werte wurden vor dem Versuch mit Hilfe eines Fragebogens ermittelt.

Die abhängigen Variablen konnten erst nach der Durchführung des Versuchs erhoben werden. Hier wurden zwei verschiedene Methoden angewendet. Die Ermittlung der Kenngrößen zur Modellier- und Programmierleistung basiert auf dem von Experten evaluierten Musterlösungen. Die Kenngrößen zur subjektiven Wahrnehmung wurden mit einem Fragebogen nach Versuchsdurchführung ermittelt.

## 5.4 Probandengruppen

Die Auswahl der Probanden stellt einen wichtigen Aspekt in der Vorbereitung von empirischen Untersuchungen dar, da sie die externe Validität und somit die Generalisierbarkeit der Ergebnisse bestimmt. Sowohl die Eigenschaften der Probanden als auch die Anzahl der Teilnehmer sind dabei ausschlaggebend. Im Bezug auf die Probandengruppen [Chr98] wird in diesem Zusammenhang unterschieden zwischen der Zielpopulation (target population), über welche eine Aussage getroffen werden soll, und der für ein Experiment zugänglichen Versuchspopulation (experimentally accessible population) aus der die eigentlichen Teilnehmer als Stichprobe (sample) ausgewählt werden. Die Generalisierung der Ergebnisse erfolgt anschließend, wie in Abb. 5.10 dargestellt, in zwei Schritten.

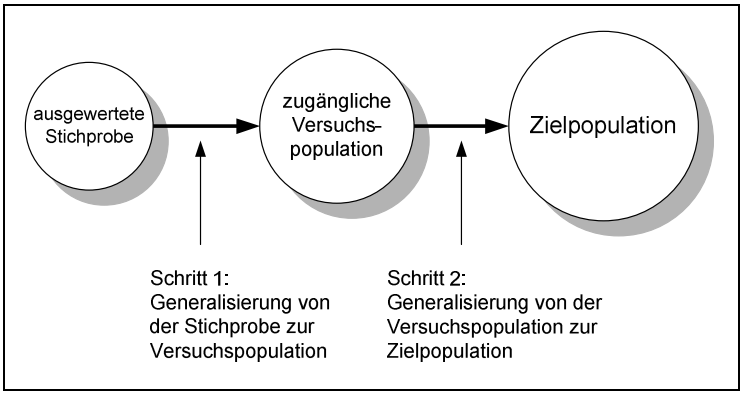



*Abb. 5.10 - Generalisierung von der Stichprobe zur Zielpopulation (in Anlehnung an [Chr98])*

Da, wie im vorangegangenen Kapitel dargestellt, die Qualifikation des Anwenders einen Einfluss auf die Leistung und das Ergebnis bei der Modellierung und Programmerstellung zu haben scheint, soll dieser Aspekt bei der Auswahl der Probanden besondere Berücksichtigung finden. Für die hier beschriebene Versuchsreihe standen Probanden unterschiedlicher Qualifikation zur Verfügung. Neben Studenten der Elektrotechnik und Informationstechnik, wurden auch Auszubildende (Industrieelekroniker), Studenten im Praxisverbund sowie Techniker in die Untersuchung mit einbezogen. Die Durchführung der Experimente mit hochqualifizierten Ingenieuren (Programmierexperten), die bereits große Erfahrung in der Steuerungsprogrammierung haben, konnte aus organisatorischen Gründen bis zur Fertigstellung der vorliegenden Arbeit nicht erreicht werden.

In den folgenden Unterkapiteln werden die Teilnehmergruppen detailliert beschrieben. Durch die Auswertung der Eingangsfragebögen ist, in Kombination mit Informationen über

den jeweiligen Studien- bzw. Ausbildungslehrgang, eine Unterscheidung der Probandengruppen im Hinblick auf ihre Qualifikation möglich.

### 5.4.1 Studenten Bachelor Information Technologies und Electrical Engineering

Insgesamt nahmen 48 Bachelorstudenten der Studiengänge Information Technologies und Electrical Engineering verteilt auf zwei Semester an der Versuchsreihe teil. Die erste Gruppe bestand aus 30 Probanden (Durchschnittsalter 24,7 Jahre, durchschnittlich 5,5 Studiensemester), von denen die Mehrheit (79%) in der Eingangsbefragung angab bereits andere Studienfächer zum Thema Modellierung besucht und Erfahrung in der Modellierung von Systemen zu haben. Die Einschätzung der Kenntnisse bei der Programmierung von Speicherprogrammierbaren Steuerungen wurde durchgehend als gering eingestuft. Im zweiten Semester nahmen 18 Probanden (Durchschnittsalter 25,4 Jahre, durchschnittlich 6 Studiensemester) am Versuch teil. Im Vergleich zur ersten Gruppe wurden die Programmierkenntnisse höher eingestuft, jedoch gaben weniger Teilnehmer (40%) den Besuch anderer Modellierungsbezogener Vorlesungen sowie Erfahrungen bei der Modellierung an.

### 5.4.2 Studenten Master Informationstechnologie

Sieben ausschließlich männliche Studenten des Studiengangs Master Informationstechnologie - Fachrichtung Information Science standen für die Durchführung der Experimente zur Verfügung. Vier Probanden hatten bereits ein Bachelorstudium, die restlichen drei Probanden ein Fachhochschulstudium abgeschlossen. Alle Probanden studierten im ersten Semester des Masterstudiengangs und hatten ein durchschnittliches Alter von 26 Jahren.

Die Eingangsbefragung zeigte in Bezug auf die Vorkenntnisse bei der Modellierung von Prozessen folgendes Ergebnis: Während Studenten mit vorangegangenem Bachelorstudium angaben, sich im Laufe ihres Studiums in mindestens einem Studienfach mit Modellierungsnotationen befasst zu haben, hatten die Fachhochschulstudenten bisher keine Erfahrungen gemacht. Die Bewertung der SPS-Programmierkenntnisse zeigte hingegen keine signifikanten Unterschiede und wurde weitestgehend als gering eingeschätzt.

### 5.4.3 Auszubildende zum Industrieelektroniker – Fachrichtung Produktionstechnik

In Kooperation mit der Volkswagen Coaching GmbH konnten Auszubildende zum Industrieelektroniker der Fachrichtung Produktionstechnik für die Evaluationsexperimente

gewonnen werden. Alle 12 Probanden waren im dritten Lehrjahr und standen kurz vor dem Abschluss ihrer Ausbildung. Für alle Auszubildenden war das die erste Berufsausbildung, das Durchschnittsalter betrug 19,8 Jahre. Die Gruppe setzte sich aus einem weiblichen und 11 männlichen Probanden zusammen

Die Auswertung der Eingangsfragebögen ergab, dass alle Teilnehmer im Rahmen der Evaluation das erste Mal mit Modellierung von Prozessen bzw. Modellierungsnotationen konfrontiert wurden. Während ihrer Ausbildung hatten alle Probanden an einer mehrwöchigen SPS-Schulung teilgenommen in deren Verlauf vier kleinere Steuerungsprogramme (Erstellungsdauer ca. 1,5-2 Stunden) erstellt werden mussten. Im Vergleich zu den anderen Teilnehmern fiel die Selbsteinschätzung der SPS-Programmierfähigkeiten folglich überdurchschnittlich hoch aus.

### 5.4.4 Studenten im Praxisverbund zum Industrieelektroniker / Industrieinformatiker

Ebenfalls in Zusammenarbeit mit der Volkswagen Coaching GmbH konnte das Evaluationsexperiment mit einer Gruppe von Studenten im Praxisverbund (StiPs) durchgeführt werden. Die Ausbildung unterscheidet sich von einer klassischen Ausbildung im dualen Berufsausbildungssystem dadurch, dass die Teilnehmer parallel zur Ausbildung im Betrieb ein Studium an einer Fachhochschule oder Universität absolvieren. Deshalb stellt die allgemeine Hochschulreife eine Einstiegsvoraussetzung für die Zulassung zu dieser Ausbildung dar.

Ingesamt nahmen 11 Studenten (10 männliche und ein weiblicher Proband, Durchschnittsalter 22,5 Jahre) an den Experimenten teil, alle absolvierten eine Ausbildung zum Industrieelektroniker in Verbindung mit einem Studium zum Industrieinformatiker (Fachhochschule Braunschweig/Wolfenbüttel) und standen kurz vor Abschluss ihrer 3-jährigen Ausbildungszeit. 9 von 11 Teilnehmern gaben in der Eingangsbefragung an, im Studium oder in der Ausbildung bereits verschiedene Modellierungsnotationen angewendet zu haben. Meist wurden Programmablaufpläne und Zustandsdiagramme genannt. Alle StiPs hatten in ihrem Studium an einem einsemestrigen SPS-Kurs teilgenommen und bereits mehrer Projekte in mittlerem Umfang selbstständig durchgeführt.

### 5.4.5 Technikerschüler - Fachrichtung Informatik

In Zusammenarbeit mit einem Berufkolleg wurden die Evaluationsexperimente mit einer Gruppe von angehenden Technikern durchgeführt. 11 ausschließlich männliche Probanden einer Abschlussklasse von Technikerschülern der Fachrichtung „Informatik“ führten das

Experiment im Rahmen des Lernfelds „Planung und Projektierung von Qualitätssicherungssystemen in industriellen Fertigungssystemen“ durch. Das Durchschnittsalter betrug 26,4 Jahre und alle Teilnehmer wiesen eine abgeschlossene Berufausbildung sowie eine mehrjährig ausgeübte Tätigkeit in dem erlernten Beruf vor. Im Rahmen der Eingangsbefragung gaben die Teilnehmer mehrheitlich an, bereits Erfahrungen bei der Programmierung von Speicherprogrammierbaren Steuerungen gesammelt zu haben, die Selbsteinschätzung ihrer Kenntnisse war jedoch überwiegend gering. Im Bezug auf Modellierungsnotationen gaben alle Teilnehmer an keine Erfahrungen zu besitzen.

### 5.4.6 Bewertung der Qualifikation und Gruppeneinteilung

Fasst man alle Probanden zusammen, so ergibt sich für die Durchführung der Evaluationsexperimente eine Gesamtteilnehmerzahl von 89 Teilnehmern. Durch entsprechende Vorlesungsveranstaltungen war der Zugang zu Studenten als Probanden deutlich einfacher und die Anzahl der Probanden in der Gruppe der Studenten somit erheblich höher als in den Gruppen mit externen Teilnehmern. In einer vorab durchgeführten Befragung wurden die Probanden dazu aufgefordert ihre für das Experiment relevanten Vorkenntnisse zu bewerten. Von Interesse war einerseits die Frage nach Vorkenntnissen im Bereich der Modellierung, die mit einer Entscheidungsfrage ermittelt wurden. Darüber hinaus mussten die Probanden ihre Erfahrung im Bezug auf die Programmierung von Speicherprogrammierbaren Steuerungen auf einer Skala von Eins (sehr gering) bis Fünf (sehr hoch) selbst bewerten. Tab. 5.2 fasst die Ergebnisse des Fragebogens zur Berufsqualifikation zusammen.

*Tab. 5.2 - Übersicht der Versuchsteilnehmer aufgeschlüsselt nach Berufsqualifikation*

| Probandengruppe | Anzahl Teilnehmer | Durchschnittliches Alter (Jahre) | Anzahl der Probanden mit Vorkenntnissen in der Modellierung | Erfahrung bei der SPS-Programmierung |
|---|---|---|---|---|
| Studenten Bachelor (Gruppe 1) | 30 | 24,7 | 24 (80,0%) | 1,91 |
| Studenten Bachelor (Gruppe 2) | 18 | 25,4 | 8 (44,4%) | 2,44 |
| Studenten Master | 7 | 26,0 | 4 (57,1%) | 2,14 |
| Auszubildende | 12 | 19,8 | 0 (0%) | 2,50 |
| Studenten im Praxisverbund | 11 | 22,5 | 9 (81,8%) | 1,56 |
| Technikerschüler | 11 | 26,4 | 0 (0%) | 2,27 |

Es wird deutlich, dass die beschriebenen Teilnehmergruppen überwiegend geringe Vorkenntnisse im Bereich der Programmierung von Speicherprogrammierbaren Steuerungen haben. Im Gegensatz zu erfahrenen Programmierern ist zu erwarten, dass diese für eine Programmieraufgabe mehr Zeit benötigen und eine höhere Fehlerrate aufweisen. Aufgrund fehlender Ergebnisse für erfahrene Programmierer, ist eine Übertragung der Ergebnisse dieser Evaluation somit nur mit Einschränkungen möglich. Gleichzeitig wurde jedoch bei der hier dargestellten Versuchskonfiguration der Einfluss der Erfahrung, der mit Kenngrößen nur schwer zu ermitteln und zu vergleichen ist, deutlich reduziert. Diese Aspekte finden bei der Interpretation der Ergebnisse besondere Berücksichtigung.

Wie in Kapitel 5.2.1 dargestellt, wurden für die Evaluation zwei unterschiedliche Versuchsdesigns entwickelt, um die Aspekte Modellerstellung und Modellanalyse in Kombination mit der Umsetzung in ein SPS-Programm getrennt voneinander untersuchen zu können. Die geringe Teilnehmerzahl an Master-Studenten sowie innerhalb der externen Probandengruppen erlaubte es nicht, die Versuche beider Designs mit jeder dieser Gruppen durchzuführen. Dies hätte entweder zu sehr kleinen Versuchsgruppen (1-2 Teilnehmer) geführt, oder die Probanden hätten beide Versuche nacheinander durchführen müssen. Im ersten Fall wäre die statistische Aussagekraft der Ergebnisse aufgrund der geringen Stichprobengröße deutlich gemindert worden. Im zweiten Fall hätte der Lerneffekt aufgrund der wiederholten Programmierung des gleichen Systems die Ergebnisse beeinflusst.

*Tab. 5.3 - Übersicht der Teilnehmer in den unterschiedlichen Versuchsgruppen getrennt nach Versuchsdesign*

| | **Versuchsdesign** | | | | | |
|---|---|---|---|---|---|---|
| | Modellierung und Programmierung | | | Programmierung nach vorgegebenem Modell | | |
| Probandengruppe | UML | ICL | Kontroll-Gruppe | ICL | UML | Kontroll-Gruppe |
| Bachelor Information Technologies | 5* | 5* | 5* | 6 | 6 | 6 |
| Bachelor Electrical Engineering | | | | | | |
| Master Information Technologies | 2 | 3 | 2 | - | - | - |
| Studenten im Praxisverbund | 3 | 4 | 4 | - | - | - |
| Auszubildende | - | - | - | 4 | 4 | 4 |
| Techniker | 4 | 3 | 4 | - | - | - |

* 2er Gruppen

Die zur Verfügung stehenden Probanden wurden, wie in Tab. 5.3 dargestellt, auf die beiden Versuchsdesigns verteilt. Dadurch war es möglich sowohl den Einfluss der unterschiedlichen Qualifikationslevel zu untersuchen als auch einen Vergleich zwischen selbst durchgeführter Analyse mit anschließender Modellierung und der Verwendung eines vorgegebenen Modells zur Programmierung zu ziehen. Die ursprüngliche Planung, alle externen Gruppen an der Evaluation im ersten Versuchsdesign teilnehmen zu lassen, musste bei der Gruppe der Auszubildenden verworfen werden, da die Auszubildenden im Rahmen der Schulungseinheiten und der Bearbeitung der Übungsaufgaben deutliche Schwächen im Umgang mit den zu verwendenden Modellierungsnotationen aufwiesen. Es war zu erwarten, dass die Qualität der erstellten Modelle deutliche Defizite aufweisen würde und eine Vergleichbarkeit nicht gegeben wäre. Folglich wurde den Auszubildenden Versuchsdesign 2 zugeordnet, bei dem ihnen ein fertiges Modell zur Verfügung gestellt wird.

# 6 Auswertung der Evaluationsexperimente

## 6.1 Grundlagen der Statistik

Zur statistischen Auswertung werden die univariate Varianzanalyse (ANOVA), die multi-variate Varianzanalyse (MANOVA) sowie die Korrelationsanalyse und die Regressionsanalyse angewendet [Bor99]. Die univariate Varianzanalyse ermöglicht es zu ermitteln, in wiefern die Variationen der abhängigen Variable durch Variationen der unabhängigen Variable hervorgerufen werden. Die ANOVA wertet somit die Zusammenhänge zwischen zwei Größen aus. Die multi-variate Varianzanalyse erlaubt eine Varianzanalyse für mehrere unabhängige und mehrere abhängige Variablen. Die MANOVA wird gewählt, wenn zwischen den unabhängigen Variablen Zusammenhänge vermutet werden, ohne dass jedoch die Möglichkeit besteht eine Variable durch eine andere zu ersetzen.

Für die Auswertung der Ergebnisse wird eine Irrtumswahrscheinlichkeit von 10% angesetzt. Das Signifikanzniveau liegt folglich bei $\alpha$=0,1. Statistische Überprüfungen der aufgestellten Thesen, deren Signifikanz unter dem festgelegten Signifikanzniveau liegen, werden somit als wahr betrachtet. Für eine Irrtumswahrscheinlichkeit bis 20% wird das Ergebnis als Tendenz gewertet, die im Weiteren noch einer genaueren Überprüfung bedarf.

Die Bewertung der Fragebögen erfolgt auf Basis der Mittelwerte der abgegebenen Wertung für eine Frage oder Aussage. Die Bandbreite der möglichen Bewertung liegt dabei für alle Fragen zwischen eins (niedrigste Bewertung bzw. Ablehnung einer Aussage) und fünf (höchste Bewertung bzw. volle Zustimmung zu einer Aussage). Eine Aussage wird dabei im Durchschnitt als gut oder mit hoher Zustimmung bewertet, wenn der Mittelwert 4,0 oder höher ist. Analog wird es als schlechte oder niedrige Zustimmung gewertet, wenn der Mittelwert 2,0 und niedriger ist.

Im Folgenden werden die Ergebnisse der Evaluationsexperimente beschrieben. In einem Unterkapitel wird dabei jeweils ein Aspekt (z.B. Einfluss der Qualifikation auf die Programmierleistung) behandelt. Der Einfluss auf die benötigte Zeit als auch auf die Fehlerrate wird dabei jeweils zusammen behandelt. Nach der Darstellung der Ergebnisse schließt jedes Unterkapitel mit einer Diskussion für den jeweiligen Aspekt ab.

## 6.2 Einfluss der Arbeitsaufgabe auf die Programmierleistung

### 6.2.1 Ergebnisse

Mentale Modelle können in Abhängigkeit vom jeweiligen Individuum stark variieren (vergl. Kapitel 4.1.1). Auch ein erstelltes Modell eines Systems oder Prozesses variiert nicht nur aufgrund des gewählten Beschreibungsmittels, sondern kann auch Unterschiede aufweisen, die z.B. daraus resultieren, wie ein System hierarchisiert oder modularisiert wurde. Folglich kann es sein, dass Anwender bei der Interpretation und Anwendung fremd erstellter Modelle Schwierigkeiten haben, wenn diese ein abweichendes mentales Modell zu Grunde legen. Dadurch verringert sich der Nutzen einer Anwendung solcher Modelle. Es ist deshalb zu vermuten, dass eine vom Probanden selbst durchgeführte Analyse und Modellierung des Systems eine stärkere positive Wirkung auf die Programmierleistung hat, als die Verwendung eines fertigen Systemmodells. Sowohl der Zeitbedarf als auch die Fehlerrate sollten deshalb in Versuchsdesign 1 signifikant geringer sein als in Versuchsdesign 2.

***These 1a:*** *Probanden, die ein vorgegebenes Modell bzw. eine textuelle Beschreibung verwenden, weisen eine geringere Anzahl bearbeiteter Schritte auf.*

Die Varianzanalyse der erhobenen Daten über alle Versuchsgruppen zeigt den Zusammenhang zwischen dem Versuchsdesign und der benötigten Zeit (Tab. 6.1). Im Vergleich der durchschnittlichen Anzahl bearbeiteter Schritte kann ein Vorteil der selbst durchgeführten Analyse und Modellierung des Systems gezeigt werden. Die Signifikanz ($p<0,01$) liegt dabei deutlich unter dem festgelegten Signifikanzniveau von $\alpha=0,1$.

These 1a ist wahr.

*Tab. 6.1 - Durchschnittliche Anzahl bearbeiteter Schritte in Abhängigkeit des Versuchsdesigns*

| Versuchsdesign | Bearbeitete Schritte | Signifikanz |
|---|---|---|
| VD1 - Modellieren und Programmieren | 19,31 | <0,01 |
| VD2 - Programmieren nach vorgegebenem Modell | 14,00 | |

Um den Einfluss der verwendeten Modellierungsnotation in Bezug auf These 1a zu bewerten, wurde die Gesamtheit der Probanden in die entsprechenden drei Gruppen unterteilt und die Varianzanalyse für jede der Gruppen erneut durchgeführt. Die Ergebnisse sind in Tab. 6.2 dargestellt.

*Tab. 6.2 - Durchschnittliche Anzahl bearbeiteter Schritte in Abhängigkeit des Versuchsdesigns und getrennt nach Modellierungsnotation*

| Modellierungsnotation | Bearbeitete Schritte Versuchsdesign 1 | Bearbeitete Schritte Versuchsdesign 2 | Signifikanz |
|---|---|---|---|
| Unified Modelling Language | 17,56 | 13,89 | 0,14 |
| Idiomatic Control Language | 20,15 | 10,00 | < 0,01 |
| Kontrollgruppe | 20,00 | 17,08 | 0,35 |

In allen drei Gruppen erzielten Probanden, die eine vorgegebene Modellierung bzw. eine textuelle Beschreibung des Systems verwendeten, ein schlechteres Ergebnis. Die detaillierte Betrachtung zeigt, dass die ICL-Gruppe bessere Ergebnisse als die UML-Gruppe erzielt, wenn das Modell selbst erstellt wird (Versuchsdesign 1). Für den Fall, dass ein vorgegebenes Modell verwendet wird (Versuchsdesign 2), ergibt sich ein gegenteiliges Resultat. Hier werden in der UML-Gruppe durchschnittlich mehr Schritte realisiert als in der ICL-Gruppe. Der Vergleich mit der Kontrollgruppe ergibt, dass durch die Verwendung einer Modellierungsnotation kein deutlicher Vorteil im Bezug auf die benötigte Zeit erzielt werden kann. In Versuchsdesign 1 erzielt die Kontrollgruppe ein ähnlich gutes Ergebnis wie die beste Modellierungsgruppe, im zweiten Versuchsdesign wird in der Kontrollgruppe sogar das beste Ergebnis erzielt. Während die Signifikanz in der ICL-Gruppe unter dem Signifikanzniveau von $\alpha$=0,1 liegt, wird die Signifikanzgrenze in der UML-Gruppe überschritten, wodurch dieses Ergebnis lediglich als Tendenz gewertet werden kann. In der Kontrollgruppe liegt die errechnete Signifikanz deutlich über dem festgelegten Signifikanzniveau, eine Abhängigkeit der Anzahl bearbeiteter Schritte vom Versuchsdesign kann hier nicht nachgewiesen werden. Insgesamt kann das Ergebnis der gemeinsamen Betrachtung somit in der getrennten Analyse weitgehend bestätigt werden.

**These 1a ist wahr für die Gruppe der ICL und kann für die UML-Gruppe tendenziell nachgewiesen werden.**

***These 1b:*** *Probanden, die ein vorgegebenes Modell bzw. eine textuelle Beschreibung verwenden, weisen eine höhere Fehlerquote auf.*

Die Auswertung zeigt, dass die Probanden, die am Versuch in Design 1 teilnahmen und den Prozess selbstständig analysiert und modelliert haben, eine deutlich geringere Fehlerrate bei der Programmierung aufweisen, als Probanden, die ein vorgegebenes Modell zur Programmierung verwendeten. Die durchschnittliche Fehlerrate in Versuchsdesign 2 ist um

fast 50 % höher (Tab. 6.3). Die Signifikanz liegt deutlich unter dem festgelegten Signifikanzniveau.

**These 1b ist wahr.**

Auch hier wird die erweiterte Analyse getrennt nach Versuchsgruppen durchgeführt, um den Einfluss der Modellierungsnotationen bewerten zu können (Tab. 6.4).

*Tab. 6.3 - Durchschnittliche Fehlerrate in Abhängigkeit des Versuchsdesigns*

| Versuchsdesign | Fehlerate | Signifikanz |
|---|---|---|
| VD1 - Modellieren und Programmieren | 43,84% | <0,01 |
| VD2 - Programmieren nach vorgegebenem Modell | 63,96% | |

In allen drei Gruppen führt die selbst durchgeführte Analyse und Modellierung zu besseren Ergebnissen. Es zeigen sich jedoch in Abhängigkeit der Art der Modellierungsnotation deutliche Differenzen im Ausmaß der Beeinflussung. In der UML-Gruppe zeigen sich lediglich geringere Unterschiede. Das Signifikanzniveau wird jedoch überschritten, sodass die These für diese Gruppe nicht belegt werden kann. Die ICL-Gruppe sowie die Kontrollgruppe weisen deutliche Unterschiede zwischen den beiden Versuchsdesigns auf und können aufgrund der errechneten Signifikanz die These für diese Gruppen belegen.

**Die Detaillierung von These 1b nach Gruppen ist wahr für die Gruppe der ICL sowie für die Kontrollgruppe.**

*Tab. 6.4 - Durchschnittliche Fehlerrate in Abhängigkeit des Versuchsdesigns und getrennt nach Modellierungsnotation*

| Modellierungsnotation | Fehlerrate Versuchsdesign 1 | Fehlerrate Versuchsdesign 2 | Signifikanz |
|---|---|---|---|
| Unified Modelling Language | 42,69% | 49,87% | 0,27 |
| Idiomatic Control Language | 45,31% | 70,40% | < 0,01 |
| Kontrollgruppe | 41,82% | 69,71% | < 0,01 |

### 6.2.2 Diskussion

Die Ergebnisse zu den Thesen 1a und 1b belegen, dass eine selbst durchgeführte Analyse des Prozesses sich förderlich auf die Programmierleistung auswirkt. In allen Gruppen erzielten Probanden aus Versuchsdesign 1 durchschnittlich bessere Ergebnisse, jedoch sind diese nicht in jedem Fall signifikant, wenn einzeln nach Gruppen analysiert wird.

Im Vergleich der untersuchten Notationen zeigt sich ein Vorteil für die ICL, da die Ergebnisse in dieser Gruppe sowohl für die benötigte Zeit als auch für die Fehlerrate signifikant sind. Obschon sich in der Gruppe der UML auch Vorteile in Versuchsdesign 1 erkennen lassen, sind diese jedoch nicht gleichermaßen signifikant wie in der ICL-Gruppe. Dies scheint vor allem an der vergleichsweise guten Fehlerrate der UML-Probanden in Versuchsdesign 2 zu liegen. Dieses Ergebnis kann einerseits dafür sprechen, dass ein UML-Modell besser in Steuerungscode übersetzt werden kann als ein ICL-Modell. Die fehlende Signifikanz der entsprechenden Varianzanalyse macht es jedoch notwendig diesen Aspekt weitergehend zu untersuchen. Andererseits kann das Ergebnis auch auf eine schlechtere Qualität der in der UML-Gruppe erstellten Modelle zurück zu führen sein.

Besonders interessant im Bezug auf die Thesen 1a und 1b ist die Tatsache, dass auch in der Kontrollgruppe bessere Ergebnisse in Versuchsdesign 1 erzielt wurden. Obwohl den Probanden in Versuchsdesign 2 eine detaillierte Funktionsbeschreibung zur Verfügung gestellt wurde, erzielten sie ein schlechteres Ergebnis. Dies ist möglicherweise darin begründet, dass bei vorhanden sein einer vorgegebenen Beschreibung eher darauf zurück gegriffen wird anstatt den Prozess selbstständig zu analysieren, was zu einem schlechteren Prozessverständnis führt. Auch dies spricht für einen Vorteil der selbst durchgeführten Systemanalyse.

## 6.3 Einfluss von Vorkenntnissen auf die Programmierleistung

### 6.3.1 Ergebnisse

Die Vorkenntnisse im Bezug auf die Programmiersprachen der IEC 61131-3 haben ebenso wie die praktische Erfahrung bei der Programmierung von SPS-Systemen einen Einfluss auf die Programmierleistung. Im Rahmen der Eingangsbefragung wurden die Vorkenntnisse der Probanden im Hinblick auf ihre SPS-Programmierkenntnisse abgefragt. Dabei wurde eine Selbsteinschätzung der Vorkenntnisse bezüglich der Programmierung in der IEC 61131-3 auf einer Skala von eins (gar keine Vorkenntnisse) bis fünf (sehr gute Vorkenntnisse) gefordert.

***These 2a:*** *Probanden, die eine höhere Selbsteinschätzung ihrer Vorkenntnisse abgeben, erzielen eine höhere Anzahl bearbeiteter Schritte und eine geringere Fehlerrate.*

Die Auswertung des Fragebogens zeigt eine verhältnismäßig niedrige Bewertung der eigenen Kenntnisse. In beiden Versuchen stuften die meisten Probanden ihr Vorwissen als gering oder gar nicht vorhanden ein. Lediglich vier Probanden gaben an gute Kenntnisse zu besitzen, keiner stufte seine Kenntnisse als sehr gut ein.

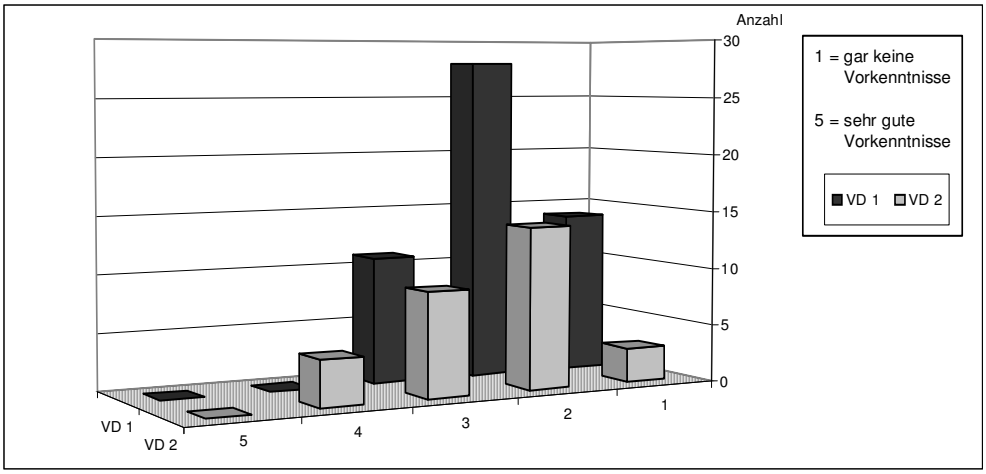



*Abb. 6.1 - Verteilung der Bewertung der Vorkenntnisse getrennt nach Versuchsdesign*

Ähnliche Ergebnisse zeigt die nach Probandengruppen aufgeschlüsselte Bewertung der Vorkenntnisse in den Fragebögen (Tab. 6.5). Hier erfolgt neben der nach der Qualifikation getrennten Betrachtung eine weitere Unterscheidung der Bachelor-Gruppe. Dies ist für einen nach Versuchsdesign getrennte Analyse der Ergebnisse notwendig, da diese Gruppe sowohl Versuchsdesign 1 als auch für Versuchsdesign 2 Probanden stellte. In der Analyse zeigt sich, dass der Mittelwert für die Bewertung der Vorkenntnisse in der ersten Gruppe der Bachelorstudenten (VD1) und in der Gruppe der Technikerschüler deutlich niedriger ist als in den restlichen Gruppen. Die Unterschiede sind jedoch nicht signifikant.

*Tab. 6.5 - Durchschnittliche Bewertung der Vorkenntnisse der Probanden getrennt nach Probandengruppen*

| Probandengruppe | Bewertung der eigenen Kenntnisse in der Programmierung |
|---|---|
| Student Bachelor (Gruppe 1) | 1,88 |
| Student Bachelor (Gruppe 2) | 2,44 |
| Student Master | 2,14 |
| Auszubildender (3. Lehrjahr) | 2,50 |
| Student im Praxisverbund (3. Jahr) | 2,27 |
| Technikerschüler (3. Jahr) | 1,56 |

Die Varianzanalyse zeigt in beiden Versuchskonfigurationen keine signifikanten Zusammenhänge zwischen der subjektiven Bewertung der eigenen Vorkenntnisse und der Programmierleistung. Sowohl bei der Analyse im Bezug auf die Anzahl bearbeiteter Schritte als auch auf die Fehlerrate wird das 10%-Niveau deutlich überschritten.

**These 2a ist nicht wahr.**

***These 2b:*** *Probanden, die bereits praktische Erfahrung gesammelt haben, erzielen eine höhere Anzahl bearbeiteter Schritte.*

Die Analyse der Anzahl bearbeiteter Schritte weist, unter Berücksichtigung der verwendeten Modellierungsnotation, für beide Versuchsdesigns einen signifikanten Zusammenhang zur praktischen Erfahrung auf (Tab. 6.6). Entgegen der These zeigt die Analyse über alle Gruppen jedoch, dass sich die praktische Erfahrung negativ auf die benötige Zeit auswirkt und Probanden mit praktischer Erfahrung unabhängig von der Aufgabe (Design 1 ⇔ Design 2) deutlich weniger Schritte bearbeiten.

In der nach Modellierungsnotation getrennten Betrachtung der Ergebnisse zeigt sich ein ähnliches Bild. In allen Gruppen programmierten Probanden ohne praktische Erfahrung in derselben Zeit mehr Schritte. Dabei zeigt sich im Vergleich der Versuchsdesigns für Probanden ohne Erfahrung nur in der UML-Gruppe ein deutlicher Unterschied. In der ICL-Gruppe sowie in der Kontrollgruppe zeigt der Vergleich der Versuchsdesigns nur Unterschiede für Probanden mit praktischer Erfahrung.

**These 2b ist nicht wahr.**

*Tab. 6.6 - Anzahl bearbeiteter Schritte in Abhängigkeit der praktischen Erfahrung getrennt nach Versuchsdesign*

| Modellierungsnotation | Versuchsdesign 1 Modellieren und Programmieren | | | Versuchsdesign 2 Programmieren nach vorgegebenem Modell | | |
|---|---|---|---|---|---|---|
| | Ohne praktische Erfahrung | Mit praktischer Erfahrung | Sig. | Ohne praktische Erfahrung | Mit praktischer Erfahrung | Sig. |
| Analyse über alle Gruppen | 20,74 | 15,00 | 0,05 | 20,75 | 9,50 | 0,02 |
| Unified Modelling Language | 19,00 | 11,33 | 0,09 | 12,00 | 9,43 | 0,54 |
| Idiomatic Control Language | 21,73 | 15,40 | 0,04 | 20,25 | 8,00 | 0,06 |
| Kontrollgruppe | 21,50 | 16,80 | 0,15 | 24,00 | 10,17 | 0,01 |

***These 2c:*** *Probanden, die bereits praktische Erfahrung gesammelt haben, erzielen eine geringere Fehlerrate.*

Die Analyse der Fehlerrate zeigt einen Zusammenhang zur praktischen Erfahrung. Dabei fällt gemessen über alle Gruppen auf, dass sich die praktische Erfahrung in Versuchsdesign 1 positiv auf die Fehlerrate auswirkt (Tab. 6.7- linker Teil). Probanden mit praktischer Erfahrung erzielen bei selbst durchgeführter Modellierung eine deutlich niedrigere Fehlerrate als Probanden, die noch keine praktische Erfahrung gesammelt haben.

Getrennt nach Modellierungsnotation ergeben sich ähnliche Ergebnisse. Dabei weisen unerfahrene Probanden in allen drei Versuchsgruppen ähnlich hohe Fehlerraten mit Werten zwischen 45% und 50% auf. Die Verringerung der Fehlerrate bei Probanden mit praktischer Erfahrung ist jedoch von der verwendeten Modellierungsnotation abhängig. In der UML-Gruppe erzielen Probanden mit praktischer Erfahrung eine durchschnittliche Fehlerrate von nur 8,39%. Im Vergleich dazu zeigt sich in der ICL-Gruppe für diese Personengruppe eine deutlich höhere Fehlerrate von 36,19%. In der Kontrollgruppe liegt die durchschnittliche Fehlerrate mit 18,41% zwischen den Werten der beiden Modellierungsgruppen.

*Tab. 6.7 - Fehlerrate in Abhängigkeit der praktischen Erfahrung getrennt nach Versuchsdesign*

| Modellierungsnotation | Versuchsdesign 1 Modellieren und Programmieren | | | Versuchsdesign 2 Programmieren nach vorgegebenem Modell | | |
|---|---|---|---|---|---|---|
| | Ohne praktische Erfahrung | Mit praktischer Erfahrung | Sig. | Ohne praktische Erfahrung | Mit praktischer Erfahrung | Sig. |
| Analyse über alle Gruppen | 48,70% | 21,60% | 0,02 | 57,36% | 68,36% | 0,24 |
| Unified Modelling Language | 46,47% | 8,39% | 0,01 | 49,04% | 50,54% | 0,74 |
| Idiomatic Control Language | 49,68% | 36,19% | 0,22 | 73,95% | 69,38% | 0,95 |
| Kontrollgruppe | 50,14% | 18,41% | 0,02 | 57,38% | 82,03% | 0,03 |

Ein abweichendes Verhalten zeigt sich in Versuchsdesign 2 (Tab. 6.7- rechter Teil). Die Fehlerraten liegen durchschnittlich über denen in Versuchsdesign 1 und Probanden ohne praktische Erfahrung erzielen im Durchschnitt eine geringere Fehlerrate. Die Aufschlüsselung nach Notation zeigt zwar, dass die Kontrollgruppe hauptverantwortlich für den Anstieg der durchschnittlichen Fehlerrate bei den erfahrenen Probanden ist, da hier mit 82,03% eine deutliche negative Abweichung zu erkennen ist. Jedoch weisen auch die UML- und die ICL-Gruppe nicht dieselben positiven Tendenzen wie in Versuchsdesign 1 auf. In beiden Gruppen gibt es lediglich geringe, nicht signifikante Unterschiede zwischen erfahrenen und unerfahrenen Probanden.

**These 2c ist wahr für Versuchsdesign 1.**

### 6.3.2 Diskussion

Die Ergebnisse zu These 2a zeigen keinen Zusammenhang zwischen der Selbsteinschätzung der Vorkenntnisse und der Programmierleistung. Dies ist möglicherweise darin begründet, dass die Probanden größtenteils angaben nur geringe Vorkenntnisse zu besitzen. Daraus könnte eine geringe Schwankung der unabhängigen Größe in der Varianzanalyse und die fehlende Signifikanz resultieren. Im Fall der Bachelorstudenten (Gruppe 1) und Masterstudenten konnte These 2a darüber hinaus mit den Ergebnissen einer SPS-bezogenen Aufgabe der Klausur im Fach Prozessinformatik überprüft werden. Jedoch zeigte sich auch hier, genau wie bei der testweisen Durchführung eines Eingangstests vor dem Versuch, kein Zusammenhang zwischen Klausur- bzw. Testergebnis und Programmierleistung. Deshalb wurde dieser Ansatz im vorliegenden Experiment nicht weiter verfolgt.

Dagegen kann der Einfluss der praktischen Erfahrung in Versuchsdesign 1 eingeschränkt nachgewiesen werden. These 2c hat sich in den Experimenten bestätigt und obschon die Ergebnisse der aufgestellten These 2b widersprechen, kann auch hier ein Vorteil festgestellt werden, wenn Zeit und Fehlerrate kombiniert betrachtet werden. Zwar haben Probanden mit Erfahrung eine geringere Anzahl an Schritten im Programm realisiert, sie weisen jedoch auch eine deutlich geringere Fehlerrate auf. Dies deutet darauf hin, dass erfahrene Programmierer, bedingt durch z.B. die Tatsache, dass sie eine bessere Vorstellung vom fertigen Programm und den möglichen Fehlerquellen besitzen, zwar langsamer dafür aber fehlerfreier Arbeiten. Bezogen auf ein komplettes Programmierprojekt könnte sich dadurch ein geringerer Nachbearbeitungsaufwand und somit ein besseres Programmierergebnis ergeben. In den Evaluationsexperimenten stand den Probanden jedoch nur eine begrenzte

Zeit zur Verfügung. Es war nicht Ziel das komplette Programm fehlerfrei zu erstellen. Das Debugging wurde in diesem Experiment nicht untersucht.

## 6.4 Einfluss der Qualität eines selbst erstellten Modells auf die Programmierleistung

### 6.4.1 Ergebnisse

Die Qualität eines Modells kann als ein Maß für den Informationsgehalt, den das Modell für die Programmierung bereitstellt, benutzt werden. Je genauer ein Modell ein System beschreibt, desto mehr Informationen stehen dem Anwender bei der Programmierung zur Verfügung. Eine höhere Qualität des Modells führt folglich zu einem geringeren Zeitbedarf und einer niedrigeren Fehlerrate.

Bei der Auswertung des Experiments wurde geprüft, ob bestehende Metriken zur Bewertung der Modelle heran gezogen werden können. Die Problematik bestand dabei jedoch darin, die bestehenden Metriken gleichermaßen auf die UML- als auch auf die ICL-Modelle anzuwenden. Darüber hinaus zeigte die Auswertung der Modelle, dass diese sich zumeist auf die Beschreibung des Systemverhaltens beschränkten, sodass die vorhandenen Metriken nur bedingt Anwendung finden konnten. Aus diesem Grund wurden die Modelle in ein an das Experiment angepasstes Schema eingeordnet, das im folgenden kurz beschrieben wird.

Die Analyse der erstellten Modelle zur Verhaltensbeschreibung zeigte die Notwendigkeit der Klassifizierung der Modelle nach ihrem Detaillierungsgrad. Es konnten zwei Formen der Detaillierung identifiziert werden, die im Folgenden als „grob“ und „detailliert“ bezeichnet sind. Die Klassifizierung erfolgt nach den folgenden Regeln:

a) Ein Modell wird als grob strukturiert bezeichnet, wenn einzelne Schritte zu Schrittfolgen zusammengefasst sind und diese im weiteren Verlauf nicht weiter detailliert werden. In einem groben Modell wird beispielsweise ein Schritt „Werkstück aufnehmen“ beschrieben, obwohl es sich dabei, gemäß der Mustermodellierung und auch im Bezug auf die Programmierung, um eine Schrittfolge aus den Einzelschritten „Kran senken“, Werkstück ansaugen“ und „Kran heben“ handelt.

b) Ein Modell gilt als detailliert strukturiert, wenn die einzelnen Schritte in ihrer Granularität denen der Mustermodellierung entsprechen und nur jeweils eine einzelne Aktion wie beispielsweise „Kran senken“, Werkstück ansaugen“ oder „Kran heben“ beschreiben. Ein detailliertes Modell darf dennoch die unter a)

beschriebenen Schrittfolgen enthalten, wenn diese im weiteren Verlauf der Modellierung verfeinert wurden (modulare Modellierung).

Die Bewertung der Qualität im Bezug auf die Detaillierung erfolgt auf Basis einer Ordinalskala. Eine mathematische Berechnung einer skalaren, quantitativen Größe für die Qualität des Modells ist aufgrund komplexer und kaum beschreibbarer Berechnungsgrundlagen nicht möglich. Qualitativ kann jedoch festgestellt werden, dass ein detailliertes Modell im Vergleich zu einen groben Modell einen höheren Informationsgehalt aufweist. Ein detailliertes Modell ist folglich als qualitativ höherwertig einzustufen als ein grobes Modell.

Der Detaillierungsgrad stellt im Bezug auf eine Modellierung jedoch nur einen Aspekt dar, der die Qualität eines Modells beeinflusst (vergl. Kapitel 4.3.1). Ein weiteres Unterscheidungsmerkmal war die unterschiedliche Ausprägung der Transitionsbedingungen. In den analysierten Modellen konnten 4 Varianten identifiziert werden (Abb. 6.2):

a) Transitionen ohne jegliche Beschreibung: Es wurden lediglich gerichtete Verbindungen zwischen Schritten oder Zuständen definiert, ohne eine Bedingungen für den Übergang zu spezifizieren.

b) Transitionen mit grober linguistischer Beschreibung: Die Bedingungen für einen Übergang zwischen Schritten oder Zuständen wurden in natürlicher Sprache beschrieben. Dabei wurde die Beschreibung grob gehalten und orientiert sich nicht direkt an den verfügbaren Prozesswerten (Sensoren und Aktoren im Prozess, die als Ein- oder Ausgangsvariablem für die Programmierung zur Verfügung stehen).

c) Transitionen mit detaillierter linguistischer Beschreibung: Die Bedingungen für einen Übergang zwischen Schritten oder Zuständen wurden in natürlicher Sprache beschrieben, sind im Gegensatz zu b) jedoch detailliert und orientieren sich dabei an den verfügbaren Prozesswerten.

d) Transitionen mit Beschreibung durch Variablen: Die Bedingungen wurden durch die logische Verknüpfung der Zustände von relevanten Variablen beschrieben.

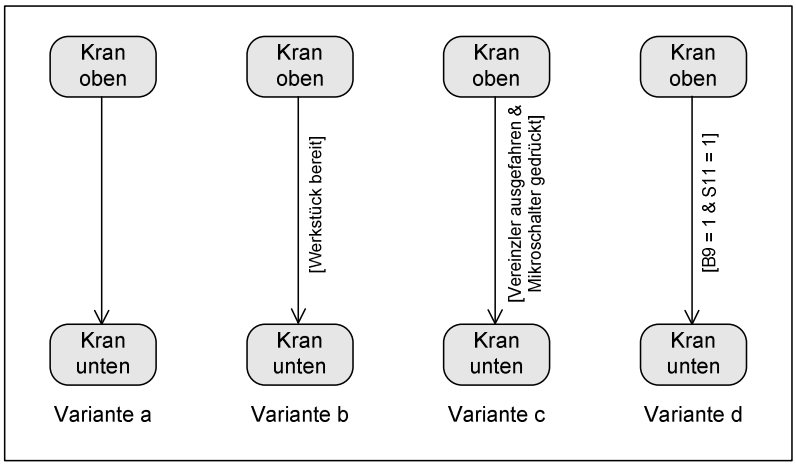



*Abb. 6.2 - Varianten der Transitionsbeschreibung am Beispiel der UML*

Auch die Art der Beschreibung von Transitionen hat einen Einfluss auf die Qualität der Modellierung, jedoch kann auch hier lediglich eine qualitative Bewertung durch die Aufstellung einer Rangfolge vorgenommen werden. Die einzelnen Klassen werden in der Reihenfolge ihrer Nennung (a➔d) einer steigenden Qualität des Modells zugeordnet. Die Beschreibung mit Hilfe von Variablen wird dabei am höchsten eingestuft, da sie vollständig ist und der Transfer vom linguistischen Ausdruck auf die entsprechende Prozessgroße und deren Zustand bereits enthalten ist.

Auf Basis dieser festgelegten Unterscheidungsmerkmale der Modelle werden im Folgenden der Zusammenhang zwischen der Qualität des Modells und der Programmierleistung sowie die Evaluation weiterer Qualitätsbezogener Thesen durchgeführt.

***These 3a***: *Probanden, die eine detaillierte Modellierung des Prozesses erstellt haben, benötigen weniger Zeit für die Programmierung und erzielen eine höhere Anzahl bearbeiteter Schritte im Programm.*

Die Untersuchung zur Abhängigkeit der Programmierleistung im Bezug auf die Qualität des erstellten Modells erfolgt zunächst nach der bereits hergeleiteten Unterscheidung der Modelle nach dem Detaillierungsgrad. Es zeigt sich sowohl für die gemeinsame als auch für die nach Modellierungsnotation getrennte Analyse ein leichter Vorteil für Probanden mit grobem Modell. Die Ergebnisse sind jedoch nicht signifikant (Tab. 6.8), da in allen Fällen ein p-Wert errechnet wird, der über dem Grenzwert von $\alpha$=0,2 liegt. Folglich kann These 3a im Bezug auf die Detaillierung des Modells nicht belegt werden.

*Tab. 6.8 - Durchschnittliche Anzahl bearbeiteter Schritte in Abhängigkeit der Modellierungsdetaillierung*

| | Modell "grob" | Modell "detailliert" | Signifikanz |
|---|---|---|---|
| Analyse über alle Gruppen | 19,43 | 17,84 | 0,71 |
| UML | 18,63 | 16,00 | 0,67 |
| ICL | 20,50 | 18,92 | 0,86 |

Die weitergehende Analyse der Daten im Bezug auf Abhängigkeit des Programmierergebnisses von der Ausprägung der Beschreibung von Transitionen zeigt ebenfalls keine signifikanten Ergebnisse. Sowohl für die gemeinsame als auch für die nach Notation getrennte Analyse wird jeweils ein p-Wert von über 0,2 errechnet. Folglich kann These 3a im Bezug auf die Ausprägung der Beschreibung von Transitionen nicht belegt werden.

**These 3a ist nicht wahr.**

***These 3b:*** *Probanden, die eine detaillierte Modellierung des Prozesses erstellt haben, erzielen bei der Programmierung eine geringere Fehlerrate.*

Wie zu These 3a erfolgt die Untersuchung zunächst in Bezug auf die Detaillierung des erstellten Modells (Tab. 6.9). Diese erbringt jedoch keine signifikanten Ergebnisse. Deshalb erfolgt auch hier eine nach Modellierungsnotation getrennte Analyse. Diese zeigt für die ICL-Gruppe nur geringe Unterschiede in der durchschnittlichen Fehlerrate zwischen groben und detaillierten Modellen und somit keine signifikanten Ergebnisse. Hingegen erzielen in der UML-Gruppe Probanden, die ein detailliertes Modell erstellten, eine deutlich geringere durchschnittliche Fehlerrate. Die errechnete Signifikanz liegt über der Grenze von $\alpha$=0,1 jedoch noch in einem Bereich, der eine Wertung als Tendenz zulässt.

*Tab. 6.9 - Durchschnittliche Fehlerrate in Abhängigkeit der Modellierungsdetaillierung*

| | Modell "grob" | Modell "detailliert" | Signifikanz |
|---|---|---|---|
| Analyse über alle Gruppen | 46,50% | 41,90% | 0,29 |
| UML | 44,30% | 33,75% | 0,19 |
| ICL | 49,75% | 47,33% | 0,85 |

Ein ähnliches Bild ergibt sich bei der Analyse der Fehlerrate in Abhängigkeit der Ausprägung der Beschreibung von Transitionen (Tab. 6.10). Die Gesamtanalyse, genauso wie die Analyse der ICL-Gruppe, ergibt keine signifikanten Ergebnisse. In der UML-

Gruppe zeigt sich dagegen, dass Probanden mit steigender Wertigkeit der Transitionsbeschreibung eine geringere Fehlerrate erzielen. Einzige Ausnahme bildet die Fehlerrate für die Klasse „linguistisch detailliert". Dieses Ergebnis ist aufgrund der Tatsache, dass dieser Klasse nur ein Proband zugeordnet werden konnte, nicht als zuverlässig zu werten. Die errechnete Signifikanz in der UML-Gruppe liegt bei p=0,161, sodass dieses Ergebnis als Tendenz angesehen wird.

**These 3b ist nicht wahr, in der UML-Gruppe lässt sich jedoch eine Tendenz zu These 3b erkennen.**

*Tab. 6.10 - Durchschnittliche Fehlerrate in Abhängigkeit der Ausprägung der Beschreibung von Transitionen*

| | keine Beschreibung | grob linguistisch | detailliert linguistisch | mit Variablen | Signifikanz |
|---|---|---|---|---|---|
| Analyse über alle Gruppen | 50,26% | 45,58% | 47,63% | 37,09% | 0,37 |
| UML | 51,95% | 40,65% | 60,00% | 30,21% | 0,16 |
| ICL | 49,42% | 56,13% | 35,27% | 42,98% | 0,66 |

***These 3c**:* *Probanden, die eine höhere Anzahl richtiger Schritte modellierten, erzielen auch bei der Programmierung eine höhere Anzahl richtiger Schritte.*

Zur Evaluation der These wird, getrennt nach Detaillierungsgrad des erstellten Modells, die Korrelation zwischen der Anzahl richtig modellierter Schritte und der Anzahl richtig bearbeiteter Schritte im Programm berechnet (Tab. 6.11). In der Klasse der „groben" Modelle kann kein Zusammenhang zwischen der Anzahl richtiger Schritte im Modell und richtiger Schritte im Programm nachgewiesen werden, da mit 0,317 keine Signifikanz vorliegt.

*Tab. 6.11 - Korrelation zwischen Anzahl richtig modellierter Schritte und Anzahl richtig bearbeiteter Schritte im Programm*

| | Modell "grob" | | | Modell "detailliert" | | |
|---|---|---|---|---|---|---|
| Korrelation zu … | Pearson-Koeffizient | Signifikanz | Anzahl Datensätze | Pearson-Koeffizient | Signifikanz | Anzahl Datensätze |
| Anzahl richtig bearbeiteter Schritte im Programm | 0,288 | 0,317 | 11 | 0,494 | 0,031 | 14 |

Im Gegensatz dazu korrelieren die beiden Größen in der Klasse der „detaillierten“ Modelle. Der errechnete positive Pearson-Koeffizient zeigt, dass eine höhere Anzahl richtig modellierter Schritte auch zu einer höheren Anzahl richtig programmierter Schritte führt. Eine anschließende lineare Regressionsanalyse ergibt Regressionskoeffizienten von $\alpha$ = 4,509 und $\beta$ = 0,318 (Koeffizienten der Geradengleichung $f(x)=\alpha+\beta*x$). Die errechnete Signifikanz liegt mit 0,031 deutlich unter dem Grenzwert, das Ergebnis ist signifikant.

**These 3c ist wahr für Modelle mit hoher Detaillierung.**

### 6.4.2 Diskussion

Die Ergebnisse zu These 3a zeigen keinen direkten Vorteil einer detaillierten Modellierung im Bezug auf den Zeitfaktor, da sich die Anzahl im Programm realisierter Schritte in diesen beiden Klassen nicht signifikant unterscheidet. Dennoch weisen die Analysen zu den Thesen 3b und 3c darauf hin, dass zumindest bei Verwendung der UML durch die detaillierte Beschreibung des Systems ein Vorteil erzielt werden kann. In dieser Gruppe führte sowohl eine höhere Detaillierung der Modelle im Bezug auf die Granularität der Zustände im Zustandsdiagramm als auch eine höhere Detaillierung der Beschreibung von Transitionsbedingungen tendenziell zu einem besseren Programmierergebnis. Die fehlende Signifikanz der Ergebnisse in der ICL-Gruppe kann dabei ein Hinweis auf eine bessere Eignung der UML sein, was jedoch in einer weiteren Untersuchung belegt werden muss.

Interessant ist auch der nachgewiesene Zusammenhang zwischen der Anzahl richtig modellierter Schritte und der Anzahl richtig bearbeiteter Schritte im Programm. Je mehr Schritte richtig modelliert wurden, desto mehr Schritte wurden anschießend auch richtig programmiert. Das bedeutet, dass durch eine Steigerung der Modellqualität auch die Programmqualität unmittelbar gesteigert werden kann. Der geringe Regressionskoeffizient $\beta$ belegt jedoch auch, dass nicht jeder Schritt trotz richtiger Modellierung auch richtig programmiert wurde. Dies spricht für Schwierigkeiten bei der Übertragung des Modells in das Programm und bietet einen Ansatzpunkt, den Nutzen der Modellierung weiter zu steigern.

## 6.5 Zusammenhang zwischen Notation und modellierten Aspekten

### 6.5.1 Ergebnisse

Die verwendeten Modellierungsnotationen enthalten Beschreibungsmöglichkeiten sowohl für strukturelle als auch für verhaltensbezogene Aspekte, deren mögliche Anwendung in den vorbereitenden Kursen vermittelt wurde. Folglich wird erwartet, dass den Probanden die Notwendigkeit beider Aspekte als Unterstützung für die Programmierung des Systems bewusst ist.

***These 4a:*** *Die erstellten Modelle beinhalten sowohl eine strukturbezogene als auch eine verhaltensbezogene Beschreibung des Systems.*

Die Analyse der erstellten Modelle nach der Art der verwendeten Diagramme (Abb. 6.3) zeigt, dass die Probanden fast ausschließlich das Verhalten des zu automatisierenden Systems modellierten. In der Gruppe der UML erstellten zwei von 13 Probanden ein Strukturdiagram, alle Probanden erstellten eine Verhaltensbeschreibung. In der Gruppe der ICL wurde hingegen gar keine Strukturbeschreibung erstellt. Alle Probanden erstellten ausschließlich eine Verhaltensbeschreibung.

| **Anzahl der UML-Gruppen:** | **13** |
|---|---|
| *- Strukturdiagramme* | *2* |
| - Klassendiagramme | 2 |
| *- Verhaltensdiagramme* | *13* |
| - Zustandsdiagramme | 12 |
| - Sequenzdiagramme | 1 |

| **Anzahl der ICL-Gruppen** | **13** |
|---|---|
| *- Strukturdiagramme* | *0* |
| *- Verhaltensdiagramme* | *13* |
| - Sequential function charts | 13 |

*Abb. 6.3 - Auswertung der modellierten Aspekte getrennt nach Modellierungsnotation*

Die Abweichungen vom Ergebnis der UML-Gruppe zur ICL-Gruppe sind nur geringfügig und nicht signifikant. Deshalb kann trotz der Tatsache, dass ausschließlich bei Verwendung der UML die Struktur modelliert wurde nicht auf eine Abhängigkeit von der verwendeten Notation geschlossen werden, sondern dies lediglich als Tendenz gewertet werden. Gleiches gilt für die Abhängigkeit von der Qualifikation der Probanden. Je ein Kandidat der Bachelor-Gruppe und der Gruppe der StiPs erstellte ein Klassendiagramm. Jedoch kann auch in dieser Hinsicht kein signifikanter Zusammenhang nachgewiesen werden.

**These 4a ist nicht wahr.**

***These 4b:*** *Die Anwendbarkeit der strukturellen und der verhaltensbezogenen Modellierung für die Erstellung des Steuerungsprogramms wird hoch bewertet.*

Im Rahmen des Fragebogens mussten die Probanden die Anwendbarkeit der strukturellen und der verhaltensbezogenen Modellelemente für die Steuerungsprogrammierung auf einer Skala von 1 (gar nicht geeignet) bis 5 (sehr gut geeignet) bewerten. Die Analyse der Ergebnisse erfolgt getrennt nach Versuchsdesigns.

Das Ergebnis für das Versuchsdesign 1 (Tab. 6.12) zeigt, dass die Bewertung für beide Beschreibungsformen im Durchschnitt jeweils unter der Grenze von 4,0 liegt und somit nicht als hoch eingestuft wird. Die Anwendbarkeit der Strukturbeschreibung wird dabei in beiden Gruppen extrem niedrig bewertet. Die durchschnittliche Bewertung in der ICL-Gruppe ist geringfügig höher, die Unterschiede zur UML-Gruppe sind jedoch nicht signifikant. Ein ähnliches Ergebnis zeigt sich für die Bewertung der der Verhaltensbeschreibung. Auch hier ist die durchschnittliche Bewertung in der ICL-Gruppe geringfügig höher, der Unterschied ist jedoch auch hier nicht signifikant.

Im Vergleich der Bewertung des Nutzens der Struktur- und der Verhaltensbeschreibung zeigen sich deutliche Unterschiede. In beiden Gruppen wird die Verhaltenbeschreibung höher bewertet, ist jedoch nicht als hoch einzustufen.

*Tab. 6.12 - Bewertung der Anwendbarkeit von Struktur- und Verhaltensbeschreibung in Versuchsdesign 1*

| Notation | Anwendbarkeit der Strukturbeschreibung | Anwendbarkeit der Verhaltensbeschreibung |
|---|---|---|
| UML | 1,33 | 3,07 |
| ICL | 1,71 | 3,60 |

Bewertung: 1=nicht geeignet, 5=sehr gut geeignet

Betrachtet man ausschließlich die Gruppe der Probanden, die bei der Modellierung auch ein Klassendiagramm erstellten, so zeigt sich im Bezug auf die Bewertung der Strukturbeschreibung nur eine geringe Abweichung gemessen an der Gesamtheit der Probanden. Die mittlere Bewertung in dieser Gruppe liegt mit 2,0 zwar etwas höher als in der Gesamtheit, die geringe Anzahl an Probanden, die die Struktur modellierten, führt jedoch dazu, dass das Ergebnis nicht signifikant und auch nicht interpretierbar ist.

**These 4b ist nicht wahr für Versuchsdesign 1.**

Versuchsdesign 2 zeigt abweichende Ergebnisse (Tab. 6.13). Die Bewertung der Strukturbeschreibung ist in Design 2 deutlich höher, der Mittelwert liegt für beide Gruppen jedoch unter dem Grenzwert und ist folglich nicht als hoch anzusehen. Auch die Bewertung der Verhaltensbeschreibung zeigt im Vergleich zu Design 1 durchschnittlich eine höhere Bewertung, die mit Mittelwerten von 3,89 bei beiden Modellierungsnotationen jedoch noch knapp unter der festgelegten Grenze von 4,0. Im Vergleich der UML mit der ICL zeigen sich in der Bewertung weder für die Strukturbeschreibung noch für die Verhaltensbeschreibung signifikante Unterschiede, die auf die verwendete Notation zurückzuführen sind.

*Tab. 6.13 - Bewertung der Anwendbarkeit von Struktur- und Verhaltensbeschreibung in Versuchsdesign 2*

| Notation | Anwendbarkeit der Strukturbeschreibung | Anwendbarkeit der Verhaltensbeschreibung |
|---|---|---|
| UML | 3,00 | 3,89 |
| ICL | 2,78 | 3,89 |

Im Vergleich der Bewertung des Nutzens der Struktur- und der Verhaltensbeschreibung zeigen wie in Design 1 signifikante Ergebnisse. In beiden Gruppen wird die Verhaltenbeschreibung deutlich höher bewertet.

**These 4b ist nicht wahr für Versuchsdesign 2.**

### 6.5.2 Diskussion

Die Ergebnisse zu den Thesen 4a und 4b in Versuchsdesign 1 belegen, dass den Probanden der Nutzen einer strukturbezogenen Beschreibung des Systems nicht bewusst war. Die geringe Anzahl an erstellten Strukturdiagrammen spiegelt sich in der durchschnittlich sehr geringen Bewertung der Anwendbarkeit dieser Beschreibung für die Programmerstellung wieder. Dabei wird deutlich, dass hier keine Abhängigkeit zu einer der verwendeten Modellierungsnotationen existiert. Obschon in den Vorlesungen und den dargestellten Beispielen darauf hin gearbeitet wurde die Notwendigkeit einer umfassenden Beschreibung zu verdeutlichen, scheint es eine allgemeine Einschätzung zu sein, dass für die Programmerstellung, also die Realisierung eines Systemverhaltens, auch nur das Verhalten und nicht die Struktur des Systems beschrieben werden muss. Dies kann durch ein Ergebnis von Détienne [Dét95] erklärt werden, wonach Programmierer, die im Bezug auf objektorientierte Programmierung eine geringe Erfahrung besitzen, die Beschreibung von Objekten und Funktionen zumeist getrennt sehen. Folglich konzentrierten sich die

Probanden im vorliegenden Versuch meist auf die Verhaltensbeschreibung, da sie die vorgabengerechte Funktion des Prozesses in den Vordergrund stellten.

Aufgrund der geringen Anzahl an Probanden, die eine strukturbezogene Systembeschreibung erstellten, konnte in den Experimenten kein messbarer Vorteil dieses Teils einer Modellierung nachgewiesen werden.

Interessant ist in diesem Zusammenhang auch, dass die Anwendbarkeit der strukturbeschreibenden Diagramme in Versuchsdesign 2 deutlich besser bewertet wurde. Hier hat die Tatsache, dass eine fertige Strukturbeschreibung zur Verfügung gestellt wurde, die Anwendbarkeit dieser Beschreibungsform signifikant positiv beeinflusst. Insgesamt zeigt sich eine, wenn auch teilweise nur geringfügig höhere Bewertung des Nutzens der Strukturmodellierung in Versuchsdesign 2. Dies kann generell als Beleg für die von den Probanden wahrgenommenen Probleme bei der Erstellung der Modellierung gewertet werden.

## 6.6 Einfluss der Modellierung auf die Modularität des Programms

### 6.6.1 Ergebnisse

Durch den objektorientierten Ansatz fördert die Unified Modelling Language eine modulare Modellierung des Steuerungsprogramms und die Wiederverwendung von Modulen in einem stärkeren Maße als der hierarchische Ansatz der ICL.

***These 5:*** *Der Anteil modular erstellter Modelle und Steuerungsprogramme ist bei Verwendung der UML höher als bei Verwendung der ICL sowie in der Kontrollgruppe.*

Diese These konnte ausschließlich in Versuchsdesign 1 evaluiert werden und wurde durch die Inspektion der erstellten Modelle und des Programmcodes bewertet. Die Ergebnisse zeigen einen relativ geringen Anteil an Probanden, die ein modulares Modell erstellt haben oder das Steuerungsprogramm modular realisiert haben. Tab. 6.14 zeigt, getrennt nach Modellierungsnotation und Qualifikation, wie viele Probanden jeweils modular modellierten und modular programmierten.

*Tab. 6.14 - Auswertung der Modularität in Modell und Programm getrennt nach Modellierungsnotation und Qualifikation*

| Modellierungsnotation | Bachelor | | Master | | StiPs | | Techniker | | Gesamt | |
|---|---|---|---|---|---|---|---|---|---|---|
| | modular modelliert | modular programmiert | modular modelliert | modular programmiert | modular modelliert | modular programmiert | modular modelliert | modular programmiert | modular modelliert | modular programmiert |
| Unified Modelling Language | 1 / 4 | 1 / 4 | 0 / 2 | 0 / 2 | 1 / 3 | 1 / 3 | 1 / 3 | 2 / 3 | 3 / 12 | 4 / 12 |
| Idiomatic Control Language | 2 / 5 | 2 / 5 | 0 / 3 | 0 / 3 | 1 / 3 | 1 / 3 | 2 / 3 | 2 / 3 | 5 / 13 | 5 / 13 |
| Kontrollgruppe | --- | 0 / 5 | --- | 0 / 2 | --- | 0 / 3 | --- | 0 / 3 | --- | 0 / 13 |

x / y : Aussage trifft zu für x von insgesamt y Probanden

Die Betrachtung der erstellten Modelle zeigt, dass gemessen über alle Qualifikationsklassen in der UML-Gruppe lediglich drei von insgesamt 12 Probanden ein modulares Modell entwarfen. In der ICL-Gruppe wurden fünf von insgesamt 13 Modellen modular ausgeführt. Der notationsbegründete Unterschied ist jedoch nicht signifikant. Dieselben Ergebnisse zeigen sich bei der Analyse der erstellten Steuerungsprogramme. Auch hier gibt es keine signifikanten Unterschiede zwischen der UML- und der ICL-Gruppe.

Die dargestellten Ergebnisse zeigen, dass These 5 in den Experimenten nicht bestätigt werden kann. Auch die Betrachtung getrennt nach Qualifikation zeigt keine signifikanten Ergebnisse.

**These 5 ist nicht wahr.**

Es zeigen sich jedoch zwei interessante, über die formulierten Thesen hinausgehende Zusammenhänge. Zum einen ist auffällig, dass in keiner der Kontrollgruppen ein modularer Ansatz in der Steuerungsprogrammierung verfolgt wurde. Zum anderen besteht ein eindeutiger Zusammenhang zwischen der Erstellung eines modularen Modells und dem modularen Entwurf des Programms. Genau die Gruppen, die auch modular modelliert haben, realisierten später ein modulares Programm.

### 6.6.2 Diskussion

Die Auswertung zu These 5 zeigt, dass ein genereller Vorteil für eine modulare Modellierung durch die Verwendung einer objektorientierten Modellierungsnotation nicht nachgewiesen werden kann. Eine mögliche Ursache für dieses Ergebnis könnte die fehlende Erfahrung mit einem objektorientierten Ansatz sein. So konnten Basili et al. nachweisen, dass ein positiver Effekt sich bei der Einführung objektorientierter Ansätze

nicht direkt einstellt [BCG+92]. Ein spürbarer Vorteil wurde erst nach einem angepassten Training und mehrmaliger Anwendung nachgewiesen. Die Tatsache, dass die Probanden im Evaluationsexperiment mit der UML zumeist das erste Mal eine objektorientierte Modellierungsnotation anwendeten, führt somit dazu, dass der positive Effekt noch nicht deutlicher wirksam wird. Es ist deshalb notwendig in diesem Zusammenhang den Einfluss des Wiederholungseffektes in weiteren Arbeiten zu untersuchen.

Dennoch scheint die vor der Programmierung durchgeführte Modellierung generell dazu zu führen, Module eines Systems zu erkennen und diese sowohl im Modell als auch im Programm zu realisieren und wieder zu verwenden. Dies ist auf die detaillierte Analyse des Systems vor der Programmierung zurück zu führen. Diese war bei den Kontrollgruppen nicht explizit gefordert. So konnte beobachtet werden, dass zumeist relativ schnell zur Programmierung des Systems übergegangen wurde. Es wurde zwar im Rahmen der Experimente nicht erhoben, ob Probanden der Kontrollgruppe während der Programmierung gleichartige Module erkannten. Es ist jedoch zu vermuten, dass sie in einem solchen Fall ihre Programmstruktur nicht mehr ändern würden, da dies einen erhöhten Zeitaufwand bedeutet.

## 6.7 Einfluss der Modellierung auf die subjektive Wahrnehmung

### 6.7.1 Ergebnisse

Die Durchführung der Modellierung mit Hilfe eines geeigneten Beschreibungsmittels bewirkt bei den Probanden eine Reduzierung der subjektiv wahrgenommenen Komplexität der Aufgabe. Im Vergleich der Modellierungsnotationen wirkt sich der objektorientierte Ansatz der UML in einem höheren Maße aus, als der streng hierarchische Ansatz der ICL.

Probanden, die mit den Modellierungsnotationen vertraut sind, die Aufgabe jedoch ohne diese Unterstützung durchführen mussten, erkennen die Komplexität der Aufgabe. Die Bewertung der Schwierigkeit der Aufgabe fällt in den Kontrollgruppen folglich höher aus. Die Probanden erkennen den möglichen Nutzen einer Modellierung und bewerten den Nutzen für die Programmierung hoch.

***These 6a:*** *Die Schwierigkeit der Aufgabe wird in der UML-Gruppe am geringsten eingestuft, die Kontrollgruppe stuft sie am höchsten ein.*

Zur Evaluation der These wird der Fragebogen ausgewertet, der von den Teilnehmern nach dem Versuch ausgefüllt wurde. Die Probanden mussten darin unter anderem die Aussage „Das Lösen der Aufgabe war nicht schwer/schwer“ auf einer Skala zwischen eins (nicht

schwer) und fünf (sehr schwer) bewerten. Die statistische Auswertung zeigt sowohl in der Gesamtanalyse als auch in der nach Qualifikation unterteilten Analyse keine Signifikanz gemäß der festgelegten Signifikanzgrenze von $\alpha$=0,1 (Tab. 6.15). In allen Analysen, zeigen sich nur geringe Unterschiede zwischen den Mittelwerten der Bewertung, wodurch die fehlende Signifikanz erklärt werden kann. Einzig die Gruppe der StiPs zeigt deutliche Abweichungen. Hier bewerten die Probenden den Versuch ohne die Anwendung einer Modellierung im Durchschnitt mindestens einen Skalenpunkt schwieriger als solche die den Prozess modellieren. Die Ergebnisse sind jedoch nicht signifikant, da die errechnete Signifikanz sowohl für die gemeinsame als auch für die nach Qualifikation getrennte Betrachtung jeweils über dem festgelegten Grenzwert liegt.

**These 6a ist nicht wahr.**

*Tab. 6.15 - Bewertung der Schwierigkeit der Aufgabe getrennt nach Notation und Qualifikation*

| Probanden | UML | ICL | Kontrollgruppe | Signifikanz |
|---|---|---|---|---|
| Gesamt | 3,29 | 3,16 | 3,57 | 0,29 |
| Bachelor | 3,50 | 3,20 | 3,40 | 0,55 |
| Master | 3,00 | 2,67 | 3,00 | 0,81 |
| StiPs | 2,67 | 3,00 | 4,00 | 0,23 |
| Techniker | 3,75 | 3,67 | 3,50 | 0,92 |

***These 6b:*** *Der Nutzen einer Modellierung für das Lösen der Aufgabe wird in der UML-Gruppe höher eingestuft als in der ICL-Gruppe.*

Für diese These wurden die Bewertungen der vorgegebenen Aussagen „Die Modellierung hilft bei der Lösung der Aufgabe“ bzw. „Die Modellierung würde bei der Lösung der Aufgabe helfen“ (Kontrollgruppen) ausgewertet. Der Fragebogen ermöglichte auch hier eine Einstufung zwischen eins (keine Zustimmung) und fünf (volle Zustimmung). Dabei zeigt die Gesamtanalyse kein signifikantes Ergebnis. Erst bei der nach Qualifikationen getrennten Auswertung treten in der Gruppe der Bachelor- und Masterstudenten sowie bei den StiPs signifikante Ergebnisse auf.

Wie in Tab. 6.16 zu erkennen ist, schätzen Bachelor- und Masterstudenten die UML jeweils am schlechtesten ein, hier zeigt sich im Durchschnitt eine Bewertung die bei 3,0 bzw. 3,5 liegt. Für die ICL wird die durch die Modellierung gebotene Unterstützung durchschnittlich mindestens einen Skalenpunkt höher eingestuft und die errechneten Mittelwerte von 4,3 bzw. 4,36 sind, gemessen an der Gesamtskala, als eine hohe Zustimmung zu werten. In der Gruppe der StiPs zeigt sich eine abweichende Bewertung. Hier wird der Nutzen der

Modellierung in der Gruppe der UML durchschnittlich als hoch bewertet, der Mittelwert liegt bei 4,0. Deutlich schwächere Bewertungen zeigen sich für die ICL (3,3).

**These 6b ist wahr für die Gruppe der StiP's.**

*Tab. 6.16 - Bewertung des Nutzes der Modellierung für die Programmierung*

| Probanden | UML | ICL | Kontrollgruppe | Signifikanz |
|---|---|---|---|---|
| Gesamt | 3,65 | 4,05 | 3,86 | 0,29 |
| Bachelor | 3,50 | 4,36 | 4,60 | 0,02 |
| Master | 3,00 | 4,30 | 4,00 | 0,05 |
| StiPs | 4,00 | 3,30 | 3,00 | 0,10 |
| Techniker | 4,00 | 3,30 | 3,67 | 0,77 |

***These 6c:*** *Probanden ohne Modellierungsunterstützung stufen den möglichen Nutzen einer vorherigen Modellierung hoch ein.*

Im Rahmen des Fragebogens wurde den Probanden der Kontrollgruppen die Frage gestellt, ob eine vorherige Modellierung einen Nutzen bezüglich der Programmieraufgabe dargestellt hätte. Es wurde dabei nicht explizit nach einer der hier verglichenen Notationen gefragt. Die Bewertung des Nutzens über alle Probanden der Kontrollgruppe (Tab. 6.16) ergibt dabei eine durchschnittliche Bewertung, die gemäß den festgelegten Grenzen nicht als hoch einzustufen ist. Die nach Qualifikation getrennte Analyse zeigt ein uneinheitliches Bild. Während die durchschnittliche Bewertung der Bachelor und Master als hoch einzustufen ist, wird der mögliche Nutzen in der Gruppe der StiP's und der Techniker geringer eingestuft und bleibt mit Durchschnittswerten von 3,0 (StiP's) und 3,67 (Techniker) unter dem Grenzwert.

**These 6c ist wahr für Bachelor- und Master-Studenten.**

### 6.7.2 Diskussion

Die Schwierigkeit der Aufgabe wird fast ausschließlich hoch bewertet und zeigt dabei hinsichtlich These 6a keine signifikanten Unterschiede, die auf die Verwendung einer Modellierungsnotation zurück zu führen sind. Ein möglicher Grund dafür ist die meist erstmalige Anwendung einer Modellierung zum Zweck der Steuerungsprogrammierung. Zwar wurde in der Befragung nicht zwischen der Schwierigkeit des Modellierungsteils und des Programmierteils unterschieden, dennoch scheint es möglich, dass Probanden, die den Prozess modellierten, die Schwierigkeit der Aufgabe in diesem Teil sahen, während

Probanden der Kontrollgruppe die Schwierigkeit der Programmieraufgabe bewerteten. Eine getrennte Ermittlung der Schwierigkeitsbewertung für diese beiden Versuchsphasen sowie die Betrachtung des Lerneffektes durch wiederholte Anwendung einer Modellierung würden diesbezüglich eine genauere Analyse ermöglichen. Dies sollte in weiteren Arbeiten untersucht werden.

Die Auswertung der Kontrollgruppen hat gezeigt, dass nur in der Gruppe der Bachelor- und Masterstudenten der mögliche Nutzen einer vorherigen Modellierung als hoch eingestuft wurde. Dies scheint in den unterschiedlichen Vorkenntnissen der Probanden im Bereich der Modellierung begründet zu sein. Die Analyse der Fragebögen ergab, dass die studentischen Probanden während des Studiums in verschiedenen Fächern schon mehrfach mit dem Thema Modellierung und der damit verbundenen detaillierten Analyse eines Systems in Kontakt kamen. Die Probanden der restlichen Gruppen (Auszubildende, Techniker und StiPs) gaben hingegen, an nur geringe oder gar keine Vorkenntnisse im Bezug auf die Modellierung zu haben. Den Studenten war somit der Zweck einer Modellierung bewusster, wodurch sie den Nutzen höher einstuften.

Die Ergebnisse zu den Thesen 6 a)-c) zeigen, dass eine Modellierung generell als sinnvoll und bei der gestellten Aufgabe als nützlich angesehen wird. Das wird insbesondere durch die Bewertung der Probanden der Kontrollgruppe, die im Bereich der Modellierung mehr Erfahrung besaßen, deutlich. Die Tatsache, dass die verwendeten Modellierungsnotationen jedoch durchschnittlich keine hohen Bewertungen fanden, macht jedoch auch die Schwierigkeiten insbesondere bei der Anwendung der UML deutlich. Es ist denkbar, dass die uneingeschränkte UML hier als nicht angemessen für die Domäne Automatisierungstechnik angesehen wird. Ein möglicher Grund könnte die Diskrepanz zwischen der objekt-orientierten Modellierung der UML und einer nicht objekt-orientierten sondern funktionalen Programmierung der Steuerung sein.

## 6.8 Einfluss von Qualifikation auf die Qualität des erstellten Modells

### 6.8.1 Ergebnisse

Eine unterschiedliche Ausbildung der Anwender beeinflusst maßgeblich ihre Fähigkeit einen Prozess zu modellieren. Die verschiedenartige Ausbildung führt zu unterschiedlichen Qualitäten der Modelle in den einzelnen Probandengruppen.

***These 7:*** *Probanden mit verschiedenen Qualifikationen erstellen Modelle unterschiedlicher Qualität.*

Aus den in Kapitel 6.4 genannten Gründen erfolgt zunächst eine Unterteilung der Gesamtheit der Probanden gemäß dem Detaillierungsgrad ihres erstellten Modells und im Folgenden getrennte Evaluation der Thesen zum Einfluss der Qualität des Modells. Die statistische Auswertung der Modelle im Bezug auf den Detaillierungsgrad ist in Tab. 6.17 dargestellt.

*Tab. 6.17 - Aufteilung der Modelle nach Qualifikation der Probanden und der Detaillierung der Modelle*

| | Anzahl grobe Modelle | Anzahl detaillierte Modelle |
|---|---|---|
| Bachelor-Studenten | 5 | 6 |
| Master-Studenten* | 3 | 1 |
| Techikerschüler* | 3 | 3 |
| StiPs | 1 | 5 |

* je ein Modell der Gruppe nicht auswertbar

Es zeigt sich ein weitgehend ausgewogenes Verhältnis zwischen groben und detaillierten Modellen über fast alle Qualifikationsgruppen. Eine Ausnahme bildet die Gruppe der StiPs, in der fünf von sechs Modellen detailliert erstellt wurden. Aufgrund der relativen Gleichverteilung in den meisten Gruppen, sind die Ergebnisse nicht signifikant. Deshalb kann These 7 in Bezug auf die Detaillierung nicht belegt, mit Blick auf das Ergebnis der StiPs jedoch als Tendenz gewertet werden.

Auch die Betrachtung der Art der Transitionsbeschreibung (Tab. 6.18) zeigt keine statistisch signifikanten Ergebnisse in der Abhängigkeit von Qualifikation der Probanden. Betrachtet man jedoch die Verteilung der Häufigkeiten innerhalb einer Qualifikationsgruppe und vergleicht diese zwischen den einzelnen Gruppen, so zeigt sich, dass in den Gruppen Studenten (Bachelor- und Masterstudenten zusammengefasst) eine weite Streuung über alle Klassen auftritt. Es zeigt sich dabei aber auch eine Tendenz zu den Beschreibungsformen, die in der aufgestellten Rangfolge (Kapitel 6.4.1) einer geringeren Qualität zugeordnet sind. In den Gruppen der Techniker sowie der StiPs erstellten die Probanden hingegen fast ausschließlich eine detaillierte Beschreibung der Transitionen mit einer Tendenz zur qualitativ am höchsten zu bewertenden Beschreibung mit Variablen. Lediglich ein Proband verwendete Transitionen ohne jegliche weitere Detaillierung.

*Tab. 6.18 - Aufteilung der Modelle nach Qualifikation der Probanden und der Detaillierung der Transitionsbedingungen*

| | keine Beschreibung | grob linguistisch | detailliert linguistisch | mit Variablen |
|---|---|---|---|---|
| Bachelor-Studenten | 3 | 4 | 2 | 2 |
| Master-Studenten* | - | 3 | 1 | - |
| Techikerschüler* | - | - | 1 | 5 |
| StiPs | 1 | - | 1 | 4 |

* je ein Modell der Gruppe nicht auswertbar

Die Auswertung im Bezug auf Abhängigkeit zu den verwendeten Modellierungsnotationen zeigt keine signifikanten Ergebnisse. Sowohl in der UML-Gruppe als auch in der ICL-Gruppe ist die Häufigkeit der einzelnen Beschreibungsklassen in einer Qualifikationsgruppe ausgeglichen. Auch ein Zusammenhang zwischen der Detaillierung des Modells und der Art der Transitionsbeschreibung kann nicht nachgewiesen werden.

**These 7 ist nicht wahr, es kann jedoch eine Tendenz nachgewiesen werden.**

### 6.8.2 Diskussion

Die Analyse der Ergebnisse zu These 7 ergibt zwar keine signifikanten Ergebnisse, dennoch lässt sich erkennen, dass Techniker und StiPs zu detaillierten Beschreibungen des Systems tendieren. Diese Probanden gaben gleichzeitig an, bereits mehrfach SPS-Systeme programmiert zu haben und wiesen somit eine größere Erfahrung in diesem Bereich auf. Eine mögliche Erklärung für das Ergebnis ist, dass das Wissen über die Eigenschaften des Zielsystems und die Anforderungen an ein SPS-Programm sich in der Modelldetaillierung widerspiegeln. Dies hat dazu geführt, dass Probanden dieser Gruppen in der Mehrzahl sowohl in Bezug auf die Schritte als auch in Bezug auf die Transitionsbedingungen eine detailliertere Beschreibung wählten.

## 6.9 Einfluss der Qualifikation auf die Programmierleistung

### 6.9.1 Ergebnisse

Eine unterschiedliche Ausbildung der Anwender beeinflusst maßgeblich seine Fähigkeit das entsprechende Steuerungssystem zu programmieren. Die verschiedenartige Ausbildung führt zu einer abweichenden Programmierleistung in den einzelnen Probandengruppen.

***These 8a:*** *Probanden mit unterschiedlicher Qualifikation benötigen für die Erstellung des Steuerungsprogramms unterschiedlich lange und weisen somit aufgrund des konstanten Zeitrahmens eine abweichende Anzahl bearbeiteter Schritte auf.*

Die Analyse der Anzahl bearbeiteter Schritte (Tab. 6.19) zeigt sowohl bei der gemeinsamen als auch bei der nach Notationen differenzierten Betrachtung signifikante Ergebnisse, die auf den Einfluss der Qualifikation zurückgeführt werden können. Im Vergleich der Notationen und der Kontrollgruppe variieren Ergebnisse jedoch derart, dass eine einheitliche Rangfolge der Qualifikation, auf Basis der Anzahl bearbeiteter Schritte, nicht festgelegt werden kann.

*Tab. 6.19 - Durchschnittliche Anzahl bearbeiteter Schritte getrennt nach Qualifikation*

| Modellierungsnotation | Bearbeitete Schritte Bachelor | Bearbeitete Schritte Master | Bearbeitete Schritte StiPs | Bearbeitete Schritte Techniker | Signifikanz |
|---|---|---|---|---|---|
| Analyse über alle Gruppen | 20,26 | 19,86 | 15,44 | 12,78 | 0,02 |
| Unified Modelling Language | 20,00 | 15,50 | 18,00 | 11,33 | <0,01 |
| Idiomatic Control Language | 19,96 | 22,00 | 13,67 | 11,67 | <0,01 |
| Kontrollgruppe | 23,50 | 21,00 | 14,67 | 15,33 | <0,01 |

In der UML-Gruppe zeigt sich, dass die Gruppe der Bachelor das beste Ergebnis erzielten. Es folgen StiPs und Master, Techniker erzielen durchschnittlich das schlechteste Ergebnis. In der ICL-Gruppe führen die Masterstudenten die Rangfolge vor Bachelorstudenten, StiPs und Technikern an. Das Ergebnis der Kontrollgruppe führt wiederum die Bachelor-Gruppe mit dem besten Ergebnis an, gefolgt von Masterstudenten und Techniker. In der Kontrollgruppe erzielten die StiPs das schlechteste Ergebnis.

Die detaillierte Betrachtung der Ergebnisse innerhalb der Qualifikationsgruppen zeigt, dass durch die Verwendung einer Modellierung das Programmierergebnis generell schlechter geworden ist. Lediglich in der UML-Gruppe der StiPs konnte ein besseres Ergebnis als in der zugehörigen Kontrollgruppe erzielt werden.

**These 8a ist wahr.**

***These 8b:*** *Probanden mit unterschiedlicher Qualifikation machen bei der Erstellung des Steuerungsprogramms unterschiedlich viele Fehler und weisen somit eine abweichende Fehlerrate auf.*

Analog zu These 8a erfolgt die Bewertung der Qualifikationsgruppen auf Basis der erzielten Fehlerrate. Auch hier kann trotz signifikanter Ergebnisse (Tab. 6.20) innerhalb der einzelnen Versuchsgruppen keine allgemeine Rangfolge festgelegt werden. Bei Probanden, die eine Modellierung mit der UML durchführten, weist die Gruppe der Techniker die geringste Fehlerrate auf, gefolgt von StiPs, Masterstudenten und Bachelorstudenten.

Dieselbe Rangfolge ergibt sich bei der Betrachtung der Probanden, welche die ICL zur Modellierung verwendeten. Jedoch sind alle erzielten Fehlerraten in den einzelnen Qualifikationsgruppen im Vergleich zur UML-Gruppe höher. Auch in der Kontrollgruppe weisen die Techniker die geringste Fehlerrate auf und sie führen die Rangfolge vor den Masterstudenten, den StiPs und den Bachelorstudenten an.

**These 8b ist wahr.**

*Tab. 6.20 - Durchschnittliche Fehlerrate getrennt nach Qualifikation*

| Modellierungsnotation | Fehlerrate Bachelor | Fehlerrate Master | Fehlerrate StiPs | Fehlerrate Techniker | Signifikanz |
|---|---|---|---|---|---|
| Analyse über alle Gruppen | 53,56% | 41,16% | 40,26% | 13,64% | <0,01 |
| Unified Modelling Language | 50,14% | 41,13% | 28,84% | 8,39% | <0,01 |
| Idiomatic Control Language | 56,24% | 46,34% | 39,19% | 34,76% | <0,01 |
| Kontrollgruppe | 53,98% | 33,41% | 52,74% | 3,04% | <0,01 |

### 6.9.2 Diskussion

Obschon die Ergebnisse sowohl für These 8a als auch für These 8b signifikant, sind lassen sich lediglich für jede These getrennte Rangfolgen aufstellen. Der Vergleich zwischen den Thesen ergibt dabei in weiten Teilen umgekehrte Rangfolgen. Es zeigt sich, dass die studentischen Gruppen zwar eine hohe Anzahl an Schritten realisierten, sie dabei jedoch auch eine hohe Anzahl an Fehlern erzeugten. Techniker und StiPs machten bei der Programmierung hingegen deutlich weniger Fehler, benötigen dabei jedoch mehr Zeit und erzielten somit weniger realisierte Schritte. Auch dieses Verhalten kann auf die höheren Kenntnisse im Bereich der Speicherprogrammierbaren Steuerungen zurückgeführt werden, da Techniker und StiPs, wie bereits in Kapitel 6.3.2 geschildert, eine größere praktische Erfahrung im Bezug auf SPS-Programmierung vorweisen konnten.

## 6.10 Einfluss von Gruppenarbeit auf die Programmierleistung

### 6.10.1 Ergebnisse

Die Verwendung einer Modellierungsnotation unterstützt Gruppenarbeit, da ein einheitliches Modell als gemeinsame Basis für eine Diskussion und den Abgleich der mentalen Repräsentation der einzelnen Gruppenmitglieder dient. Die Verwendung der Modellierung wirkt sich deshalb positiv auf den Zeitaufwand für die Programmierung und die erzielte Fehlerrate aus.

***These 9a:*** *Gruppen, die bei der gemeinsamen Analyse des Systems eine Modellierungsnotation verwenden, weisen im Vergleich zur Einzelarbeit eine höhere Anzahl bearbeiteter Schritte auf. Die Steigerung ist dabei für Gruppen, die modellieren, höher als bei Gruppen, die ohne Modellierung arbeiten.*

Zur Bewertung dieser These werden ausschließlich die Ergebnisse aus Versuchsdesign 1 herangezogen, da der Einfluss der Gruppenarbeit in der Analysephase beurteilt werden soll. Bei der Betrachtung der Ergebnisse, unabhängig von der verwendeten Notation, zeigt sich, dass Gruppen im Vergleich zu Einzelpersonen eine um durchschnittlich 55,54% höhere Anzahl an bearbeiteten Schritten erzielen. Das Ergebnis ist signifikant (Tab. 6.21), These 9a ist somit wahr.

Im Vergleich der drei Gruppen wird deutlich, dass der Einfluss der verwendeten Modellierungsnotation, bezogen auf den Absolutwert der Anzahl bearbeiteter Schritte, nur gering ist. In allen drei Experimentgruppen wurde durch die Gruppenarbeit eine Steigerung des Programmierergebnisses erzielt. Für Probanden mit Gruppenarbeit ist das Ergebnis der ICL-Gruppe jedoch nur geringfügig höher als das der Kontrollgruppe, die UML-Gruppe erzielte sogar ein schlechteres Ergebnis als die Kontrollgruppe.

Berechnet man in Abhängigkeit der Modellierungsnotation die relative Steigerung des Programmierergebnisses der Probanden mit Gruppenarbeit im Vergleich zu den Probanden ohne Gruppenarbeit, so verdeutlicht sich das Resultat. Die Kontrollgruppe weist mit 39,47% den geringsten Zuwachs auf. In der UML-Gruppe wird ein um 50,52% besseres Ergebnis erzielt, welches jedoch immer noch schlechter ist, als das der Kontrollgruppe. Durch die Verwendung der Idiomatic Control Language konnte sogar eine Verbesserung um 78,54% erreicht werden. Hier liegt das Ergebnis leicht über dem der Kontrollgruppe.

**These 9a ist wahr.**

*Tab. 6.21 - Durchschnittliche Anzahl bearbeiteter Schritte in Abhängigkeit von Einzel-/Gruppenarbeit*

| Modellierungsnotation | Bearbeitete Schritte Einzelarbeit | Bearbeitete Schritte Gruppenarbeit | Steigerung | Signifikanz |
|---|---|---|---|---|
| Analyse über alle Gruppen | 15,07 | 23,44 | 55,54% | 0,01 |
| Unified Modelling Language | 14,35 | 21,60 | 50,52% | <0,01 |
| Idiomatic Control Language | 13,84 | 24,71 | 78,54% | 0,05 |
| Kontrollgruppe | 16,85 | 23,50 | 39,47% | 0,05 |

Der generelle Einfluss der Gruppenarbeit auf die Programmiergeschwindigkeit ist somit bewiesen.

***These 9b:*** *Gruppen, die bei der gemeinsamen Analyse des Systems eine Modellierungsnotation verwendet haben, weisen im Vergleich zur Einzelarbeit eine geringere Fehlerrate auf. Die Verbesserung ist dabei für Gruppen, die modellieren, höher als für Gruppen, die ohne Modellierung arbeiten.*

Bei der Betrachtung des Einflusses der Gruppenarbeit auf die durchschnittliche Fehlerrate zeigt sich, dass ohne Unterscheidung nach der verwendeten Notation durch die Gruppenarbeit alleine kein Einfluss auf die Fehlerrate nachgewiesen werden kann (Tab. 6.22). Deshalb erfolgt auch hier die nach Modellierungsnotation differenzierte Betrachtung. Es zeigt sich, dass in der Kontrollgruppe nur ein sehr geringer Unterschied zwischen Einzel- und Gruppenarbeit besteht. Für Probanden, die eine Modellierungsnotation verwendeten, kann hingegen sowohl für den Absolutwert der Fehlerrate als auch für die relative Steigerung ein Unterschied nachgewiesen werden. Während sich bei der Anwendung der Idiomatic Control Language die Fehlerrate deutlich erhöht, tritt in der UML-Gruppe eine fast ebenso deutliche Verringerung der Fehlerrate auf.

*Tab. 6.22 - Durchschnittliche Fehlerrate in Abhängigkeit von Einzel-/Gruppenarbeit*

| Modellierungsnotation | Fehlerrate Einzelarbeit | Fehlerrate Gruppenarbeit | Steigerung | Signifikanz |
|---|---|---|---|---|
| Analyse über alle Gruppen | 50,61% | 50,09% | -1,03% | 0,38 |
| Unified Modelling Language | 48,26% | 40,70% | -15,67% | 0,13 |
| Idiomatic Control Language | 48,26% | 54,21% | 12,33% | 0,66 |
| Kontrollgruppe | 53,98% | 53,68% | -0,56% | 0,81 |

Die Ergebnisse sind sowohl für die gemeinsame als auch für die nach Notationen getrennte Betrachtung in der ICL- sowie in der Kontrollgruppe nicht signifikant, die Ergebnisse der UML-Gruppe können nur als Tendenz gewertet werden.

**These 9b ist nicht wahr, in der UML-Gruppe lässt sich jedoch eine Tendenz zu These 9b erkennen.**

## 6.10.2 Diskussion

Eine einheitliche Bewertung der Eignung der Modellierungsnotationen zur Gruppenarbeit kann durch die getrennte Betrachtung der Thesen 9a und 9b nicht erfolgen. So ist beispielsweise allein durch die Tatsache, dass die Probanden in Gruppen zusammen gearbeitet haben, ein messbarer Vorteil erzielt worden. Betrachtet man ausschließlich die

Ergebnisse der Kontrollgruppe, so zeigt sich ein Nutzen im Bezug auf die benötigte Zeit. Die durchschnittliche Fehlerrate war hingegen nicht von der Gruppenarbeit abhängig.

Vergleicht man die Ergebnisse der Kontrollgruppe mit der UML-Gruppe so zeigt sich, dass die relative Steigerung im Bezug auf die Anzahl bearbeiteter Schritte zwar nur geringfügig höher ist, die UML-Gruppe jedoch eine signifikant verbesserte Fehlerrate aufweist. Das Ergebnis der UML-Gruppe ist somit besser einzustufen als das der Kontrollgruppe. Die höhere Steigerung beider Leistungsfaktoren in der UML-Gruppe kann dabei auf die Verwendung dieser Notation zur Modellierung zurückgeführt werden.

Ein entsprechender Vergleich der ICL-Gruppe mit der Kontrollgruppe zeigt, dass durch die Verwendung der ICL zwar eine Steigerung der bearbeiteten Schritte erzielt werden konnte, dieses Resultat durch die ebenfalls gestiegene Fehlerrate wieder relativiert wird. Beide Effekte sind somit gegenläufig, wodurch in der Addition der beiden Effekte für die ICL somit kein signifikanter Vorteil bei der Gruppenarbeit durch die Verwendung der ICL nachgewiesen werden kann.

## 6.11 Weitere Ergebnisse

Neben der subjektiven Bewertung von vorgegebenen Aussagen auf einer vordefinierten Skala, wurden die Probanden in den Fragebögen auch aufgefordert ihre Erfahrungen in Form von Freitexten zu dokumentieren. Dabei gab es keinerlei thematische Einschränkungen. Darüber hinaus wurden Besonderheiten im Verlauf der Experimente durch den Versuchsleiter notiert. Somit war es möglich weitere Aspekte herauszuarbeiten, die für die Bewertung der Anwendbarkeit einer Modellierung für die Steuerungsprogrammierung relevant sind. Es ist jedoch nur eingeschränkt möglich daraus allgemeingültige Aussagen abzuleiten, da die einzelnen Aspekte nur für Probanden bewertet werden können, die ein Statement abgaben oder für die durch den Versuchsleiter eine entsprechende Notiz angelegt wurde. Eine definierte Aussage für die anderen Probanden, sei sie positiv oder negativ, kann hingegen nicht gemacht werden. Dennoch zeigen sich dadurch Aspekte deren Evaluation in weiterführenden Arbeiten sinnvoll ist.

***Fehlende Softwareunterstützung bei der Modellierung***

Die fehlende Unterstützung der Modellierung durch ein Softwarewerkzeug wurde, hauptsächlich durch die studentischen Probanden, teilweise sehr negativ bewertet. Es wurde als ungewohnt und schwierig empfunden das Modell auf Papier oder einer Tafel zu entwickeln. Fehlende Flexibilität und schlechte Änderbarkeit bereits erstellter Modellteile waren die Hauptgründe für diese Bewertung. Die Tatsache, dass dies überwiegend bei den

Studenten der Fall war, könnte darin begründet sein, dass diese im Verlauf des Studiums bereits häufiger mit Modellierung konfrontiert wurden und dies meist durch Softwaretools unterstützt wird. Beim Entwurf des Experiments wurde jedoch bewusst auf die Verwendung von Softwarewerkzeugen verzichtet, um Störeffekte durch unterschiedlich gute Werkzeuge zu vermeiden. Für die UML existiert eine große Anzahl industrietauglicher Werkzeuge. Im Gegensatz dazu steht für die ICL nur ein prototypisches Tool, mit unvollständiger Unterstützung der Funktionalität, zur Verfügung. Dadurch wäre, unter Anwendung dieser Modellierungswerkzeuge, kein objektiver Vergleich der Ergebnisse zwischen der UML-Gruppe und der ICL-Gruppe möglich gewesen.

Darüber hinaus beeinflusst die Verwendung eines Modellierungstools möglicherweise die Bewertung einer Modellierung und wäre dabei auch stark von der Qualität und der Usability des verwendeten Werkzeugs abhängig gewesen. Auch der Funktionsumfang entsprechender Werkzeuge, im Bezug auf eine Vorgehensweise, hätte einen Einfluss auf die Evaluation dargestellt. So bieten z.B. UML-Werkzeuge einerseits teilweise explizit eine Modellierung nach einem Vorgehensmodell (z. B. Rational Unified Process) an. Andererseits sind auch implizit bestimmte Modellierungsregeln implementiert, die den Anwender bei einer Modellierung mehr unterstützen als die ausschließliche Verwendung von „Papier und Stift“.

***Anwendbarkeit der Modellierungsnotation für die Steuerungsprogrammierung***

Die Bewertung der textuellen Beschreibung, mit deren Hilfe die Kontrollgruppe in Versuchsdesign 2 das Steuerungsprogramm erstellen sollte, zeigte Unterschiede in Abhängigkeit von der Qualifikation der Probanden. An diesem Versuch nahm ein Teil der Bachelorstudenten sowie die Gruppe der Auszubildenden teil. Hier zeigte sich, dass in der Gruppe der Studenten vier von sechs Probanden anmerkten, dass die textuelle Beschreibung der Struktur und des Verhaltens des Systems keine geeignete Hilfe für die Programmierung darstellte. Stattdessen gaben sie an, eine Modellierungsnotation nutzen zu wollen, da dies eine bessere Unterstützung darstellt. In der Gruppe der Auszubildenden bewerteten hingegen drei von vier Probanden den Text als gut verständlich und für die Aufgabe angemessen. Die Vorlesungen und Übungen, mit denen beide Probandengruppen auf das Experiment vorbereitet wurden, waren jedoch vollkommen identisch. Dies könnte darauf hindeuten, dass die Vorkenntnisse, die die meisten Studenten im Bereich der Modellierung aufwiesen, zu einem höheren Abstraktionsvermögen führen. Folglich fällt ihnen das Erlernen der Modellierungsnotation leichter und sie stufen den Nutzen einer solchen Unterstützung dadurch höher ein als die Auszubildenden.

***Auswirkung der Modellierungsnotationen auf die Gruppenarbeit***

Es zeigte sich in Versuchsdesign 1, dass ein Teil der Probanden, die mit einer Modellierung in Gruppen zusammen arbeiteten, die Eignung eines Modells für die Diskussion mit dem Partner bewerteten. Im Fragebogen wurde nicht explizit danach gefragt, ob das gemeinsame grafische Modell dazu genutzt wurde, Probleme oder Unklarheiten mit dem anderen Gruppenmitglied zu besprechen. Dennoch gaben acht von insgesamt 20 Probanden im Rahmen der freien Kommentare zum Versuch an, dass sich das Modell gut für die Diskussion von Problemen eignet. Von den acht Probanden modellierten fünf mit der UML, drei Probanden verwendeten die ICL. Ein Vorteil für eine der Notationen kann folglich nicht erkannt werden.

***Fehlen einer Vorgehensweise bei Verwendung einer Modellierung***

In dem abschließenden Fragebogen mussten die Probanden, wie in Kapitel 5.2.3 bereits dargestellt, verschiedene Aussagen zur Anwendbarkeit einer Modellierung im vorhandenen Kontext bewerten. Die Analyse der Fragebögen ergab, dass darüber hinaus eine Vielzahl der Probanden den Wunsch nach einer vorgegebenen Vorgehensweise äußerten. Dies bezog sich dabei auf zwei Bereiche:

Bei der Modellierung war die Unsicherheit, welche Diagramme verwendet werden sollten, sehr groß. Es wurde zwar angegeben, dass die Bedeutung und Anwendung der verschiedenen Diagramme generell verstanden wurde und klar gewesen sei und in den meisten Gruppen auch schnell festgelegt, welche Diagramme erstellt werden sollten (Zustandsdiagramm bzw. SFC). Dennoch lassen verschiedene Aussagen in den Fragebögen vermuten, dass die Probanden, im Nachhinein betrachtet, nicht sicher waren, ob ein anderer Weg oder die Verwendung einer anderen Diagrammart nicht geeigneter gewesen wäre.

Im Bezug auf die Programmierung gaben viele Probanden an, dass nicht klar sei, wie das erstellte oder vorgegebene Modell in ein Programm umgesetzt werden sollte.

Beide Aussagen lassen darauf schließen, dass die Zusammenhänge zwischen Modell und Programm nicht ausreichend klar waren. Viele Probanden forderten eine genauere Anleitung, wie modelliert und das Modell anschließend in ein Programm überführt werden könnte. Die Ursache dafür könnten zu geringe Kenntnis sowohl im Bereich der Modellierung als auch der Programmierung sein, da das Wissen in beiden Bereichen vorhanden sein muss, um die gestellte Aufgabe zu lösen. Einen signifikanten Beleg liefern die beschrieben Experimente dafür jedoch nicht, dies könnte Untersuchungsgegenstand weiterer Arbeiten sein.

***Überdurchschnittlich hohe Fehlerrate bei der Programmierung***

Wie die Analyse der Thesen im Bezug auf die erzielte Fehlerrate zeigt, hat sich im Rahmen der Evaluationsexperimente durchweg eine sehr hohe durchschnittliche Fehlerrate bei der Programmierung der Steuerung gezeigt. Bezogen auf die Anzahl der realisierten Programmschritte waren in Abhängigkeit der betrachteten Gruppe durchschnittlich bis zu ca. 55 % des Programmcodes fehlerhaft (vergleiche Tab. 6.20). Diese hohen Fehlerraten sind für eine industrielle Anwendung der Software selbstverständlich inakzeptabel. Die Ursachen für diese Ergebnisse können vielfältig sein und wurden im Rahmen der Ergebnisdiskussion bereits teilweise genannt. Als ein Hauptgrund wird die geringe Erfahrung der Probanden im Bezug auf die Steuerungsprogrammierung vermutet. Wie in Kapitel 2.3.1 beschrieben, arbeitet eine Speicherprogrammierbare Steuerung das Programm zyklisch ab und verwendet dabei ein Prozessabbild, welches an definierten Punkten gelesen und zurück geschrieben wird. In Kombination mit der meist in Funktionsbausteinsprache durchgeführten Programmierung scheint das dazu zu führen, dass nicht alle Bedingungen, die für die Ausführung einer Aktion notwendig sind, berücksichtigt wurden. Häufig wurden nur diejenigen Bedingungen verwendet, die sich im aktuellen Zustand unmittelbar ändern müssen, um eine nachfolgende Aktion einzuleiten. In der Denkweise entspricht dies einem Zustandsdiagramm in der UML oder einem SFC in der ICL. Bei Umsetzung in ein Steuerungsprogramm ist dies jedoch entweder durch die Anwendung einer entsprechenden Programmiersprache (in der IEC 61131-3 die Ablaufsprache) oder einer bestimmten Programmierphilosophie (z.B. der Schrittkettenprogrammierung) zu berücksichtigen oder alle für die Aktion notwendigen Bedingungen müssen programmiert werden. Aufgrund fehlender Erfahrung in der Programmierung Speicherprogrammierbarer Steuerungen schien den Probanden dieser Zusammenhang teilweise nicht bewusst zu sein. Leider war es jedoch im Rahmen dieser Evaluation nicht möglich das Experiment mit erfahrenen Programmierern durchzuführen.

Trotz der hohen Werte im Bereich der Fehlerraten erlauben die Ergebnisse jedoch, wie in den vorangegangenen Abschnitten gezeigt, eine Aussage über den Einfluss einer Modellierung auf die Fehlerrate bei der Programmierung. Eine angepasste Untersuchung in weiteren Arbeiten scheint dabei jedoch sinnvoll, um die oben diskutierten Ursachen empirisch zu untersuchen. Dabei sollten entweder erfahrene Programmierer am Versuch teilnehmen oder eine zielgerichtete Anpassung des Schulungsteils im Bezug auf diesen Aspekt erfolgen.

## 6.12 Korrelation der Ergebnisse verschiedener Thesen

Neben der nach Thesen getrennten Betrachtung, wie sie im vorliegenden Kapitel bisher beschrieben wurde, wurden die einzelnen Ergebnisse auch in Beziehung zu einander gesetzt und untersucht, ob inkongruente oder kongruente Ergebnisse vorliegen. Tab. 6.23 fasst die Ergebnisse in übersichtlicher Form zusammen. Im Folgenden werden auffällige Wechselbeziehungen oder Zusammenhänge aufgezeigt und diskutiert.

*Tab. 6.23 - Übersicht der Ergebnisse getrennt nach Thesen und Modellierungsnotation*

| These | 1a | 1b | 2a | 2b | 2c | 3a | 3b | 3c | 4a | 4b | 5 | 6a | 6b | 6c | 7 | 8a | 8b | 9a | 9b |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **Gesamtheit aller Probanden** | ++ | ++ | O | O | ++* | O | O | + | O | O | O | O | +** | +*** | + | ++ | ++ | ++ | O |
| **UML-Gruppe** | + | O | | O | ++* | O | + | | O | O | | | | | | ++ | ++ | ++ | + |
| **ICL-Gruppe** | ++ | ++ | | O | O | O | O | | O | O | | | | | | ++ | ++ | ++ | O |
| **Kontrollgruppe** | O | ++ | | O | ++* | | | | | | | | | | | ++ | ++ | ++ | O |

++ : These ist signifikant wahr
+ : These ist tendenziell wahr
O : Ergebnisse nicht signifikant
- : Ergebnisse sprechen tendenziell gegen die These
-- : Ergebnisse sprechen signifikant gegen die These

* These wahr für Versuchsdesign 1, nicht wahr für Versuchsdesign 2
** These wahr für StiPs, nicht wahr für Gruppen anderer Qualifikationen
** These wahr für Ba/Ma, nicht wahr für Gruppen anderer Qualifikationen

Der Vergleich der subjektiven Bewertung des Nutzens der Modellierung in Abhängigkeit der Qualifikation (These 6b - Kapitel 6.7.1) mit der Programmierleistung in Abhängigkeit der Qualifikation (Thesen 8a und 8b - Kapitel 6.9.1) zeigt, dass für die einzelnen Qualifikationen unterschiedliche Ergebnisse auftreten. Betrachtet man ausschließlich die Gruppe der StiPs so fällt auf, dass der Nutzen der Modellierung in der UML-Gruppe deutlich höher bewertet wird als in der ICL-Gruppe. Gleichzeitig werden in der UML-Gruppe mehr Schritte bearbeitet und eine geringere Fehlerrate erzielt als in der ICL-Gruppe. Dieser Zusammenhang könnte darauf hindeuten, dass die UML in der Gruppe der StiPs die Aufgabe derart unterstützt hat, dass die Leistung nicht nur höher war als in der ICL-Gruppe, sondern den Probanden diese Unterstützung auch bewusst war. Dies führte dann zu einer besseren Bewertung des Nutzens der UML. In der Gruppe der Techniker ist der Zusammenhang in ähnlicher Tendenz vorhanden. Hier sind die Zusammenhänge jedoch nicht gleichermaßen bewertbar, da die Ergebnisse zu These 6b in dieser Gruppe nicht signifikant sind und die Unterschiede in der Programmierleistung deutlich geringer sind.

Führt man eine entsprechende Betrachtung für Bachelor- und Masterstudenten durch, so zeigt sich zwar eine signifikant bessere Bewertung der ICL. Diese korreliert jedoch keineswegs mit den Ergebnissen der Programmierung. Nur die Master-Studenten erzielen bei Verwendung der ICL eine höhere Anzahl bearbeiteter Schritte und die Fehlerraten in der ICL-Gruppe sind jeweils höher als in der UML-Gruppe.

Die Gründe für dieses Ergebnis können durch das Experiment nicht geliefert werden. Es ist jedoch denkbar, dass zwei Aspekte dies zumindest begünstigt haben. Auf der einen Seite besitzen StiPs und Techniker im Bezug auf die Programmierung von Speicherprogrammierbaren Steuerungen eine gewisse Erfahrung, die höher ist als bei Bachelor- und Masterstudenten. Im Bereich der Modellierung hatten StiPs und Techniker hingegen kaum Erfahrung mit der UML. Dies könnte dazu geführt haben, dass sie die Modellierungsnotation unvoreingenommen und unabhängig von vorgegebenen Vorgehensweisen angewendet haben. Dabei haben sie sich von ihrem Wissen über SPS-Programmierung leiten lassen, was gleichzeitig zu einem guten Programmierergebnis und einer positiven Bewertung geführt hat. Auf der anderen Seite besitzen Bachelor- und Masterstudenten im Bezug auf die Modellierung mit der UML höhere Vorkenntnisse. Diese wurden jedoch, zumindest im Rahmen des Studiums, meist im Zusammenhang mit der Programmierung in objektorientierten Hochsprachen erlangt. Gemeinsam mit der vergleichsweise geringeren Erfahrung bei der SPS-Programmierung kann das dazu geführt haben, dass sie ihnen bekannte Vorgehensweise angewendet haben, die jedoch für die geforderte Arbeitsaugabe weniger geeignet war. Ein Hinweis dafür könnte die häufige Forderung der studentischen Probanden nach einer Vorgehensweise im Rahmen der Beantwortung der Fragebögen sein.

Der Vergleich der Ergebnisse, zum Einfluss der Arbeitsaufgabe und der Analyse der modellierten Aspekte, zeigt, dass die Probanden in Versuchsdesign 1 ein besseres Programmierergebnis als in Versuchsdesign 2 erzielten (Thesen 1a und 1b - Kapitel 6.2.1), obschon sie überwiegend ausschließlich das Verhalten des Prozesses modellierten und die Struktur vernachlässigten (These 4a - Kapitel 6.5.1). Dies widerspricht zwar vom Grundsatz her der These, dass ein vollständiges Modell eine bessere Unterstützung bietet. Unberücksichtigt bleibt bei dieser Betrachtung jedoch die Tatsache, dass auch das erstellte Modell nur ein unvollständiges Abbild des mentalen Modells des Anwenders darstellt. Der Anwender modelliert nur das, was er für die eigentliche Aufgabe als wichtig erachtet oder lässt eventuell auch Teile weg, die er für zu trivial oder einfach hält. Die Analysetätigkeit als solches erzeugt jedoch beim Nutzer bereits ein mentales Modell der Struktur des Prozesses, welches, unabhängig von dessen Dokumentation, vorhanden ist und auch zur späteren Programmierung verwendet wird. Das mentale Modell wurde im Rahmen der vorgestellten Evaluation nicht explizit ermittelt oder abgefragt, sodass keine Aussage darüber gemacht werden kann, in wiefern Wissen über strukturelle Zusammenhänge des Prozesses, bei den Probanden vorhanden war. Das mentale Modell des Systems könnte in weiterführenden Arbeiten durch eine Anpassung des Versuchsdesigns, z.B. mit Hilfe der Kartenlegetechnik oder geführten Interviews, ermittelt werden.

Interessant ist auch der Vergleich der Ergebnisse zu These 7 (Einfluss von Qualifikation auf die Qualität des erstellten Modells) und These 3b (Einfluss der Qualität eines selbst erstellten Modells auf die Fehlerrate). Hier zeigt These 7, dass Probanden mit höher einzustufender Qualifikation im Bezug auf die Programmierkenntnisse einer SPS, also StiPs und Techniker, dazu tendieren ein Modell mit höherer Detaillierung zu erstellen. Gleichzeitig kann durch These 3b tendenziell nachgewiesen werden, dass bei Anwendung der UML eine hohe Detaillierung im Modell zu einer geringeren Fehlerrate führt, wohin gegen bei der ICL kein Effekt nachgewiesen werden kann. Folglich sollten Probanden, welche die UML zur Modellierung verwenden, mit steigender Qualifikation ein besseres Ergebnis erzielen. Dies kann durch These 8b (Einfluss der Qualifikation auf die Fehlerrate) belegt werden. Gleichzeitig kann gezeigt werden, dass dieser Einfluss bei Anwendung der ICL nicht gleichermaßen signifikant ist. In dieser Gruppe sind die Ergebnisse zu These 3b nicht signifikant und auch die Unterschiede in der Fehlerrate in Abhängigkeit der Qualifikation sind deutlich geringer als bei der UML. Die Ergebnisse zeigen einerseits, dass die Qualifikation ein wichtiger Faktor im Bezug auf den Nutzen einer Modellierung im Rahmen der Steuerungsprogrammierung darstellt. Anderseits zeigen sie auch die unterschiedliche Ausprägung des Nutzens, der darauf hindeutet, dass die UML für diesen Anwendungsfall besser geeignet ist als die ICL. Da die entsprechenden Ergebnisse nur als Tendenz gewertet werden können, ist eine detailliertere Untersuchung im Rahmen weiterer Arbeiten sinnvoll.

Neben dem reinen Vergleich der Ergebnisse zu einzelnen Thesen, wurden auch Zusammenhänge zu den weiteren Ergebnissen aus Kapitel 6.11 untersucht. Dabei fällt besonders ein möglicher Zusammenhang zwischen dem Thesenpunkt 4 und dem geäußerten Wunsch nach einer Vorgehensweise auf. Die Analyse zu These 4a (Modellierte Aspekte) zeigte, dass fast ausschließlich Verhalten modelliert und der Nutzen von strukturbeschreibender Modellierung sehr niedrig eingestuft wurde. Dies kann unter anderem damit erklärt werden, dass den Probanden nicht klar war, wie sie bei einer Modellierung am besten vorgehen. Dass eine entsprechende Beschreibung des Prozesses durchaus als sinnvoll angesehen wird, zeigen die Ergebnisse zu These 4b (Subjektive Bewertung der Strukturbeschreibung), die in Versuchsdesign 2 deutlich besser ausfällt.

Abschließend kann bei der Betrachtung der Ergebnisse festgestellt werden, dass zwar nicht alle Thesen belegt werden konnten. Es ist jedoch auch festzuhalten, dass keine Ergebnisse erzielt wurden, deren Aussagen sich vom Grundsatz her unmittelbar widersprechen.

# 7 Schlussfolgerungen aus der empirischen Evaluation

Ziel der Arbeit war es zu evaluieren, ob die Entwicklung der Steuerungssoftware automatisierungstechnischer Systeme durch den Einsatz einer Modellierung sinnvoll unterstützt werden kann, um die Qualität der Software und die Effizienz dieses Entwicklungsprozesses zu steigern. Darüber hinaus sollte ermittelt werden, wie sich die hauptsächlichen Probleme bei der Anwendung einer Modellierung darstellen und welche Maßnahmen ergriffen werden können, um den Programmierer bei seiner Aufgabe geeignet zu unterstützen. Dies ist mit Hilfe der erzielten Ergebnisse zwar nur in Teilen möglich, da nicht alle Thesen belegt werden konnten. Dennoch bietet die Evaluation die Grundlage für eine Diskussion, wie eine Modellierung des Prozesses im Rahmen des Engineeringprozesses aussehen könnte und was dabei im Speziellen zu berücksichtigen ist. Zudem liefern die durchgeführten Experimente auch Ergebnisse, die eine Bewertung der zur Evaluation gewählten Vorgehensweise ermöglichen. Darüber hinaus können daraus Vorschläge für die Durchführung weiterer Experimente abgeleitet werden.

## 7.1 Auswirkungen auf das Engineering automatisierungstechnischer Prozesse im Rahmen der Steuerungsprogrammierung

Die Ergebnisse der durchgeführten Experimente zeigen, dass Aussagen über den Nutzen einer der vorgestellten Modellierungsnotationen nur mit Einschränkungen möglich sind. Die Frage nach den Schlussfolgerungen aus der Evaluation kann daher auch nicht generell beantwortet werden. Einzelne Aspekte der Evaluation sollen daher herausgegriffen und fokussiert gezeigt werden, wie eine Modellierung dazu beitragen kann die Effektivität im Engineering zu steigern, und welche Voraussetzungen für eine effektive Anwendung erfüllt bzw. geschaffen werden müssen.

Die Versuche zeigten, dass eine Modellierung des zu automatisierenden Systems nicht zu einer höheren Bearbeitungsdauer durch den Programmierer führte. Probanden aus den Modellierungsgruppen erzielten trotz der zusätzlichen Aufgabe und damit geringerer Zeit für die Programmierung durchschnittlich ein ähnlich gutes Programmierergebnis wie die Kontrollgruppen. Dies kann dadurch erklärt werden, dass die während der Modellierung durchgeführte Analyse des Systems auch bei der Programmierung der Speicherprogrammierbaren Steuerung notwendig ist. Sie wird stattdessen nur unbewusst durchgeführt und nicht explizit dokumentiert. Eine Modellierung führt in der Anwendung somit nicht zu einem zeitlichen Mehraufwand. Vielmehr wird die Tätigkeit des

Systementwurfs von der Programmierung in den Bereich der Modellierung verlagert. Dafür spricht auch die Tatsache, dass bei Anwendern, die im Bezug auf die SPS-Programmierung eine höhere Erfahrung aufweisen, mit steigender Anzahl richtig modellierter Schritte auch die Anzahl richtig programmierter Schritte steigt. Darüber hinaus bringt die Modellierung jedoch auch einen nutzbaren Mehrwert mit sich. So konnte in der Tendenz gezeigt werden, dass durch eine Modellierung die Modularität des Programms steigt. Eine explizite, von der Programmierung getrennte Analyse und Modellierung des Systems ergab in der Evaluation, unabhängig von der verwendeten Notation, einen höheren Anteil an modularer Programmierung. Dies kann bei häufiger Anwendung zu einem höheren Grad an Wiederverwendung führen. Weiterer Nutzen kann durch die Wiederverwendung des dokumentierten Modells der Steuerungsfunktionen erzielt werden. Ein möglicher Ansatz ist im Bereich der Programmdokumentation zu sehen, wo das UML-Modell bisher gewählte Beschreibungsformen oder einen frei formulierten Text ergänzen und teilweise ersetzen kann. Dies führt zu einer weiteren Zeiteinsparung und Effizienzsteigerung.

Die Evaluation zeigte, dass die Wahl eines geeigneten Beschreibungsmittels den erzielbaren Nutzen beeinflusst. Die UML ist in ihrer uneingeschränkten Form für die Steuerungsprogrammierung zwar nur bedingt geeignet. Ein Vergleich mit den Ergebnissen der ICL-Gruppen zeigt jedoch, dass die ICL in keinem der untersuchten Punkte signifikant bessere Ergebnisse erbrachte. Folglich scheint es der sinnvollere Weg zu sein, die UML durch geeignete Maßnahmen für den Einsatz bei der Steuerungsprogrammierung anzupassen.

Die große Anzahl der Diagramme in der UML hat die Probanden schnell verwirrt und die Zusammenhänge zwischen den einzelnen Darstellungsformen waren nicht klar. Hier kann dem Anwender in der industriellen Anwendung durch verschiedene Maßnahmen eine Hilfestellung gegeben werden. Ein möglicher Ansatz ist die Anpassung der UML an die Domäne der Automatisierungstechnik durch eine „domain specific language“. Wie in Kapitel 3.1.10 beschrieben, steht mit der UML-PA mittlerweile eine für die Automatisierungstechnik angepasste Variante der UML zur Verfügung, in der unter anderem versucht wurde, durch Zusammenführung oder Eliminierung redundanter Diagrammtypen, die Komplexität des Beschreibungsmittels zu reduzieren. Evaluationsergebnisse, die denen der hier vorgestellten Versuchsreihe entsprechen, liegen für die UML-PA nicht vor. Zwar konnte in Experimenten nachgewiesen werden, dass Probanden, die ein UML-PA-Modell verwendeten, darin modellierte Kommunikationsstrukturen schneller erkennen als bei Verwendung eines UML-Modells [Kat08]. Untersuchungen zu Vor- oder Nachteilen der UML-PA gegenüber der uneingeschränkten UML für die Modellierung steuerungstechnischer Aufgaben wurden

jedoch nicht durchgeführt, sodass entsprechende Aussagen erst nach einer Evaluation beider Beschreibungsmittel im direkten Vergleich möglich sind. Auch eine Anwendung der SysML für die Modellierung, im hier besprochenen Kontext, ist zu prüfen. Diesbezüglich liegen ebenfalls keine entsprechenden Ergebnisse vor.

Neben der Anpassung der Modellierungssprache stellt auch die Anwendung einer optimierten Vorgehensweise eine Möglichkeit dar, den Nutzen der Modellierung zu steigern. Wie in Kapitel 3.1.11 dargestellt, schreibt die UML von sich aus keine Vorgehensweise vor. Diese Freiheit in der Anwendung stellt besonders für unerfahrene Anwender eine nicht zu unterschätzende Hürde dar. Darauf deutete im Rahmen der Evaluation einerseits die starke Fokussierung auf bestimmte Diagrammformen, andererseits auch die freien Kommentare der Fragebögen, in denen häufig nach einer Vorgehensweise verlangt wurde, hin. Die Bereitstellung eines Rahmenwerks kann hier dazu genutzt werden, erprobte Vorgehensweisen zu dokumentieren und als Standard zu etablieren, der für unerfahrene Anwender gleichzeitig als Leitfaden für die Modellierung dient. Es ist jedoch darauf zu achten, dass Vorgaben im Bezug auf die Vorgehensweise nicht zu starr sind. Dies würde ihre Anwendbarkeit erheblich einschränken, da sich die Bedingungen, die beim Start der Modellierung herrschen, in Abhängigkeit vom jeweiligen Entwurfsprozess individuell erheblich unterscheiden können. Eine entsprechende Vorgehensweise muss auf vorhandene Unterschiede flexibel reagieren können und sollte dabei eher als Leitfaden denn als restriktives Regelwerk gesehen werden. In der Praxis werden entsprechende Vorgehensweisen zur Anwendung der UML für verschiedene Anwendungsbereiche bereits eingesetzt (vergleiche Kapitel 3.1.11). Für die Modellierung zum Zweck der Steuerungsprogrammierung existiert ein solches Modell noch nicht.

Für die effektive Anwendung einer Modellierungsnotation reichen gute Kenntnisse der entsprechenden zu verwendenden Notation und des damit verbundenen Paradigmas alleine nicht aus. Auch Erfahrung im Bezug auf die verwendete Zielumgebung und Programmiersprache sind notwendig, um ein für die Steuerungsprogrammierung nutzbringendes Modell des zu automatisierenden Systems zu erstellen. Die Versuche haben gezeigt, dass Probanden mit einer größeren Erfahrung im Bereich der Programmierung (StiPs und Techniker) dazu tendieren, eine Beschreibung des Systems in einer hohen, der Programmierung angemessenen Detaillierung des Modells zu wählen. Für Bachelor und Masterstudenten, die größtenteils eine vergleichsweise geringere Erfahrung in der Programmierung von Speicherprogrammierbaren Steuerungen aufwiesen, zeigte sich diese Tendenz nicht. Es konnte belegt werden, dass ein qualitativ höherwertiges Modell eine bessere Unterstützung für die Programmierung darstellt. Für die Anwendung einer Modellierungsnotation ist folglich darauf Wert zu legen, dass die Nutzer sowohl im Bereich

der Modellierung als auch in der Programmierung einen Wissensstand aufweisen, der eine effektive Nutzung der Modellierung ermöglicht. Dies ist bei der Einführung einer durch die UML unterstützten Vorgehensweise für die Steuerungsprogrammierung in der industriellen Anwendung zu beachten.

Unerfahrene Anwender können zusätzlich durch die Bereitstellung von Beispielapplikationen unterstützt werden. Dabei sollten für den Anwendungsbereich typische Anwendungsbeispiele so aufbereitet werden, dass sie die Vorgehensweise sowohl im Bezug auf die Modellierung als auch die Programmierung abbilden und den Zusammenhang zwischen Modell und Programm anschaulich erklären. Anhand dieser Beispiele hat der Nutzer die Möglichkeit einen Einstieg in die Modellierung zu finden und es wird veranschaulicht, wie ein Modell aufgebaut werden kann, damit es für die anschließende Programmierung eine möglichst große Unterstützung darstellt. Im Rahmen der Experimente gaben viele Probanden an, dass nicht klar sei, wie das Modell in Programmcode umgesetzt werden soll. Die Diskrepanz zwischen objektorientierter Modellierung auf der einen Seite und der prozeduralen Programmierung der SPS auf der anderen Seite stellt dabei einen Haupthinderungsgrund dar. Durch die Verwendung von Beispielapplikationen kann besonderer Wert auf die Erläuterung dieses Aspektes gelegt werden und so der Transfer vom Modell zu Programm verdeutlicht und erleichtert werden.

Die Evaluation machte bezüglich einer Trennung der Aufgaben „Modellierung“ und „Programmierung“ deutlich, dass es von Vorteil ist, wenn ein Anwender beide Aufgaben nacheinander selbst abarbeitet. Im Rahmen der Versuche erbrachten Probanden, die zuerst modellierten und anschließend programmierten, ein besseres Ergebnis, als jene die mit einem vorgegebenen Modell arbeiteten. Vor dem Hintergrund der geringen Erfahrung der Evaluationsteilnehmer, entweder auf dem Gebiet der Modellierung oder der Programmierung, kann daraus abgeleitet werden, dass unerfahrene Anwender, bei der hier gewählten Vorgehensweise, beide Aufgaben erfüllen sollten, um größtmöglichen Nutzen aus der Modellierung zu ziehen. Ob diese Aussage ebenfalls für erfahrene Anwender gilt, muss in weiteren Untersuchungen ermittelt werden.

Für die Versuche wurden die Modelle, wie beschrieben, ausschließlich mit Papier und Stift bzw. einer Tafel und Kreide erstellt. Diese Einschränkung wurde im Versuchsdesign lediglich zum Zweck der Vergleichbarkeit der Ergebnisse für die verschiedenen untersuchten Notationen getroffen und ist für eine industrielle Nutzung natürlich nicht notwendig. Ganz im Gegenteil sind für grafische Modellierungssprachen gerade entsprechende Softwarewerkzeuge, welche die Eigenheiten der jeweiligen Sprache genau abbilden und den Gebrauch in einer effektiven Weise unterstützen, von enormer

Wichtigkeit ([KKP+99],[BaVo01]). Sie ermöglichen unterschiedliche Sichten auf das Modell zu präsentieren und stellen dabei auch Beziehungen zwischen Elementen verschiedener Sichten her, die eine Beschreibung komplexer Systeme erleichtern. Auch im Bezug auf einen bereitzustellenden Leitfaden, stellt die Anwendung eines Softwarewerkzeugs eine Möglichkeit dar, ein Rahmenwerk zu implementieren und den Nutzer durch Hinweise und Vorschläge zur weiteren Vorgehensweise zu unterstützen.

## 7.2 Bedeutung für die Durchführung empirischer Evaluationsexperimente in der Automatisierungstechnik

In der vorliegenden Arbeit wurde erstmalig der Einfluss einer Modellierung auf das Resultat der Steuerungsprogrammierung auf Basis einer umfangreichen Stichprobe an Probanden in einer empirischen, experimentellen Evaluation untersucht. Zwar beschäftigten sich bereits seit den 80'er Jahren diverse Arbeiten zum Beispiel mit Fragen, welche Faktoren als förderlich im Bezug auf die Programmierleistung einzustufen sind (vergl. Kapitel 4.2). Die Ergebnisse waren jedoch nur bedingt auf den hier untersuchten Anwendungsbereich übertragbar, da sie zumeist unter anderen Rahmenbedingungen erfasst wurden. Aus diesem Grund stellte der Entwurf einer entsprechenden Vorgehensweise für die Evaluation und das Design des dazugehörigen Versuchsaufbaus eine besondere Herausforderung dar. Dabei konnten im Verlauf der Versuchsvorbereitung, -durchführung und -auswertung besondere Aspekte identifiziert werden, welche die Durchführung eines derartigen Feldversuches beeinflussen.

Die Grenzen der Evaluation wurden bewusst groß gewählt, um eine möglichst große Anzahl von Einflussfaktoren identifizieren zu können. So wurde zum einen eine komplexe Aufgabe gewählt, in deren Verlauf die Probanden mehrere Arbeiten (modellieren und programmieren) zu bearbeiten hatten. Darüber hinaus wurden nur sehr wenig einschränkende Vorgaben zur Vorgehensweise gemacht, um zu ermitteln, wie die Probanden unbeeinflusst an eine solche Aufgabe herangehen. Ein derart offenes Versuchsdesign ist notwendig, um mögliche Bereiche oder Zusammenhänge zu identifizieren, für die eine fokussierte Untersuchung sinnvoll erscheint. Dies führt zusammen mit dem hohen Maß an Freiheitsgraden im Prozess der Systemmodellierung und des Programmentwurfs dazu, dass die Menge an aufzunehmenden Daten erheblich ansteigt. Dabei erschwert die Vielzahl, im Vorfeld bekannter aber auch noch unbekannter Zusammenhänge zwischen den einzelnen Größen, die Analyse der aufgestellten Thesen, da auch die Wahrscheinlichkeit, dass relevante Daten beim Versuchsdesign nicht identifiziert und berücksichtigt wurden, steigt. Dies ist bei der Bewertung der Evaluationsergebnisse besonders zu berücksichtigen. Nicht signifikante Ergebnisse belegen nur, dass eine These

für die vorhandene Versuchskonstellation nicht bewiesen werden konnte. Es ist im Einzelfall jedoch zu prüfen, ob und in welchem Maße unbekannte oder nicht aufgenommene Variablen einen Einfluss auf das Ergebnis gehabt haben können. Beobachtungen der Versuchsleiter und die freien Antworten der Fragebögen dienen dabei als wertvolle Hinweise. Gegebenenfalls ist der Versuch mit geänderten Parametern zu wiederholen.

Die Versuche haben gezeigt, dass neben dem Entwurf des Versuchsdesigns auch die Auswahl der Probanden das Ergebnis im besonderen Maße mit beeinflusst. Die breite Streuung der Ausbildungsqualifikation führte dazu, dass Ergebnisse zu einzelnen Thesen teilweise gar nicht oder nur für bestimmte Probandengruppen signifikant sind. Zudem führte die geringe Erfahrung im Bereich der Modellierung und der Programmierung durchschnittlich zu relativ hohen Bearbeitungszeiten und Fehlerraten. Dennoch konnten aus den Ergebnissen bestimmte Aussagen abgeleitet werden, die als Grundlage für eine fokussierte Untersuchung in weiteren Arbeiten dienen. Die geplante Versuchsdurchführung mit erfahrenen SPS-Programmierern (z.B. Anwender mit mehrjähriger Erfahrung in der Industrie) musste leider entfallen, da es nicht möglich war, derartig qualifizierte Probanden in einer ausreichenden Anzahl für den Versuch zu gewinnen. In der Nachbetrachtung ist dies jedoch nicht als Nachteil zu bewerten. Vielmehr scheint es bei der hier dargestellten Vorgehensweise sinnvoll, eine erste Evaluation mit einer großen Anzahl von Probanden durchzuführen, auch wenn diese vergleichsweise unerfahren sind. Dadurch können Aspekte identifiziert werden, die in der Folge mit Hilfe von fokussierten Versuchen und einer geringeren Anzahl erfahrener Probanden evaluiert werden können.

Trotz der beschriebenen Schwierigkeiten hat sich die gewählte Vorgehensweise zur Untersuchung als geeignet herausgestellt, um Zusammenhänge auf einem bisher kaum untersuchten Gebiet erstmalig zu evaluieren. Es zeigte sich, dass die UML auch für die Modellierung und den Programmentwurf in der Steuerungstechnik anwendbar ist. Zwar stellt die UML, in der untersuchten Form, nicht das beste Mittel der Wahl dar, da ein effizienz- oder qualitätssteigernder Effekt einer derartigen Vorgehensweise nur unter bestimmten Randbedingungen nachgewiesen wurde. Es ist dabei jedoch zu berücksichtigen, dass die Probanden ein relativ geringes Vorwissen im Bereich der Steuerungsprogrammierung und/oder der Modellierung vorwiesen und ihnen im Rahmen der Versuchsvorbereitung nur eine relativ kurze Schulung in den zu verwendenden Modellierungsnotationen und Programmiersprachen gegeben werden konnte. Zudem beschränkte sich der praktische Übungsteil aus Zeitgründen auf ein Beispiel, welches mit den Probanden zusammen erarbeitet wurde. Somit ist zu erwarten, dass durch eine Fokussierung der Untersuchungen auf einzelne Teilaspekte und eine Anpassung der UML

z.B. durch die Verwendung von Profilen oder die Vorgabe einer Vorgehensweise bessere Ergebnisse erzielt werden können.

## 7.3 Vorschläge für die Durchführung weiterer Evaluationen

Die in dieser Arbeit beschriebenen Experimente stellen eine der ersten Untersuchungen dar, die den Einfluss einer Modellierung auf das Resultat der Steuerungsprogrammierung empirisch evaluiert. Grundlage für diese Untersuchungen waren die Fragestellungen, die in Kapitel 1 auf Basis der psychologischen Grundlagen sowie verschiedener empirischer Studien abgeleitet wurden. Die Durchführung der Experimente und die damit erzielten Ergebnisse ergaben darüber hinaus weitere Hinweise, welche Aspekte differenzierter betrachtet oder zusätzlich zu den hier evaluierten untersucht werden sollten.

Wie in Kapitel 7.1 dargestellt, ist die UML in ihrer uneingeschränkten Form für die Steuerungsprogrammierung zwar nur bedingt geeignet, die ICL ergibt jedoch in keinem der untersuchten Punkte signifikant bessere Ergebnisse. Folglich ist es sinnvoller, die Anwendung der UML weiter zu untersuchen und diese dabei durch geeignete Maßnahmen für den Einsatz bei der Steuerungsprogrammierung anzupassen.

Für alle weiteren Untersuchungen sollte die Modellierung mit Hilfe eines Modellierungswerkzeugs erfolgen. Im vorliegenden Experiment stellte die gewählte Vorgehensweise, nur mit Papier und Bleistift zu modellieren, für die Probanden offensichtlich eine Einschränkung dar. Zur Reduzierung dieses Effektes ist es notwendig, ein geeignetes offenes und gegebenenfalls anpassbares UML-Werkzeug zu verwenden. Dies kann zu einer deutlicheren Ausprägung des Nutzens der Modellierung gegenüber einer herkömmlichen, nicht unterstützten Programmierung führen.

Die Evaluation hat die Probleme unerfahrener Anwender, bei der Modellierung des Systems und der Übertragung des Modells auf das Steuerungsprogramm, gezeigt. In der Bewertung der Versuche durch die Probanden, im Rahmen der Beantwortung der Fragebögen, wurde dies meist mit dem Fehlen einer Vorgehensweise beschrieben. Dieser Aspekt muss weiter fokussiert werden. Es ist zu untersuchen, welchen Einfluss eine Vorgehensweise auf Zeit und Qualität bei der Programmerstellung hat. Dabei können zwei unterschiedliche Ansätze verfolgt werden: Zum einen besteht die Möglichkeit dem Probanden in der Trainingsphase explizit einen Leitfaden zur Verfügung zu stellen und ihn so durch die direkte Vorgabe einer Vorgehensweise bei seiner Aufgabe zu unterstützen. Zum anderen kann der Proband auch indirekt durch eine erhöhte Anzahl an Trainings- oder Beispielaufgaben und die daraus resultierende Steigerung seiner Erfahrung unterstützt werden. In diesem Fall wird der Leitfaden nicht explizit bereitgestellt, sondern es wird

versucht, dem Probanden durch eine größere Anzahl von Beispielen implizit einen Leitfaden anzutrainieren. Beide Ansätze haben dabei sowohl Vor- als auch Nachteile. Bei der expliziten Vorgabe eines Leitfadens werden die Freiheitsgrade und somit die Anzahl der unabhängigen Variablen deutlich eingeschränkt. Allerdings lassen die Ergebnisse ausschließlich Aussagen zu der untersuchten Vorgehensweise zu. Bei einer Änderung der Vorgehensweise sind jeweils neue Untersuchungen durchzuführen. Beim impliziten Training einer Vorgehensweise besteht hingehen die Unsicherheit, wie und in welchem Maße der Proband in der Lage ist, aus den Beispielen selbständig eine Vorgehensweise zu identifizieren. Individuelle Unterschiede in der Abstraktions- und Analysefähigkeit der Probanden machen es deshalb notwendig, die vom Probanden gewählte Vorgehensweise gesondert zu erheben. Zudem steigt durch die höhere Anzahl unabhängiger Variablen auch die für eine Evaluation benötigte Anzahl an Probanden. Vorteil dieses Trainings ist, dass die Vorgehensweisen erfasst werden, die die Probanden als anwendbar und zielführend erachten.

Für eine weiterführende Untersuchung im Bezug auf die Vorgehensweise ist deshalb ein zweistufiger Ansatz sinnvoll, bei dem in einem ersten Schritt ermittelt wird, welche Vorgehensweisen durch die Anwender beim impliziten Training bevorzugt werden. Anschließend besteht in einem zweiten Schritt dann die Möglichkeit, die Favoriten des ersten Schritts mit einer auf Basis einer Expertenevaluation hergeleiteten und optimierten Vorgehensweise mit explizitem Training vergleichend zu evaluieren.

Im Zusammenhang mit der Erfahrung ist auch der Einfluss eines Wiederholungseffektes näher zu untersuchen. Bei der erstmaligen Anwendung einer Modellierung zur Programmierung eines Steuerungssystems greift der Anwender auf Wissen zurück, welches er im Rahmen der Schulung erlangt hat. Da im vorliegenden Versuch bei der Schulung bewusst auf die Vorgabe einer Vorgehensweise verzichtet wurde, konnte er sich ausschließlich auf sein Wissen bezüglich der Modellierungsnotation und der Programmierumgebung stützen. Eine wiederholte Anwendung führt dazu, dass der Anwender ein Erfahrungswissen aufbaut, welches er bei folgenden gleichartigen Aufgaben anwendet. Fehler, die er begangen und auch als solche erkannt hat, wird er in ähnlichen Situationen bewusst versuchen zu vermeiden. Bei Problemen mit der von ihm gewählten Vorgehensweise, wird er versuchen diese anzupassen. So ist bei mehrmaliger Anwendung ein Effekt zu erwarten, der sich sowohl auf die Zeit als auch auf die Fehlerrate positiv auswirkt. In der vorliegenden Arbeit war der Wiederholungseffekt kein Bestandteil der Untersuchung, eine Evaluation in weiteren Arbeiten ist deshalb notwendig.

Auch im Bezug auf Modularität und Wiederverwendung, ist eine Verbesserung des Ergebnisses durch den Wiederholungseffekts zu erwarten. In der vorliegenden Untersuchung konnten die Ergebnisse die aufgestellten Thesen zur Modularität nicht bestätigen. Die geringe Erfahrung der Probanden mit objektorientierten Ansätzen wurde als ein möglicher Grund für geringe oder fehlende Modularität in Modell und Programm identifiziert. Wie in Kap. 6.6.2 dargestellt wiesen Basili et al. nach, dass ein positiver Effekt der Objektorientierung sich erst nach mehrmaliger Anwendung einstellt. Es sollte deshalb untersucht werden, ob eine wiederholte Anwendung der Modellierung sich förderlich auf Modularität und Wiederverwendung auswirkt. Darüber hinaus sollte für weitere Evaluationen ein komplexerer Prozess verwendet werden, der im Bezug sowohl auf Struktur als auch auf Verhalten ein höheres Maß an Wiederverwendung ermöglicht.

Durch die Untersuchung zusätzlicher Arbeitsaufgaben besteht die Möglichkeit den Nutzen einer Modellierung in weiteren Szenarien zu ermitteln. Die in der vorliegenden Arbeit untersuchten Anwendungsfälle behandeln ausschließlich die Erstellung neuer Steuerungssoftware. Es besteht darüber hinaus jedoch häufig die Anforderung, bestehende Automatisierungssysteme auf Basis der bestehenden Systemdokumentation zu warten oder zu modifizieren. Für den Bereich der Hardware ist diese Dokumentation in Form von Gerätespezifikationen, Stromlaufplänen oder Netzwerkplänen weitestgehend standardisiert und wird einheitlich angewendet. Für die Dokumentation von Steuerungssoftware wird jedoch keine standardisierte Dokumentationsform verwendet. Hier wird häufig direkt auf den Programmcode zurückgegriffen, um Fehler zu finden oder Anpassungen vorzunehmen. Beim Ersteller des ursprünglichen Steuerungsprogramms und dem Wartungs- oder Servicepersonal handelt es sich meist nicht um dieselben Personen. Individuelle Unterschiede der Programmierer bei der Programmerstellung erfordern folglich, dass das bestehende Programm analysiert und nachvollzogen wird, bevor entsprechende Änderungen vorgenommen werden können. Littmann et al. konnten nachweisen, dass eine geplante systematische Vorgehensweise zur Informationsbeschaffung bei der Modifikation bestehender Programme zu einem besseren Ergebnis führt als eine ungeplante, der momentanen Anforderung angepasste Vorgehensweise [LPL+86]. Folglich kann die Anwendung einer einheitlichen Dokumentation zur Beschreibung des Softwaresystems dazu beitragen, dass der dafür notwendige Zeitaufwand reduziert und die Wahrscheinlichkeit von Fehlern, z.B. durch falsche Interpretation des bestehenden Programmcodes, minimiert wird. In weiteren Experimenten muss deshalb untersucht werden, wie sich die Anwendung einer einheitlichen, auf der UML basierenden Programmdokumentation auf den Zeitbedarf und die Fehleranfälligkeit bei der Wartung und Modifikation fremder Steuerungssoftware auswirken.

Neben der Untersuchung der IEC 61131-3, die derzeit Standard in der Steuerungsprogrammierung ist, sollten weitere Arbeiten sich auch mit der Untersuchung neuer Ansätze im Bereich der Programmiersprachen und der Engineeringwerkzeuge befassen. So wird mit CoDeSys 3.0 [Hes05] erstmalig ein Ansatz zur Verfügung gestellt, der Eigenschaften der Objektorientierung auch für die Programmierung von Speicherprogrammierbaren Steuerungen nutzt. Dies verringert die Diskrepanz, die zwischen der objektorientierten Modellierung und der prozeduralen Programmierung existiert. Es ist anzunehmen, dass dies im Vergleich zur Standard-IEC 61131-3 zu einer Leistungssteigerung führt.

Darüber hinaus existieren erste Ansätze, die eine direkte Integration einer UML-basierten Modellierung in die Programmierumgebung realisieren [WSV08]. Dabei ermöglicht die Generierung des Programmcodes direkt aus dem UML-Modell eine Programmerstellung ohne die Notwendigkeit der Programmierung. Die Effektivität dieser Vorgehensweise sowie die Auswirkungen auf die Wartbarkeit und Änderbarkeit des Programmcodes sind bisher jedoch nicht bekannt und könnten in ähnlichen Experimenten evaluiert werden.

# 8 Zusammenfassung und Ausblick

In der vorliegenden Arbeit wurde eine Evaluation vorgestellt, deren Ziel es war, eine wissenschaftliche Grundlage für Aussagen über Anwendbarkeit von Methoden und Werkzeugen des konventionellen Softwareengineering für die Steuerungsprogrammierung zu schaffen. Es wurde eine Reihe von kontrollierten Experimenten mit insgesamt 89 Probanden unterschiedlicher Qualifikation (Studenten, Auszubildende, Technikerschüler und Studenten im Praxisverbund) durchgeführt, um den Nutzen der Unified Modeling Language (UML) und der Idiomatic Control Language (ICL) empirisch zu untersuchen.

Die Ergebnisse zeigen, dass eine Modellierung nicht zu einer höheren Bearbeitungsdauer des Softwareprojekts führte und widersprechen somit der weit verbreiteten Meinung, dass die Effizienz im Engineering durch die zusätzliche Aufgabe zwangsläufig sinkt. Darüber hinaus konnte für bestimmte Aspekte sogar ein positiver Effekt durch die Modellierung festgestellt werden. So erzielten Probenden, welche die UML zur Modellierung nutzten und angaben in SPS-Programmierung erfahren zu sein, im Vergleich zu unerfahrenen Probanden, eine signifikant niedrigere Fehlerrate bei der Programmierung.

Es wurde jedoch auch deutlich, dass die UML in ihrer uneingeschränkten Form nur bedingt geeignet ist, um die Steuerungsprogrammierung zu unterstützen. Die große Anzahl von Diagrammen und die große Freiheit bei der Modellierung führen, besonders bei unerfahrenen Anwendern, schnell zur Verwirrung. Dies führte zu einer negativen Beeinflussung des Gesamtergebnisses und steuerte unter anderem zur geringen Qualität der erstellten Modelle und der hohen Fehlerrate bei der Programmierung bei.

Im Vergleich der beiden untersuchten Beschreibungsmittel konnte kein signifikanter Vorteil für eine der beiden Notationen nachgewiesen werden. Die existierenden Möglichkeiten zur Anpassung der UML sowie die Tatsache, dass die ICL in keinem der untersuchten Punkte signifikant bessere Ergebnisse erzielte, führen jedoch zu der Folgerung, dass eine weitere Untersuchung der UML den sinnvolleren Weg darstellt.

Die gewählte Vorgehensweise der Evaluation mittels kontrollierter Experimente hat sich für den untersuchten Anwendungsbereich als geeignetes Mittel herausgestellt, um Zusammenhänge auf einem bisher kaum untersuchten Gebiet erstmalig zu evaluieren. Neben der Evaluation der aufgestellten Thesen konnten durch das offene Versuchsdesign Bereiche und mögliche Zusammenhänge identifiziert werden, für die eine fokussierte Untersuchung in weiterführenden wissenschaftlichen Arbeiten sinnvoll ist.

Darüber hinaus wurden durch die Experimente konkrete Hinweise darauf gesammelt, welche Maßnahmen ergriffen werden können, um Nutzen einer UML-basierten Modellierung für die Steuerungsprogrammierung zu steigern. Durch eine Anpassung der UML an die Anforderungen im Softwareengineering der Automatisierungstechnik besteht die Möglichkeit, auf relevante Diagrammformen und Beschreibungsmechanismen zu fokussieren. Auch eine optimierte und standardisierte Vorgehensweise wirkt der Verwirrung beim Nutzer entgegen, indem ihm ein Leitfaden für eine zielführende Anwendung des Beschreibungsmittels zur Verfügung gestellt wird.

Auf Basis der erzielten Ergebnisse und aktueller Entwicklungen im Bereich der Steuerungsprogrammierung ist zu erwarten, dass der Aspekt der Systemmodellierung mit Hilfe eines für die Automatisierungstechnik angepassten Beschreibungsmittels zunehmend ein integraler Bestandteil des Engineeringprozesses wird. Obschon entsprechende Beschreibungsmittel mit der UML-PA oder SysML bereits entworfen sind, muss ihre Eignung für die gestellte Aufgabe erst noch nachgewiesen werden. Doch auch heute gehören Beschreibungsmittel, wie die UML und deren Anwendung zur Modellierung automatisierungstechnischer Systeme, immer häufiger zum Lehrplan der Ingenieurs- oder Technikerausbildung.

Darüber hinaus wird durch eine Kombination der Programmiersprachen der IEC 61131 mit objektorientierten Konstrukten die Schere zwischen objektorientierter Modellierung und prozeduraler Steuerungsprogrammierung geschlossen. Mit der Umsetzung in industrietaugliche Programmierwerkzeuge, wie CoDeSys V3, erfolgte eine Einführung objektorientierter Methoden in die industrielle Steuerungsprogrammierung. In einem aktuellen Forschungsvorhaben des Fachgebiets Eingebettete Systeme der Universität Kassel erfolgt derzeit gemeinsam mit den Unternehmen 3S und Beckhoff sowie verschiedenen Anwendern des Maschinen- und Anlagenbaus zudem eine Integration von UML-Diagrammen in die Entwicklungsumgebung CoDeSys. Diese Integration ermöglicht eine direkte Generierung des Steuerungscodes aus dem Modell und macht zusätzliche Schnittstellen zwischen den verschiedenen Planungswerkzeugen oder die händische Umsetzung des Modells in den Code überflüssig.

Die Gesamtheit der beschriebenen Maßnahmen und Entwicklungen wird dazu beitragen, den Nutzen einer Modellierung im Rahmen der Steuerungsprogrammierung zu steigern, um dadurch die Effizienz des Engineering und die Qualität der Steuerungssoftware in der Automatisierungstechnik zu steigern.

# 9 Literaturverzeichnis

[ABH+05] Arisholm, E.; Briand, L.C.; Hove, S.E.; Labiche, Y.: *The Impact of UML Documentation on Software Maintenance: An Experimental Evaluation*. In: IEEE Transactions on Software Engineering, Vol. 32(6), 2006, S. 365-381.

[AhPo89] Ahrens, W.; Polke, M.: *Informationsstrukturen in der Leittechnik*. In: Polke, M.: Prozeßleittechnik, 2. Auflage, Oldenbourg- Industrieverlag, München, 1994.

[All97] Allen, R.B.: *Mental Models and User Models*. In: Helander, M.; Landauer, T.K.; Prabhu, P: Handbook of Human-Computer Interaction, 2nd completely revised edition, Esevier Science B.V., North-Holland, 1997.

[AnTr00] Anderl, R.; Trippner, D.: *STEP – Standard for the Exchange of Product Model Data*. Teubner Verlag, Stuttgart, 2000.

[APW06] Astleitner, H.; Pasuchin, I.; Wiesner, C.: *Multimedia und Motivation - Modelle der Motivationspsychologie als Grundlage für didaktische Mediengestaltung*. In: MedienPädagogik - Online-Zeitschrift für Theorie und Praxis der Medienbildung, www.medienpaed.com, Zürich, Heft 12, 2006.

[ArHo74] Arvey, R.D.; Hoyle, J.C.: *A Guttmann approach to the development of behaviorally based scales for system analysts and programmer analysts*. In: Journal of Applied Psychology, American Psychological Association, Washington DC, Vol. 59(1), 1974, S. 61-68.

[Bal01] Balzert, H.: *Lehrbuch der Software-Technik*. Spektrum Akademischer Verlag, Heidelberg, 2001.

[BaVo01] Bartels, J.; Vogel, B.: *Systementwicklung für die Automatisierung im Anlagenbau*. In: Automatisierungstechnik (at), Oldenbourg Wissenschaftsverlag, München, Vol. 49(5), 2001, S. 214-224.

[BCG+92] Basili, V.; Caldiere, G.; McGarry, F.; Pajerski, R.; Page, G.; Waligora, S.: *The software engineering laboratory - an operational software*

*experience factory*. In: Proceedings of the 14th international conference on Software engineering (ICSE), Melbourne, 1992, S. 370-381.

[BeRe05a] Bergin, S.; Reilly, R.: *The influence of motivation and comfort-level on learning to program*. In: 17th Workshop of the Psychology of Programming Interest Group, Sussex University, 2005.

[BeRe05b] Bergin, S.; Reilly, R.: *Programming: Factors that Influence Success*. In: Proceedings of the thirty-fifth SIGCSE technical sympsium on Computer Science Education (SIGCSE), St. Louis, 2005.

[BFS05] Bonfe, M.; Fantuzzi, C.; Secchi, C.: *Verification of Behavioral Substitutability in Object-Oriented Models for Industrial Controllers*. In: Proceedings of the IEEE International Conference on Robotics and Automation, Vol. 4, Barcelona, 2005.

[Bla74] Blaiwes, A.S.: *Formats for Representing Procedural Instructions*. In: Journal of Applied Psychology, Vol. 59(6), 1974.

[Bor99] Bortz, J.: *Statistik für Sozialwissenschaftler*. Springer-Verlag, Berlin, 1999.

[Bri00] Bristol, E.H.: *A Field Device Language in a Process Control Language Family*. ISA Expo, New Orleans, 2000.

[Bri05] Bristol, E.H.: *Idiom Based Documenation of Continuous (Feedback) Process Control*. Webseite des Authors, http://homepage.mac.com/ebristol, abgerufen 15.01.2009.

[Bri06] Bristol, E.H.: *Large System ICL, Rough Specification*. Webseite des Authors, http://homepage.mac.com/ebristol, abgerufen 15.01.2009.

[Bri94] Bristol, E.H.: *A Language for integrated Process Control Application*. Retirement Symposium in Honor of Prof. Ted Williams, Purdue University, West Lafayette, 1994.

[Bri95] Bristol, E.H.: *Not a Batch Language; A Control Language*. ISA Transactions, Jahrgang 34, Ausgabe 4, 1995, S. 387-403.

[BRJ99] Booch, G.; Rumbaugh, J.; Jacobson, I.: *Das UML-Benutzerhandbuch*. Addison-Wesley, München, 1999.

[Car88] Caroll, J.M.: *Mental Models in Human-Computer Interaction*. In: Helander, M.: Handbook of Human-Computer Interaction, Elsevier Science B.V., North-Holland, 1988.

[Chr98] Christensen, L. B.: *Experimental Methodology*. 7. Auflage, Allyn & Bacon, Boston, 1998.

[CSK+89] Curtis, B.; Sheppard, S.B.; Kruesi-Bailey, E.; Bailey, J.; Boehm-Davis, D.: *Experimental evaluation of Software Documentation Formats*. In: The Journal of System and Software, Vol. 9(2), 1989, S. 167-207.

[CTM80] Carol, J.M.; Thomas, J.C.; Malhotra, A.: *Presentation and representation in design problem solving*. In: British Journal of Psychology, Cambridge University Press, Cambridge, Vol 71(1), 1980, S. 143-153.

[Cur88] Curtis, B.: *Five Paradigms in the Psychology of Programming*. In: Helander, M.: Handbook of Human Computer Interaction. Ed. North Holland, Amsterdam, 1988, S. 87-106.

[CuTa87] Cunniff, N.; Taylor, R.P.: *Graphical vs. textual representation: An empirical study of novices' program comprehension*. In: Soloway, E.; Sheppard, S.: Empirical Studies of Programmers: Second Workshop, Intellect Books, Bristol, 1987.

[Dét95] Détienne, F.: *Design Strategies and Knowledge in Object-Oriented Programming: Effects of Experience*. In: Human-Computer Interaction, Vol. 10(2/3), 1995.

[DHM98] Dröschel, W.; Heuser, W.; Midderhoff, R.: *Inkrementelle und objektorientierte Vorgehensweisen mit dem V-Modell 97*. Oldenbourg-Verlag, München,1998.

[DIN19226] *DIN 19226: Leittechnik, Regelungstechnik und Steuerungstechnik*. Beuth-Verlag, Berlin, 1987.

[DIN69901] *DIN 69901: Projektwirtschaft, Projektmanagement, Begriffe*. Beuth-Verlag, Berlin, 1994.

[DIN61131-1] *DIN EN 61131-1: Speicherprogrammierbare Steuerungen - Teil 1: Allgemeine Informationen.* Beuth-Verlag, Berlin, 2003.

[DIN61131-2] *DIN EN 61131-2: Speicherprogrammierbare Steuerungen - Teil 2: Betriebsmittelanforderungen und Prüfungen*. Beuth-Verlag, Berlin, 2007.

[DIN61131-3] *DIN EN 61131-3: Speicherprogrammierbare Steuerungen - Teil 3: Programmiersprachen*. Beuth-Verlag, Berlin, 2003.

[DINV19233] *DINV19233: Automatisierung mit Prozeßrechensystemen, Begriffe*. Beuth-Verlag, Berlin, 1998.

[DrFe04a] Drath, R.; Fedai, M.: *CAEX - ein neutrales Datenaustauschformat für Anlagendaten - Teil 1*. In: atp - Automatisierungstechnische Praxis, Oldenbourg Industrieverlag, München, Vol 46(2), 2004, S. 52-56.

[DrFe04b] Drath, R.; Fedai, M.: *CAEX - ein neutrales Datenaustauschformat für Anlagendaten - Teil 2*. In: atp - Automatisierungstechnische Praxis, Oldenbourg Industrieverlag, München, Vol 46(3), 2004, S. 20-27.

[Dut94] Dutke, S.: *Mentale Modelle: Konstrukte des Wissens und Verstehens*. Verlag für angewandte Psychologie, Göttingen, Stuttgart, 1994.

[ErMa04] Ermlich, R.; Maier, U.: *Speicherprogrammierbare Steuerungen (SPS) und SPS-Systeme*. In: Früh, K.F.; Maier, U.: Handbuch der Prozessautomatisierung. 3. Auflage, Oldenbourg Industrieverlag, München, 2004.

[Fay05] Fay, A.: *Engineering in verteilten, offenen, durchgängigen Systemen*. In: at - Automatisierungstechnik, Oldenbourg Wissenschaftsverlag, München, Vol 53(4-5), 2005, S. 205-210.

[For07] Forbig, P.: *Objektorientierte Softwareentwicklung mit UML*. 3. Auflage, Hanser Fachbuchverlag, München , 2007.

[Fri02] Friedrich, D.: *Integriertes Engineering von automatisierungstechnischen Systemen entlang des gesamten Lebenszyklus*. Tagungsband zum Workshop "Elektrotechnik CAD", Stuttgart, 2002.

[Fri09] Friedrich, D.: *Anwendbarkeit von Methoden und Werkzeugen des konventionellen Softwareengineering zur Modellierung und Programmierung von Steuerungssystemen*. `urn:nbn:de:0002-7319`, Online-Fassung der Dissertation mit Anhang, Kassel University Press, 2009.

[FrKo03] Frey, G.; Kowalewski, S.: *Klassifizierung und Bewertung von Beschreibungsmitteln in der Automatisierungstechnik*. In: VDI-Berichte

1756, Proceedings of GMA Kongress 2003, VDI-Verlag, Düsseldorf, 2003.

[FrVo02] Friedrich, D.; Vogel-Heuser, B.: *Integrated automation engineering along the life-cycle*. In: Proceedings of the 28th Annual Conference of the IEEE Industrial Electronics Society (IECON), Sevilla, 2002.

[FVB03] Friedrich, D.; Vogel-Heuser, B.; Bristol, E. H.: *Evaluation of Modelling Notations for Basic Software Engineering in Process Control*. In: Proceedings of the 29th Annual Conference of the IEEE Industrial Electronics Society (IECON), Roanoke, 2003.

[FrVo05a] Friedrich, D.; Vogel-Heuser, B.: *Evaluating the Benefit of Modelling Notations for PLC-Programming Quality*. In: 11th International Conference on Human-Computer Interaction (HCI international), Las Vegas, 2005.

[FrVo05b] Friedrich, D.; Vogel-Heuser, B.: *Einfluss der Modellierung auf die Qualität der Steuerungsprogrammierung in der Ingenieurausbildung*. In: VDI-Berichte 1883, Proceedings of GMA Kongress 2005, VDI-Verlag, Düsseldorf, 2005.

[FrVo06] Friedrich, D., Vogel-Heuser, B.: *Nutzen von Modellierung für die Qualität und Effizienz der Steuerungsprogrammierung in der Automatisierungstechnik*. In: atp - Automatisierungstechnische Praxis, Oldenbourg Industrieverlag, München, Vol 48(3), 2006, S. 54-60.

[GaCh87] Gardiener, M.M.; Christie, B.: *Applying cognitive psychology to user-interface design*. John Wiley & Sons Inc., Hoboken, 1987.

[Grö04] Grötsch, E.: *SPS - Speicherprogrammierbare Steuerungen als Bausteine verteilter Automatisierung*. Oldenbourg Industrieverlag, München, 2004.

[GrPa78] Grell, J.; Pallatsch, W.: *Selbstgesteuertes Lernen in einer Kultur der Fremdsteuerung*. In: Neber, H.; Wagner, C.; Einsiedler, W.: Selbstgesteuertes Lernen. Beltz-Verlag, Weinheim, 1978, S. 88-108.

[HaKi99] Hahn, J.; Kim, J.: *Why are some diagrams easier to work with? Effects of diagrammatic representation on the cognitive intergration process of systems analysis and design*. In: ACM Transactions on Computer-Human Interaction (TOCHI), Vol. 6(3), 1999, S. 181-213.

[Hes05] Hess, D.: *Objektorientierte Erweiterung der IEC 61131-3*. In: atp - Automatisierungstechnische Praxis, Oldenbourg Industrieverlag, München, Vol. 47(11), 2005, S. 72-75.

[HrRu02] Hruschka, P.; Rupp, C.: *Agile Softwareentwicklung für Embedded Real-Time Systems mit der UML*. Oldenbourg Industrieverlag, München, 2004.

[IEC19501] *ISO/IEC 19501 - Information technology -- Open Distributed Processing -- Unified Modeling Language (UML) Version 1.4.2*. ISO International Organization for Standardization, 2005.

[IEEE1061] *IEEE1061: Software quality metrics methodology*. IEEE Institute of Electrical and Electronics Engineers, 1998.

[Jon98] Jones, C.: Estimating Software Costs. McGraw-Hill Osborne Media, New York, 2007.

[JoTi00] John, K.-H.; Tiegelkamp, M.: *SPS-programmierung mit IEC 61131-3: Konzepte und Programmiersprachen, Anforderungen an Programmiersysteme, Entscheidungshilfen*. 3.Auflage, Springer-Verlag, Berlin, 2000.

[Kat08] Katzke, U.: *Spezifikation und Anwendung einer Modellierungssprache für die Automatisierungstechnik auf Basis der Unified Modeling Language (UML)*. Dissertation Universität Kassel, Kassel, 2008.

[KBB02] Kutar, M.; Britton, C.; Barker, T.: *A Comparison of Empirical Study and Cognitive Dimensions Analysis in the Evaluation of UML Diagrams*. In: Proceedings of the 12th IEEE International Workshop on Program Comprehension (IWPC), West London, 2004.

[KKP+99] Kamsties, E.; von Knethen, A.; Philipps, J.; Schätz, B.: *Eine vergleichende Fallstudie mit CASE-Werkzeugen für objektorientierte und funktionale Modellierungstechniken*. In: OMER Workshop Proceedings, Bericht Nr. 1999-01, 1999.

[KVW05] Katzke, U.; Vogel-Heuser, B.; Wannagat, A.: *Möglichkeiten der Darstellung von Zustandsautomaten in der IEC 61131-3*. In: Vogel-Heuser, B; Holleczek, P.: Koordinierung autonomer Systeme, Springer-Verlag, Berlin, 2005.

[LaCh04] Lange, C.; Chaudron, M.: *An Empirical Assessment of Completeness in UML Design*. In: Proceedings of the 8th International Conference on Empirical Assessment in Software Engineering (EASE), Edinburgh, 2004, S. 111-121.

[LaCH06] Lange, C.; Chaudron, M.: *Effects of defects in UML models: an experimental investigation*. In.: Proceedings of the 28th international conference on Software engineering, Shanghai, 2006, S. 401-411.

[LaGö99a] Lauber, R.; Göhner, P.: *Prozessautomatisierung 1*. Springer-Verlag, Berlin, 1999.

[LaGö99b] Lauber, R.; Göhner, P.: *Prozessautomatisierung 2*. Springer-Verlag, Berlin, 1999.

[Lon89] Long, J.: *Cognitive Ergonomics and Human-Computer Interaction: an introduction*. In: Long, J.; Whitefield, A.: Cognitive Ergonomics and Human-Computer Interaction. Cambridge University Press, ORT, 1989.

[LPL+86] Littman, D.C.; Pinto, J.; Letovsky, S.; Soloway, E.: *Mental Models and Software Maintenance*. In: Iyengar, S.S.; Soloway, E.: Empirical Studies of Programmers: First Workshop. Ablex Publishing Corp., Norwood, 1986.

[Lun03] Lunze, J.: *Automatisierungstechnik – Methoden für die Überwachung und Steuerung kontinuierlicher und ereignisdiskreter Systeme*. Oldenbourg Wissenschaftsverlag, München, 2003.

[May76] Mayer, R.E.: *Comprehension as Affected by the Structure of Problem Representation*. In: Memory & Cognition, Vol. 4(3), 1976.

[May01] Mayer, R.: *Multimedia Learning*. Cambridge University Press, New York, 2001.

[MBW95] Mossenbock, H.; Bach, R.; Wirth, N.: *Object-oriented Programming in Oberon-2*. Springer-Verlag Berlin, 1995.

[Mil56] Miller, G.A.: *The magical number seven, plus or minus two: Some limits on our capacity for processing information*. In: The Psychological Review, Vol. 63(2), 1956, S. 81-97.

[NA35] *NA 35: Abwicklung von PLT-Projekten*. Arbeitsblatt NA35 der Namur, Version vom 24.3.2003, http://www.namur.de, 2003.

[Nit08] von Nitzsch, R.: *Entscheidungslehre: Wie Menschen entscheiden und wie sie entscheiden sollten*. 5. Auflage, Verlagshaus Mainz, Aachen, 2008.

[Nor83] Norman, D.A.: *Some observations on mental models*. In: Gentner, D.; Stevens, A.L.: Mental Models. Lawrence Erlbaum Associates, Hillsdale, 1983.

[Oes01] Oestereich, B.: *Objektorientierte Softwareentwicklung: Analyse und Design mit der Unified Modeling Language*. Oldenbourg Industrieverlag, München, 2001.

[OMG02] *Unified Modeling Language Specification 1.4.2*. Object Management Group, Needham, 2002.

[OMG05] *UML Profile for Schedulability, Performance, and Time, version 1.1*. Object Management Group, Needham, 2005.

[OMG07] *SysML Specification*. Object Management Group, Needham, 2007.

[OtDo02] Otero, M.C.; Dolada, J.J.: *An Initial Experimental Assessment of the Dynamic Modelling in UML*. In: Empirical Software Engineering, Vol. 7(1), 2002, S. 27-47.

[Pai86] Paivio, A.: *Mental representations: a dual coding approach*. Oxford University Press, Oxford, 1986.

[Pen03] Pender, T.: *UML Bible*. John Wiley & Sons Inc., Hoboken, 2003.

[PISTEP94] PI-Step: *Process Plant Engineering Activity Model*. Webseite des Process Industries STEP Consortium, http://www.pistep.org.uk, abgerufen 15.01.2009.

[Pol94] Polke, M.: *Prozessleittechnik*. Oldenbourg Industrieverlag, München, 1994.

[RaBr06] Rausch, A.; Broy, M.: *Das V-Modell XT: Grundlagen, Erfahrungen und Werkzeuge*. Springer-Verlag, Berlin, Heidelberg, 2006.

[RHP+66] Reinstedt, R.N.; Hammidi, B.C.; Peres, S.; Ricard, E..: *Computer Personnel Research Group Programmer Performance Prediction Study*. The RAND Corporation, Santa Monica, 1966.

[Sca87] Scane, R.: *A historical perspective.* In: Gardiener, M.M.; Christie, B.: Applying cognitive science to user-interface design. John Wiley & Sons, New York, 1987.

[Sha76] Shaw, M.E.: Group dynamics: The Psychology of Small Group Behavior. 2nd edition, McGraw-Hill Book Company, New York, 1976.

[Sch03] Schnieder, E.: *Integration heterogener Modellwelten der Automatisierungtechnik.* In: Nagl, M.; Westfechtel, B.: Modelle, Werkzeuge und Infrastrukturen zur Unterstützung von Entwicklungsprozessen. Wiley-VHC, Weinheim, 2003, S. 311-324.

[Sch93] Schnieder, E.: *Prozessinformatik: Automatisierung mit Rechensystemen; Einführung mit Petrinetzen*. 2. Auflage, Vieweg-Verlag, Braunschweig, 1993.

[Sch97] Scheffler, E.: *Statistische Versuchsplanung und -auswertung*. Deutscher Verlag für Grundstoffindustrie, Stuttgart, 1997.

[Sch08] Schünemann, U.: *Objektorientierung CoDeSys3*. In: Vogel-Heuser, B.: Automation & Embedded Systems. Oldenbourg Industrieverlag, München, 2008.

[SeRu98] Selic, B.; Rumbaugh, J.: *Using UML for Modeling Complex Real-Time Systems*. In: Languages, Compilers, and Tools for Embedded Systems, Lecture Notes of Computer Science, Vol. 1474, Springer-Verlag Berlin, 1998.

[TiHi03] Tilley, S.; Huang, S.: *A Qualitative Assessment of the Efficacy of UML Diagrams as a Form of Graphical Documentation in Aiding Program Understanding*. In: Proceedings of the 21st annual international conference on Documentation (SIGDOC), San Francisco, 2003, S. 184-191.

[VDE3681] *VDI/VDE 3681: Einordnung und Bewertung von Beschreibungsmitteln aus der Automatisierungstechnik*. Beuth-Verlag, Berlin, 2005.

[VFK+05] Vogel-Heuser,B.; Friedrich, D.; Katzke, U.; Witsch, D. : *Usability and benefits of UML for plant automation – some research results*. In: atp international, Vol. 1(1), Oldenbourg Industrieverlag, München, 2005.

[Vog03] Vogel-Heuser, B.: *Systems Software Engineering: Angewandte Methoden des Systementwurfs für Ingenieure*. Oldenbourg Industrieverlag, München, 2003.

[VoWa08] Vogel-Heuser, B.; Wannagat, A.: *Wiederverwendung und Modulares Engineering mit CoDeSys V3*. Oldenbourg Industrieverlag, München, 2008.

[Wei02] Weidenmann, B.: *Multicodierung und Multimodalität im Lernprozess*. In: Issing, L.J.; Klimsa, P.: Information und lernen mit Multimedia und Internet. Beltz-Verlag, Weinheim, 2002.

[Wei06] Weilkins, T.: *Systems Engineering mit SysML / UML. Modellierung, Analyse, Design*. 1. Auflage, dpunkt.verlag, München, 2006.

[Wei86] Weidenmann, B.: *Psychische Prozesse beim Verstehen von Bildern*. Verlag Hans Huber, Bern, 1986.

[WeOe04] Weilkins, T.; Oestereich, B.: *UML 2- Zertifizierung*. dpunkt.verlag, München, 2004.

[Wot88] Wottawa, H.: *Psychologische Methodenlehre: Eine orientierende Einführung*. 2. Auflage, Juventa- Verlag, Weinheim, 1988.

[WSV08] Witsch, D.; Schünemann, U.; Vogel-Heuser, B.: *Steigerung der Effizienz und Qualität von Steuerungsprogrammen durch Objektorientierung und UML*. In.: atp - Automatisierungstechnische Praxis, Oldenbourg Industrieverlag, München, Vol. 50(11), 2008, S. 42-47.