Model Driven Engineering - Zukunft des Entwurfs von Elektronik im Auto

Ruben Zimmermann
250 743

Betreut von Dipl.-Inform. Christian Fuß

Zusammenfassung

Dieser Artikel beschreibt den modellbasierten Entwicklungsprozess für Software im Automobilbereich, der im Rahmen des STEP-X Projektes vom Zentrum für Verkehr der TU-Braunschweig in Zusammenarbeit mit der Volkswagen AG entwickelt wurde. Im Projekt wird ein durchgehender Prozess von der Anforderungserfassung bis zur Codegenerierung vorgeschlagen und auch die Bereiche Test und Diagnose werden frühzeitig integriert. Gleichzeitig werden verschiedene CASE-Tools zur optimalen Unterstützung des Prozesses vorgestellt.

Inhaltsverzeichnis

1 Einleitung

Im Automobilbereich ist in den letzten Jahren eine Verschiebung des Entwicklungsschwerpunktes weg von mechanisch/hydraulischen Systemen hin zu elektronischen Systemen mit großem Softwareanteil zu beobachten. Diese Entwicklung wird besonders deutlich im Bereich der Sicherheits- und Komfortsysteme, die zunehmend für den Erfolg eines Fahrzeugs verantwortlich sind. Die Prognose der Wertschöpfungsanteile, zu sehen in Abbildung 1, bestätigt diese Entwicklung.

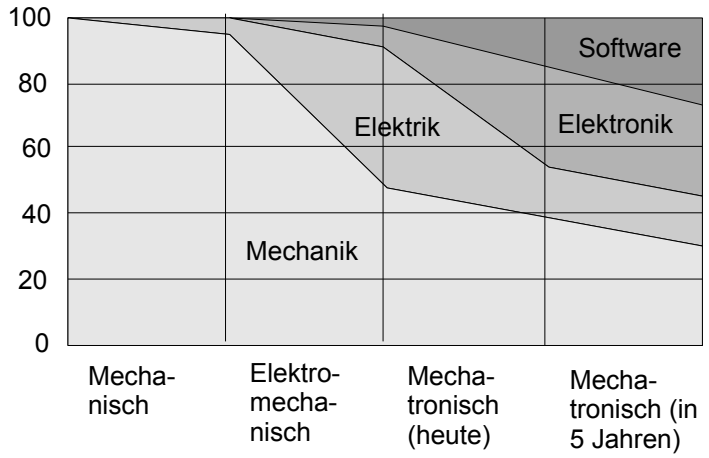

Abbildung 1: Wandel der Wertschöpfungsanteile [2]

Entfallen zur Zeit noch etwa 40% der Herstellungskosten eines Fahrzeugs auf den Bereich der Mechanik, so wird in Zukunft dieser Anteil, genau wie der der Elektrik weiter sinken, wohingegen die Bereiche Software und Elektronik weiter an Wertschöpfungsanteilen gewinnen werden. Dies geht einher mit immer komplexeren Systemen und immer schnelleren Weiterentwicklungen. Um trotz der steigenden Komplexität und der kürzeren „time-to-market cycles“ die nötige Qualität der Produkte gewährleisten zu können, bedarf es entsprechend guter Entwicklungsprozesse. Ein solcher Prozess wurde im Rahmen des STEP-X (STrukturierter EntwicklungsProzess für Automotive-Anwendungen am Beispiel X-By-Wire) Projekts vorgeschlagen und ist Thema dieses Artikels. STEP-X ist ein Gemeinschaftsprojekt zwischen dem Zentrum für Verkehr der TU-Braunschweig und der Volkswagen AG. Die beteiligten Institute des Zentrums für Verkehr sind im Einzelnen das Institut für Regelungs- und Automatisierungstechnik, das Institut für Software sowie das Institut für Messtechnik und Grundlagen der Elektrotechnik. Zur Zielsetzung des Projektes heißt es in [7]: „Ziel des Projektes STEP-X ist es, einen durchgängigen Entwicklungsprozess von der Definition der Anforderungen

bis zur automatischen Codegenerierung für elektronische Systeme mit großem Softwareanteil aufzuzeigen.“ Der folgende Artikel stellt die Bestandteile dieses Prozesses, die Verknüpfung der Prozessstufen untereinander und die Werkzeuge, durch die dieser Prozess unterstützt wird vor. Im folgenden Kapitel werden dazu zunächst die wichtigsten Werkzeuge, die im Verlaufe des Prozesses Anwendung finden, sowie das V-Modell, an dem sich der Prozess orientiert, vorgestellt. In Kapitel 3 werden anschließend die einzelnen Stufen des Prozesses erläutert und Kapitel 4 behandelt die Aktivitäten Test und Diagnose, die über alle Prozessstufen hinweg in die Entwicklung integriert sind. Letztlich folgt in Kapitel 5 eine Zusammenfassung.

2 Unterstützung der Softwareentwicklung durch V-Modell und CASE-Tools

Bei der Beschreibung des STEP-X Prozesses in Kapitel 3 wird an einigen Stellen sowohl auf das V-Modell, als auch auf verschiedene CASE-Tools Bezug genommen. Daher stellt das folgende Kapitel kurz das allgemeine V-Modell nach Boehm und die einzelnen Werkzeuge vor.

2.1 Das V-Modell

Das allgemeine V-Modell nach Boehm [3] bildet die Grundlage vieler Vorgehensmodelle in der Softwareentwicklung. Ein Vorgehensmodell dient der Projektplanung, indem es den Entwicklungsprozess in einzelne Phasen unterteilt. Die einzelnen Phasen sind dabei inhaltlich und zeitlich abgegrenzt und werden im Normalfall nacheinander durchlaufen. Als Beispiel für ein solches Vorgehensmodell zeigt Abbildung2 ein Wasserfallmodell. Hier wird der gesamte Entwicklungsprozess in sechs Phasen aufgeteilt, die von oben nach unten durchlaufen werden. Die Verknüpfungen zwischen den Phasen sind durch die Pfeile dargestellt, wobei Verbindungen immer nur zwischen einer Phase und dem direkten Vorgänger/Nachfolger besteht.
Das allgemeine V-Modell nach Boehm verzichtet auf eine strenge chronologische Phaseneinteilung. Eine typische Darstellung des Modells mit den konstruktiven Elementen des Prozesses auf der linken Seite und den integrativen und validierenden Elementen im rechten Ast, zeigt Abbildung 3. Die Übergänge im absteigenden Ast sollen dabei weniger eine zeitliche Abfolge darstellen, als vielmehr den Detaillierungsgrad der auf der jeweiligen Stufe entstehenden Produkte. Im

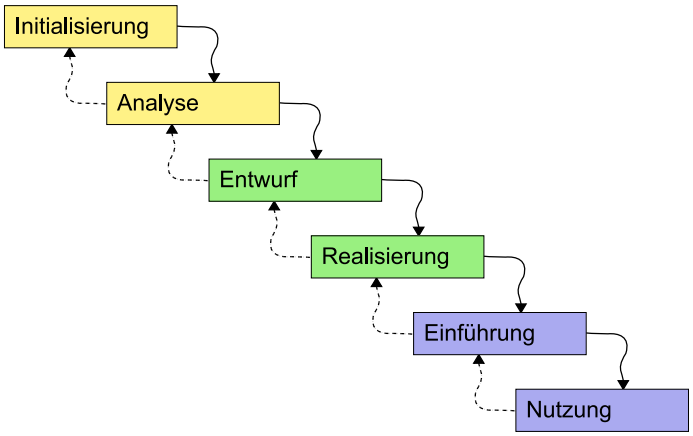

Abbildung 2: Wasserfallmodell [4]

aufsteigenden Ast wächst demgegenüber mit der Höhe der Ebene die Größe des Systems. Wesentlich für das V-Modell sind die vertikalen Validierungsbeziehungen zwischen den konstruktiven und den validierenden Aktivitäten, die jeweils anzeigen, gegen welche Anforderungen die Produkte des aufsteigenden Astes validiert werden.
Der STEP-X Prozess sieht eine etwas andere Einteilung der Prozessphasen vor, als in 3 gezeigt. Eine Übersicht der Stufen, in die der STEP-X Entwicklungsprozess unterteilt wird bietet Abbildung 7. Auf den einzelnen Ebenen kommen unterschiedliche Werkzeuge zur Unterstützung der Entwicklung zum Einsatz. Im Folgenden werden diese Werkzeuge sowie die Phase, in der sie eingesetzt werden vorgestellt.

2.2 DOORS 6.0

Eine Kernaktivität bei der Softwareentwicklung ist die Anforderungsanalyse (Requirements Engineering). Der Begriff Anforderung bezeichnet hierbei Fähigkeiten und Eigenschaften, die ein System haben muss, um eine benötigte Funktion bereitzustellen oder um einen Vertrag, Standard oder eine Spezifikation zu erfüllen.

Abbildung 4 zeigt die Anforderungen im V-Modell und ihre Beziehungen untereinander. Gemäß dem in 2.1 beschriebenen allgemeinen V-Modell steigt linksseitig der Detaillierungsgrad der Anforderungsdokumente mit absteigender Ebene. Gleichzeitig zeigen die vertikalen Beziehungen die Einflussnahme der Produkte auf der rechten Seite auf die Dokumente der linken Seite. Ein wesentlicher

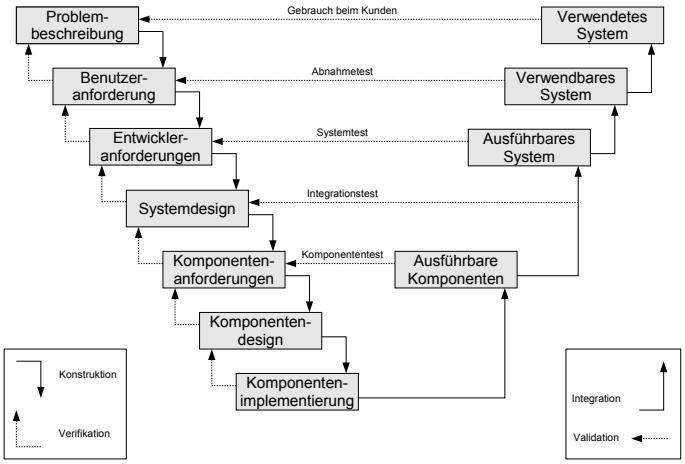

Abbildung 3: Grafische Darstellung des allgemeinen V-Modells nach Boehm [3]

Aspekt im Requirements Management ist die Verfolgbarkeit von Anforderungen entlang des V-Modells, die durch Requirements Management Systeme unterstützt werden soll. Im STEP-X Projekt greift man zum Requirements Management auf das Tool „DOORS 6.0“ [1] der Firma Telelogic zurück. Durch die intuitive Benutzbarkeit ähnlich einer Textverarbeitung ist der Umgang mit DOORS 6.0 von allen am Prozess beteiligten Personen schnell zu erlernen. Mit Hilfe von Links ist es möglich Beziehungen zwischen Anforderungen herzustellen oder externe Quellen einzubinden. Diese Links dienen der Rückverfolgbarkeit der Anforderungen, beispielsweise von den ursprünglichen Benutzeranforderungen über die Systemanforderungen zur Softwarespezifikation.

Abbildung 5 zeigt schematisch die Verlinkung von Anforderungen auf verschiedenen Ebenen, die Pfeile zwischen den einzelnen Anforderungen symbolisieren eine „erfüllt“-Beziehung. Treten Änderungen während der Entwicklung auf, bietet DOORS 6.0 Analysefunktionen, um Bereiche entlang dieser Pfade zu identifizieren, die von den Änderungen betroffen sind. Die Identifizierung betroffener Bereiche durch Links und die Auswirkung von Änderungen entlang des V-Modells zeigt Abbildung 6. Auf der Ebene der Kunden Anforderungen tritt hierbei eine Änderung auf. Durch die Verlinkung der Anforderungen können in DOORS automatisch andere Bereiche identifiziert werden, die von der Änderung betroffen sind, dargestellt durch die roten Verbindungslinien [5; 8].

[1] http://www.telelogic.de/products/doors/index.cfm

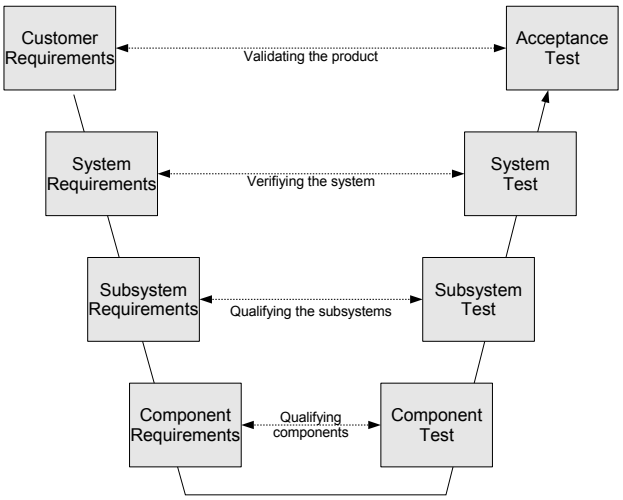

Abbildung 4: Anforderungen im V-Modell [5]

2.3 Artisan RealTime Studio 4.3

In den frühen Entwicklungsphasen des STEP-X Prozesses, von der Analyse bis zum Architekturentwurf, bedient man sich der UML als Modellierungssprache. Während der Analyse nutzt man dabei Anwendungsfall-, Sequenz- und Klassendiagramme. Der funktionale Grobentwurf wird unterstützt durch Klassen- und Zustandsdiagramme und im Architekturentwurf bedient man sich der Verteilungsdiagramme.
Zur Modellierung mittels UML findet im STEP-X Projekt das Tool „RealTime

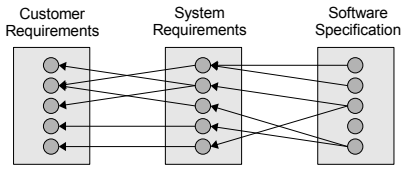

Abbildung 5: Verknüpfung zwischen Anforderungen auf verschiedenen Ebenen [5]

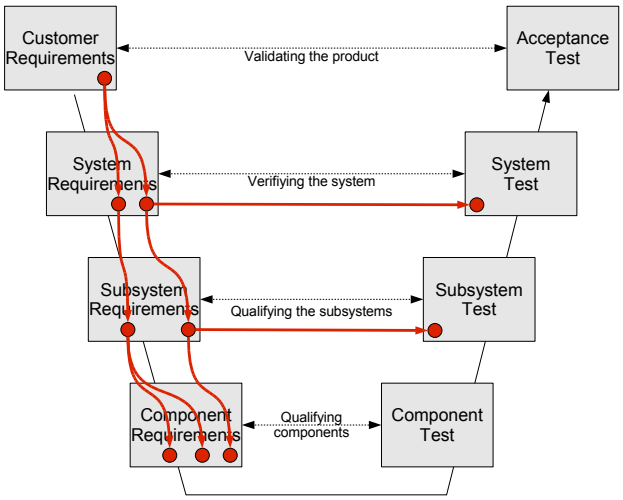

Abbildung 6: Verfolgbarkeit von Anforderungen im V-Modell [5]

Studio“[2] von Artisan Anwendung. Der Schwerpunkt von RealTime Studio liegt in der Modellierung von Echtzeit- und eingebetteten Systeme. Dazu bietet es neben UML-Diagrammen weitere Arten von Diagrammen an, um die Modellierung von Echtzeitsystemen zu verbessern. Mittels „System Architecture“ Diagrammen kann man beispielsweise die Systemarchitektur aus technischer Sicht modellieren. Dazu baut man ein System aus Subsystemen, Schnittstellen, Prozessoren, Bussen etc. auf. Anschließend kann man das Modell der Systemarchitektur mit Modellen der Software verbinden und so eine Beziehung zwischen Hard- und Software herstellen. Weitere Diagramme dienen beispielsweise der Erfassung und Modellierung nicht-funktionaler Bedingungen oder Nebenläufigkeit. Darüber hinaus beinhaltet RealTime Studio Simulationstechniken zur Validierung von Modellen und bietet die Möglichkeit anhand von Modellen automatisch Programmcode in C, C++, Java oder Ada zu erzeugen [10; 9; 8].

2.4 ASCET-SD

Die Firma ETAS stellt mit ASCET-SD [3] ein weiteres Modellierungstool zur Verfügung, das im Gegensatz zu Artisans RealTime Studio nicht UML, sondern eine eigene Modellierungssprache verwendet. Zur Anwendung kommen hier unter an-

[2] http://www.artisansw.com/products/
[3] http://de.etasgroup.com/downloads/ascetsd _br.shtml

derem Blockdiagramme und Zustandsautomaten. Wie auch RealTime Studio bietet ASCET-SD die Möglichkeit, erstellte Modelle zu testen, zum einen in einer reinen Simulation (Offline-Test), zum anderen aber auch in Echtzeit unter realen Bedingungen (Online-Test). Eine komfortable Dokumentationsfunktion erlaubt das Einfügen von textuellen Ergänzungen in Modelle und die Erzeugung von Berichten im HTML-, PS-, ASCII-, oder RTF-Format. Wesentlich für den Einsatz von ASCET-SD im STEP-X Prozess ist aber die Fähigkeit, optimierten C-Code für verschiedene Steuergeräte automatisch generieren zu können [8; 6; 1].

2.5 MATLAB/Simulink

MATLAB[4] ist eine Sprache zur Formulierung und Lösung technisch-wissenschaftlicher Probleme, deren Notation eng an die „übliche“ mathematische Schreibweise angelehnt ist. Die Datenbasis von MATLAB (MATrix LAboratory) bilden Matrizen, d.h. Berechnungen werden in MATLAB anhand von Matrizen ausgeführt. Im Gegensatz zu klassischen Computeralgebrasystemen wird MATLAB vielfach für numerische Berechnungen genutzt. Neben der Fähigkeit zur numerischen Lösung von Problemen bietet MATLAB verschiedene Möglichkeiten zur grafischen Darstellung und Analyse von Daten. Anweisungen können im so genannten „Command Window“ interaktive eingegeben werden oder in Skripten und Funktionen zusammengefasst werden. Zahlreiche vorgefertigte Skripten und Funktionen werden in verschiedenen „Toolboxen“ gebündelt und erweitern bei Bedarf den Funktionsumfang von MATLAB. Simulink dient der Modellierung und Simulation von Systemen und ist nahtlos in die MATLAB-Entwicklungsumgebung integriert. Für die Modellierung benutzt Simulink Blockdiagramme, deren Bestandteile in Bibliotheken zusammengefasst sind. Neben den mitgelieferten Schaltblöcken gibt es verschiedene Erweiterungen zu den Simulink Bibliotheken, durch die man den Funktionsumfang entsprechend erhöhen kann.

3 Der STEP-X Entwicklungsprozess

In diesem Kapitel werden die Phasen des STEP-X Entwicklungsprozesses detailliert beschrieben. Abbildung 7 zeigt die Einteilung des Prozesses in neun Phasen. Gegenstand dieses Kapitels sind die Aktivitäten im mittleren Block des Schaubildes. Die prozessbegleitenden Aktivitäten Test und Diagnose sind Thema des nächsten Kapitels.

[4] http://www.mathworks.de/products/matlab/

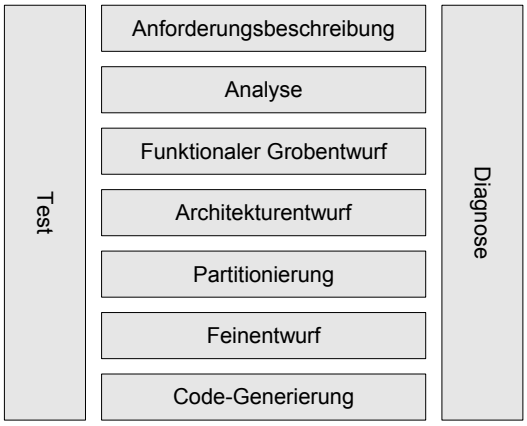

Abbildung 7: Phasen des Entwicklungsprozesses im Projekt STEP-X [7]

Große Beachtung schenkt man im STEP-X Prozess den Anforderungsdokumenten und ihrer Struktur. Besonders die Übergänge zwischen den einzelnen Prozessphasen profitieren von einer guten Anforderungsbeschreibung. Abbildung 8 zeigt die während des Prozesses entstehenden Dokumente in einer am V-Modell orientierten Darstellung. Zu erkennen ist außerdem eine Strukturierung der Dokumente in vier so genannte „Modellierungsebenen“, die eine Betrachtung des Systems aus verschiedenen Sichtweisen ermöglichen.

Die *Benutzer-/Produktebene* betrachtet ein Teilsystem aus dem Blickwinkel des Benutzers und der ihm zur Verfügung gestellten Funktionalität. Die Funktionen werden hier so beschrieben, wie sie der Benutzer bei der Verwendung des (Teil-)Systems wahrnimmt. Die Dokumente der Produktebene entstehen in der Phase der Anforderungsbeschreibung und werden in der Analyse in die Systemebene überführt. Zur Modellierung greift man bei der Anforderungsbeschreibung und der Analyse auf UML-Anwendungsfalldiagramme, Sequenzdiagramme und Klassendiagramme zurück.

Die *Systemebene* rückt die technische Umsetzung in den Mittelpunkt der Betrachtung und erlaubt die Beschreibung von Funktionen, Schnittstellen und Vorgaben, wie sie sich bei der Integration des Teilsystems ins Gesamtsystem darstellen. Nach dem Übergang von der Benutzer- auf die Produktebene in der Analysephase resultieren die Dokumente dieser Ebene aus dem funktionalen Grobentwurf und dem Architekturentwurf. Die Modellierung während dieser Prozessphasen geschieht durch Klassen-, Zustands- und Verteilungsdiagramme

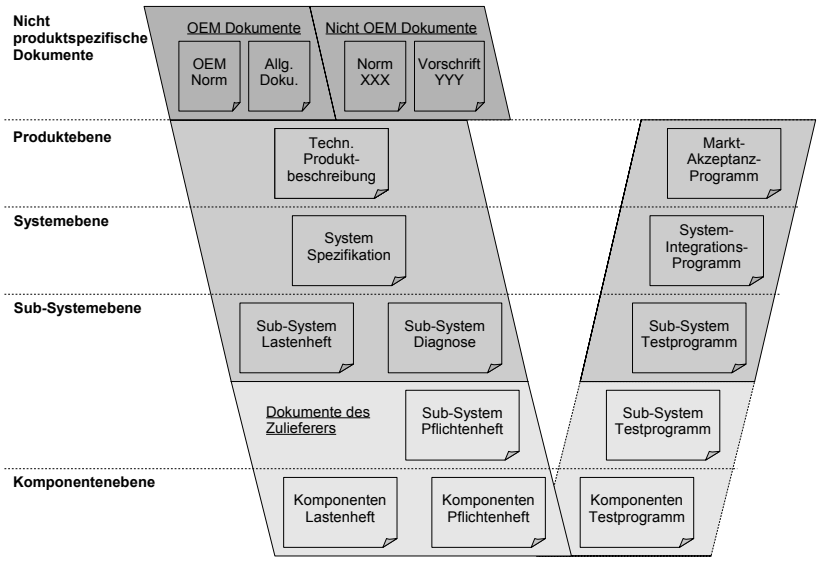

Abbildung 8: Ebenen der Anforderungserfassung für Automotive Anwendungen [7]

Auf der *Subsystem-/Komponentenebene* werden Anforderungen erfasst, die sich aus der Interaktion verschiedener Teilsysteme ergeben. Die Erkenntnisse hierzu gewinnt man in den Phasen Partitionierung und Feinentwurf. In der Feinentwurfsphase wird die UML-Notation abgelöst von Blockdiagrammen und Zustandsautomaten für den Gebrauch in ASCET SD.

Neben diesen vier Ebenen wird in [7] noch eine weitere definiert, die so genannte „Verifikationsebene“. Diese ist aber kaum als eigenständige Ebene zu sehen, vielmehr ist sie ein Bestandteil der anderen Ebenen. Jede Ebene enthält einen Verifikationsanteil, in dem man Angaben zu Verifikation und Test festhält.

Die *„Ebene der mitgeltenden Unterlagen“* dient der Erfassung von gesetzlichen Bestimmungen oder Vorgaben von Seiten des OEM. Sie beeinflusst alle oben genannten Ebenen, geht aber nicht aus dem Prozess selbst hervor. Daher steht sie in Abbildung 8 im linken Ast des V-Modells oberhalb des eigentlichen „V“. Eine solche Einflussnahme geschieht beispielsweise durch eine Norm, die dieser Ebene entnommen wird. Diese beeinflusst einerseits die Anforderungen auf den nach ge-

lagerten Ebenen, andererseits aber auch deren Verifikationsanteile, da das System beim Testen auch auf die Einhaltung dieser Norm geprüft werden muss.

Über diese Strukturierung der Anforderungsdokumente in Modellierungsebenen hinaus schlägt der STEP-X Prozess eine weitere Unterteilung der einzelnen Ebenen vor. Dabei werden verschiedene Arten von Anforderungen unterschieden. Die *Funktionalen Anforderungen* umfassen dabei Funktionen und Unterfunktionen des Systems sowie Vor- und Nachbedingungen für das Aktivieren und Sperren von Funktionen, Ausnahmen und Fehlerfälle. Unter dem Begriff *Diagnoseanforderungen* fasst man alle Vorgaben zusammen, die später für die Diagnose relevant sind. Als *nicht funktionale Anforderungen* bezeichnet man alle Anforderungen an ein (Sub-)System, die nicht mit der eigentlichen Funktion des Systems zusammenhängen und nicht aus den mitgeltenden Unterlagen hervorgehen. Darunter fallen beispielsweise Vorgaben an die Zuverlässigkeit und Performanz eines (Sub-)Systems. Über die *Schnittstellen* beschreibt man die Kommunikation eines Systems mit seiner Umgebung durch Aktoren, Sensoren und Bedienelemente. *Parameterlisten* dienen der Erfassung von (Zahlen-)Werten, die für verschiedene Anforderungen relevant sind und tragen damit zur Konsistenz der Anforderungsdokumente bei. Im *Glossar* werden schließlich alle relevanten Begriffe für die zu entwickelnde Anwendung festgehalten.

Das Konzept der Modellierungsebenen wird in DOORS über Module umgesetzt, dazu bildet jede Ebene ein solches Modul. Die Zusammengehörigkeit von Anforderungen über Modellierungsebenen (Module) hinweg wird in DOORS über Links ausgedrückt. Dabei verweist eine abgeleitet Anforderung per Konvention auf die ursprüngliche Anforderung. Hierdurch erreicht man eine optimale Verfolgbarkeit von Anforderungen, wie sie bereits in 2.2 beschrieben wurde. Betrachten wir erneut die Beeinflussung von Anforderungen durch eine Norm, so bestehen in der Umsetzung der Anforderungsdokumente in DOORS verschiedene Verbindungen, die diese Einflussnahme wiedergeben. Die Anforderung, die aus der Norm hervor geht, und auch die Anforderungen an Verifikation und Test, die im Verifikationsanteil der jeweiligen Ebene festgehalten sind, verweisen auf die Norm selbst. Gleichzeitig werden die Resultate eines später durchgeführten Tests wieder mit der Anforderung und dem Verifikationsanteil verlinkt, der für die Tests relevant war. Letztlich besteht durch eine Kette von Links eine Verbindung von der eigentlichen Norm bis zu den Ergebnissen der Tests zu ihrer Einhaltung.

Um die Übersicht, die Konsistenz und auch die Vollständigkeit der Anforderungsdokumente zu erhöhen, werden im STEP-X Prozess generische Lastenhefte erstellt. Diese beinhalten für eine Domäne allgemein gültige Anforderungen und Attribute sowie eine vorgefertigte Gliederung zur Beschreibung der Funktion. Aus diesen „Schablonen“ werden die spezifischen Lastenhefte abgeleitet. Nach

der eigentlichen Erfassung der Anforderungen folgen im STEP-X Prozess noch eine semantische und linguistische Analyse der Anforderungen. Die semantische Analyse soll unvollständige, widersprüchliche, missverständliche oder fehlende Anforderungen aufdecken, die linguistische Analyse bietet Regeln zur optimalen Formulierung von Anforderungen.

3.1 Anforderungsbeschreibung

Die erste Phase innerhalb des STEP-X Prozesses ist die Anforderungsbeschreibung. Diese ist nicht zu verwechseln mit den allgemeinen Ausführungen zur Anforderungsbeschreibung innerhalb des gesamten Prozesses. Die Phase der Anforderungsbeschreibung wird unterteilt in die Spezifikation der Funktionalen Anforderungen und die der nicht-funktionalen Anforderungen. In den Funktionalen Anforderungen wird das gewünschte Systemverhalten aus der Sicht des Anwenders beschrieben. Stellenweise werden Teilaspekte schon mittels UML modelliert. Die nicht-funktionalen Anforderungen beschreiben Eigenschaften wie z.B. Performanz oder Standartkonformität eines Systems. Zur Verwaltung der Anforderungsbeschreibung, sowie der UML-Diagramme kommt DOORS zum Einsatz, so dass gegebenenfalls Links erstellt werden können zwischen nicht-funktionalen Anforderungen und den Dokumenten der Ebene der mitgeltenden Unterlagen, aus denen sie hervorgegangen sind. Ebenso werden Verknüpfungen zwischen Anforderungen und evtl. damit in Bezug stehenden UML-Diagrammen erzeugt, so dass am Ende dieser Prozessphase eine Anforderungsbeschreibung vorliegt, die neben der textuellen Beschreibung auch UML-Modelle enthält und dadurch ein besseres Verständnis des Systems ermöglichen soll. Die in dieser Phase erzeugte Anforderungsbeschreibung bildet im V-Modell der Anforderungsdokumente die Benutzerebene.

3.2 Analyse

Ziel der Analyse ist es, die überwiegend textuellen Beschreibungen aus der Anforderungsbeschreibung um grafische Beschreibungen in Form von UML-Diagrammen zu erweitern. In dieser Phase kommt es also zu einer ersten Formalisierung der zuvor frei formulierten Anforderungen. Zu diesem Zweck verwendet man in der Analysephase unter anderem *Anwendungsfalldiagramme*. Diese Diagramme dienen allgemein der Modellierung des erwarteten Systemverhaltens und zeigen Akteure und Anwendungsfälle mit ihren Abhängigkeiten und Beziehungen. Die benötigten Informationen zur Erstellung der *Anwendungsfalldiagramme* liegen als textuelle Beschreibung bereits als Ergebnis der Anforderungsbe-

schreibung vor. Für die Beschreibung von Anwendungsfällen, bei denen zeitliche Aspekte wie die Abfolge von Ereignissen eine Rolle spielen, nutzt man zusätzlich *Sequenzdiagramme*. Mittels *Sequenzdiagrammen* modelliert man allgemein eine Interaktion zwischen Objekten, die das dynamische Verhalten eines Systems verdeutlicht. Zur Darstellung der Interaktion müssen die Nachrichten spezifiziert werden, die die Objekte untereinander austauschen. In dieser Phase werden demnach auch schon Objekte, ihre Methoden und ihre Kommunikation untereinander festgelegt und in *Klassendiagrammen* festgehalten.

3.3 Funktionaler Grobentwurf

In der Phase des *funktionalen Grobentwurfs* spezifiziert man auf Grundlage der Anforderungsdokumente und der Ergebnisse der Analyse die Funktionen des Systems und modelliert sie in Klassendiagrammen. Die einzelnen Funktionen bilden dabei modulare logische Einheiten und komplexe Funktionen werden in Subfunktionen zerlegt.

Anschließend wird aus den einzelnen Funktionen und Subfunktionen das Gesamtsystem zusammengesetzt, wodurch man eine modulare, hierarchische Struktur des Systems erzeugt. Für die Komposition ist es unabdingbar, dass jede Funktion Schnittstellen kapselt, um eine Kommunikation zwischen den Funktionen zu ermöglichen. Eine direkte Kommunikation zwischen Funktionen ist hierbei allerdings nicht vorgesehen. Jede Funktion verfügt über einen Koordinator in Form einer Klasse, über den die gesamte Kommunikation zwischen anderen Funktionen und den zugehörigen Subfunktionen abgewickelt wird. Dazu nimmt ein Koordinator eingehende Signale entgegen und leitet sie an die zugehörigen Funktionsklassen weiter. Durch diese Form der Kommunikation erzeugt man eine flexible Systemstruktur, in der (Sub-) Funktionen leicht entfernt oder geändert werden können, da dies keinen Einfluss auf die verbleibenden Funktionen hat. In Abbildung 9 erkennt man beispielsweise den Koordinator für die Komfortfunktionen des Fensterhebers. Alle Signale des Systems gehen beim Koordinator ein und werden dort an die zuständige Funktion (dargestellt als Klasse) weitergeleitet. Diese Ausführungen lassen bereits erkennen, dass das gezeigte Diagramm kein Klassendiagramm im eigentlichen Sinn darstellt, sondern lediglich die Notation eines Klassendiagramms benutzt wurde. Dies wird zunächst daran deutlich, dass es sich bei den beiden Elementen „Comfort opening“ und „Comfort closing“ um Funktionen und weniger um Klassen handelt. Diese Funktionen verfügen über Schnittstellen, die hier in „Lollipop“-Notation festgehalten sind. Über diese Schnittstellen kommunizieren die Funktionen mit ihrem Koordinator, so dass auch die Verbindungen zwischen dem Koordinator und den Schnittstellen der Funktionen keine

Beziehungen zwischen Klassen ausdrücken, sondern den Signalfluss im System modellieren.

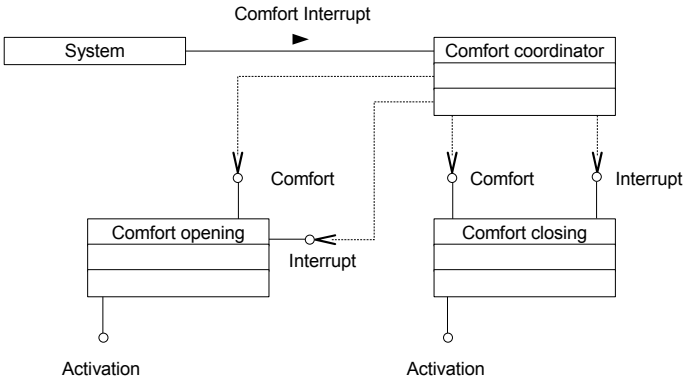

Abbildung 9: Klassendiagramm zur Fensterheber-Komfort-Funktion [8]

Gleichzeitig werden Varianten von Funktionen erzeugt, falls mehrere ähnliche Funktionen benötigt werden. Als Beispiel nennt [8] eine Variantenbildung für elektrische Fensterheber. Dabei wird eine Variante für die Fahrertür, eine für die Beifahrertür und eine für die hinteren Türen erzeugt, die in ihrer Grundfunktionalität alle gleich sind. Der Fahrer hat aber als einziger die Möglichkeit alle anderen Fenster zu steuern. Der Beifahrer hat lediglich die Möglichkeit sein eigenes Fenster zu bedienen und in die Variante für die hinteren Türen muss eine Kindersicherung integriert werden. In Abbildung 10 sieht man die Darstellung der Variantenbeziehung in Klassendiagrammen, wie sie im STEP-X Prozess festgelegt ist. Treten bei der Bildung der Varianten Aspekte zu Tage, die in der Analysephase noch nicht ersichtlich waren, werden entsprechende Änderungen an den Modellen vorgenommen. Auch die Schnittstellen werden angepasst und zunächst in Form von parametrisierten Klassen modelliert. Hierbei werden sowohl die Signale selbst, als auch deren Quellen parametrisiert angegeben und erst bei der Instanzierung konkretisiert.

In einem weiteren Schritt innerhalb des funktionalen Grobentwurfs erfolgt eine Modellierung des Systemverhaltens durch *Zustandsdiagramme*, wodurch erste Tests und Simulationen möglich werden. *Zustandsdiagramme* zeigen verschiedene Zustände, die ein Objekt einnehmen kann, sowie Bedingungen und Aktionen, die zu einer Zustandsänderung führen. Um eine korrekte Modellierung mittels dieser *Zustandsdiagramme* zu gewährleisten, werden im STEP-X Projekt verschiedene Modellierungsrichtlinien festgelegt. Dazu zählt etwa die Vorgabe, dass Transitionen immer einen Start- und einen Endzustand haben müssen.
Über Designregeln werden die Grundlagen geschaffen für eine automatische

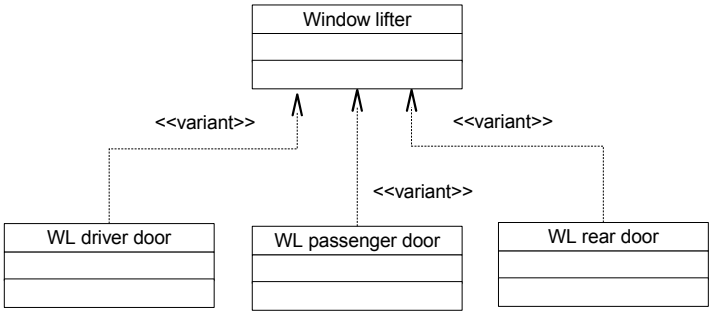

Abbildung 10: Varianten des Fensterhebers [8]

Überführung der UML-Zustandsdiagramme in die Notation von ASCET-SD, wie sie beim Übergang in die Implementierungsphase notwendig wird. Gleichzeitig wird versucht die Kompatibilität und Übersichtlichkeit der *Zustandsdiagramme* zu erhöhen. Zu den Designregeln zählt etwa die Konvention, dass Initialzustände im Zustandsdiagramm möglichst links oben stehen sollten. Andere Designregeln besagen etwa, dass Transitionen gradlinig und überschneidungsfrei verlaufen sollen usw. Während die Einhaltung der erwähnten Modellierungsrichtlinien von den meisten UML-Tools automatisch überprüft werden kann, ist dies bei den Designregeln nicht der Fall. Aus diesem Grund wurde im STEP-X Projekt ein „Regel Checker“ entwickelt, der Zustandsdiagramme anhand vorgegebener Designregeln überprüft.

3.4 Architekturentwurf

Für den Architekturentwurf bedient man sich der UML *Verteilungsdiagramme*. Dieser Diagrammtyp zeigt die Verteilung eines Systems auf die vorhandene Hardware und die Kommunikationsbeziehungen zwischen verschiedenen Hardwarekomponenten.
Im Bereich der Automobilelektronik bezeichnet man als Architektur meist die Hardware-Architektur, beispielsweise die Hardwaretopologie oder das Layout von Kabelbäumen. Dementsprechend steht in der Phase des Architekturentwurfes nicht die Softwarearchitektur im Mittelpunkt, wie man leicht vermuten könnte. Vielmehr bezieht man an dieser Stelle erstmals die Hardware in die Betrachtung ein. Um später die Verteilung der Softwarekomponenten auf Steuergeräte realisieren zu können, müssen zunächst die vorhandenen Steuergeräte, sowie ihre Verbindung durch Kommunikationsbusse ermittelt werden. Gleichzeitig fließen die Informationen über die Hardware in die Instanzierung der zuvor erstellten Klassen ein. Beispielsweise hängt die Zahl der Instanzen einer Klasse unter anderem davon

ab, ob sie verteilt auf mehreren Steuergeräten zum Einsatz kommt, oder nur auf einem Steuergerät genutzt wird. Anschließend können die im funktionalen Grobentwurf identifizierten Funktionen zu Softwarekomponenten zusammengefasst und mit einer Schnittstelle versehen werden. Beispielsweise sieht man in Abbildung 11 die beiden Funktionen „Comfort opening“ und „Comfort closing“ die wir bereits im Funktionalen Grobentwurf kennengelernt haben, diesmal jedoch zusammengefasst als Softwarekomponente und verteilt auf ein Steuergerät. Im Verteilungsdiagramm werden die einzelnen Funktionen innerhalb der Komponenten als Objekte dargestellt. Die Schnittstellen der Softwarekomponenten werden in der „lollipop“ Notation festgehalten.

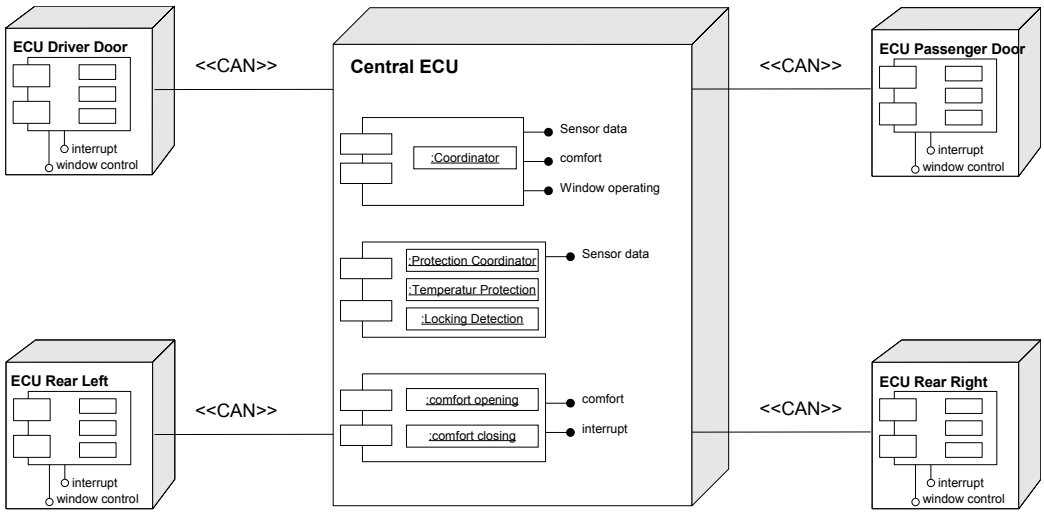

Abbildung 11: Verteilung von Komponenten auf logische Steuergeräte

Abschließend werden die Komponenten auf die logischen Steuergeräte verteilt. Wie bereits beschrieben zeigt Abbildung 11 beispielhaft ein Verteilungsdiagramm. Zu sehen sind fünf Steuergeräte sowie ihre Verbindung untereinander. Die Softwarekomponenten der einzelnen Steuergeräte entnimmt man einer detaillierten Darstellung der einzelnen Steuergeräte. In Abbildung 11 ist eine solche Detailansicht für das Steuergerät „Central ECU“ zu sehen.

3.5 Partitionierung

Für die Verteilung des Softwaresystems auf Hardwarekomponenten, werden in der Partitionierung zunächst die konkreten funktionalen Anforderungen an die Software ermittelt. Diese variieren je nach Typ und Ausstattungsvariante des Fahrzeugs, in dem sie eingesetzt werden sollen. Stehen die funktionalen Anforderun-

gen fest, so erfolgt die Zusammenstellung eines entsprechenden Softwaresystems aus den zuvor erstellten Softwarekomponenten. Dabei ist wiederum die strukturierte Erfassung und Verwaltung der Anforderungen, Dokumente und Modelle in DOORS hilfreich. Durch Rückverfolgung von Links kann eine Anforderung bis auf die Softwarekomponente/n auf der Subsystem und Komponentenebene verfolgt werden, durch die sie erfüllt wird. Die Zusammenstellung eines Softwaresystems durch Bestimmung seiner funktionalen Anforderungen wird dadurch stark vereinfacht.

Im Anschluss an die Zusammenstellung der Software ermittelt man die Hardwaretopologie, definiert Aktorik- und Sensorikschnittstellen und die zugeordneten Bussysteme, um die konkrete Verteilung der Softwarekomponenten auf die Hardware im Fahrzeug festlegen zu können. Im so genannten „Mapping“ vollzieht man die Abbildung des Softwaresystems auf die zugehörige Hardware. Durch die hier vorgenommene Verteilung der Softwarekomponenten auf Steuergeräte, wird gleichzeitig auch die Buskommunikation der Geräte untereinander festgelegt, da die Kommunikationsbeziehungen auf Softwareebene bereits feststehen. Nach der Bestimmung der Struktur der Buskommunikation, werden den einzelnen Bussignalen im *Kommunikationsmapping* Botschaften zugewiesen. Gleichzeitig bestimmt man hier Priorität, Sendeart und Zykluszeit der einzelnen Botschaften.

3.6 Feinentwurf

In der Phase des Feinentwurfs kommt es zu einem Notationswechsel. Die bisher verwendete UML-Notation wird abgelöst von Blockdiagrammen und Zustandsautomaten, wie sie in ASCET-SD Verwendung finden. Dieser Schritt wird hauptsächlich dadurch begründet, dass zwar viele UML-Tools eine automatische Codegenerierung aus UML-Modellen heraus erlauben, der erzeugte Code aber nicht für Mikrocontroller optimiert ist und in der Regel nur Hüllencode ist. ASCET-SD bietet diese Möglichkeit der Generierung optimierten Codes, weshalb in der Implementierungsphase auf dieses Werkzeug zurückgegriffen wird. Gleichzeitig will man aber nicht auf die umfangreichen Möglichkeiten der UML-Modellierung verzichten. Die daher notwendige Überführung der UML-Diagramme in die Notation von ASCET-SD wird als eines der Hauptprobleme bei der Entwicklung des Prozesses beschrieben. Die in Kapitel 3.3 angesprochenen Designregeln und der Regelchecker zur automatischen Überprüfung dieser Regeln bilden die Grundlage für eine automatische Modelltransformation. Dabei werden die Designregeln für UML-Zustandsdiagramme so gewählt, dass sie sich für eine Transformation in die ASCET-SD Notation anbieten. In weiteren Schritten soll dann durch einen

„Model Transformator“ einen „Optimizer“ und einen „Verificator“ die Modelltransformationen automatisiert werden. Die Arbeiten in diesem Bereich sind noch nicht abgeschlossen.
Ziel der Modelltransformation ist es, die Verteilungsdiagramme aus dem Architekturentwurf und die Zustandsdiagramme aus dem Grobentwurf in ASCET-SD Diagramme zu überführen. Die einzelnen Softwarekomponenten werden dabei durch Blöcke in ASCET-SD Blockdiagrammen repräsentiert. Eine Beschreibung des Verhaltens der einzelnen Funktionen erfolgt durch ASCET-SD Zustandsautomaten.

3.7 Code-Generierung

Im Anschluss an den Mapping-Prozess erfolgt die automatische Code-Generierung. Den Hüllencode der Softwarekomponenten generiert man dabei aus den mit Real-Time Studio erzeugten UML-Modellen und ergänzt diesen Hüllencode durch den Code der einzelnen Funktionen, die aus den ASCET-SD Modellen heraus erzeugt werden. Im Anschluss erfolgt die Zusammenstellung des Systems aus den Codefragmenten in einer Integrationsphase. Nach der Übersetzung der Codefragmente stehen diese in einer Restbussimulation, in der Nachrichten von noch nicht real im Netz vorhandenen Steuergeräten simuliert werden, für Test zur Verfügung.

4 Prozessbegleitende Aktivitäten

4.1 Test und Co-Simulation

Sind in klassischen Entwicklungsprozessen erst relativ spät im Prozess Tests möglich, liefert ein modellbasierter Prozess wie der des STEP-X Projektes schon früh erste verifizierbare Modelle, die zu Testzwecken genutzt werden können. Im STEP-X Prozess ergibt sich eine besonders starke Verflechtung von Entwicklung und Test schon in der Phase der Anforderungsbeschreibung. In Kapitel 3.1 wurde bereits auf die verschiedenen Modellierungsebenen zur Strukturierung der Anforderungsdokumente eingegangen und die Verifikationsebene vorgestellt. Für die Testphase können einzelne Testfälle meist schon aus der Benutzer- oder der Systemebene abgeleitet werden und bilden mit den Anforderungen der Verifikationsebene die Basis für die Testfallspezifikation. Aus den Parameterlisten der verschiedenen Ebenen können gegebenenfalls notwendige Daten zur Vervollständigung der Testfälle ermittelt werden. Durch die enge Verflechtung der Testspezifikation mit der Anforderungsbeschreibung integriert man im STEP-X Projekt

das Testmanagement in das Requirements Management. Zu diesem Zweck werden Testziele und Testkriterien ebenfalls in DOORS-Modulen erfasst und im Anschluss an die Testdurchführung und Auswertung um die Testdokumentation ergänzt. Durch diese Integration wird zum einen eine Verknüpfung von Anforderungen und Testergebnissen ermöglicht, zum anderen bietet die Nutzung einer einzigen gemeinsamen Datenbank zur Erfassung der Anforderungen und der Tests Vorteile im Hinblick auf Konsistenz und Pflege der Daten.
Die frühe Modellbildung im STEP-X Prozess stellt bereits gegen Ende des Grobentwurf erste ausführbare Modelle zur Verfügung, die grob das Systemverhalten wiedergeben. Bereits hier kann durch erste Abnahmetests überprüft werden, ob alle Anforderungen durch das System erfüllt werden. Im Gegensatz zu klassischen Entwicklungsprozessen, in denen mit den Modultests begonnen wird, sind nun zuvor schon Abnahmetests auf virtuellen Prototypen durchführbar. Diese frühe Gelegenheit Fehler zu erkennen senkt die Kosten der Fehlerbehebung beträchtlich.
Ein wesentlicher Bestandteil der Testmethodik im STEP-X Projekt ist die Toolkopplung. Durch die ExITE Toolbox der Firma Extessy AG werden die eingesetzten CASE-Tools zu einem Toolverbund gekoppelt. Innerhalb dieses Toolverbundes werden die verschiedenen Werkzeuge synchronisiert und es können somit verschiedene Modelle zu einer Simulation kombiniert werden. Diese so genannte Co-Simulation kommt beispielsweise in der Phase des Feinentwurfs zum Einsatz, in der ASCET-SD und MATLAB/Simulink gekoppelt werden, indem eine in ASCET modellierte Steuerung mit einer in MATLAB/Simulink modellierten Umgebung verknüpft wird.
Erst durch die Toolkopplung und die Möglichkeit über Toolgrenzen hinweg zu simulieren werden frühe Tests möglich und eventuelle Designfehler können früh erkannt und behoben werden.

4.2 Diagnose

Die hohe Anzahl möglicher Kombinationen von Steuergeräten und Software in einem Fahrzeug, unterschiedlich z.B. je nach Typ und Ausstattung, steigern die Bedeutung von Diagnosefunktionen. Für die Wartung eines Fahrzeugs ist die Diagnose unerlässlich, ermöglicht sie doch eine Eingrenzung von Fehlerkandidaten. Gleichzeitig erfordert die hohe Komplexität der Systeme ein strukturiertes Vorgehen bei der Erstellung der Diagnosefunktionen. Später unterstützen solche Diagnosefunktionen beispielsweise Werkstätten bei der Reparatur und Wartung von Fahrzeugen, indem sie durch Fehlerlokalisation auszutauschende Bauteile aufzeigen, oder sie liefern dem Fahrzeughersteller durch eine detaillierte Fehlererkennung wichtige Anhaltspunkte zur Verbesserung künftiger Fahrzeuge.

Allgemein ist für eine automatische Diagnose Wissen über das zugehörige System notwendig. In der Literatur wird unterschieden zwischen *beschreibendem Diagnosewissen*, das Ursache-Wirkung-Beziehung aus Beobachtungen am System beschreibt und *modellbasiertem Diagnosewissen*. Dieses *modellbasierte Diagnosewissen* beschreibt die Zusammenhänge zwischen Systemgrößen und Fehlern auf Bauteilebene. Für die Diagnose der im STEP-X Prozess entwickelten Systeme bedient man sich des *modellbasierten Diagnosewissens* und definiert Arbeitsschritte um dieses Wissen aus dem Entwicklungsprozess zu entnehmen. Zur Generierung dieses Wissens ist ein Modell der zu diagnostizierenden Anwendung notwendig. Innerhalb der Modelle bilden Systemfunktion Komponenten und das Modell selbst zeigt die logischen und physikalischen Abhängigkeiten dieser Komponenten untereinander.
Im Entwicklungsprozess wird die Diagnose erstmals in der Phase des Grobentwurfes integriert. Durch die Zusammenstellung der Systemstruktur aus den einzelnen Funktionen und Subfunktionen des Systems, können an dieser Stelle Informationen über die logischen Abhängigkeiten der Komponenten extrahiert werden, die dann für eine systemweite modellbasierte Diagnose genutzt werden können.
Auch in die Entwicklungsphasen *Architekturentwurf* und *Partitionierung* ist die Diagnose eingebunden, da hier benötigte Informationen über die Kommunikation im System extrahiert werden könne. Diese Kommunikationsinformationen werden beispielsweise von einer systemweiten Diagnosefunktion genutzt, wenn durch das Ausbleiben einer Nachricht eines Steuergerätes auf dessen Ausfall geschlossen wird. Diese Fehlererkennung erleichtert unter anderem die Reparatur- und Wartungsarbeiten am Fahrzeug. Da nämlich das Ausbleiben der Nachrichten des ausgefallenen Steuergerätes von allen Kommunikationspartnern dieses Gerätes als Fehler erfasst wird, erschwert dies die Erkennung des eigentlichen Fehlers.

5 Zusammenfassung

Der in diesem Artikel vorgestellte Entwicklungsprozess für die Softwareentwicklung im Automobilbereich, der im Rahmen des STEP-X Projektes vorgeschlagen wurde, ist ein durchgängiger modellbasierter Entwicklungsprozess von der Erfassung der Anforderungen bis zur automatischen Codegenerierung. Großes Augenmerk wird auf eine ausführliche und gut strukturierte Erfassung der Anforderungen gelegt, da diese als Basis für den gesamten folgenden Entwicklungsprozess und das prozessbegleitende Testen genutzt wird. Die textuellen Anforderungen aus der ersten Prozessphase werden in der Analyse um grafische Beschreibungen in UML-Notation ergänzt. Die Funktionen des Systems werden im funktionalen Grobentwurf festgelegt und im Architekturentwurf mit der Hardware im Auto in

Verbindung gebracht. Anschließend erfolgt das Mapping der Softwarekomponenten auf Steuergeräte und die automatische Codegenerierung.
Der Testprozess wird früh in die Entwicklung integriert, indem schon bei der Anforderungsanalyse relevante Aspekte für Tests ermittelt werden und die Tests zusammen mit den Anforderungen in einer gemeinsamen Datenbank verwaltet werden. Durch Toolkopplung und die frühe Verfügbarkeit von Modellen wird ein sehr frühes Testen im Prozess möglich. Auch die Diagnose ist früh in die Entwicklung integriert.

Literatur

[1] *http://de.etasgroup.com/products/ascet/*.

[2] *www.automobil-produktion.de*.

[3] *www.software-kompetenz.de*.

[4] *www.wikipedia.de*.

[5] DICK, JEREMY: *What is Requirements Management*. Technischer Bericht, Telelogic, 2004.

[6] HORSTMANN, MARC: *Verflechtung von Test und Entwurf für eine verlässliche Entwicklung eingebetteter Systeme im Automobilbereich*, 2005.

[7] M.MUTZ, M.HARMS, M.HORSTMANN, DR. M. HUHN, G. BIKKER, C. KRÖMKE, K. LANGE, E. SCHNIEDER U. GOLTZ und J.-U. VARCHMIN: *Ein durchgehender modellbasierter Entwicklungsprozess für elektronische Systeme im Automobil*. VDI Berichte, 1789, 2003.

[8] MUTZ, MARTIN, MICHAELA HUHN, URSULA GOLTZ und CARSTEN KRÖMKE: *Model Based System Development in Automotive*. 2002.

[9] SCHÄTZ, BERNHARD, TOBIAS HAIN, FRANK HOUDEK, WOLFGANG PRENNINGER, MARTIN RAPPL, JAN ROMBERG, OSCAR SLOTOSCH, MARTIN STRECKER und ALEXANDER WISSPEINTNER: *CASE Tools for Embedded Systems*. Technischer Bericht, Technische Universität München, Institut für Informatik, 2003.

[10] SCHWERDTFEGER, BJÖRN: *Analyse eines Realtime CASE Tools: Artisan Realtime Studio*.