Entwicklung und Einsatz innovativer HMI-Software zur Diagnose elektronischer Steuergeräte in der Automobilindustrie

Inaugural-Dissertation
zur Erlangung des Grades eines
Doktors der Wirtschaftswissenschaften
an der
Wirtschaftswissenschaftlichen Fakultät der
Universität Passau

vorgelegt von
Diplom Wirtschafts-Informatiker
Florian Schwarz

Hauptberichterstatter:	Prof. Dr. Franz Lehner, Universität Passau
Mitberichterstatter:	Prof. Dr. Peter Kleinschmidt, Universität Passau
Eingereicht am:	30.04.2008

Danksagung

An dieser Stelle möchte ich mich bei Prof. Dr. Franz Lehner und Prof. Dr. Peter Kleinschmidt herzlich für ihre tatkräftige Unterstützung bedanken. Sie haben mit zahlreichen Anregungen und konstruktiver Kritik zum Gelingen dieser Arbeit beigetragen.

Meinen Kollegen, Vorgesetzten und Vertretern des Kooperationspartners bin ich für das angenehme Arbeitsklima, ihre Unterstützung und ihr Entgegenkommen während der Dauer meiner Dissertationstätigkeit auf die gleiche Weise zu Dank verpflichtet.

Besonderen persönlichen Dank möchte ich meinen Eltern aussprechen. Sie haben mich zeitlebens unterstützt und mir geholfen, der Menschen zu werden, der ich heute bin. Ihnen möchte ich diese Arbeit widmen.

Nicht zuletzt gilt mein Dank meinen beiden besten Freunden Matthias und Johannes, die mich durch diese anstrengende Lebensphase begleitet haben, sowie meiner Freundin Cornelia für ihr liebevolles Verständnis.

Florian Schwarz Passau, den 30.04.2008

Inhaltsverzeichnis

Abkürzungsverzeichnis

(siehe Anhang A, Glossar und Abkürzungsverzeichnis)

Abbildungsverzeichnis

Tabellenverzeichnis

1 Einleitung

1.1 Motivation

Die deutschen Automobilhersteller erfreuen sich in den letzten Jahren an "Rekordumsätzen"[1] und profitieren vom weltweiten Konjunkturanstieg in der Fahrzeugindustrie. Vor allem Märkte im asiatischen und osteuropäischen Raum bescheren den Fahrzeugherstellern steigende Exportzahlen[2]. Trotz des gesteigerten Absatzes stehen die Hersteller künftig einer massiven Kosten- und Qualitätsproblematik gegenüber.

So plant der Fahrzeugbauer VW, der mit fast 9 % der gesamten Weltproduktion mit Abstand die meisten Fahrzeuge aller deutschen Hersteller produziert[3], seine Produktionskosten bis 2008 um insgesamt 1,3 Milliarden Euro zu senken[4], wie Reinhard Jung, VW-Markenvorstand für Produktion und Logistik, im September 2005 ankündigte. Die Steigerung der Produktivität sei, so Jung, ein wichtiger Schritt um weiterhin "wettbewerbsfähig" zu bleiben. Er begründete diese drastische Maßnahme damit, dass "... aufgrund der zu hohen Produktionskosten in Deutschland zuletzt Verluste eingefahren …" worden seien.

Neben diesen einschneidenden Kostensenkungsvorhaben steht die Automobilindustrie noch einem weiteren, bisher in der Geschichte erstmaligen Problem bei der Qualitätssicherung gegenüber: den Rückholaktionen. So musste beispielsweise DaimlerChrysler im Jahr 2005 die größte Aktion dieser Art der gesamten Automobilindustrie starten. 1,3 Millionen davon betroffene Fahrzeuge mussten aufgrund qualitativer Mängel in der Elektronik der Lichtmaschine und Bremsanlage zurückgeholt werden[5]. Auch andere Marken wie Porsche, BMW und VW haben mit diesem Problem zu kämpfen[6]. BMW musste schon ein Jahr vorher 2004, rund 75.000 Fahrzeuge aus der 5er und 7er Reihe wegen Fehler in der Komfortelektronik zurückrufen[7]. Des Weiteren erfolgte im Januar 2005 durch die US-Verkehrssicherheitsbehörde der Rückruf von

[1] VDA 2005, S. 202
[2] Vgl. Automobil Produktion, November 2005 (Online-Quelle)
[3] Vgl. VVDA 2004, S. 44-45
[4] Vgl. FAZ 2005, S. 16
[5] Vgl. NZ 2005, Nr. 332006
[6] Vgl. Automobil Produktion, Juni 2005, S. 24
[7] Vgl. NZ 2004, Nr. 318822

359.000 Fahrzeugen des Herstellers Ford wegen Probleme mit der Schließelektronik an der Heckklappe[8].

Als häufigste Fehlerquelle an modernen Fahrzeugen zeigt sich also derzeit die Elektronik. Sie ist aber, so auch, Burkhard Göschel, BMW-Vorstand für Einkauf und Entwicklung, ein wesentliches Merkmal des Produkts und leistet einen wichtigen Beitrag zur Differenzierung gegenüber der Konkurrenz. Die innovative Elektronik im Fahrzeug trägt für ihn nicht nur zur "Verbesserung der aktiven und passiven Sicherheit bei, sondern auch zu wesentlich mehr Komfort"[9].

Eine Gemeinschaftsstudie des Center of Automotive Research (CAR), PriceWaterhouseCoopers, des VDA und der Stadt Leipzig zeigt die rasante Zunahme des Wertschöpfungsanteils der Elektrik/Elektronik gemessen an den Produktionskosten des Fahrzeugs. Beginn des Jahres 2002 betrug dieser Anteil bereits 25 Prozent (siehe Abbildung 1.1).

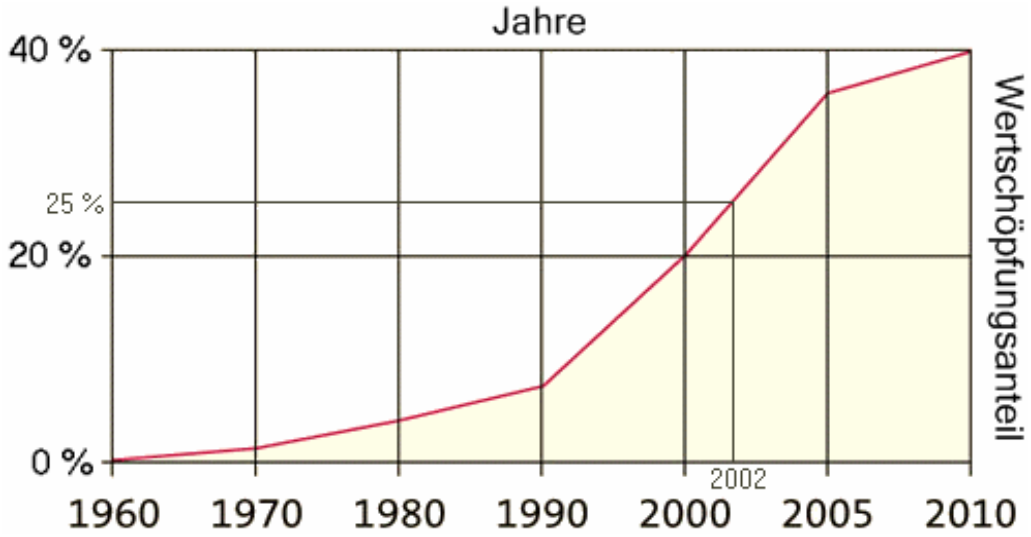


Abbildung 1.1 Wertschöpfungsanteil der Elektrik/Elektronik am Fahrzeug[10]

Alle derzeit aktuellen Studien zu den Themen kommen zu dem Ergebnis, dass dieser Anteil weiter zunehmen wird. So zeigt Abbildung 1.2, dass bei Audi 2003 der Anteil von Elektronik, Elektrik und Software an den Herstellkosten bereits bei 60 % liegt[11]. Für die nächsten 5 Jahre wird prognostiziert, dass dieser noch auf über 70 % steigen wird.

[8] Vgl. NZ 2005, Nr. 324641
[9] Automobil Produktion, Juni 2005, S. 24
[10] Vgl. Automobil Produktion, November 2002 (Online-Quelle)
[11] Vgl. Automobil Produktion, Juli 2003 (Online-Quelle)

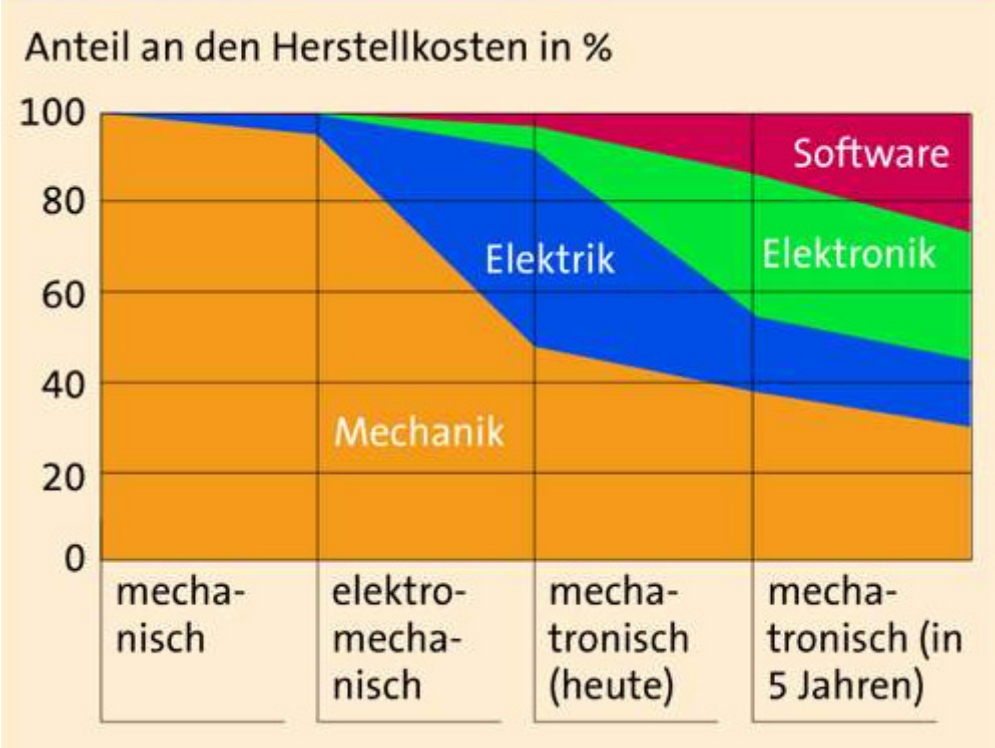


Abbildung 1.2 Wandel der Wertschöpfungsanteile (horizontal: Zeitspanne von 1960-2010, ~12,5 Jahre je Block)[12]

Elektronisch gesteuerte Systeme machen also heute häufig bereits 30 % eines modernen Fahrzeugs aus. Das bedeutet, dass derzeit Fahrzeuge auf den Markt kommen, die mit 40 bis 70 elektronischen Steuergeräten ausgestattet sind. Neben den altbekannten Systemen, wie z. B. elektronischer Zündung, ABS oder Benzineinspritzsysteme, sind dies heute auch Innenraum- und Komfortsysteme, wie z. B. Speicherung der Sitzposition (Elcode) oder Einparkassistenten (Park Distance Control). Laut einer Studie von Mercer soll die Anzahl der elektrischen Steuergeräte bis ins Jahr 2010 auf über 80 steigen[13]. Diese sind komplett miteinander vernetzt und werden in Zukunft untereinander Informationen mit elektronischen Systemen aus den unterschiedlichsten Bereichen des Fahrzeugs austauschen.

[12] Vgl. Automobil Produktion, Juli 2003 (Online-Quelle)
[13] Vgl. Automobil Produktion, Februar 2004 (Online-Quelle)

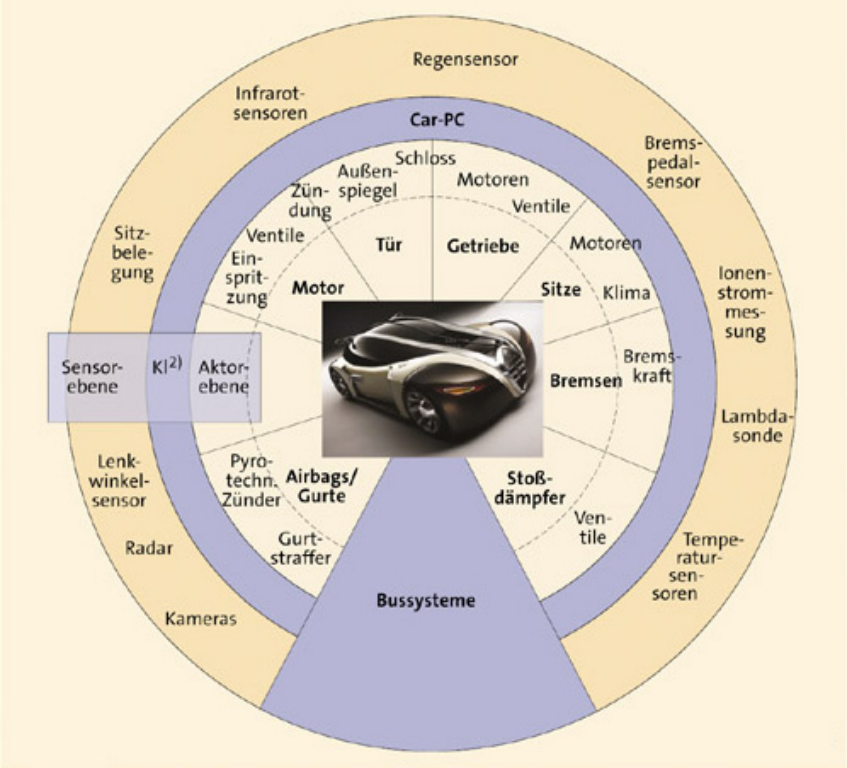


Abbildung 1.3 In Zukunft vernetzt: "Automobil 2010"[14]

Als Folge der Zunahme der Anzahl von elektronischen Steuergeräten und deren Aufgaben im Fahrzeug steigt mit dem Funktionsumfang auch die Komplexität dieser elektronischen Systeme. Studien belegen, dass für die Jahre nach 2005 ein überproportionaler Anstieg der Softwarekomplexität erwartet wird[15].

Durch die hohe Zahl an Rückholaktionen und die Elektroniklastigkeit der zu behebenden Fehler müssen vor allem die Fehlerdiagnosesysteme an die heutige Komplexität der Elektronik angepasst werden. Aber nicht nur die Rückholaktionen der Hersteller belegen, dass sich die Diagnosesysteme neuen Herausforderungen stellen müssen, sondern diese Aufgabe ergibt sich auch durch andere Bereiche der Automobilindustrie.

So wird im Branchenbericht 2004 des Zentralverbandes für Karosserie- und Fahrzeugtechnik (ZKF) auf die Tatsache hingewiesen, dass Werkstätten, die in Zukunft noch neuere Fahrzeuge reparieren möchten und somit wettbewerbsfähig bleiben wollen, "modernste Test- und Diagnosegeräte sowie Hard- und Software [benötigen], die nur durch aufwendige Investitionen erfüllbar sind"[16].

Des Weiteren trifft dieser Druck auch die Automobilzulieferer. 72 % der Fertigungskosten eines deutschen Fahrzeugs entfallen auf die Zulieferindustrie. Darin ist nahezu der gesamte Anteil der Elektronikkosten enthalten. Deshalb versucht die Automobilindustrie auch ihre Zuliefe-

[14] Vgl. Automobil Produktion, Februar 2004 (Online-Quelle)
[15] Vgl. Hupfer 2004, S. 16
[16] ZKF 2004, S. 6

rer bezüglich der Prüfung und Diagnose der elektronischen Steuergeräte in die Verantwortung zu nehmen[17].

Aufgrund dieser neuen technischen Herausforderungen, denen sich die Steuergerätediagnose gegenübersieht, müssen sich die unterschiedlichsten national und auch international tätigen Firmen der Fahrzeugbranche des Themas annehmen.

Durch die immer stärkere Verschmelzung von Mechanik und Elektronik mit der Software wird die Diagnose des Fahrzeugs mit neuen technischen Disziplinen konfrontiert. Das früher eher mechanische Diagnosewerkzeug wird mehr und mehr von softwaretechnischen Anforderungen durchdrungen. Im Zusammenhang mit der Produktionskostenproblematik vereinigt der Diagnoseprozess daher wirtschaftliche und informationstechnische Aspekte.

Deshalb erscheint es lohnend, sich der geschilderten Probleme im Bereich der Wirtschaftsinformatik anzunehmen. Die vorliegende Arbeit versucht, hierfür einen Beitrag zu leisten.

Zudem bietet die Automobilindustrie aus wirtschaftlicher Sicht ein Erfolg versprechendes Umfeld für Prozessoptimierungen. Laut einer Veröffentlichung des Mannheimer Zentrums für Europäische Wirtschaftsforschung verbessern im Fahrzeugbau über die Hälfte aller Prozessinnovationen die Qualität und reduzieren die Kosten des Produkts[18].

Abbildung 1.4 Was bringen Prozessinnovationen[19]

Im Fahrzeugbau ist auch ein forschungs- und entwicklungsfreudiger Markt anzutreffen. Laut einem Bericht des VDA aus dem Jahr 2004 hat die deutsche Automobilindustrie in den letzten fünf Jahren ihre Aufwendungen für Forschung und Entwicklung verdoppelt. 2003 betrug dieser 14,5 Mrd. Euro, also durchschnittlich 7 % des Umsatzes[20].

Die vorliegende Dissertation soll neben den konzeptionellen Prozessverbesserungen auch deren praktische Umsetzung liefern. Hierfür wird die Kooperation mit einem Vertreter aus der Automobilindustrie gesucht.

Bei näherer Betrachtung der einzelnen Fahrzeughersteller ist festzustellen, dass einzelne Vertreter bereits begonnen haben, ihr Diagnosesystem zu verbessern. So veröffentlichte VW Anfang 2006, dass testweise neue Software zur Fehlerdiagnose der Kfz-Elektronik des Modells

[17] Vgl. Dudenhöffer 2003, S. 6-10
[18] Vgl. Automobil Produktion, August 2004 (Online-Quelle)
[19] Vgl. Automobil Produktion, August 2004 (Online-Quelle)
[20] Vgl. VDA 2004, S. 205

Passat eingesetzt wurde. Nach dem erfolgreichen Abschluss der Testphase soll die neue Software schrittweise in den VW-Partnerbetrieben ausgerollt werden[21].

In Zusammenarbeit mit dem Kooperationspartner dieser Dissertation, einem führenden deutschen Kfz-Hersteller, sollen Verbesserungspotenziale bei der Diagnose von elektronischen Steuergeräten aufgespürt werden. Dabei sollen die damit in Verbindung stehenden Geschäftsprozesse erfasst und im Rahmen dieser Arbeit optimiert werden.

Angesichts der geschilderten Problematik, der die Automobilindustrie gegenübersteht, werden mit der Prozessoptimierung die Ziele Reduktion der Durchlaufzeiten und Qualitätssteigerung verfolgt. Die Prozesse der Steuergerätediagnose finden sich in den Kerngeschäftsbereichen wie Endmontage (Nacharbeit), Verwaltung, Entwicklung und Analyse wieder. Eine Optimierung, sprich zeitliche Verkürzung der Prozesse und qualitative Steigerung der Diagnosearbeit, wird sich folglich positiv auf die Produktionskosten und Qualitätssicherheit der Fahrzeuge auswirken.

1.2 Zielsetzungen und Aufbau der Arbeit

Die zentrale Zielsetzung der Arbeit besteht darin, einen innovativen Softwarebeitrag zur Diagnose elektronischer Steuergeräte in der Automobilindustrie zu leisten.

Den innovativen Charakter erhält die Software durch eine umfassende Modellierung, Analyse und Optimierung aller relevanten Geschäftsprozesse. Die betroffenen Geschäftsprozesse werden beginnend mit den Kernprozessen bis zu den einzelnen Arbeitsschritten erfasst und ausgewertet. Dadurch können bisher nicht erkannte Verbesserungspotenziale aufgedeckt werden. Durch deren Umsetzung soll eine Software entstehen, die die derzeitigen Arbeitsabläufe rund um die Steuergerätediagnose optimiert.

Aus der Aufgabenstellung, über die Prozessoptimierung zu einer neuartigen Diagnosesoftware zu gelangen, ergeben sich diverse Einzelziele für die Arbeit. Um deren strukturierte Abarbeitung zu gewährleisten, werden sie zunächst explizit formuliert und in die richtige Reihenfolge gebracht. Die einzelnen Ziele verstehen sich als Meilensteine und sollen schrittweise abgearbeitet werden. Sie sind somit richtungweisend für die Gliederung und das weitere Vorgehen innerhalb der Arbeit. Der Aufbau der Gliederung orientiert sich aus gegebenem Anlass an einem Ablaufschema zur Geschäftsprozessoptimierung (siehe Staud[22] und Becker[23]):

1. Darstellung der technischen Grundlagen und wesentlicher Komponenten bei der Diagnose von Steuergeräten (allgemein und kooperationspartnerspezifisch)
2. Abgrenzung der Arbeit (Ansatzpunkt und Zielsetzung der Modellierung)
3. Erfassung der relevanten Geschäftsprozesse (Ist-Modellierung)
4. Schwachstellenanalyse der gegenwärtigen Prozesse (Ist-Modell-Analyse)
5. Entwurf von verbesserten Prozessen (Soll-Modellierung)

[21] Vgl. Wehner 2006 (Online-Quelle)
[22] Vgl. Staud 2001, S. 17
[23] Vgl. Becker 2000, S. 18

6. Nachweis der Behebung der Schwachstellen (Soll-Modell-Simulation)
7. Definition der neuen Anforderungen anhand der verbesserten Prozesse und deren Vergleich mit bestehenden Angeboten auf dem Softwaremarkt und an Forschungsprojekten
8. Entwurf eines neuen Konzepts auf Basis der Anforderungen (Softwarearchitektur) oder Verwendung eines Marktangebots
9. Falls der Entwurf eines neuen Systems notwendig ist, erfolgt dessen Umsetzung (Implementierung)
10. Leistungsbewertung des verbesserten Diagnosesystems im Rahmen einer Pilotprojektierung (Softwareevaluation)

Die Zielsetzungen werden im Folgenden näher erläutert. Da die Abarbeitung der Ziele die Gliederung der Arbeit reflektiert, wird hierbei darauf verwiesen, in welchen Kapiteln die entsprechenden Inhalte wiedergegeben werden.

zu (1) Aufgrund der Eingliederung der Arbeit in die Thematik der Wirtschaftsinformatik, werden die Ansatzpunkte für die Prozessverbesserung im Bereich der IT gesucht. Aus technischer Sicht besteht die Diagnose von Steuergeräten im Grunde darin, unter Zuhilfenahme von Steuergerätefunktionen die Fehlerursache zu identifizieren.

In Kapitel 1.1 werden zwei Komponenten des Diagnoseablaufs vorgestellt, die aufgrund des Wandels in der Kfz-Elektronik starken Änderungen unterworfen sind: das Steuergerät und die Diagnosesoftware.

Das Kapitel 2 befasst sich damit, die Grundlagen dieser beiden Komponenten und deren Zusammenspiel darzulegen. Es wird hierbei auch auf die verschiedenen Ebenen bei der Kommunikation zwischen Diagnostiker und Steuergerät eingegangen (Kapitel 2.2).

In Kapitel 3 werden die Technologien behandelt, die speziell beim Kooperationspartner zur Steuergerätediagnose eingesetzt werden.

zu (2) Aus den Bauteilen (IT-Komponenten) des Diagnosesystems wird in Kapitel 4 der Ansatzpunkt für die Prozessverbesserungen abgegrenzt.

Die HMI-Software wird als für die Arbeit relevante Komponente bestimmt. Sie umfasst die Oberfläche und die Verarbeitungslogik der Diagnosesoftware. Über sie kann der Diagnostiker mit dem Steuergerät interagieren. Diese Begriffe und Zusammenhänge werden in Kapitel 2.2 ausführlich erläutert.

Ferner werden in Kapitel 4 die Gründe für die Auswahl dieses Ansatzpunktes dargelegt. Es wird ein großes Innovationspotenzial in den Bereichen Mobilität, Funktionalität und grafischer Gestaltung gesehen.

Vor dem Hintergrund der Abgrenzung und der technischen Grundlagen der Diagnose von Steuergeräten kann der Ist-Zustand erfasst werden.

zu (3) Ziel von Kapitel 5 ist es, alle Prozesse zu erfassen, in denen die HMI-Diagnosesoftware zum Einsatz kommt.

Hierbei wird die Top-down-Methode angewendet. Angefangen mit den Kernprozessen bis hin zu den einzelnen Arbeitsschritten werden die Prozessdaten in eine ARIS-Datenbank eingepflegt. In Kapitel 5 werden nur die für die Analyse relevanten Prozesse dokumentiert.

zu (4) Anhand der in (2) vermuteten Innovationspotenziale werden die erfassten Prozesse analysiert. In Kapitel 6 werden hierzu die verschiedenen Analysemethoden gegeneinander abgewogen. Gängige Methoden sind Referenzmodelle, Benchmarking, Schwachstellenanalysen, Fragenkataloge und Kennzahlenanalysen.

zu (5 + 6) Die Kritiken aus Kapitel 6 werden in Kapitel 7 in Form von Verbesserungsvorschlägen umgesetzt. Auch hier werden verschiedene Herangehensweisen in Betracht gezogen – neben radikalen Business-Process-Reengineering-Maßnahmen auch einmalige Geschäftsprozessoptimierungen. Die verbesserten Entwürfe werden als Soll-Prozesse modelliert.

Anhand einer Soll-Prozess-Simulation wird zunächst theoretisch nachgewiesen, dass die Änderungen die Schwachstellen beheben.

zu (7 + 8) Als Folge der optimierten Abläufe werden klare Anforderungen definiert, die die Diagnosesoftware erfüllen muss. Die Definition erfolgt in Form eines Anforderungskatalogs, der als Pflichtenheft für eine verbesserte HMI-Software gilt (Kapitel 7.3). Die Anforderungen betreffen diverse technische Bereiche der Software (Betriebssystem, Netzwerk, Entwicklungssprache, Datenformat etc.).

Im Anschluss an die Bestimmung der Anforderungen wird das bestehende Marktangebot entsprechend der Erfüllung dieser Kriterien untersucht (Kapitel 8).

Zu (9) Falls das Marktangebot, wie zu erwarten, den gestellten Anforderungen nicht entspricht, erfolgt eine eigenständige softwaretechnische Umsetzung des Anforderungskatalogs. Die Arbeit soll damit nicht nur einen konzeptionellen Vorschlag, sondern auch eine einsatzfähige Software liefern. Die Basistechnologien und Architekturschichten des neuen Konzepts werden erklärt, sowie die Umsetzung (Programmierung) nachvollziehbar dargestellt. In Kapitel 9 und 10 wird diese Aufgabestellung bearbeitet.

zu (10) Nach der Implementierung wird untersucht, ob die entwickelte Software die gestellten Anforderungen erfüllt.

Der Software wird hierfür beim Kooperationspartner im Rahmen einer testweisen Pilotprojektierung als Prototyp eingeführt. Die Projektierung erfolgt innerhalb ausgewählter Geschäftsprozesse. Anhand der Daten aus nachgestellten Anwendungsfällen und der Produkteigenschaften (Fähigkeiten) des Prototyps wird dieser bewertet.

Die Bewertung schließt eine analytische und experimentelle Evaluation ein. Die analytische Evaluation soll anhand qualitativer und quantitativer Daten (hauptsächlich Fähigkeiten des Prototyps) Ergebnisse dafür liefern, ob das neue System alle Kriterien des Anforderungskatalogs erfüllt. Die experimentelle Evaluation zeichnet sich durch einen hohen Grad an Beobach-

tungsdaten aus. Die Daten werden aus kontrollierten Experimenten, den Anwendungsfällen, gewonnen. Die Untersuchung basiert auf objektiv ermittelten Werten wie Durchlaufzeiten. Das Ergebnis der Evaluation soll die Bestätigung liefern, dass mit dem erarbeiteten Diagnosesystem eine Verbesserung der erfassten Geschäftsprozesse möglich ist.

Die Arbeit schließt mit einer Zusammenfassung und einem Ausblick ab. Das Kapitel 12 liefert hierfür eine Einschätzung, inwieweit das Ergebnis der Arbeit zukunftsfähig ist. In diesem Rahmen werden aktuelle Markttendenzen beschrieben sowie die Potenziale betrachtet, die sich für den Prototyp innerhalb des Pilotprojekts bieten.

2 Grundlagen

Im ersten Kapitel wurde die Kfz-Elektronik als Markt mit hohem Innovationspotenzial identifiziert[24]. Als Folge gravierender Rückholaktionen bei den Kfz-Herstellern erscheint vor allem deren Fehlerdiagnose als der Bereich, auf den das Augenmerk für Verbesserungen zu legen ist.

In Kapitel 1.1 wurde als Hauptgrund der genannten Rückholaktionen die Elektronik genannt. Heute kontrollieren Steuergeräte den entscheidenden Teil der Funktionalität der Elektronik. Sie sind aus einem modernen Fahrzeug nicht mehr wegzudenken[25]. Ihre gestärkte Stellung im Fahrzeug ist die Ursache für neue Herausforderungen bei der Diagnose.

Als Einstieg in die Arbeit werden deshalb die Grundlagen der elektronischen Steuergeräte im Kraftfahrzeug behandelt. Die Ausführungen in Kapitel 2.1 beschäftigen sich mit dem Aufbau und der Funktionalität der Geräte. Auch ihr historischer Verwendungszweck sowie Beispiele für ihren momentanen Einsatz, ihre Funktionsweise und Kommunikation werden erläutert.

Für die Diagnose ist neben dem zu prüfenden Steuergerät das Diagnosewerkzeug von entscheidender Bedeutung. Es bildet die Schnittstelle zwischen Automobildiagnostiker und Steuergerät. Orientiert an dieser Schnittstelle wird in Kapitel 2.2 der Vorgang der Diagnose erläutert. Es werden die beteiligten Komponenten des Vorgangs und deren Zusammenhänge aufgezeigt.

2.1 Grundlagen elektronischer Steuergeräte

2.1.1 Historische Entwicklung

Elektronische Systeme haben sich über die letzten 40 Jahre hinweg mit fast dramatischer Geschwindigkeit entwickelt. Inzwischen haben sie in allen wichtigen Anwendungsbereichen des Fahrzeugs wie Motor, Fahrzeugbewegung, Sicherheit, Komfort oder Multimedia Einzug erhalten[26].

[24] Vgl. Schilder 2005, S. 172
[25] Vgl. Bosch 2002, S. 288
[26] Vgl. Gevatter 2006, S. 509

Als Geburtsstunde und Vorreiter der Fahrzeugelektronik wird in den meisten Quellen die Einführung der Transistorzündung in den 70er Jahren genannt[27]. Damit wurde bei der Zündung der Nachteil des Kontaktabbrands an den Unterbrecherkontakten beseitigt. Die Transistorzündung löste den rein mechanischen Vorgänger der Spulenzündung ab.

Die Transistorzündung wurde zunächst, wie andere Elektronikbauteile hauptsächlich aufgrund der hohen Kosten nur in Luxuskarossen eingebaut. Der Wertanteil der Elektronik pro Fahrzeug lag 1980 nur bei einem halben Prozent (Gesamtdurchschnitt)[28].

In Kombination mit der Mechanik dienten elektronische Systeme vorerst dazu, die Genauigkeit zu verbessern[29]. So konnte mit der Transistorzündung die Genauigkeit des Zündzeitpunkts über einen sehr viel längeren Zeitraum erhalten werden.

Das Konzept, Mechanik und Elektronik zu verbinden, war anfänglich nur wenig verbreitet. Trotzdem etablierte sich hierfür sehr früh der Begriff "Mechatronik", den 1969 die japanische Firma Yaskawa Electric Cooperation erstmals als interdisziplinäres Zusammenwirken der beiden Kernelemente: Mechanik und Elektronik definierte[30].

Allerdings erst zwanzig Jahre nach ihrer ersten Definition begann die Mechatronik, für die Fahrzeugindustrie interessant zu werden. Mit den immer kostengünstiger werdenden Elektronikteilen in den 90ern konnten diese auch in Fahrzeuge der Mittelklasse oder in Kleinfahrzeuge eingebaut werden. 1990 lag der Wertanteil der Fahrzeugelektronik bereits bei 7 % pro Fahrzeug[31]. Dies sollte aber erst den Anfang eines von diesem Zeitpunkt an rapide wachsenden Marktes sein. Die folgende Abbildung veranschaulicht diese Entwicklung.

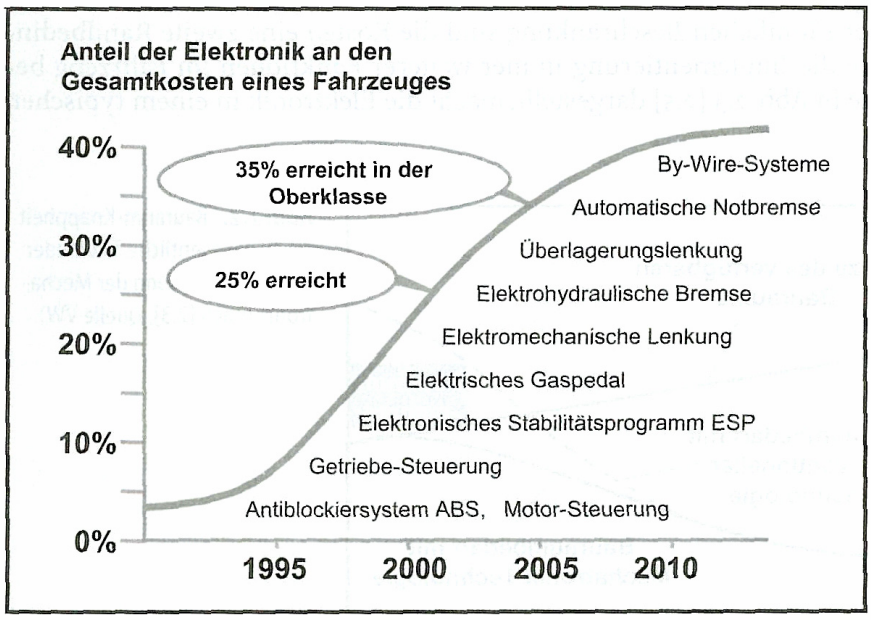


Abbildung 2.1 Entwicklung des Anteils der Elektronikkosten an den Fahrzeuggesamtkosten[32]

[27] Vgl. Bosch 2002, S. 288
[28] Vgl. Bosch 2002, S. 290
[29] Vgl. Gevatter 2006, S. 9
[30] Vgl. Gevatter 2006, S. 9
[31] Vgl. Bosch 2002, S. 290
[32] Vgl. Gevatter 2006, S. 8

Aber nicht nur der Preisverfall und die Leistungssteigerung der Elektronik, sondern auch noch andere wichtige Entwicklungstendenzen der Mechatronik begünstigten die starke Zunahme ihrer Verbreitung.

So wird z. B. auch die Miniaturisierung als wichtiger Grund für die Zunahme der Elektronik angesehen. Die immer kleiner werdenden Bauteile konnten leichter im Fahrzeug montiert und aufgrund ihrer geringen Größe vor allem vor den Einflüssen der Umgebung des Kraftfahrzeugs wie Nässe, Öle oder hohe Temperaturen geschützt werden, wodurch sich der Mechatronik viel mehr Einsatzmöglichkeiten eröffneten.

Die Mechatronik stellt "heute eine der größten Herausforderungen in der Fahrzeugbranche" dar[33]. Eine weitere Entwicklung in der Elektronik, die zu dieser raschen Zunahme ihrer Bedeutung beigetragen hat, ist die Steigerung der Rechen- und Speicherleistung. Durch die Verarbeitungsmöglichkeit von immer mehr Daten in kürzerer Zeit können immer komplexere Funktionsbereiche gesteuert werden.

Die Logik der Steuerung und Verarbeitung der Daten in den elektronischen Systemen liegt in der Software, oder allgemeiner bezeichnet, in der Informatik. Mit der Zunahme der Komplexität der Aufgaben wurde auch die Software für die Mechatronik immer bedeutender. Dies führte dazu, dass man ihre Definition nochmals aufbrach. Die Mechatronik soll als Interaktion von jetzt drei Kerndisziplinen verstanden werden: Mechanik, Elektronik und Informatik[34].

Auch die Bedeutung der Kerndisziplinen für die Wirtschaft hat sich über die Jahre hinweg verschoben. Die Komponente Elektronik und die neue Komponente Software, die ursprünglich eingeführt wurden, nur um die Genauigkeit zu verbessern, sind heute die Treiber der Innovationen[35]. Die inzwischen als selbstverständlich erscheinenden Funktionen wie ABS, ESP oder elektronische Zündung hätten ohne Elektronik und Software nicht realisiert werden können[36]. Einige dieser Beispiele werden im folgenden Kapitel veranschaulicht.

2.1.2 Anwendungsbeispiele

Inzwischen unterliegen fast alle rein mechanischen Systeme einem Wandel hin zu mechatronischen Systemen[37]. Abbildung 2.2.2 vermittelt einen ersten Eindruck, wie vielfältig diese Systeme bereits heute sind. Eine vollständige Auflistung erscheint derzeit kaum möglich, da der Markt in zu starker Bewegung ist und für jedes Kfz-Modell weitere Anwendungen erschlossen werden.

Zur Veranschaulichung werden Beispiele aus den Anwendungsbereichen Sicherheit, Antriebsstrang und Komfort kurz erläutert.

[33] Gevatter 2006, S. 509
[34] Vgl. Schweitzer 1989, VDI Bericht 787
[35] Vgl. Schilder 2005, S. 172
[36] Vgl. Gevatter 2006, S. 509 oder Bosch 2002, S. 288
[37] Vgl. Gevatter 2006, S. 15

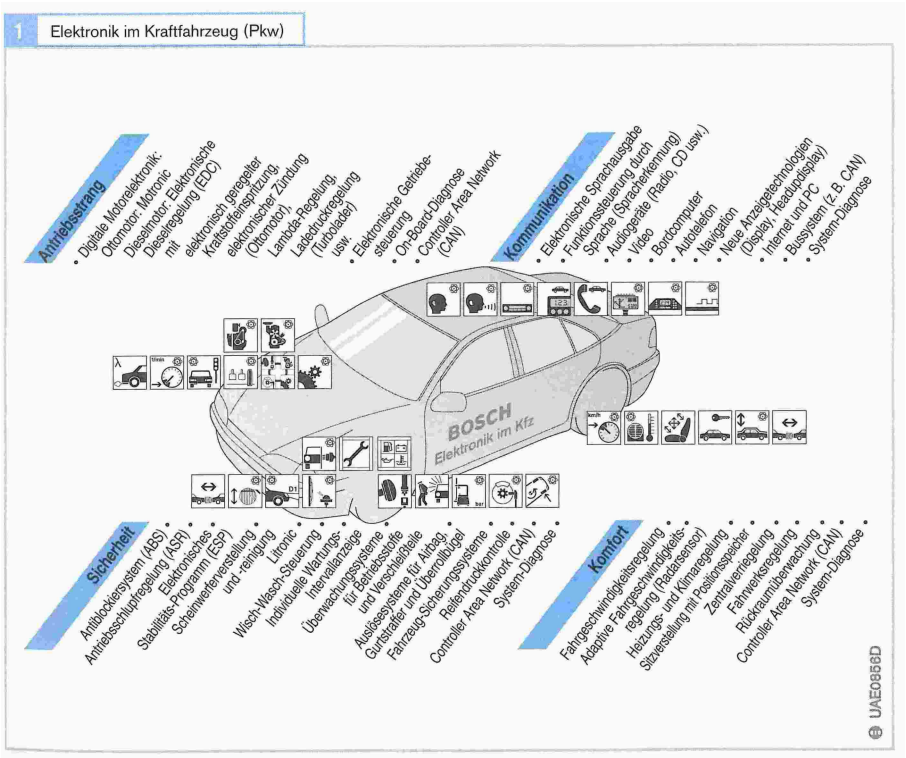


Abbildung 2.2 Elektronik im Kraftfahrzeug (Pkw)[38]

Ein historischer Schritt in der Entwicklung von Mechatronik-Systemen vollzog sich im Fahrwerkssystem. Anfang der 80er Jahre überraschte Mercedes mit der revolutionären Entwicklung des Antiblockiersystems (ABS). Es verhindert, dass die Räder bei einer Vollbremsung blockieren und damit unlenkbar einfach geradeaus rutschen. Nur geübte Fahrer konnten bis dahin diesen Zustand durch Stotterbremsen verhindern[39].

Infolge der nun höheren Leistung der Elektronikbauteile wurde 1995 das System serienmäßig um ein elektronisches Stabilitätsprogramm (ESP) erweitert. Durch sensorische Überwachung der Raddrehzahlen (ABS-Sensor) und durch Vergleich von Lenkwunsch des Fahrers (Lenkwinkel-Sensor) und tatsächlicher Fahrzeugbewegung (Querbeschleuniger-Sensor) wird gezielt auf die Bremsen der einzelnen Räder (ESP-Steuergerät und -Aktor) Einfluss genommen. Dadurch wird das Schleuderrisiko erheblich vermindert.

Inzwischen ist durch Unfallstatistiken nachgewiesen, dass diese Systeme die Anzahl von Unfällen deutlich reduziert haben[40]. ESP sorgte auch dafür, dass die zunächst nicht kippsichere A-Klasse schließlich doch zuverlässig den Kurventest bestand[41].

Die Einführung der Mechatronik vollzog sich im Triebstrang zwar weniger populär, aber dennoch mit gleicher Intensität. Die Einführung der Transistorzündung als Vorreiter der Mechatronik war nur der erste Schritt. In vielen Fahrzeugen übernimmt bereits ein digitales Motormanagement (DME) die Steuerung des Zündzeitpunkts. Kernstück ist ein Steuergerät, das mit einer flexibel einstellbaren Steuerungssoftware ausgestattet ist. Daten über Drehzahl,

[38] Vgl. Bosch 2002, S. 289
[39] Vgl. Schilder 2005, S. 32
[40] Vgl. Lingnau 2003, S. 29
[41] Vgl. Schilder 2005, S. 33

Zündzeitpunkt, Einspritz- und Luftmenge, Abgaswerte (Lambda-Sonde), Kurbelwellenstellung, Gaspedalstellung, Öltemperatur und viele andere Details laufen dort zusammen. Je nach den Daten werden Steuerbefehle an die Aktoren des Motors gegeben. Beispielsweise wird der Zündzeitpunkt früher gesetzt, wenn der Motor "klopft"[42].

Ein anderes elektronisches System, die On-Board-Diagnose (OBD), überwacht permanent das Abgasverhalten des Motors. Das aus den USA übernommene System wurde 2001 in den EU4-Abgasrichtlinien verankert[43]. Das System sorgt dafür, dass die Abgaswerte kontinuierlich im vorgegebenen Rahmen gehalten werden. Ist das System gestört, leuchtet ein Warnsignal im Cockpit auf[44].

Im Komfortbereich des Kraftfahrzeugs ist die Klimaregelung ein Hauptgebiet der Mechatronik. Lüfter, Heizelemente, Klimakompressor und Klappen in den Luftkanälen werden auf der Basis von Sensorinformationen über Temperatur, Sonnenstand und Strahlungsintensität geregelt. Moderne Klimaanlagen sind auch bei niedrigen Temperaturen stets im Einsatz. Die Anlage kühlt die Luft zuerst ab und trocknet sie dabei, um sie anschließend auf die gewünschte Temperatur zu erwärmen (Reheat-Funktion). Dadurch bleiben die Scheiben immer beschlagsfrei. Intelligente Systeme (Thermotronic) berücksichtigen hierbei sogar noch die Luftfeuchtigkeit. Ein Taupunkt-Sensor misst die Luftfeuchtigkeit und das Klima-Steuergerät kann so entscheiden, wie stark die Temperatur abgekühlt und getrocknet werden muss[45]. Die Innovationen an dieser Stelle reißen nicht ab. Eine neue Möglichkeit der Temperaturmessung in der Fahrzeugkabine erfolgt ist die Zonen-Infrarot-Kontrolle. Durch mehrere Infrarot-Sensoren wird dabei die Temperatur direkt von der Oberfläche der Insassen abgelesen. Das System orientiert sich auf diese Weise noch stärker an der tatsächlich gemessenen Temperatur[46].

Die Anwendungsgebiete Komfort und Sicherheit liegen in der Kfz-Elektronik eng beieinander. Das Ein- und Ausschalten der Scheinwerfer geschieht in einigen Fahrzeugen in Abhängigkeit von den Beleuchtungsbedingungen. Helligkeitssensoren werten die Lichtverhältnisse aus. Sie dunkeln z. B. bei störender Überbeleuchtung durch heranfahrende Fahrzeuge die Rückspiegel ab. Ein anderes Beispiel sind adaptive Kurvenlichter. Sie richten beim Fahren in der Kurve die Lichtkegel des Scheinwerfers in Abhängigkeit vom Lenkwinkel wieder auf die Straße. In Zukunft sollen hierfür sogar Informationen aus dem Navigationssystem berücksichtigt werden[47].

Das Angebot von mechatronisch gesteuerten Systemen im Fahrzeug wächst immer noch weiter. Airbags werden je nach der Aufprallstärke stufenweise gezündet. Durch das neue Pre-Safe-Konzept sollen mittels der Auswertung von Umfeldsensoriken Sitzlehnen in eine aufrechte Position gebracht und Gurtstraffer vorgespannt werden. Der Fahrer wird so kurz vor einem Unfall in eine stabile Lage gebracht. Erweiterungen sind angedacht, um auch mit der Sitzfederung Aufprallschwingungen abzufangen.

[42] Vgl. Schilder 2005, S. 135
[43] Vgl. Birnbaum 2000, S. 10
[44] Vgl. Schilder 2005, S. 135
[45] Vgl. Schilder 2005, S. 85
[46] Vgl. Valldorf 2003, S. 361
[47] Vgl. Gevatter 2006, S. 19

2.1.3 Aufbau und Funktionsweise

Trotz der vielfältigen und permanent zunehmenden Einsatzbereiche von mechatronischen Systemen sind die Systeme grundsätzlich einheitlich aufgebaut.

Im vorherigen Kapitel war immer wieder von folgenden Komponenten zu lesen: Sensor, Steuergerät und Aktor. Hinzu kommt die Vernetzung (Kommunikation) als vierter elementarer Funktionsblock eines elektronischen Systems.

Prinzipiell lässt sich der Aufbau wie folgt erklären. Das Steuergerät bedient sich der Aktoren und Sensoren, um mit der Mechanik und Elektronik zu kommunizieren. Folgende Abbildung schematisiert den Zusammenhang dieser vier Komponenten noch einmal:

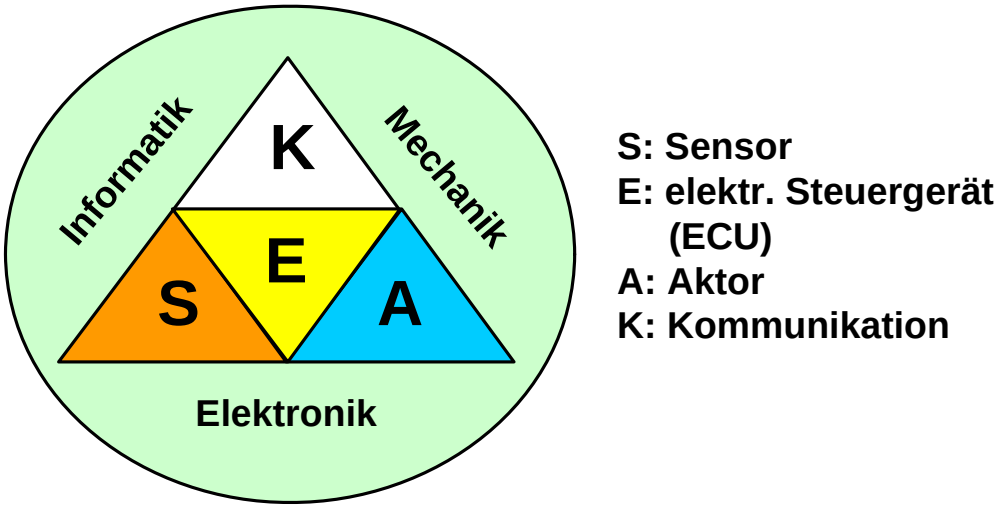


Abbildung 2.3 Mechatronisches System[48]

Sensoren dienen der Erfassung von Betriebszuständen und -bedingungen (z. B. Motordrehzahl, Raddrehzahl, Temperatur). Sie wandeln physische Größen in elektrische Signale um.

Das elektrische Steuergerät (engl. electronic control unit, ECU) verarbeitet die Informationen der Sensoren nach bestimmten Steuer- und Regelalgorithmen. Entsprechend dem Ergebnis der Informationsverarbeitung steuert es die Aktoren mit elektrischen Ausgangssignalen an.

Die Aktoren setzen die elektrischen Ausgangssignale des Steuergeräts in mechanische Größen um.

Es finden sich auch hier die Kernkomponenten der Mechatronik wieder:

- die Elektronik als Basis, um die Informationen zu übermitteln (elektrische Signale),
- die Informatik, um diese zu verarbeiten und
- die Mechanik, auf die das System wirkt.

Deshalb werden diese Systeme auch als mechatronische Systeme bezeichnet.

Bei vielen modernen Anwendungen wie ESP ist es notwendig, dass Steuergeräte Informationen von anderen Steuergeräten oder mehreren Sensoren einholen. Hierbei kommt die Kompo-

[48] angelehnt an Gevatter 2006, S. 12

nente Kommunikation bzw. Vernetzung zum Tragen. Außerdem dient sie dazu, um bei der Diagnose von außen auf die Funktionen des Steuergeräts zuzugreifen.

Das Zusammenwirken der vier Einheiten kann an fast jedem elektronischen System im Fahrzeug nachvollzogen werden, wie im folgenden Beispiel dargestellt.

Das ABS-System besteht aus Radsensoren (an jedem Vorderrad mindestens einer), einem ABS-Steuergerät und Magnetventilen (Aktoren) an den Bremszylindern. Die Radsensoren messen die Raddrehung an jedem Rad und melden an das Steuergerät, wenn das Rad blockiert. Daraufhin sendet das Steuergerät ein entsprechendes Signal an das Magnetventil und reduziert so den Bremsdruck. Das Rad kommt auf diese Weise vor der vollständigen Blockade wieder zum Drehen. Das ABS-Steuergerät korrigiert somit den Bremsdruck für jedes Rad individuell[49].

In dem Steuergerät verbinden sich die zwei Kernkomponenten der Mechatronik – die Elektronik und Informatik. Wie sich diese Verbindung darstellt, zeigt folgende Abbildung:

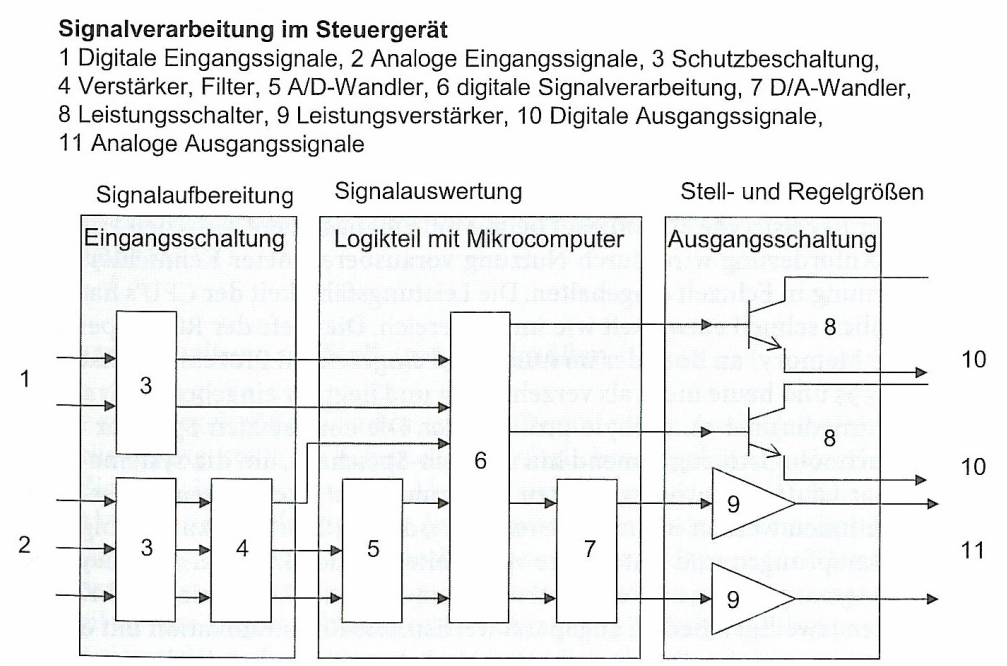


Abbildung 2.4 Signalverarbeitung im elektronischen Steuergerät[50]

Die Eingangs- und Ausgangsschaltung dient zum Empfang bzw. zur Weitergabe elektronischer Signale. Diese beiden Einheiten machen die Kommunikation mit Sensor und Aktor sowie mit anderen Steuergeräten und auch der Diagnosesoftware möglich. Dieser Teil ist von der Elektronik dominiert.

Im Kern des Steuergeräts vollzieht sich die Signalverarbeitung. Hier werden durch die Software die empfangenen Informationen verarbeitet und entsprechende Antworten formuliert. Man spricht von dem sogenannten Logikteil des Steuergeräts. Mit den immer neuen Anwendungsgebieten der elektronischen Systeme steigt vor allem der Anspruch an den Logikteil (Datenverarbeitung) enorm. Die Software des Steuergeräts muss immer mehr Signale verschie-

[49] Vgl. Schilder 2005, S. 34
[50] Vgl. Gevatter 2006, S. 511

denster Sensoren aufnehmen und in immer kürzerer Zeit verarbeiten. Dieser Teil ist folglich in den letzten Jahren mehr und mehr durch die Informatik (IT-Komponenten) geprägt. Er macht Errungenschaften, wie z. B. ESP, Pre-Crash-Systeme oder automatische Abstandsregelung, überhaupt erst möglich.

Der Logikteil ist Ansatzpunkt für die Diagnosesoftware. Mit ihm gilt es zu kommunizieren, um die Funktionalität des Steuergeräts zu untersuchen. Allerdings ebnete erst die zentrale Vernetzung der Steuergeräte den Weg, um die Diagnose effizient durchführen zu können.

2.1.4 Kommunikation und Vernetzung

Zu Beginn der Einführung der elektronischen Systeme in den 70ern waren die Systeme selbstständig und von anderen Systemen unabhängig (autark). Sie waren nur für eine begrenzte Aufgabe zuständig, wie z. B. die Zündauslösung bei der Transistorzündung.

Aber noch im gleichen Jahrzehnt sollten verschiedene Informationen von mehreren elektronischen Systemen gleichzeitig genutzt werden. Ein Beispiel ist die L-Jetronic, die in Abhängigkeit von mehreren Faktoren, wie Motordrehzahl oder Last, die nötige Kraftstoffmenge aus den elektrischen Einspritzventilen einfließen lässt. Es wurde begonnen, die Steuergeräte untereinander zu vernetzen.

Der nächste Schritt war, dass sich mehrere Steuergeräte gegenseitig beeinflussten, wie z. B. das Schaltverbot bei der Antriebschlupfregelung (ASR). ASR verhinderte mit seiner Entstehung in den 80ern das Durchdrehen der Antriebsreifen beim Anfahren.

Die gegenseitige Einflussnahme und Informationsverarbeitung der Steuergeräte untereinander ist bei einem modernen Fahrzeug nicht mehr wegzudenken[51]. Zum Beispiel Sicherheitssysteme wie die Zentralverriegelung, die mit der Wegfahrsperre kommunizieren muss, um sie an- und auszuschalten.

Diese Entwicklung blieb auch für die Hardware im Fahrzeug (technische Ausrüstung) nicht ohne Folgen. Ein heutiges Fahrzeug mit z. B. 67 Steuergeräten verfügt über eine "gesamte Kabellänge aller verbauten Leitungen von mehr als 3 km"[52]. Hinzu kommen mehr als 3000 elektrische Anschlüsse (Pins).

Die Vielzahl der elektronischen Systeme und der damit verbundene umfangreiche Verdrahtungsaufwand machten für den Datenaustausch auch ein dafür geeignetes Kommunikationssystem erforderlich. Um dies zu bewerkstelligen, bediente man sich hierbei einer Lösung aus der Informatik. Hier war das Problem, mehrere Rechner zu koppeln, bereits gelöst. Das sogenannte Bussystem hielt Einzug in die Kfz-Welt. Ein Bus bezeichnet in der Datenverarbeitung eine "Verbindungsleitung innerhalb eines Rechners bzw. zwischen mehreren Rechnern, auf denen Informationen übertragen werden"[53].

Trotz der damaligen Einigung, auf die Bussystem-Technologie hat sich bis heute aufgrund der verschiedensten Einsatzgebiete der Steuergeräte eine "nahezu unübersehbare Vielfalt von öf-

[51] Vgl. Herner 2001, S. 423
[52] Herner 2001, S. 424
[53] Herner 2001, S. 427

fentlichen und proprietären Standards" entwickelt[54]. Eine Aufzählung der wohl bekanntesten Vertreter wie CAN, MOST, Firewire oder Flexray soll an dieser Stelle genügen.

Die Steuergeräte förderten diese vielfältige Entwicklung, da sie den Leitungen verschiedenste Aufgaben abverlangen. Die Anforderungen unterscheiden sich nach Datenmenge, Schnelligkeit der Übertragung, Priorität oder auch in den Sicherheitsmechanismen.

Aber neben den Bussen selbst tragen auch noch die darauf aufsetzenden Protokolle zur Steigerung der "Unübersichtlichkeit" im Kommunikationsbereich bei. Das Problem ist, dass bei Bussen und Protokollen selbst Normierungsgremien "Begriffe unterschiedlich, gar nicht oder sogar unterschiedliche Dinge gleich benennen"[55].

Protokolle sind vor allem dann von Bedeutung, wenn es um die Auswertung von Nutzdaten des Steuergeräts von außen geht. Beispielsweise wenn eine Diagnosesoftware auf Daten des Steuergeräts zugreifen möchte, müssen für die Kommunikation zwischen Diagnosesoftware und Steuergerätsoftware beide das gleiche Protokoll unterstützen.

So sind speziell für den Bereich der Diagnose eine Reihe von Protokollen und Bussen entstanden. ISO 9141-CARB, KWP 2000 oder CCP sind Beispiele für öffentliche Diagnoseprotokolle mit entsprechenden Bussen, wie L-Line oder K-Line bzw. ISO 9141. Hinzukommen eine Reihe von proprietären Lösungen der einzelnen Kfz-Hersteller.

Deshalb erscheint es bei der Diagnosesoftware sinnvoll, den Softwarebereich, der für die eigentliche Kommunikation mit dem Steuergerät zuständig ist, von der übrigen Diagnosesoftware abzutrennen. Dadurch kann der Kommunikationsbereich leichter aktualisiert werden, wenn sich z. B. Protokolle oder Bussysteme verändern oder neu hinzukommen. Ein weiterer Vorteil ist, dass der andere Bereich der Diagnosesoftware, der z. B. der Darstellung der Diagnoseoberfläche dient, sich nicht mit Protokollen oder Bussystemen auseinandersetzen muss.

Die Aufteilung der Softwarebereiche bzw. Schichten verdeutlicht das nächste Kapitel. Es beschreibt, welche funktionelle Bedeutung diese Schichten bei der Diagnose eines Steuergeräts einnehmen.

2.2 Bedeutung der Mensch-Maschine-Schnittstelle (HMI) in der Diagnose elektronischer Steuergeräte

Ziel dieses Kapitels ist es zu erläutern, welche Schichten bei der Kommunikation zwischen Diagnostiker und Steuergerät im Fahrzeug durchlaufen werden. Anhand dieser Schichten wird in Kapitel 4 der Ansatzpunkt der Arbeit abgegrenzt. Die Erläuterung der Schichtenaufteilung wird durch die Abbildung 2.5 unterstützt. Sie ist angelehnt an die Grafik "Benutzeroberfläche in automatisierten Produktionsanlagen" von Charwat[56]. Beginnend mit einer allgemeinen Darstellung werden in der Abbildung die Begrifflichkeiten immer weiter auf die Diagnosetechnik zugeschnitten. Die Grafik wird aus Sicht der Spalten von links nach rechts also immer speziali-

[54] Zimmermann 2006, S. V
[55] Zimmermann 2006, S. 6
[56] Vgl. Charwat 1992, S. 62

sierter. Die vorgestellten Farben und Farbtöne werden signifikant für die einzelnen Schichten und deren Untergliederung genutzt. Sie werden in dem weiteren Verlauf der Arbeit mit gleicher Bedeutung eingesetzt.

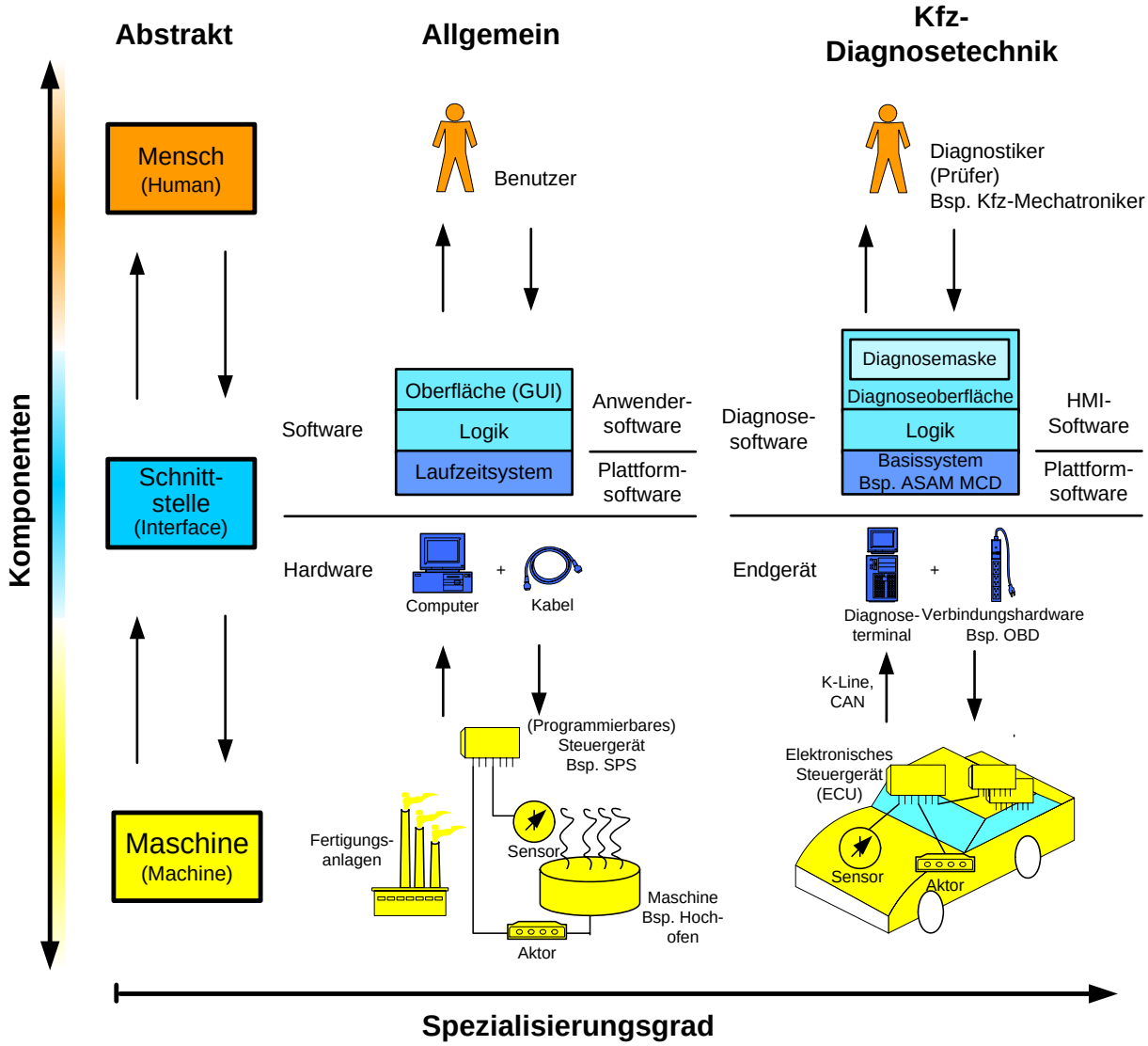


Abbildung 2.5 Mensch-Maschine-Schnittstelle (HMI) in der Steuergerätediagnose

Die begriffsgenaue Erläuterung der einzelnen Schichten ist aus zwei Gründen an dieser Stelle wichtig. Zum einen dient sie dazu, in Kapitel 4 den Ansatzpunkt der Arbeit abzugrenzen. Zum anderen öffnet sich mit der begrifflichen Bestimmung der Schnittstelle zwischen Mensch und Maschine ein weites Feld. Wie kaum in einem anderen Gebiet der Informationstechnik werden so viele unterschiedliche Begriffe für denselben Gegenstand verwendet und umgekehrt denselben Begriffen unterschiedliche Bedeutungen zugeordnet[57]. Es wird daher zunächst sichergestellt, dass Leser und Autor mit den Begriffen den gleichen Inhalt verbinden.

Bestes Beispiel hierfür ist der Begriff der Schnittstelle selbst. Um nur einige Bezeichnungen zu nennen, die auf das Gleiche verweisen können: Mensch-Maschine-Schnittstelle (MMS), Be-

[57] Vgl. Voss 1998, S. 2

nutzerschnittstelle, Benutzungsschnittstelle, Benutzeroberfläche, Mensch-Maschine-System (MMS), Mensch-Maschine-Interaktion (MMI) oder Mensch-Computer-Schnittstelle (MCS). Hinzukommen die gängigeren englischen Pendants wie Human Machine Interface (HMI), Man Machine Interface (MMI), User Interface (UI), Graphical User Interface (GUI) oder Man Computer Interface (MCI).

Die Grafik zeigt, dass auf der abstrakten Ebene (links) Mensch und Maschine durch eine Schnittstelle getrennt sind. Der Überbegriff Mensch-Maschine-Schnittstelle (MMS) erscheint in diesem Kontext deshalb am besten geeignet. Synonym wird der englische Begriff Human Machine Interface (HMI) verwendet.

Im eigentlichen Sinn geht man beim Menschen von einem Benutzer aus, der beispielsweise eine Fertigungsanlage oder ein Fließband bedienen will. Er bedient sich dafür einer Benutzerschnittstelle. Dies führt zunächst zu der einfachen Erklärung, dass jede Maschine, mit der ein Mensch kommunizieren will, eine Benutzerschnittstelle benötigt[58]. Implizit wird damit ausgedrückt, dass durch diese Schnittstelle ein bidirektionaler (hin und zurück) Informationsfluss ermöglicht wird[59]. Dies wird durch die Pfeile in Abbildung 2.1 simuliert.

Für die Zwecke dieser Arbeit muss die Schnittstelle selbst noch detaillierter betrachtet werden. Als Komponenten umfasst eine Benutzerschnittstelle Hardware und Software[60].

Die Hardware umfasst den physischen Teil. Bei einer Leitwarte in einer Fabrik wären dies das Anzeigepult mit Schaltern bzw. Tastatur, der Rechner selbst und dessen Verkabelung. Im Bereich der Diagnosetechnik wird an dieser Stelle von Endgeräten gesprochen werden. Das Endgerät kann z. B. ein Handterminal oder eine fest installierte Diagnosestation mit Tower-PC und Monitor sein. Zugehörige Verbindungskabel zum Fahrzeug können z. B. OBD-Stecker sein. In der Diagnosetechnik verbreiten sich hierfür neben OBD-Kabeln auch Funkadapter. Die Kommunikation der Steuergeräte bzw. deren Informationsaustausch untereinander erfolgt gängigerweise über K-Line bzw. CAN-Bussysteme (siehe Kapitel 2.1.4).

Tiefere technische Kenntnisse über die Hardware sind für die Arbeit nicht vonnöten. Entscheidende Aufmerksamkeit kommt hingegen dem Softwareteil der Schnittstelle zu.

Als Diagnosesoftware wird die Summe aller Softwareschichten bezeichnet, die für die Schnittstellenfunktionalität relevant sind. Die Software ist auf dem Endgerät installiert. Bei der Kombination von Hardware und Software wird in der Diagnosetechnik vom Diagnosewerkzeug oder Diagnosesystem gesprochen.

Die Aufteilung der hier dargestellten Schichten des Diagnosesystems orientiert sich an einer Standardisierung durch den Kfz-Standardisierungsverein ASAM[61].

Die oberste Schicht der Software aus Benutzersicht nimmt die Oberfläche ein. Die Oberfläche bezeichnet den nach außen sichtbaren Anteil der Benutzerschnittstelle[62]. Egal ob ein Kfz-Steuergerät geprüft oder ein Fließband gestartet/gestoppt werden soll, der Benutzer benötigt eine Oberfläche, um der Maschine den Befehl mitzuteilen, im umgangssprachlichen Gebrauch, auch als User Interface oder GUI (Graphical User Interface) bezeichnet. Sie stellt für den Benutzer die optische Möglichkeit dar, auf die Funktionalität der Maschinen zuzugreifen. Hierbei

[58] Vgl. Langmann 1994, S. 1
[59] Vgl. Langmann 1994, S. 2
[60] Vgl. Triebe 1996, S. 11
[61] Vgl. Schäuffele 2003, S. 32
[62] Vgl. Voss 1998, S. 2

besteht das Problem, eine für den Benutzer verständliche Abbildung der Funktionalität zu schaffen. Die Oberfläche ist folglich ein wesentlicher Teil der Schnittstelle zwischen Mensch und Maschine.

Im Fall der Diagnosetechnik wird bei der Oberfläche von einer Diagnoseoberfläche gesprochen. Der Softwareteil der Diagnoseoberfläche liefert hier zumeist nur den Rahmen für die Anzeige. Erst in Verbindung mit der Diagnosemaske (ladbaren Datei) wird für den Diagnostiker eine funktionelle Diagnoseoberfläche dargestellt. Zu vergleichen ist sie mit dem Schreibprogramm Word, das den Rahmen bietet, um verschiedene Dokumente (.doc-Dateien) anzuzeigen. Der eigentliche Inhalt über die Funktionalität steckt in der Diagnosemaske wie der Text in der Word-Datei.

Um den Informationsfluss zwischen Mensch und Maschine anzustoßen und zu erhalten, muss die Diagnosesoftware Dynamiken und Interaktion unterstützen. Diesen Teil übernimmt größtenteils die Logik-Schicht. Teilweise teilt sie sich diese Aufgabe mit der Oberfläche.

Bei der Diagnosesoftware steht die Interaktion im Vordergrund. Die Interaktion beschreibt im Allgemeinen "die wechselseitige Beeinflussung zweier, voneinander weitgehend unabhängiger (…) Funktionseinheiten"[63]. Bei der HMI-Interaktion geht die Beeinflussung weitgehend von der Einheit Mensch aus[64].

Dies bedeutet, dass z. B. ein Diagnostiker, der die Funktionalität "rechtes Fernlicht" untersuchen will, den zugehörigen Knopf auf der Diagnosemaske drückt. Die Logik-Schicht der Diagnosesoftware reagiert nun und gibt den Befehl an das Fahrzeug weiter. Das Licht wird aktiviert – der Zustand des Lichts (des Fahrzeugs) wurde beeinflusst.

Die Dynamik steht in der Diagnosesoftware nun dafür ein, dass der geänderte Status des Lichts auch auf der Diagnoseoberfläche in der Diagnosemaske angezeigt wird. Dynamische oder aktive Oberflächen sind folglich Darstellungen die "ihre Gestalt ändern können"[65], in der Diagnosetechnik eben dem Fahrzeugstatus entsprechend.

Die Oberfläche und Logik sind meistens in einer Anwendung verankert. Diese Kombination wird allgemein als Anwendersoftware bezeichnet[66]. In dieser Arbeit wird hierfür der in der Fertigungsindustrie gängige Begriff HMI-Software verwendet. Die HMI-Software dient der Darstellung der Diagnosemasken. Die Diagnosemasken sind in Dateien (hier: Diagnoseskripten) gespeichert. Die Diagnoseskripte werden durch die HMI-Software geladen. Nach dem Laden kann die HMI-Software die Inhalte der Diagnosemasken darstellen und dem Diagnostiker eine funktionale Diagnoseoberfläche bieten.

Neben der HMI-Software vervollständigt das Laufzeitsystem (die Plattformsoftware) die Diagnosesoftware. Die Plattformsoftware übernimmt die eigentliche Kommunikation mit der Maschine. Wie in Kapitel 2.1.4 bereits erwähnt, ist dieser Softwareteil von der Anwendersoftware (HMI-Software) getrennt. Daraus ergibt sich für die Anwendersoftware der Vorteil, dass sie grundsätzlich von der Maschine bzw. dem Steuergerät unabhängig entwickelt werden kann[67]. Es muss demnach bei der Entwicklung von HMI-Software nicht auf die verschiedenen Protokolle und Bussystemtypen der Steuergeräte Rücksicht genommen werden. Die Plattformsoft-

[63] Charwat 1992, S. 227
[64] Vgl. Charwat 1992, S. 228
[65] Langmann 1994, S. 110
[66] Vgl. Schäuffele 2003, S. 33
[67] Vgl. Schäuffele 2003, S. 33

ware tritt zu dem Zeitpunkt in Aktion, sobald die HMI-Software läuft und mit dem Steuergerät kommunizieren soll. Deshalb auch die Bezeichnung Laufzeitsystem. Im der Umgangssprache wird auch die nicht ganz korrekte Bezeichnung Treiber benutzt. Der Treiber ist aber vielmehr nur ein Teil des Laufzeitsystems.

In der Diagnosetechnik ist das MCD-System von ASAM ein Beispiel für ein solches Laufzeitsystem. In der Kfz-Branche ist für das Laufzeitsystem der Begriff Basissystem geläufig.

Durch die Kommunikation des Basissystems mit dem Steuergerät über die entsprechende Hardware schließt sich die Verbindung zwischen Mensch und Maschine.

Erreichen die Befehle oder Abfragen des Basissystems das Steuergerät, führt dieses entweder den Befehl aus oder liest bei einer Abfrage den Wert aus. Das Auslesen von Fahrzeugdaten erfolgt über Sensoren und das Wirken auf das Fahrzeug, im Zuge der Ausführung von Steuerbefehlen, erfolgt über Aktoren (siehe Kapitel 2.1.3). Damit knüpft die Erläuterung über das Zusammenspiel der HMI-Komponenten in der Diagnosetechnik an die Beschreibung der Funktionsweise und Kommunikation von Steuergeräten aus dem vorherigen Kapitel an.

3 Rahmenbedingungen beim Kooperationspartner

In diesem Kapitel wird die allgemeine Beschreibung der Kfz-Diagnosetechnik auf die Technologien des Kooperationspartners übertragen (Kapitel 3.1).

Hierbei werden speziell die beim Kooperationspartner im Einsatz befindlichen IT-Komponenten der Diagnosesoftware erklärt. Das Kapitel 3.2 beschäftigt sich mit dem Basissystem EDIABAS und das Kapitel 3.3 mit der HMI-Software INPA.

3.1 Diagnose elektronischer Steuergeräte beim Kooperationspartner

Mitte der 80er Jahre wurden die ersten elektronischen Systeme beim Kooperationspartner eingesetzt. Sie wurden über die Jahre durch Bussysteme (siehe Kapitel 2.1.4) und Computerchips ergänzt. Mit dieser Kombination der elektronischen Systemen mit Software erhielten die Mechatronik und folglich die Steuergeräte Einzug beim Kfz-Hersteller (siehe Kapitel 2.1.3). Erster Anwendungsbereich war die Heizungsregelung.

Bis heute setzt sich der Anstieg der mechatronischen Komponenten und deren Komplexität jedes Jahr weiter fort. Mitte des letzten Jahrzehnts führte der Kooperationspartner deshalb ein eigenes Basissystem zur Diagnose (EDIABAS) und eine speziell dafür entwickelte HMI-Software (INPA) ein. Diese beiden Komponenten der Diagnosesoftware (siehe Kapitel 2.2) befinden sich bis heute im Einsatz.

In diesem Kapitel wird erklärt, wie sich die beiden proprietären Komponenten der Diagnosesoftware (EDIABAS und INPA) beim Kooperationspartner in das allgemeine Kfz-Diagnosetechnikschema eingliedern. Für die Erläuterung wird auf das Grundlagekapitel 2.2 Bezug genommen. Folgende Abbildung zeigt die Übertragung der Kfz-Diagnosetechnik aus Kapitel 2.2 in die Begrifflichkeit der Systeme des Kfz-Herstellers.

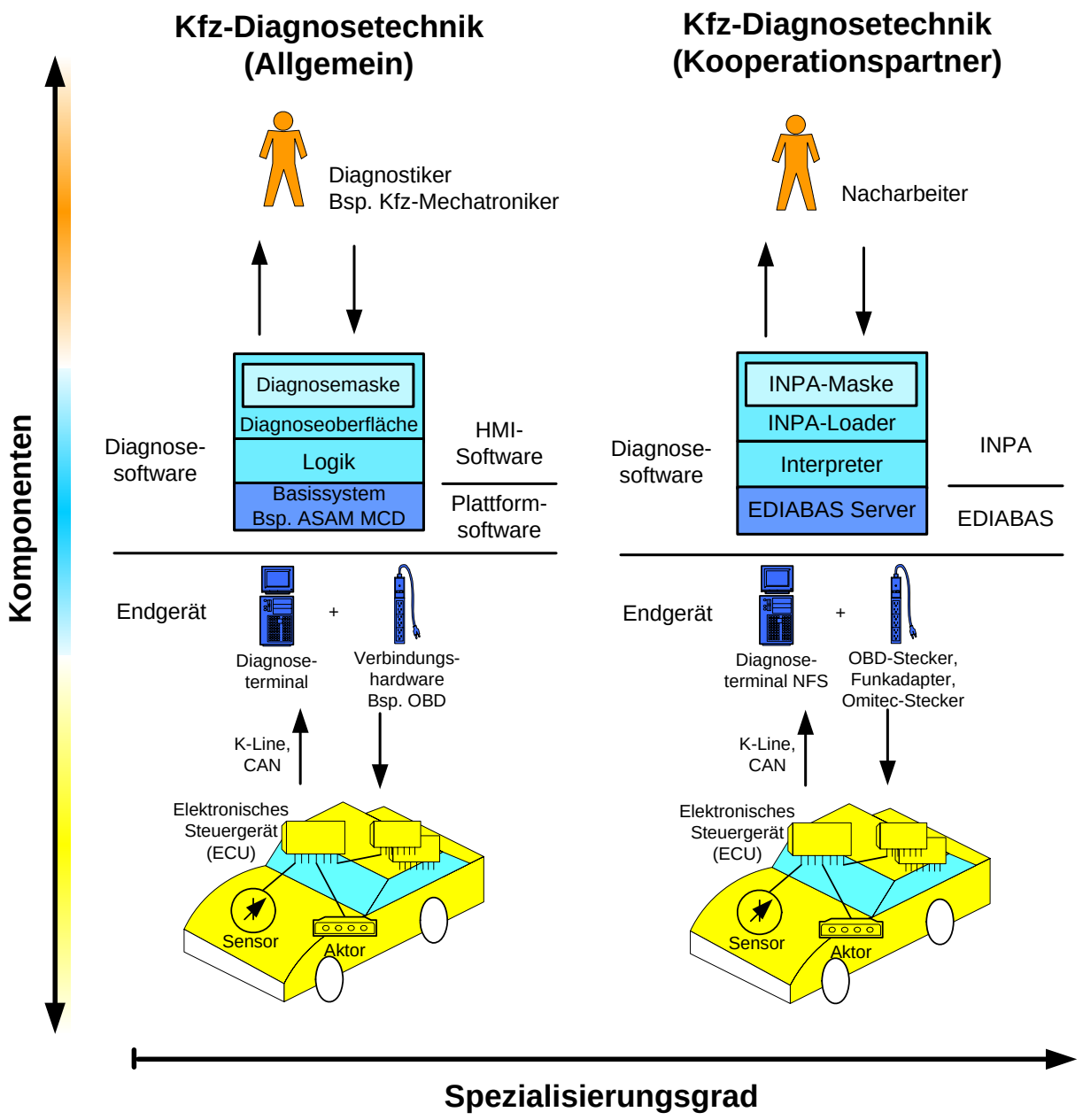


Abbildung 3.1 Kfz-Diagnosetechnik: Allgemein und spezifisch für Kooperationspartner

Im Hardwarebereich der Diagnosetechnik kommen beim Kooperationspartner neben OBD-Kabel noch Funkadapter oder Omitec-Stecker zum Einsatz. Die Kommunikation der Steuergeräte bzw. deren Informationsaustausch untereinander erfolgt vorwiegend über K-Line- bzw. CAN-Bussysteme (siehe Kapitel 2.1.4).

Beim Kooperationspartner ist das Konzept der Diagnosesoftware in die HMI-Software INPA und das Basissystem EDIABAS unterteilt.

Die Komplexität der Fahrzeuge und insbesondere die starke Zunahme der Anzahle Steuergeräte machte ein neues Basissystem für die Diagnose erforderlich. Früher waren die Kommunikationsfunktionen innerhalb der HMI-Software verankert und auf einen Steuergerätetyp zugeschnitten. Dies schränkte die Einsatzfähigkeit der HMI-Software stark ein. Deshalb wurde die

Programmierung der Kommunikationsfunktionen durch eine einheitliche ausgelagerte Kommunikationsbasis mit dem Name EDIABAS ersetzt.

Der Name EDIABAS steht für *E*lektronik *Dia*gnose *Bas*issystem. Dieses Konzept wurde als Steuergeräte-Kommunikationsbasis entwickelt. Seit 1992 wurde es schrittweise bei allen auf Diagnose basierenden Prüf-, Codiersystemen und HMI-Systemen in der Fertigung des Kooperationspartners eingesetzt. Mit einer genaueren technischen Erläuterung dieser Komponente befasst sich das folgende Kapitel.

Etwa im gleichen Jahr wie EDIABAS wurde auch die HMI-Software INPA proprietär für den Kfz-Hersteller entwickelt. Die Software besteht vorrangig aus dem INPA-Loader und dem INPA-Interpreter. Der Interpreter ist in die Loader-Anwendung integriert. Allerdings erst in Verbindung mit den INPA-Masken bietet der INPA-Loader eine funktionale Diagnoseoberfläche. Dieses Zusammenspiel wird in Kapitel 3.3 näher beschrieben.

3.2 Diagnosebasissystem

3.2.1 Hintergrund

In der Diagnosesoftware ist vorzugsweise das Laufzeitsystem von der Anwendersoftware (HMI-Software) getrennt (siehe Kapitel 2.2). Diese Trennung hat mehrere Gründe.

Neben der stetig steigenden Anzahl an Steuergeräten im Fahrzeug nehmen auch das System und die Vielfalt von deren Einsatzbereichen zu. In Kapitel 2.1.4 wurde erläutert, dass in den verschiedenen Einsatzbereichen unterschiedliche Anforderungen an das Kommunikationssystem gestellt werden. Diese Anforderungsvielfalt führt zu einem breiten Spektrum an verwendeten Protokollen und Bussystemen. Dieser Zustand wird noch verstärkt durch den Bezug der Steuergeräte von verschiedenen Herstellern. Durch das Laufzeitsystem soll eine einheitliche Basis für die HMI-Software geschaffen werden, sodass sie trotz verschiedener Protokolle, auf die Steuergeräte zugreifen kann.

Ein weiterer Grund für die Trennung ist der vielfältige Zugriff auf ein Steuergerät während seines Lebenszyklus. Der Zyklus umschließt z. B. Labortests während der Entwicklung, die Abfrage von Daten in Versuchsfahrzeugen, die Prüfung nach dem serienmäßigen Einbau, die Diagnose bei aufgetretenen Fehlern oder den Austausch bzw. die Neuprogrammierung im Service nach dem Verkauf. Durch ein abgekoppeltes Laufzeitsystem wird vermieden, dass jede HMI-Software aus den einzelnen Bereichen die Kommunikation mit den Steuergeräten selbst implementieren muss. In allen Bereichen können die unterschiedlichen HMI-Softwareprodukte das gleiche Laufzeitsystem verwenden.

Ein dritter nennenswerter Grund für ein eigenständiges Laufzeitsystem ist die Aufbereitung der Daten. Steuergeräte liefern bei Abfragen verschiedene Wertetypen (Zahlen, boolesche Werte, Zeichenketten etc.). Das Laufzeitsystem bietet den Komfort, die Werte bei der Abfrage bereits im richtigen Typ zu liefern, um sie direkt im Programmcode der HMI-Software verarbeiten zu können. In der Erläuterung des ASAM-Basissystems wird verdeutlicht, dass der HMI-Software

die "Unannehmlichkeit" genommen wird, in einer für sie "untypischen Sprache" kommunizieren zu müssen[68].

Beim Kooperationspartner wird als Laufzeitsystem eine proprietäre Entwicklung der Softing AG mit dem Namen EDIABAS eingesetzt. Die Struktur und Architektur, die sich aufgrund der Softwaretrennung für EDIABAS ergeben, wird im Folgenden beschrieben.

3.2.2 Funktionsweise

EDIABAS stellt ein "Basissystem zur Kommunikation mit Steuergeräten" dar[69].

Um die in Kapitel 3.1.1 beschriebene Flexibilität zu garantieren, darf das System weder fest mit der HMI-Software noch mit dem Steuergerät verankert sein. Es stellt vielmehr einen frei gekoppelten "Dienst" zur Verfügung, der die Daten zwischen Steuergerät und Oberflächenanwendung austauscht.

Der Dienst kann von der HMI-Software aufgerufen werden. Das Ergebnis dieses Dienstes kann nach dessen Abarbeitung von der HMI-Software abgefragt werden. Die HMI-Software in ihrer Funktion als Client tauscht mit dem Basissystem Daten aus. Zu vergleichen ist diese Kommunikationsweise mit der von Internet-Browsern (Clients), die auf einem Server Seiten aufrufen. Mit dem Basissystem als Server ist EDIABAS folglich wie eine Art Client/Server-Architektur konzipiert.

Da der Begriff Client/Server-Architektur in diesem Zusammenhang jedoch äußerst missverständlich wäre, wurde der Begriff Basissystem eingeführt. Die Abbildung 3.2 stellt die beschriebene Funktionsweise graphisch dar. Es wird auf die Trennung der Softwareschichten verwiesen, die bereits in Kapitel 2.2 thematisiert wurde (Farbwahl aus Abbildung 2.5 und 3.1).

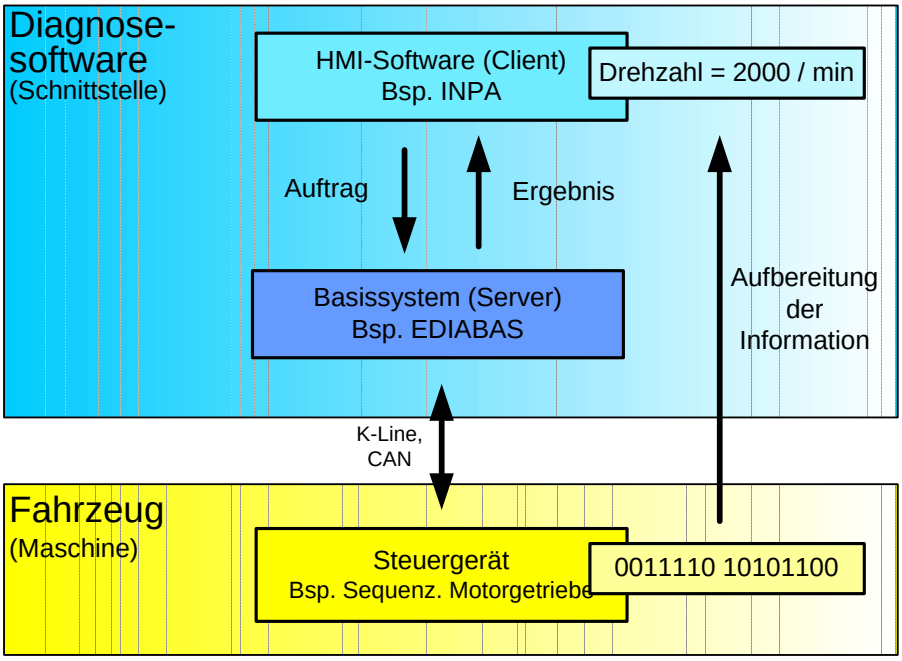


Abbildung 3.2 Arbeitsweise EDIABAS, EDIABAS USER 2003, S. 12

[68] Wenzel 2001, S. 6

Anhand des Beispiels aus in Abbildung wird die Funktionsweise nochmals verdeutlicht.

Die Diagnosesoftware, genauer betrachtet die HMI-Software, soll die Anzahl gewisser Umdrehungen (z. B. des Antriebsstrangs des Motors) anzeigen. Hierfür ist ein Befehl definiert, der in der Diagnosemaske verankert ist. Der Befehl wird als Auftrag an das Basissystem weitergegeben. Der Client (HMI-Software) nützt den Dienst, den der EDIABAS-Server dafür zur Verfügung stellt.

Das Basissystem wandelt unter Zuhilfenahme entsprechender Parametrierungsinformationen den Befehl in binären Code (Maschinencode) um, sodass das Steuergerät (ECU) diesen einlesen und ausführen kann.

Über das vom Steuergerät unterstützte Protokoll und Bussystem übermittelt das Basissystem den Maschinencode an das Steuergerät. Im Beispiel wird die Anfrage nach Motordrehzahl im Schaltgetriebe an das sequenzielle Motorgetriebe-Steuergerät (SMG) gestellt. Über den entsprechenden Sensor fragt das Steuergerät die Drehanzahl ab und schickt diese an den EDIABAS-Server zurück.

Das Basissystem wandelt die Antwort des Steuergeräts in eine für die HMI-Software lesbare Sprache um. Somit kann die HMI-Software dem Benutzer das Ergebnis auf seiner Diagnoseabfrage darstellen. Das dargestellte Ergebnis ist eine Umdrehungszahl von 2000 U/min. Auf diesem Funktionsprinzip beruhen die meisten Basissysteme.

3.2.3 Architektur

Um die Hintergründe der Verarbeitung bei EDIABAS besser zu verstehen, wird in diesem Kapitel auf die Architektur eingegangen. Sie wird zunächst in Abbildung 3.3 dargestellt:

[69] EDIABAS User 2004, S. 11

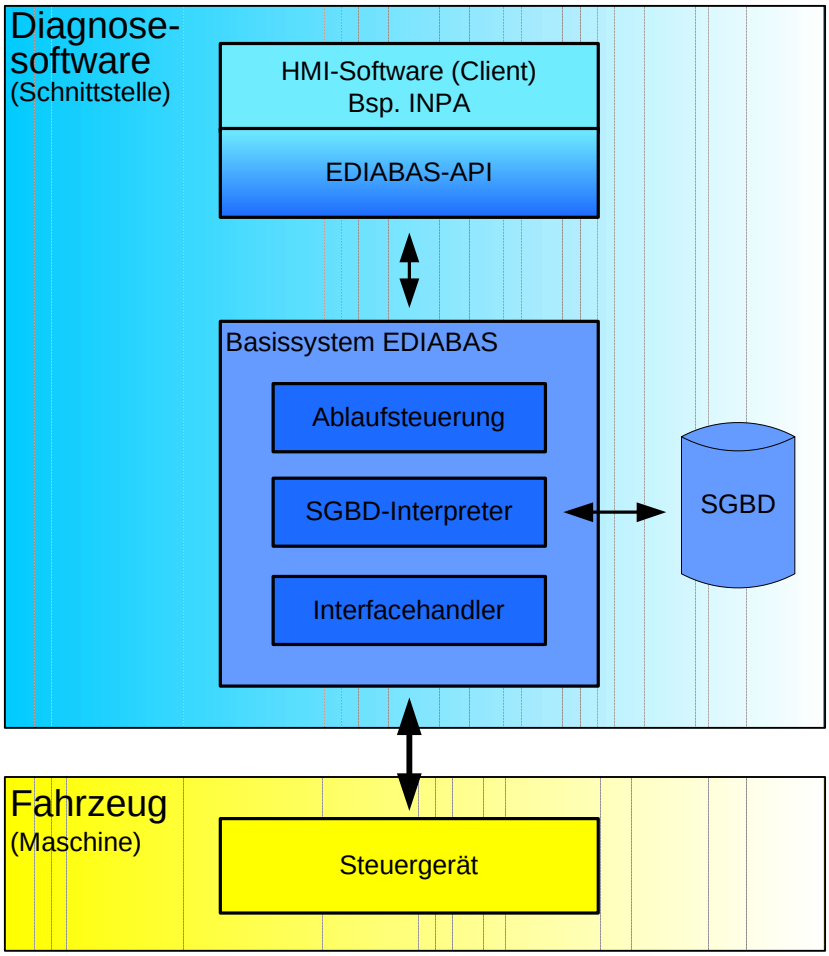


Abbildung 3.3 EDIABAS-Struktur, EDIABAS USER 2003, S. 12

Wie einleitend erläutert, ist der EDIABAS-Server nicht auf eine spezifische HMI-Software oder einen Steuergerätetyp beschränkt. Er agiert von beiden Komponenten lose gekoppelt.

Dies ist möglich, da der Server sein Wissen über die einzelnen Steuergeräte in austauschbaren Datenmodulen speichert. Diese sogenannten "*S*teuer*g*eräte *B*eschreiungs*d*ateien" (SGBDen) enthalten das benötigte Wissen, um mit den Steuergeräten kommunizieren zu können. Dies umfasst vor allem das Wissen bezüglich der Übersetzung der binären Steuergerätedaten in symbolische für die HMI-Software verarbeitungsfähige Daten.

Die SGBDen sind also vom Server losgelöst und werden deshalb in der Abbildung als externe Bibliothek dargestellt. Zu verstehen sind sie als Binärdateien, die zur Laufzeit geladen werden, sobald auf das Steuergerät zugegriffen wird. Der Dateiname jeder SGBD (ohne Endung) ist der Name, mit dem die HMI-Software ein Steuergerät referenziert. Er verweist auf die Kurzbezeichnung des Steuergeräts und die Baureihe. Im Beispiel von Abbildung 3.2 trägt die SGBD für das sequenzielle Motorgetriebe und die Baureihe E60 den Namen SMG_60.

Um die SGBD zu verarbeiten und mit dem Steuergerät und der HMI-Software zu kommunizieren, ist EDIABAS in drei wesentliche Bestandteile gegliedert:

- Ablaufsteuerung (Kernel)
- SGBD-Interpreter
- Interfacehandler (IFH)

Die Ablaufsteuerung bestimmt das Verhalten des Gesamtsystems und beinhaltet die Kommunikationsschnittstelle zur HMI-Software. Sie stellt softwaretechnisch die Verarbeitungslogik dar.

Der SGBD-Interpreter liest die SGBD ein und übernimmt die eigentliche Umsetzung der vom Steuergerät erhaltenen Binärdaten in symbolische Daten. Er interpretiert die in den SGBDen enthaltenen Daten, Methoden und Abläufe, initiiert die Kommunikation mit dem Steuergerät und liefert die gewonnenen Ergebnisse zur Weiterleitung an die HMI-Software zurück.

Der Interfacehandler (IFH) ist ein austauschbares Softwaremodul, das je nach verwendeter Verbindungshardware (z. B. OBD-Stecker, siehe Kapitel 2.2) in verschiedenen Versionen eingebunden werden kann. Eine Anfrage an das Steuergerät wird vom IFH entsprechend der verwendeten Verbindungshardware umgesetzt. Die vom Steuergerät zurückgelieferten Daten werden vom IFH ausgewertet und an die Ablaufsteuerung (bzw. HMI-Software) weitergegeben. Der Interfacehandler bietet außerdem die Möglichkeit der Simulation von Steuergerätedaten. Somit kann die auf EDIABAS basierende HMI-Software auch dann getestet werden, wenn die verwendeten Steuergeräte nicht vorhanden sind.

Um auf den EDIABAS-Server zugreifen zu können, wird eine Softwareschnittstelle zur Verfügung gestellt. Diese ist betriebssystemabhängig und muss in die HMI-Software integriert werden.

Dieses sogenannte »Application Programmers Interface« (EDIABAS-API) übernimmt die systemabhängige Kommunikation zum Basissystem. Ist die EDIABAS-API in die HMI-Software integriert, kann man mittels dieser Schnittstelle auf das Steuergerät zugreifen. Hierfür müssen, wie erwähnt, Aufträge (sogenannte Jobs) abgeschickt werden.

3.2.4 Konzept der Befehlsverarbeitung

Ein Job (dt. Steuergerätebefehl) ist ein in sich abgeschlossener Auftrag. Mit einem Job können Daten vom Steuergerät ausgelesen und zum Steuergerät geschickt werden. Er kann ohne Berücksichtigung der Auftragsreihenfolge aufgerufen werden (Grundsatz der Abgeschlossenheit). Der Job liefert nach Bearbeitungsende Ergebnisse, die von der HMI-Software verwendet werden können (Results). Die Jobs sind nicht fest in EDIABAS verankert, sondern Bestandteil der SGDB. Sie können demnach flexibel geladen werden.

Der Aufruf eines Jobs geschieht in der HMI-Software über die EDIABAS-API-Funktionen apiJob, apiJobData oder apiJobExt. Diese Funktionen benötigen mindestens zwei Angaben: das anzusprechende Steuergerät bzw. die Steuergerätegruppe und den Job, der ausgeführt werden soll.

In Kapitel 2.3.2 wurde das Beispiel vorgestellt, die Motordrehzahl auszulesen und darzustellen. Übertragen auf dieses Beispiel muss in der Diagnosemaske der apiJob mit den Parametern SMG_60 (für das Steuergerät) und "DREHZAHL_LESEN" (für den Jobnamen) ausgeführt werden. Diese Aufrufverarbeitung stellt folgende Abbildung dar:

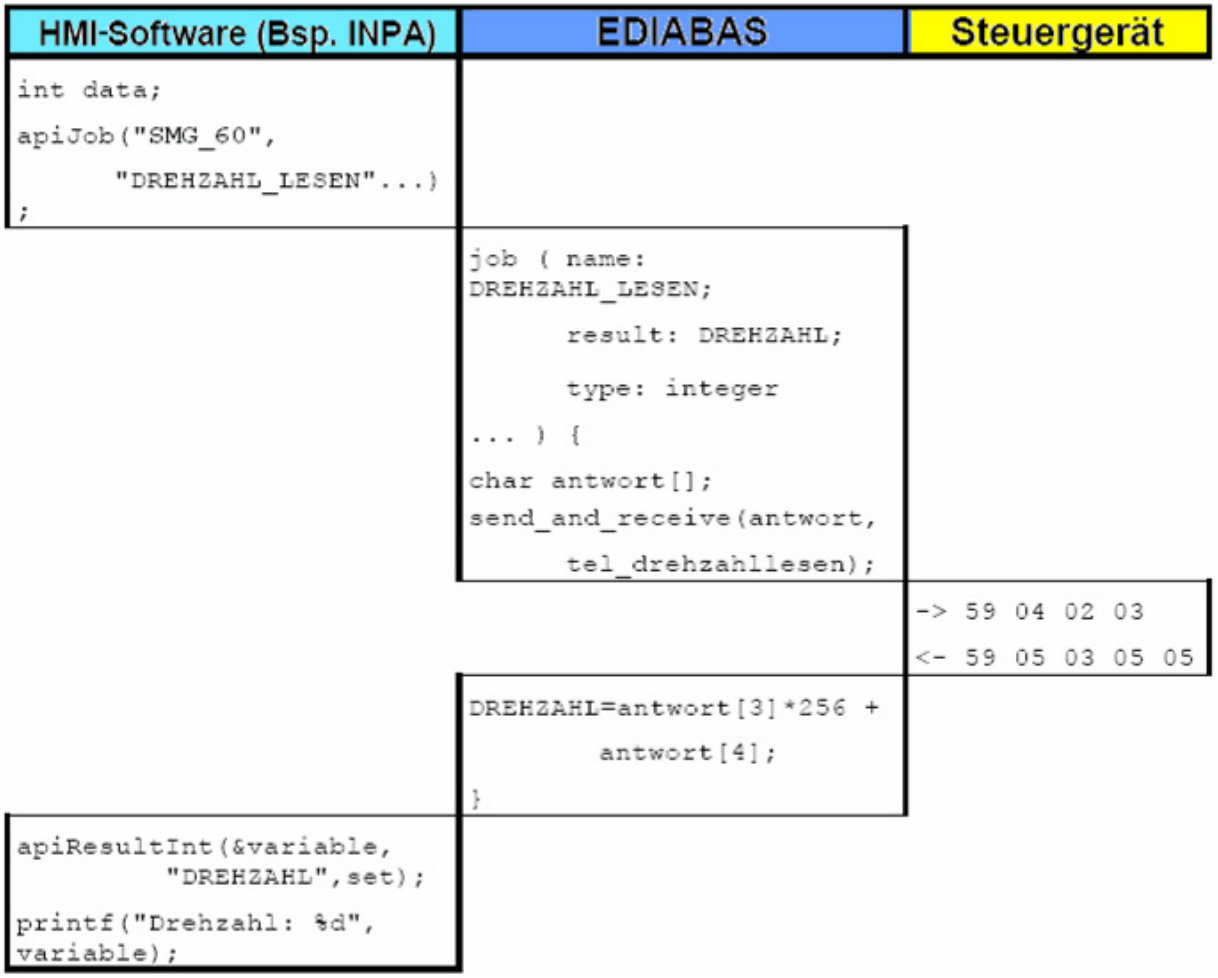


Abbildung 3.4 Quellcode: Verarbeitungsschema Aufruf des Steuergerätebefehls[70]

Nach dem Absetzen des Jobs wird der Auftrag in die für das Steuergerät verständlichen Binärdaten umwandelt und über den Interfacehandler übergeben. Das Ergebnis wird in symbolischer (lesbarer) Form der HMI-Software bereitgestellt (siehe Kapitel 3.2.3).

In der HMI-Software können diese Ergebnisse dann mit der Funktion apiResult in verschiedenen Formaten ausgelesen werden (Zahlen, boolesche Werte, Zeichenketten etc.). So kann z. B. das Ergebnis der "Drehzahl", die Zahl 2000 als Integer von der HMI-Software angefordert werden.

Wie die HMI-Software INPA die Antworten des EDIABAS-Servers verarbeitet und an den Benutzer weitergibt, wird im anschließenden Kapitel 2.4 beschrieben.

[70] EDIABAS USER 2004, S. 20

3.3 HMI-Software

Bei der Einführung von neuen Steuergeräten oder Steuergerätevarianten in der Fertigung muss für die Behandlung von Problemfällen eine Software zur Diagnose des Steuergeräts bereitgestellt werden. Dies ist unter anderem in den Bereichen Nacharbeit, Analyse oder Entwicklung notwendig. Vor INPA geschah dies beim Kooperationspartner durch eine Erweiterung einer Diagnosesoftware, die für den Einsatz im Kundendienst (Bereich Service) entwickelt wurde.

Um auf die Anforderungen im Werk flexibler und vor allem schneller reagieren zu können, wurde ein eigenständiges Prüfablaufsystem entwickelt. Diese Software trägt den Namen INPA (*In*terpreter für *P*rüf*a*bläufe) und ist entsprechend dem Verständnis der Arbeit, nach Kapitel 2.2 vom Typ her eine HMI-Software.

3.3.1 Funktionsweise und Architektur

Wie in Kapitel 2.2 beschrieben, sind die Diagnosemasken bei der HMI-Software nicht hart codiert, sondern in ladbare Dateien ausgelagert. Die Dateien werden als Diagnoseskripte bzw. INPA-Skripte bezeichnet. Dadurch ist es INPA möglich, für jedes Steuergerät eine individuelle Diagnoseoberfläche bereitzustellen. Die Diagnosemasken werden spezifisch für jedes Steuergerät erstellt.

Um die Masken bzw. Skripte zu erstellen, wurde für INPA eine proprietäre, C-ähnliche Sprache mit dem Namen PABS (Prüfablauf-Beschreibungssprache) entwickelt. In dieser Sprache werden die Quellen für die späteren Masken erstellt. Die Quelldatei mit der Endung SRC wird in der Abbildung 3.5 als INPA-Skriptquelle bzw. als Diagnoseskriptquelle bezeichnet.

Diese Quellen enthalten unter anderem Codefragmente zum Aufruf von Jobs (Steuergerätebefehle). Ein solches Fragment kann in Abbildung 3.4 betrachtet werden. Deshalb ist beim Erstellen einer Diagnosemaske der Zugriff auf die Steuergerätedaten essenziell. Es müssen z. B. der Jobname, Parameter, Ergebnistypen etc. ausgewertet werden. Die Steuergerätedaten sind in den SGBDen gespeichert (siehe Kapitel 3.2.3).

Mit dem PABS-Parser werden die Skriptquellen kompiliert. Die kompilierten Skriptquellen werden als INPA-Skript oder Diagnoseskript bezeichnet. Die Datei trägt die Endung IPO. Als Zwischenprodukt der Kompilierung erzeugt der Parser eine IPS-Datei. Sie ist nicht ausführbar und erfüllt keinen funktionalen Zweck für die Diagnosesoftware. Sie wird deshalb im Diagramm ausgespart und erscheint nur beiläufig in den Erläuterungen der Arbeit.

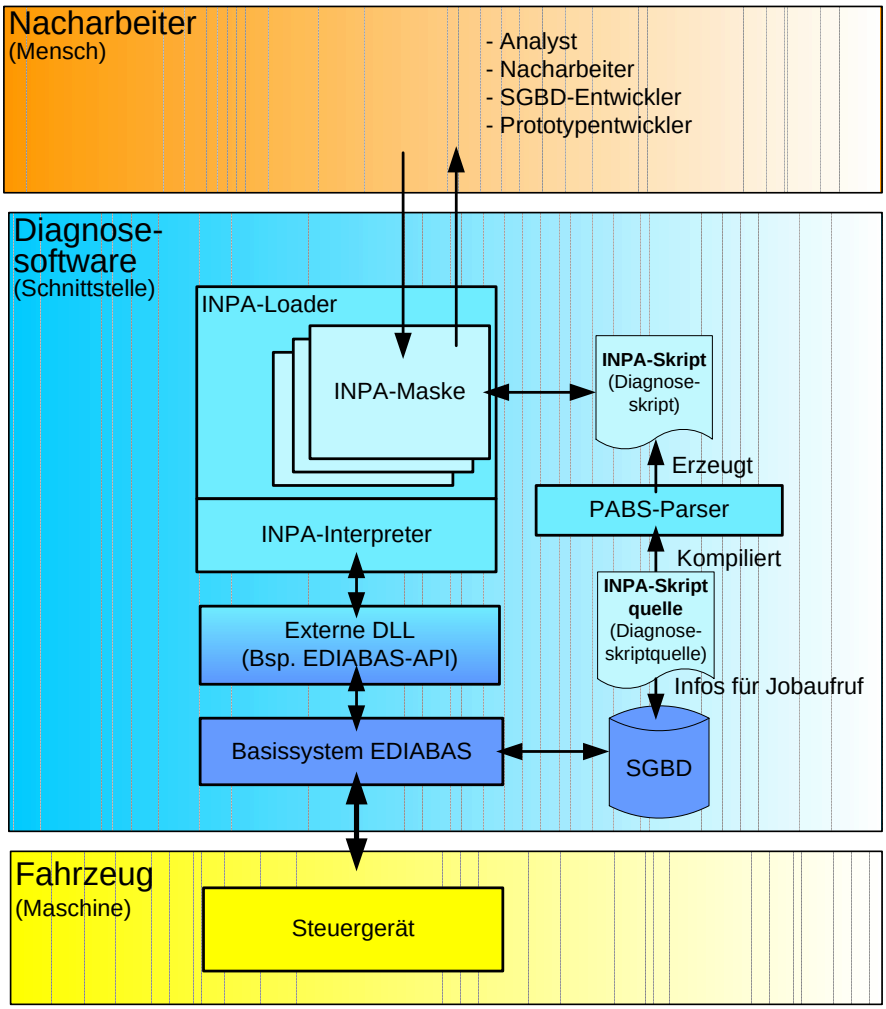


Abbildung 3.5 Komponenten von INPA, INPA 2003, S. 9

Die Diagnoseskripte (IPO-Dateien) sind eigenständig nicht lauffähig, sondern benötigen eine entsprechende Laufzeitumgebung. Diese wird mit dem INPA-Loader bereitgestellt. Innerhalb dieser Umgebung können die Diagnoseskripte geladen werden. Der INPA-Loader ist eine für Windows XP und NT entwickelte ausführbare Anwendung.

Mit dem Laden eines Diagnoseskripts im INPA-Loader werden dem Diagnostiker die verschiedenen Diagnosemasken (INPA-Masken) am Monitor angezeigt. Daraufhin kann er die Diagnosearbeit beginnen. Wie in der Abbildung 3.5 dargestellt, kann ein Diagnoseskript mehrere Masken enthalten. Die Navigation in den Masken und deren Bedienung erfolgt vorrangig über die Tastatur.

Der Interpreter verarbeitet intern den kompilierten Maschinencode der IPO-Datei. Er verwaltet softwaretechnisch die Objekte und ermöglicht die Anbindung externer DLLs. Als externe DLL ist die EDIABAS-API von entscheidender Bedeutung, um mit dem EDIABAS-Server zu kommunizieren. Der Zugriff auf das Basissystem über die HMI-Software wird im Verlauf der Arbeit noch vertieft.

3.3.2 Begriffserläuterung

Für die in dieser Arbeit verwendeten Begriffe ist zu erwähnen, dass sie auf Bezeichnungen beruhen, die sich beim Kooperationspartner etabliert haben. Besonders für die Prozessmodellierung (Kapitel 5) ist es wichtig, die Begriffe exakt in dieser Weise zu benutzen. Dadurch bleiben die Modelle für die Mitarbeiter verständlich. Die Begriffe sind teilweise fachlich allerdings nicht ganz korrekt.

Der Begriff Skript ist an sich falsch, da PABS keine Skriptsprache, sondern eine Programmiersprache ist. Bei INPA wird zwar von einem Interpreter gesprochen, im Grunde wird aber der kompilierte Quellcode (die IPO-Datei) in einer Laufzeitumgebung ausgeführt und nicht interpretiert. Interpreter verarbeiten eigentlich direkt Programmcode in Textform und keine Kompilate in Maschinencode. Da der Begriff INPA-Skript und Skriptquelle fest verankert sind, sollen diese auch bei der Prozessbeschreibung verwendet werden.

In den Erläuterungen wird verallgemeinernd der Begriff Diagnoseskript für INPA-Skript bzw. IPO-Datei verwendet sowie Diagnoseskriptquelle für INPA-Skriptquelle. Die INPA-Begriffe finden sich aufgrund der Lesbarkeit in den abgebildeten Prozessmodellen (Kapitel 5-7) wieder (Begründung siehe oben). In den Erläuterungen der Dissertation werden die allgemeinen Begriffe verwendet.

Wie erwähnt, ist die Bezeichnung Diagnoseskript für INPA fragwürdig. Ein Skript ist ein (für Menschen) lesbarer Code, der nur interpretiert wird. Bei INPA ist dies allerdings kompilierter Code, nur die Diagnoseskriptquelle ist lesbar. Da INPA allerdings in verschiedenen Bereichen wie der SGBD-Entwicklung, Nacharbeit oder Analyse werksübergreifend zum Einsatz kommt, ist es notwendig, die Begriffe im gebräuchlichen Sinne zu verwenden, um eine gemeinsame Kommunikationsbasis zu wahren.

4 Abgrenzung und weiteres Vorgehen für die Erfassung und Analyse

In diesem Kapitel werden die Gründe dafür genannt, warum in der vorliegenden Arbeit die HMI-Software ausgewählt wurde, um die Abläufe der Diagnosetätigkeiten zu optimieren.

In Zusammenarbeit mit dem Kooperationspartner wurden verschiedene Ansatzpunkte erarbeitet. Die Ansatzpunkte ergeben sich aus den allgemeinen Herausforderungen, die der Wandel des Steuergerätemarkts mit sich bringt und denen sich auch der Kooperationspartner stellen muss. Die Ansatzpunkte wurden in den Bereichen Mobilität (Kapitel 4.2.1), flexible Funktionalität (Kapitel 4.2.2) und grafische Darstellung (Kapitel 4.2.3) identifiziert.

Anhand der Ansatzpunkte kann bestimmt werden, welche IT-Komponenten in der Kfz-Diagnosetechnik demzufolge von den neuen Herausforderungen betroffen sind. Die einzelnen IT-Komponenten, die im Bereich der Kfz-Diagnosetechnik zusammenspielen, wurden in Kapitel 2.2 definiert. Aufgrund der dort beschriebenen Komponenten grenzt Kapitel 4.3 die HMI-Software als zentralen Ausgangspunkt für Verbesserungen ab.

Da die Abgrenzung auf den Grundlagen der Diagnosetechnik basiert (Kapitel 2 und 3), werden die technischen Rahmenbedingungen im folgenden Unterkapitel 4.1 eingangs nochmals kurz aufgeführt.

Das Kapitel schließt mit einem Ablaufplan ab (Kapitel 4.4). Darin wird beschrieben, wie bei der Erfassung und Analyse weiter vorgegangen wird.

4.1 Rahmenbedingungen bei der Steuergerätediagnose

Aus den vorherigen Kapiteln lassen sich folgende Grundlagen für die Diagnose der Steuergeräte zusammenfassen:

Elektronische Steuergeräte nehmen im Kraftfahrzeug die verschiedensten Aufgaben wahr. Diese reichen von der Motorsteuerung, über die Fahrsicherheit bis hin zu Komfortfunktionen wie

beispielsweise der Speicherung der Sitzposition. Weitere Beispiele sind in Kapitel 2.1.2 zu finden. Je nach Aufgabenkomplexität sind die Steuergeräte miteinander vernetzt und kommunizieren auch miteinander. Um im Rahmen der Fehlersuche die Funktionsweise der Steuergeräte prüfen zu können, bedient sich der Diagnostiker eines Diagnosewerkzeugs oder Diagnosesystems. Dieses setzt sich aus Hardware (Endgerät) und Software (Diagnosesoftware) zusammen.

Die Diagnosesoftware ist in die zwei Bereiche Basisystem und Diagnosesoftware aufgeteilt. Das Basissystem übernimmt die direkte Kommunikation mit dem Steuergerät. Ein solches Basissystem ist z. B. das ASAM MCD-System. Damit der Diagnostiker gezielt eine Funktion des Steuergeräts ansprechen kann, muss die Diagnosesoftware, bestehend aus Oberfläche und Logikverarbeitung, eine entsprechende Oberfläche zur Verfügung stellen. Sie wurde als HMI-Software definiert.

Dieser Abriss der Diagnose von elektronischen Steuergeräten stellt die technische Ausgangssituation der Arbeit dar. Allerdings unterliegen diese technischen Rahmenbedingungen der Diagnose derzeit einem starken Wandel. Dieser Wandel resultiert vorrangig aus zwei Veränderungen der letzten Jahre: Zum einen aus der rasant steigenden Anzahl und Vielfalt an Funktionen der Steuergeräte und zum anderen aus deren Bedeutsamkeit für die Funktionstüchtigkeit des Kraftfahrzeugs. Beides sind Veränderungen, deren Intensität in den nächsten fünf bis zehn Jahren erwartungsgemäß zunehmen wird (siehe Kapitel 1.1).

Warum die Diagnosesoftware an diese veränderten Bedingungen angepasst werden muss, wurde im ersten Kapitel bereits erläutert. Hier wurden z. B. die erheblichen Kosten für Kfz-Rückholaktionen genannt.

Wie im Grundlagenkapitel verdeutlicht, ist die Diagnose von Steuergeräten eine technisch komplexe Aufgabe. Sie erfordert das Zusammenspiel von verschiedenen IT-Komponenten. Diese finden sich sowohl im Hardware- als auch im Softwarebereich wieder. Aufgrund der Komplexität und des Umfangs der Thematik erscheint es deshalb sinnvoll, zunächst die Komponenten zu identifizieren, in denen Verbesserungspotenzial vermutet wird.

In ersten Gesprächen mit dem Kooperationspartner wurden Verbesserungsvorschläge gesammelt. Aufgrund des Wandels auf dem Steuergerätemarkt und der Ausgangssituation beim Kooperationspartner wurden dabei die folgenden drei Bereiche identifiziert:

Diese Bereiche sind

- der Einsatz der Diagnosesoftware auf mobilen Endgeräten (Mobilität),
- ein flexibler parametrierbarer Funktionsumfang der Diagnoseoberfläche (Funktionalität) und
- die grafisch hochqualitative Darstellung der Diagnoseoberfläche (Grafik).

Diese drei Bereiche sollen die Ansatzpunkte bilden, um die Kfz-Diagnosetechnik beim Kooperationspartner gezielt zu verbessern. Im Folgenden wird kurz erläutert, warum die Praxis der Kfz-Steuergerätediagnose diese Herausforderungen hervorbringt.

4.2 Ansatzpunkte

4.2.1 Mobilität

Noch vor einigen Jahren war eine Reparatur des Fahrzeugs vor Ort durch den Pannenservice zumindest provisorisch mit gängigen Werkzeugen aus der Mechanik und Elektrik möglich. Heute ist für die Reparaturdienste selbst die Fehlereingrenzung ohne die entsprechende Diagnosesoftware nicht mehr möglich[71]. Die Diagnosesoftware für Steuergeräte findet ihren heutigen Einsatz also sowohl bei mobilen Pannendiensten als auch in Kfz-Werkstätten. Die Einsatzbereiche erstrecken sich unter anderem von der Steuergeräteentwicklung, Labortests, Versuchen in Testfahrzeugen und der Fertigung über die Endprüfung, den Funktionstest, der Nacharbeit und der Fehleranalyse bis hin zum Service bzw. der Fehlersuche im Rahmen des Kundendienstes, sowie dem Austausch und der Reparatur von Steuergeräten[72].

Als Folge des weiträumigen Einsatzes muss die Diagnosesoftware auf verschiedensten Typen von Endgeräten lauffähig sein.

Für das junge Anwendungsgebiet bei den Pannendiensten ist z. B. eine elementare Anforderung an das Diagnosegerät, dass es mobil und ortsunabhängig eingesetzt werden kann. Es muss folglich Software verwendet werden, die auch auf mobilen Geräten, wie z. B. Notebooks oder Tablet PCs, lauffähig ist.

Auch bei der Nacharbeit im Rahmen der Endmontage eröffnen sich neue Einsatzbereiche für mobile Diagnosegeräte. Wird bei der finalen Fertigungsprüfung eines Fahrzeugs ein Fehler festgestellt, besteht aufgrund der Produktionstaktzeiten keine Möglichkeit für eine intensive Diagnose. Deshalb muss das Fahrzeug an dieser Stelle aus dem Produktionszyklus "ausgeschleust" werden. Die Ausschleusung ist allerdings einen sehr zeitaufwendigen Prozess dar. Beim Kooperationspartner wurde während der Prozessdatenerhebung eine Mindestdauer von zwei Stunden erfasst, bis das Fahrzeug wieder vollständig in den Prüfprozess zurückgeschleust werden konnte (siehe Quelle INPA-Prozessdatenerfassung 2006, "Montagebericht, Nacharbeit"). Um sich die Kosten solcher Ausschleusungen zu ersparen, stellt sich z. B. für BMW die aktuelle Aufgabe, dass die "Diagnose (…) in allen Phasen der Montage durchführbar sein muss"[73]. Man will versuchen, den zeitintensiven Zyklus der Ausschleusung zu umgehen, indem direkt am Montageband nach dem Fehler gesucht wird. Aufgrund der permanenten Leistungssteigerung mobiler Handgeräte, wie Mini-Notebooks oder PDAs, ist man bestrebt im Gange, diese für die Diagnose zu verwenden. Neben der schnelleren Handlungsfähigkeit soll damit dem Diagnostiker bei seiner täglichen Arbeit auch mehr Bewegungsfreiheit am Steuergerät ermöglicht werden.

Allerdings bringt die Erweiterung der klassischen Endgeräte (fest montierte PC Anlage) um mobile Endgeräte auch eine Reihe neuer Anforderungen an die Diagnosesoftware mit sich. Durch die Vielfalt an Geräten ergibt sich eine breite Palette von unterschiedlichen Anzeigeflächen (Monitor). Vom 21-Zoll-Monitor der Diagnoseanlage bis hin zum handflächengroßen PDA-Display. Teilweise variieren die mobilen Geräte auch sehr stark in ihrer Anzeigefähig-

[71] Vgl. ZKF 2004, S. 8
[72] Vgl. Softing AG Pressestelle 2003, S. 2
[73] Hammerschmidt 2004 (Online-Quelle)

keit, z. B. in Auflösung oder Farbtiefe. Zudem kommen unter anderem die unterschiedlichen Betriebssysteme hinzu.

Als Folge daraus sind viele einsatz- und endgerätspezifische Lösungen von Diagnosesoftware entstanden[74].

Besonders für die Automobil-Großkonzerne bedeutet allerdings der Einsatz verschiedener Versionen der Diagnosesoftware einen enormen Mehraufwand in der Verwaltung.

An einer innovativen Diagnosesoftware wird somit erwartet, dass sie endgerätübergreifend eingesetzt werden kann. Dieser Aspekt soll bei der Analyse der eingesetzten Diagnosesoftware beim Kooperationspartner als ein Kernbereich berücksichtigt werden.

4.2.2 Funktionalität

Neben der Mobilität stellt die steigende Funktionsvielfalt der Steuergeräte die heutige Diagnosesoftware vor weitere Herausforderungen.

Nachdem sich die Variationsvielfalt der Steuergerätefunktionalität erst infolge der kostengünstigen Elektronikbauteile Ende der 90er Jahre ergeben hat (siehe Kapitel 2.1.1), waren die mechatronischen Systeme anfänglich nur auf spezielle Anwendungsbereiche beschränkt. Beispiele sind die Abgas- und Motorsteuerung oder Zündsteuerung.

Deshalb war es zunächst praktikabel, unterschiedliche Diagnosesoftware für die jeweiligen Einsatzbereiche bzw. Steuergerätetypen einzusetzen.

Um z. B. die abgasrelevanten Steuergeräte zu testen, gibt es eine Reihe verschiedenster OBD-Softwareangebote. OBD ist ein Kontrollsystem, das ursprünglich dazu entwickelt wurde, die durch den Fahrzeugverkehr steigende Luftverschmutzung in den Ballungszentren Los Angeles und San Francisco zu reduzieren (siehe Kapitel 2.1.2). Mit der OBD-Diagnosesoftware wurden die Fahrzeuge auf ihr Emissionsverhalten hin geprüft und Ursachen für zu hohe Werte ermittelt. In der Diagnosesoftware ist also von Grund auf fest programmiert (hart kodiert), welcher Diagnosebefehl für welchen Wert im Fahrzeug anzusprechen ist. Die Oberfläche ist speziell auf die Diagnose dieses Anwendungsbereiches ausgelegt. Eine Verwendung der gleichen Software für einen anderen Elektronikbereich ist nicht vorgesehen.

Laut Dietmar Peters, dem Leiter der Diagnoseentwicklung der Volkswagen AG, wurde zu Beginn der Einführung der Steuergeräte zu jedem Steuergerät ein eigenes Diagnosewerkzeug mit zugehöriger Software geliefert. Angesichts der Vielzahl von Steuergeräten, nach Peters, "heute undenkbar"[75].

Für eine zeitgenössische Diagnosesoftware ergibt sich somit die Anforderung eine Diagnoseoberfläche zu bieten, deren Funktionalität, flexibel an die Diagnose verschiedener Steuergeräte angepasst werden kann.

[74] Vgl. Wenzel 2001, S. 2
[75] Hammerschmidt 2004 (Online-Quelle)

4.2.3 Grafik

Zudem stellt die steigende Vielfalt der Steuergeräte auch höhere Anforderungen an die grafische Gestaltungsweise der Diagnoseoberfläche. Dieses Phänomen lässt sich in den Bereichen erkennen, die schon länger mechatronisch dominiert werden.

Ein Beispiel ist die Klimatechnik (siehe Kapitel 2.1.2). Um Komfortelektronik wie eine Klimaanlage zu prüfen, muss sich der Diagnostiker bei der Prüfung im Fahrzeuginnern befinden. Er benötigt am Diagnosestand folglich ein Handterminal, damit er sich frei bewegen kann. Für die anfänglich noch manuell gesteuerte Heizanlage war die einfache Textausgabe der Handterminals vollkommen ausreichend. Über ein Textmenü konnte z. B. das Gebläse auf die einzelnen Stufen gestellt werden. Um allerdings eine Bedienkonsole eines modernen automatischen Infrarot-unterstützten Klimamanagementsystems auf der Diagnoseoberfläche abzubilden, werden unter anderem Schieberegler, die Anzeige von Diagrammen oder gleichzeitig anwählbare Bedienknöpfe benötigt. Eine einfache Textausgabe ist an dieser Stelle nicht mehr benutzerfreundlich und teilweise sogar nicht mehr ausreichend.

Als dritter Kernbereich für die Analyse soll deshalb überprüft werden, ob das derzeitige System noch hinreichend grafische Darstellungsmöglichkeiten bietet.

4.3 Ergebnis der Abgrenzung

Als Ergebnis der Abgrenzung wird in diesem Kapitel bestimmt, welche IT-Komponenten der Kfz-Diagnosetechnik laut den Ansatzpunkten neuen Herausforderungen gegenüberstehen. Alle drei erfassten Anforderungsbereiche finden sich in einer spezifischen Komponente des Diagnosesystems wieder.

Die Bereiche Mobilität und die grafische Ausgabe stellen klare Ansprüche an die Gestaltung der Diagnoseoberfläche und somit auch an die Software, die diese bereitstellt. Mit dem Ansatzpunkt Funktionalität soll überprüft werden, ob in dem jetzigen System die Diagnoseoberfläche flexibel genug gestaltet werden kann, um die Vielfalt an Steuergerätefunktionalitäten abzudecken. Die Aufgabenstellung richtet sich folglich auch hier an die Diagnoseoberfläche und deren Verarbeitungslogik. Angelehnt an die Darstellung der Kfz-Diagnosetechnik (Abbildung 2.5) sind diese Bereiche der Diagnosesoftware zuzuordnen – beim Kooperationspartner also der HMI-Software. Die HMI-Software bildet mit ihrer Diagnoseoberfläche die äußerste Schnittstelle zum Menschen (Diagnostiker).

Im Rahmen der Arbeit soll deshalb überprüft werden, ob die beim Kooperationspartner eingesetzte HMI-Software dem derzeitigen Anforderungsprofil gewachsen ist. Gegebenenfalls werden Schritte zur Verbesserung eingeleitet.

4.4 Ablaufplan für die Erfassung und Analyse

In Kapitel 3.3 wurde die beim Kooperationspartner im Einsatz befindliche HMI-Software vorgestellt. Die Beschreibung der Ansatzpunkte Mobilität, Funktionalität und Grafik hat allgemei-

ne Herausforderungen aufgezeigt, mit denen die HMI-Software konfrontiert ist. Mithilfe der Prozessanalyse sollen diese Ansatzpunkte konkretisiert werden.

Um die Prozessanalyse auf eine fundierte Basis zu stellen, mussten zunächst alle von der HMI-Software betroffenen Prozesse erfasst werden. Von Seiten des Kooperationspartners mussten hierfür Prozessablaufmodelle und zugehörigen Prozessdaten zur Verfügung gestellt werden.

Erste Gespräche mit der zentralen Verwaltungsstelle[76] der HMI-Software haben gezeigt, dass die Informationen beim Kooperationspartner nicht in dem erforderlichen Maße vorlagen. Da die Diagnosetechnik einen wettbewerbskritischen Faktor darstellt (siehe Kapitel 1.1), war auch der Kooperationspartner an einer vollständigen Erfassung aller Abläufe und Daten interessiert. Die gewonnenen Informationen sollten deshalb nach der Erfassung dem Kfz-Hersteller in Form eines Fachkonzepts zur internen Weiterverarbeitung zur Verfügung gestellt werden. Des Weiteren wurde die Nutzung dieser Informationen für die Dissertation genehmigt. Um den Ansprüchen von Doktorand und Kooperationspartner nachzukommen und einen geregelten Ablauf der Erfassung sowie der Analyse zu gewährleisten, wurde folgender Ablaufplan erstellt. Er entspricht dem mit dem Kooperationspartner vereinbarten Projektplan[77].

Da die Arbeit das Ziel in Aussicht stellt, eine verbesserte HMI-Software zu liefern, wurden an dieser Stelle auch weiterführende Schritte in die Planung aufgenommen. Sie sind im Ablaufplan mit Klammern markiert und kennzeichnen optionale Schritte.

1. Abgrenzung der relevanten Geschäftsprozesse

Als Erstes müssen alle Geschäftsprozesse identifiziert werden, in denen die HMI-Software zum Einsatz kommt. Die Abgrenzung der relevanten Geschäftsprozesse erfolgt in Gesprächen mit den Verantwortlichen der zentralen Verwaltungsstelle. Danach werden mit den Leitern der Geschäftsprozessbereiche Termine vereinbart, wann die Prozessbeobachtung durchgeführt werden kann.

Geplanter Zeitraum: Juli bis August 2006[78]

2. Prozessbeobachtung aller relevanten Geschäftsprozesse[79]

Die Prozessbeobachtung sieht vor, dass alle die HMI-Software betreffenden Arbeitsabläufe vor Ort in der Produktion und Verwaltung untersucht werden. Neben der Beobachtung der Arbeitsschritte sollen auch Prozessdaten wie Durchlaufzeiten, Wartezeiten, Wiederholungen und Prozesshäufigkeiten manuell erfasst werden. Die gesamte Prozessbeobachtung und Datenerfassung wird vom Doktoranden eigenständig durchgeführt. Des Weiteren stellt der Kooperationspartner Auswertungen der Geschäftsprozessbereiche zur Verfügung. Sie werden z. B. in Form von Tätigkeitsberichten, Arbeitsprotokollen oder Ähnlichem ausgehändigt. Abschließend sollen im Rahmen der Prozessbeobachtung Mitarbeiter und Prozessverantwortliche nach ihrer Zufriedenheit mit der derzeitigen Software und ihren Verbesserungsvorschlägen befragt werden.

[76] Abteilung Basissystementwicklung des Kooperationspartners (Verantwortlich für Verwaltung und Betreuung des INPA-Systems)
[77] Der Ablaufplan wurde entsprechend der vorliegenden Spezifikation von 2006 bis 2008 umgesetzt. Die Termine sind in Fußnoten zu den geplanten Zeiträumen vermerkt.
[78] Im Juli 2006 wurden Gespräche mit dem Leiter der Abteilung TI-430, Basissystementwicklung E/E (Kfz-Hersteller), Vertretern eines Softwarezulieferer des Kfz-Hersteller und dem Betreuer der Dissertation geführt.
[79] Als relevante Geschäftsprozesse werden in Kapitel 5 die Nacharbeit, Analyse, Entwicklung und Verwaltung bestimmt.

Geplanter Zeitraum: September bis Dezember 2006[80]

3. Modellierung der Prozessabläufe mittels ARIS

Im Anschluss an die Beobachtung erfolgt die Übertragung der Prozessabläufe und Prozessdaten in eine ARIS-Datenbank. Diese Technologie steht dem Kooperationspartner zur Verfügung. Die Abläufe sind mit ARIS grafisch in Form von Prozessmodellen abzubilden. Die Erstellung der Modelle wird durch die Aufnahme von Prozessdaten, wie z. B. Durchlaufzeiten oder Wiederholungen, ergänzt.

Geplanter Zeitraum: Januar bis März 2007[81]

4. Zusammenführung der Prozessmodelle in einem Fachkonzept[82]

Ergänzend zu den Prozessmodellierungen wird dem Kooperationspartner ein Fachkonzept zur Verfügung gestellt. Das Konzept enthält neben den Prozessmodellen, auch Anwendungsfälle. Es handelt sich dabei um Fallbeispiele der Prozesse, die von den Arbeitern während der Prozessbeobachtung bearbeitet wurden. Zum Teil sind die Anwendungsfälle auch den Berichten entnommen. Ergänzt wird das Konzept durch die Beschreibung der Softwarearchitektur der im Einsatz befindlichen HMI-Software. Das Fachkonzept ist vom Doktoranden zu erstellen.

Geplanter Zeitraum: März bis Mai 2007[83]

5. Automatisierte und verbale Schwachstellenanalyse der Prozesse über ARIS

Abschließend wurde vereinbart, dass dem Kooperationspartner auch die Ergebnisse der Analyse zur Verfügung gestellt werden. Die Analyse soll Schwachstellen aufzeigen und die Ansatzpunkte bzw. Herausforderungen (Mobilität, Funktionalität und Grafik) konkretisieren. Sie ist somit auch eine wichtige Zielsetzung der Dissertation. Dem Kooperationspartner werden daraus die Ergebnisse der automatisierten Schwachstellenanalyse und die Auswertung der Mitarbeiterkritiken bereitgestellt.

Geplanter Zeitraum: Juni 2007[84]

(6.) Erarbeitung eines Vorschlags zur Optimierung der Prozesse

Wie eingangs erläutert, geht die Zielsetzung der Arbeit über die Analyse hinaus. Falls, wie zu erwarten, verbesserungswürdige Prozessbereiche erschlossen werden, sollen sie durch optimierte Soll-Prozesse ersetzt werden. Als optionale Weiterführung der Kooperation sieht dieser

[80] In einem Werk des Kfz-Herstellers wurde die Nacharbeit drei Wochen (20.11.06 bis 8.12.06) und die Analyse zwei Wochen (11.12.06 bis 21.12.06) lang beobachtet. Die Arbeiten in der Abteilung der SGBD-Entwicklung wurde eine Woche (13.11.06 bis 17.11.06) begleitet. Der Verwaltungsprozess wurde für einen Zeitraum von zwei Wochen (16.10.06 bis 27.10.06) beobachtet. Die erfassten Daten finden sich in den Anhängen C und D.

[81] Die Übertragung in die ARIS-Datenbank erfolgte teilweise bereits während der Erfassung von Dezember 2006 bis Februar 2007. Sie wurde im Verlauf des Jahres 2007 nachkorrigiert. Die genauere Durchführung und die Ergebnisse der Modellierung werden detailliert in Kapitel 5 erläutert. Die eingepflegten Prozessdaten können in der Quelle INPA-Prozessdatenerfassung 2006 nachgelesen werden. Es ist dort ein Auszug der ARIS-Datenbank beigelegt.

[82] Titel: "Prozessanalyse des Ablaufsystems INPA, Teil 1: Prozesserfassung und Softwarearchitektur". Nicht der Öffentlichkeit zugängliches Fachkonzept, das der Kooperationspartner zur internen Verarbeitung nutzt.

[83] Die zur Vorlage bestimmte Version wurde am 25.05.07 fertiggestellt (Quelle: INPA-Analyse 2007)

[84] Die automatisierten Analyseberichte wurden zwischen 18. und 22.06.07 angefertigt. Die Berichte sind in der Quelle INPA-Analyseberichte 2007 hinterlegt. Die vollständige Analyse ist ausführlich in Kapitel 6 dokumentiert.

Punkt vor, dem Kooperationspartner die optimierten Prozessmodelle zur Ansicht zukommen zu lassen. Die Prozesse sollen nach Möglichkeit mit ARIS modelliert werden.

Geplanter Zeitraum: Juli 2007[85]

(7.) Erstellung eines Anforderungskatalogs bzw. Pflichtenhefts

Da die Dissertation die Zielsetzung hat, eine optimierte HMI-Software zu erstellen, wird im Falle der Prozessoptimierung aus den Verbesserungsvorschlägen ein Anforderungskatalog erstellt. Mit dem Anforderungskatalog wird sichergestellt, dass eine neue HMI-Software allen Ansprüchen gerecht wird und damit die Soll-Prozesse realisiert werden können.

Falls es zu einem Anforderungskatalog kommt, wird dieser in Gesprächen mit den Bereichsleitern und Arbeitern vor Ort diskutiert.

Geplanter Zeitraum: Juli 2007[86] (Schritt läuft parallel zur Optimierung)

(8.) Entwicklung/Beschaffung einer verbesserten Software

Anhand des Anforderungskatalogs soll im Rahmen der Dissertation versucht werden, eine verbesserte HMI-Software zu entwickeln oder zu beschaffen. Eine mögliche Neuentwicklung erfolgt in Form eines Prototyps.

Geplanter Zeitraum: Juli bis Januar 2008[87] (der Zeitraum kann sich verkürzen, falls keine Prototypentwicklung notwendig ist)

(9.) Pilotprojektierung der Software in den ausgewählten Prozessen

Abschließend wird vereinbart, die aus der Arbeit hervorgehende Software beim Kooperationspartner im Rahmen eines testweisen Pilotprojekts einzusetzen. Die Pilotprojektierung erfolgt eingeschränkt auf ausgewählte Prozessbereiche, um das Tagesgeschäft nicht zu beeinträchtigen.

Geplanter Zeitraum: Januar/Februar 2008[88]

(10.) Beurteilung der neuen Software auf Basis von Mitarbeiterbefragungen und Anwendungsfällen

Es ist sowohl im Interesse des Kooperationspartners als auch eine der zentralen Punkte der Dissertation die tatsächliche Leistung der neuen HMI-Software zu beurteilen. Deshalb wird auch eine Vereinbarung zu einer fakultativen Evaluation der HMI-Software getroffen. Dabei ist vorgesehen, anhand von Anwendungsfällen die Ergebnisse der derzeitigen und der neuen HMI-Software zu vergleichen. Auch sollen Mitarbeiter nach ihrem Urteil über die neue Software

[85] Die optimierten Prozesse wurde im Anschluss an die Analyse verfasst (25.06. bis 13.07.07). Die optimierten Prozesse sind in Kapitel 7.2 der Arbeit beschrieben.
[86] Der Anforderungskatalog wurde im Juli 2007 erstellt und bis September 2007 mit dem Kooperationspartner weiter verfeinert. Er findet sich in Kapitel 7.3 der Arbeit.
[87] Die Konzepterstellung und Architektur hat bereits zu Beginn des Projekts nach der Bestimmung der Ansatzpunkte in Kapitel 4 begonnen. Erste Testimplementierungen wurden Anfang 2007 durchgeführt. Hauptzeit der Entwicklung erfolgte zwischen Mai und Oktober 2007. Parallel dazu wurde eine Marktanalyse durchgeführt. Kapitel 8 bis 10 beschäftigt sich mit der Marktanalyse und Neuentwicklung.
[88] Die Einführung der neuen HMI-Software beim Kooperationspartner erfolgte im Dezember 2007 in der Nacharbeit und Verwaltung. Kapitel 11.3 beschreibt die Schritte der Einführung.

befragt werden. Die Gesamtbeurteilung entscheidet über die weitere Verwendung der HMI-Software.

Geplanter Zeitraum: Februar 2008[89] (Schritt läuft parallel zur Einführung)

Da die Abarbeitung des Ablaufplans im Rahmen der Dissertationstätigkeit erfolgt, entsprechen einige Punkte des Ablaufplans den Zielsetzungen aus Kapitel 1.2. Elementare Teile beider Vorgehensmodelle sind die Modellierung und Analyse der Prozesse mit dem Ziel der Optimierung. Dieses Ziel wird mit in folgenden Kapiteln erarbeitet.

[89] Die Anwendungsfälle mit der neuen und der bestehenden Software wurden Januar und Februar 2008 durchgeführt. Genaue Termin sind den Protokollen der Quelle INPA-Analyseberichte 2007 zu entnehmen. Die Durchführung und das Gesamturteil der Evaluation sind in Kapitel 11.4 ausführlich beschrieben.

5 Modellierung der gegenwärtigen Prozesse zur Diagnose elektronischer Steuergeräte

Zentrale Zielsetzung der Dissertation ist die Optimierung der Prozesse zur Diagnose von Steuergeräten.

Die Bedeutung der Optimierung von Prozessen ist bei den Unternehmen in den letzten Jahren stark gestiegen[90]. So beschäftigten sich laut einer Umfrage des Instituts IDS Scheer im Jahre 2003 mit steigender Tendenz 80 % aller befragten Unternehmen intensiv mit dem Prozessmanagement. Eine Folge davon war, dass sich hierfür sehr viele unterschiedliche Methoden, Konzepte und Werkzeuge entwickelt haben. Sie werden im Folgenden unter dem Begriff Geschäftsprozessmanagements (GPM) zusammengefasst.

Zunächst gilt es festzulegen, auf welche Weise die Prozesse optimiert werden sollen und welche Methode sich dafür am besten eignet. Die Bandbreite reicht hier von einer radikalen Erneuerung durch Business Process Reengineering bis hin zu einer punktuellen Verbesserung durch die Total-Cycle-Time-Methode. Die Auswahl der Methode ist von entscheidender Bedeutung bei der Definition der neuen Anforderungen bzw. des Sollkonzepts. Die nähere Betrachtung der Vorgehensweise folgt im Rahmen der Analyse und Soll-Modellierung in den Kapiteln 6 und 7.

Unabhängig davon, mit welcher Methode die Prozesse optimiert werden, ist die Bestandsaufnahme ein elementarer Schritt der Geschäftsprozessoptimierung (GPO). Sie beinhaltet die Erfassung, welche Geschäftsprozesse in welcher Form ablaufen[91].

Wenngleich der Intensität und dem Umfang der Prozesserfassung in den einzelnen GPM-Konzepten unterschiedliche Priorität zugeordnet wird[92], bildet in den meisten Konzepten die Ist-Aufnahme und deren Analyse den ersten Schritt[93].

In dieser Arbeit besteht die Ist-Aufnahme vorrangig in der Modellierung der gegenwärtigen Prozesse und in der Erfassung der zugehöriger Prozessdaten.

---

[90] Vgl. Schmelzer 2004, S. 34
[91] Vgl. Staud 2001, S. 17
[92] Vgl. Ellis 2004, S. 105
[93] Vgl. Schmelzer 2004, S. 254

Für die Geschäftsprozessmodellierung ist die Auswahl des Werkzeugs von elementarer Bedeutung. Denn das Werkzeug beeinflusst den Aufbau, die Syntax und somit die Lesbarkeit der Modelle sehr stark. Wie bei den Methoden des GPM hat sich auch bei der Geschäftsprozessmodellierung ein breites Angebot herausgebildet.

5.1 Beschreibung des ausgewählten Modellierungswerkzeugs ARIS

Unter einem Werkzeug zur GPO versteht man aus praktischer Sicht vorrangig ein Software Toolset – eine Sammlung von Programmen, die die Arbeit bei der Prozesserfassung und Weiterverarbeitung der Daten erleichtern soll.

Den bekannten Vertreter der Branche bieten aber weit mehr und sie nur als Modellierungssoftware zu bezeichnen, würde ihren Funktionsumfang nicht gerecht. Die meisten von ihnen enthalten auch ein umfassendes Konzept, um die Anforderungen des GPM im Unternehmen fast vollständig abzudecken. Sie bieten dahingehend Möglichkeiten zur Analyse, Simulation, Auswertung, Überwachung und Verbesserung der Prozesse an. Dies unterstreicht nochmals die Wichtigkeit bei der Auswahl des richtigen Werkzeugs.

Einer der populärsten Vertreter hierfür ist die von A. W. Scheer entwickelte Architektur Integrierter Informationssysteme (ARIS). ARIS beschreibt "eine ganzheitliche Methode zur umfassenden Beschreibung von Geschäftsprozessen"[94]. Aufbauend auf dieser Methode bietet ARIS ein Toolset, das einen umfangreichen Werkzeugkasten zur Modellierung, Analyse und Optimierung von Geschäftsprozessen vereinigt. Nicht zuletzt macht es diese Vollständigkeit zum "weltweit meistverkauften Werkzeug für die Prozessmodellierung" und führten 2003 zum Einsatz in 90 % der Dax-30-Unternehmen[95].

Die Entscheidung für die Verwendung von ARIS für diese Arbeit fiel zum einen wegen dieser umfangreichen Modellierungsfunktionalität und seiner starken Verbreitung, zum anderen, da es bei der GPO sowohl zur Verbesserung bestehender Prozesse als auch zu deren Neueinführung geeignet ist, vgl. Ellis[96]. Des Weiteren ist das Werkzeug beim Kooperationspartner bereits im Einsatz. Dies verschafft den Vorteil, dass die Mitarbeiter zum Teil bereits mit der Darstellungsweise von ARIS vertraut sind.

Bei der Erfassung eines so umfangreichen Arbeitsbereiches wie der Diagnose von Steuergeräten ist es wichtig, die Aussagekraft der Prozessbilder nicht durch zu viele Informationen zu verringern. Dies gilt besonders, wenn man sich auf ein spezielles Anwendungssystem in den Prozessen konzentrieren will – wie im vorliegenden Fall auf INPA.

Eine elementare Eigenschaft von ARIS ist es, diese Komplexität von Prozessen reduzieren und damit beherrschen zu können[97]. Die Reduktion wird vorrangig durch zwei grundlegende Konzepte erreicht. Dies ist zum einen die Zerlegung in Beschreibungssichten und zum anderen in Beschreibungsebenen.

---

[94] Ellis 2004, S. 17
[95] Vgl. Pressemitteilung von: IDS Scheer, veröffentlicht am 01.01.2004
[96] Vgl. Ellis 2004, S. 17
[97] Vgl. Scheer 2001, ARIS-Methode, S. 2-1

Folgende Abbildung zeigt die vier Sichten Organisationssicht, Datensicht, Steueurungssicht und Funktionssicht und jeweils deren Unterteilung in die drei Ebenen Fachkonzept, Datenverarbeitungskonzept und Implementierung. Die Sichten und Ebenen werden in den folgenden Kapiteln näher erläutert.

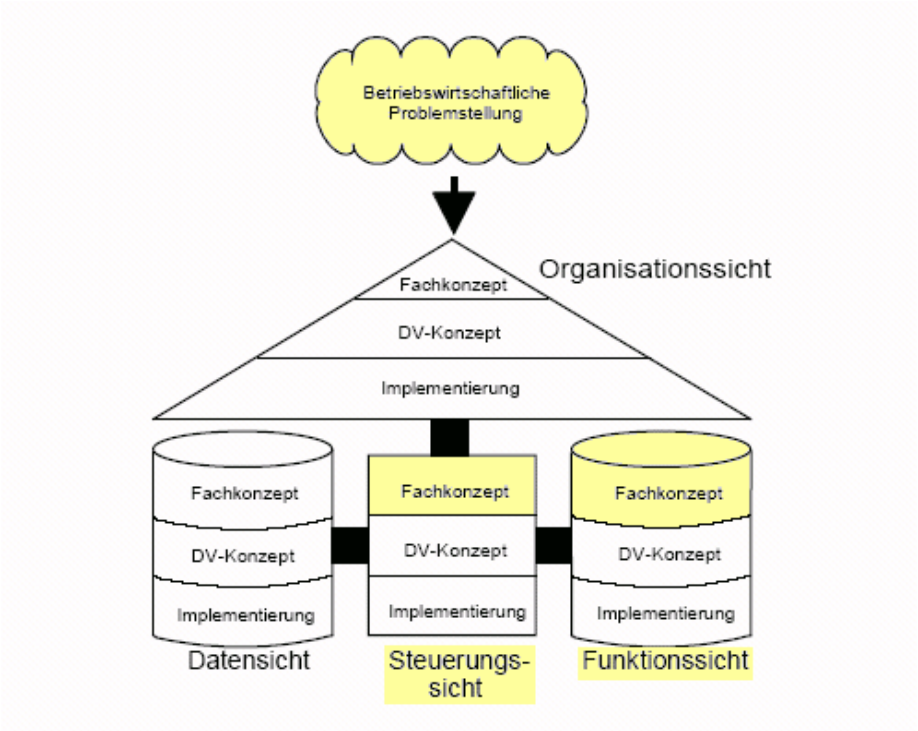


Abbildung 5.1 ARIS-Konzept aus Scheer 2001, ARIS-Methode, S. 2-9

5.1.1 ARIS-Beschreibungssichten des bestehenden Diagnosesystems

Bei der Darstellung mittels Beschreibungssichten ist es das Ziel, einen Prozess zunächst als Ganzes zu betrachten, um ihn anschließend in einzelne Sichten zu zerlegen. Dadurch reduziert sich die Komplexität. Die Prozessabschnitte können in den einzelnen Sichten nur durch die Beziehungen und Zusammenhänge abgebildet werden, die für den Themenbereich bzw. die Aussage relevant sind. Wie in der Grafik dargestellt, sind bei ARIS vier Sichten von grundlegender Bedeutung: die Datensicht, Organisationssicht, Steuerungssicht und Funktionssicht[98].

Die Datensicht enthält die Zustände und Ereignisse eines Prozesses. Trifft z. B. ein fehlerhaftes Fahrzeug in der Nacharbeit ein, wird mittels dieser Sicht dargestellt, z. B. welche Fahrgestellnummer das Fahrzeug hat, welchen Test das Fahrzeug nicht bestanden hat oder zu welcher Uhrzeit. Die Datensicht erfasst also den Bereich, der bei der Umsetzung der Datenbank von Bedeutung ist. Scheer formuliert die Aufgabenstellung für diese Sicht wie folgt: "Welche Daten müssen wie gespeichert werden, damit der Prozess durchgeführt werden kann"[99].

[98] Vgl. Staud 2001, S. 27
[99] Scheer 2001, ARIS-Methode, S. 4-22

Die Organisationssicht gibt vorrangig Organigramme wieder. Die Fragestellung lautet hier, welcher Personentyp (Diagnostiker) bzw. welche Organisationseinheit (Abteilung) ist für den Bereich Nacharbeit verantwortlich.

Die Prozesse (bei Scheer Funktionen genannt) werden in der Funktionssicht abgebildet. Hierbei können die Funktionen in Teilfunktionen aufgeteilt, sowie deren Zusammenhänge dargestellt werden. Beispielsweise muss ein Diagnostiker, um den Prozess Fehlersuche am Fahrzeug durchführen zu können, die Diagnosesoftware starten. So stellt sich das Starten der Software als Unterfunktion der Funktion Fehlersuche dar. Die Funktionssicht wird deshalb verwendet, um einen Überblick über die Geschäftsprozesse mit ihren Teilprozessen zu gewinnen.

In der Steuerungssicht wird die Verbindung der einzelnen Sichten zueinander hergestellt. Für das Beispiel der Fehlersuche bedeutet dies, dass z. B. das Ereignis des Eintreffens eines fehlerhaften Fahrzeugs den Diagnostiker veranlasst, die Diagnosesoftware am Fahrzeug anzuschließen und zu starten. Die Elemente aus Datensicht, Organisationssicht und Funktionssicht treffen zusammen und werden in einem Diagramm dargestellt. Es würden hier in einem Diagramm die Fahrgestellnummer, der Nacharbeitsmitarbeiter und die Software INPA mit den einzelnen Ereignissen und Funktionen wie "INPA starten" dargestellt.

In der Steuerungssicht können je nach gewünschter Komplexität und je nach Informationsgrad Elemente der einzelnen Sichten mitaufgenommen oder weggelassen werden. Hieraus ergibt sich eine gute Möglichkeit, den Blick auf das Wesentliche im Prozess zu lenken. Die meisten hier dargestellten Prozessketten sind deshalb in der Steuerungssicht angesiedelt.

5.1.2 ARIS-Beschreibungsebenen des bestehenden Diagnosesystems

Das andere Konzept, das zur Komplexitätsreduktion beträgt, ist die Zerlegung in Beschreibungsebenen. Diese Methode beschreibt Scheer auch als Life-Cycle-Konzept. Denn mittels der verschiedenen Ebenen werden die einzelnen Stufen zur Entwicklung eines Informationssystems dargestellt. Die Stufen führen von einem formalen Planungsmodell bis hin zur tatsächlichen Implementierung.

Da jede Sicht in diese Ebenen unterteilt werden kann, eröffnet dies die Möglichkeit, Prozesse in ARIS sowohl in allgemeiner abstrakter Form bis hin zur programmiertechnischen Umsetzung zu erfassen (siehe Abbildung 5.1).

Scheer wählt hierfür die Beschreibungsebenen Fachkonzept, DV-Konzept und technische Implementierung.

Das Fachkonzept stellt die erste strukturierte Darstellung eines Prozesses dar. Die Darstellung muss also in einer so weit in einer "formalisierten Sprache" beschrieben werden, sodass sie Ansatzpunkt für eine "konsistente Umsetzung in die Informationstechnik sein kann"[100]. Das Fachkonzept nimmt deshalb eine so besondere Bedeutung ein, weil es "langfristiger Träger des betriebswirtschaftlichen Gedankengutes ist"[101]. Am Fachkonzept lässt sich am besten der fach-

[100] Scheer 1998, S. 15
[101] Staud 2001, S. 28

liche Nutzen des Informationssystems erkennen[102]. Aus diesem Grund kommt besonders bei der Ist-Modellierung in diesem Kapitel das Fachkonzept zum Einsatz.

Das DV-Konzept und die Implementierung sind Ebenen, die schon sehr nah an der Technik sind. Sie dienen der Übertragung der Prozesse auf hardware- und softwaretechnische Komponenten. Die Implementierungsebene spiegelt teilweise direkt den Programmcode der Software wider. Da aber zu vermuten ist, dass die Software ohnehin geändert werden muss, sind diese beiden Ebenen für die Ist-Erfassung von geringerem Interesse. Außerdem sollen Programmcode und Prozessdaten zum Schutz firmeninterner Daten möglichst ausgespart werden.

Um die betroffenen Prozesse in dieser Arbeit abzubilden, liegt der Schwerpunkt also in den Fachkonzepten der Funktions- und Steuerungssicht.

Der Überblick über die Geschäftsprozesse (Kapitel 5.4 und 5.5) wird als Funktionsbaum (Fachkonzept der Funktionssicht) abgebildet. Die Beschreibung der einzelnen Teilprozesse (Fachkonzept der Steuerungssicht) erfolgt unter Zuhilfenahme von erweiterten ereignisorientierten Prozessketten (eEPKs). Die Bereiche sind in Abbildung 5.1 gelb markiert. Als Vorarbeit zu den Fachkonzepten soll bei ARIS das Vorgehen bei der Erfassung festgelegt werden.

5.2 Vorgehen bei der Erfassung

In der Abbildung 5.1 kann man erkennen, dass dem formalen Beginn der Prozessmodellierung (den Fachkonzepten) eine weitere Beschreibungsebene vorausgeht. Diese beschreibt Scheer als halbformal. Sie dient der Beschreibung der allgemeinen Problemstellung und liefert als Ergebnis ein Vorgehen für die Erfassung. Mit ihr soll der Einstieg in die Modellierung nach ARIS gefunden werden.

Die Problemstellung soll demnach die Ausgangssituation anhand "grober Tatbestände" erläutern[103]. Da sich diese Beschreibung nahe an der fachlichen Sprachwelt des Problembereichs bzw. Branchenbereichs orientiert, ist hierbei die Nutzung von "fachweltspezifischen Vokabulars" durchaus üblich und teilweise notwendig. Deshalb sieht es Scheer auch nur als Vorschlag, sich hierbei bereits einer Modellierungstechnik zu bedienen. In der Praxis ist meist eine rein textliche Beschreibung üblich[104].

Die Erläuterung des Ist-Zustandes der Prozesse soll an dieser Stelle weniger eine komplett umfassende Beschreibung sein, als dazu dienen, den Beweggrund für die Modellierung aufzuzeigen. Sie versucht folglich, die Schwachstellen des derzeitigen Systems zu umreißen. Neben der Benennung der Probleme kann sie auch Gründe und Vorschläge für Verbesserungen, also "Soll-Konzepte", zum Inhalt haben[105].

Zur Begründung der Prozessoptimierung in der Diagnose sei auf das erste Kapitel dieser Arbeit verwiesen. Die stark gestiegene Komplexität der Steuergerätetechnik in den letzten Jahren sowie die daraus resultierenden kostenintensiven Rückholaktionen sind mit die ausschlaggeben-

[102] Vgl. Scheer 2001, ARIS-Methode, S. 2-8
[103] Scheer 2001, ARIS-Methode, S. 2-6
[104] Vgl. Staud 2001, S. 29
[105] Vgl. Scheer 2001, ARIS-Methode, S. 3-1

den Beweggründe für die Kfz-Industrie, sich mit den Prozessen der Steuergerätediagnose neu zu befassen.

Ein erster Eindruck in Bezug auf die Schwachstellen des derzeitigen Systems wurde bereits im Kapitel 4.2 vermittelt. In einführenden Gesprächen mit Mitarbeitern wurden Kritikpunkte und Verbesserungsvorschläge gesammelt. Es wurden Mitarbeiter ausgewählt, die mit der Software am Fahrzeug arbeiten, diese verwalten und mit der Leitung der Diagnosebereiche betraut sind.

Beispiele aus Kapitel 4.2 sind die mangelnde Fähigkeit der Diagnosesoftware auf mobilen Systemen lauffähig zu sein (Mobilität) und die ungenügende grafische Gestaltungsmöglichkeit (Grafik). Angeführt wurde auch die aufwendige Arbeitsweise, Diagnosemasken zu erstellen und zu bearbeiten (Funktionalität).

Eine fundierte Schwachstellenanalyse ist allerdings nicht Inhalt dieser Beschreibungsebene. Diese erfolgt im nächsten Kapitel im Rahmen der Ist-Analyse. Als Voraussetzung für eine Ist-Analyse nennt Ellis die sogenannte Prozessinventur[106]. Darunter ist die Erfassung der Ist-Prozesse zu verstehen. In diesem Fall ist die Software bereits über Jahrzehnte im Einsatz. Parallel zu der steigenden Vielfalt der Steuergeräte, sind sich die Einsatzbereiche der Software teilweise sprunghaft angestiegen. Deshalb liegen keine vollständigen Dokumentationen der betroffenen Prozesse vor.

Ergebnis der Erläuterung der Problemstellung ist somit, dass zunächst alle Prozesse, an denen die bestehende Diagnosesoftware beteiligt ist, erfasst werden müssen. Die Ist-Erfassung dient also zum einen dazu, zu erkennen, welche Anforderungen an das derzeitige System gestellt werden, und zum anderen, um eine begründete Schwachstellenanalyse durchführen zu können. Welche organisatorischen Schritte für die Erfassung und Analyse in Zusammenarbeit mit dem Kooperationspartner eingeleitet wurden, zeigt der Ablaufplan in Kapitel 4.4.

Schönsleben beschreibt in seinem Kapitel zum praktischen Vorgehen bei der Modellierung von großen Informationssystemen, wie dem vorliegenden, dass die Praxis zeigt, dass es "eigentlich kein allgemein gültiges Vorgehen gibt"[107]. Er schlägt als ersten Schritt eine Systemabgrenzung vor. Deshalb werden auch nicht die Diagnoseprozesse aller Werke modelliert, sondern repräsentativ mittels ausgewählter Werke erfasst. Des Weiteren soll versucht werden, die wichtigsten Geschäfts- bzw. Kernprozesse herauszugreifen und diese zunächst überblicksartig darzustellen. Eine Modellierung aller Teilprozesse hält Schönsleben für kaum möglich und auch nicht zweckmäßig. Er schreibt von einem "konzentrierten Vorgehen" bei der Prozessmodellierung[108].

Schmelzer empfiehlt für die Ersterfassung die Top-down-Methode[109]. Bei dieser Vorgehensweise orientiert man sich an den Kernprozessen im Unternehmen. Davon ausgehend werden die Geschäftsprozesse und deren Teilprozesse bis zum nötigen Grad an Granularität abgeleitet. Deshalb werden im nächsten Schritt dieser Ist-Aufnahme die Kernprozesse des Unternehmens erfasst.

[106] Vgl. Ellis 2004, S. 104
[107] Schönsleben 2001, S. 155
[108] Schönsleben 2001, S. 156
[109] Vgl. Schmelzer 2004, S. 77-79

5.3 Erfassung der Kernprozesse beim Kooperationspartner

In Kapitel 5.2 wurden bereits unterschiedliche Typen von Prozessen, wie Kernprozesse, Geschäftsprozesse oder Teilprozesse genannt. Durch diese Typisierung legen Unternehmen die Hierarchie ihrer Prozesse fest. Allerdings haben sich in der Praxis wie auch in der Theorie, über die Jahre hinweg verschiedenste Ansätze herausgebildet[110]. In diesen Konzepten werden den Typen teilweise völlig unterschiedliche Bedeutung und Gewichtung zugewiesen. Deshalb ist es ratsam, bei der Modellierung innerhalb eines bestehenden Prozesskonzepts die bereits praktizierte Nomenklatur zu verwenden.

Infolgedessen soll der Prozess, der aus Unternehmenssicht die höchste Hierarchiestufe einnimmt, als Kernprozess bezeichnet werden. Synonym wird dieser Prozesstyp in der Praxis auch als Unternehmens-, Management- oder Geschäftsprozess bezeichnet.

Wie seine Bezeichnung wird auch die Definition des Geschäftsprozesses in der Literatur intensiv diskutiert. Staud liefert eine Sammlung verschiedenster Definitionen und versucht, daraus eine Schnittmenge an Eigenschaften abzubilden[111]. Nach diesen Überschneidungen besteht ein Geschäftsprozess "aus einer zusammenhängenden abgeschlossenen Folge von Tätigkeiten, die zur Erfüllung einer betrieblichen Aufgabe notwendig sind". Bis auf wenige deutliche Ausnahmen geht man bei Geschäftsprozessen davon aus, dass sie am Kunden orientiert sind. Der Kunde ist also Anforderungs- und Ergebnisträger des Prozesses. Folgende Abbildung skizziert die relevanten Komponenten eines Geschäftsprozesses und zeigt die Wichtigkeit des Kundenverhältnisses.

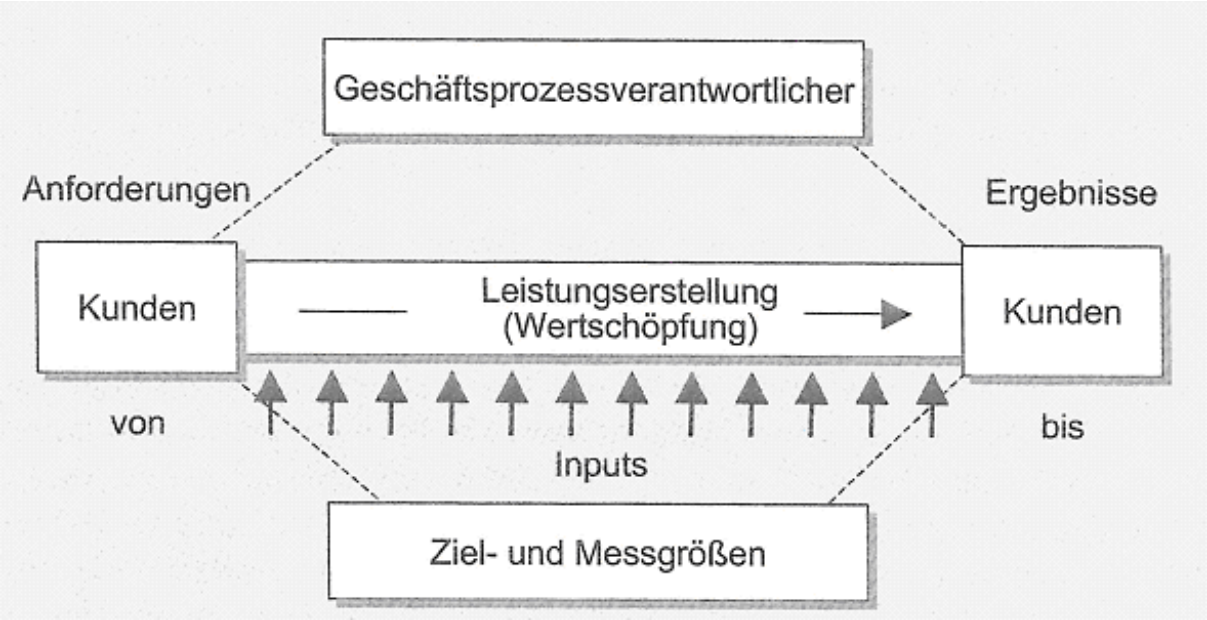


Abbildung 5.2 Komponenten eines Geschäftsprozesses, Schmelzer 2002, S. 46

[110] Vgl. Schmelzer 2004, S. 84
[111] Vgl. Staud 2001, S. 7-8

Durch die direkte Orientierung am Kunden erhält der Geschäftsprozess seine Wichtigkeit für das Unternehmen. Denn nach der ISO-Norm 9000 "Qualitätsmanagement" ist es oberstes Ziel der Unternehmen die Anforderungen des Kunden zu erfüllen und dessen Anforderungen zu übertreffen. Auf diese Weise argumentiert Schmelzer für die Position der Geschäftsprozesse.

Für Porter gibt es zudem noch Geschäftsprozesse, die im Unternehmen eine besondere Rolle spielen. Diese bezeichnet er als Kernprozesse[112]. Sie sind von ihrer Beschaffenheit her an sich Geschäftsprozesse. Allerdings wird mit ihnen die Hauptleistung des Unternehmens erbracht. Kernprozesse nehmen die zentrale Ausgangsposition ein, um weitere Geschäftsprozesse abzuleiten.

Die Anzahl der Kernprozesse orientiert sich weitgehend an Größe und Komplexität der Geschäftseinheit, sowie an der Anzahl und Varianz der Kunden und Produkte. In der Praxis und Literatur hält sich die Zahl größtenteils unter zehn. Schmelzer schreibt von zwei bis drei primären Geschäftsprozessen bzw. Kernprozessen, mit denen aus Erfahrung auszukommen ist.

Beim Kooperationspartner sind im Zuge eines Prozessreengineering (Umgestaltung) Mitte der 90er Jahre zwei Kernprozesse definiert worden. Diese stellen sich wie folgt dar:

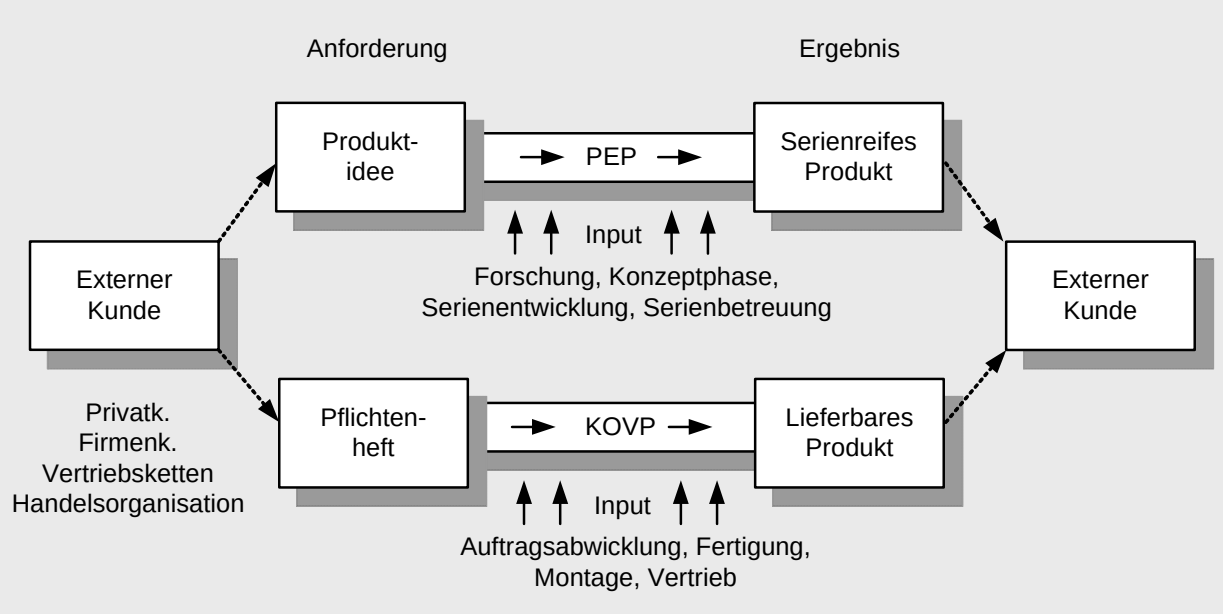


Abbildung 5.3 Kernprozesse beim Kooperationspartner

Der Produktentstehungsprozess (PEP) beschreibt den Weg von der allerersten Produktidee bis hin zum serienreifen Fahrzeug. Die wesentliche Herausforderung besteht in diesem Prozess darin, den Entwicklungszyklus zu verkürzen. Ziel ist es, die Produkte schneller auf den Markt und zum Kunden zu bringen. Durch die zunehmende Wichtigkeit der Elektrotechnik als Wettbewerbsfaktor wird auch die Verkürzung der Innovationszyklen als signifikantes Ziel gesehen.

Der kundenorientierte Vertriebsprozess (KOVP) beinhaltet die Herausforderungen einer kundenindividuellen Fertigung. Der Prozess beschreibt den Weg vom Kundenauftrag, also der Bestellung des Kunden beim Händler oder in der Niederlassung, bis hin zur Auslieferung des Fahrzeugs an den Kunden. Ziel ist es, die Änderungsflexibilität zu steigern, sodass beispiels-

[112] Vgl. Staud 2001, S. 11

weise ein Kunde Sonderausstattungen oder die Farbe seines Autos nach der Bestellung noch ändern kann und trotzdem sein Fahrzeug fristgerecht ausgeliefert wird.

Im folgenden Kapitel werden aus den Kernprozessen die für diese Arbeit relevanten sekundären Geschäftsprozesse abgeleitet.

5.4 Erfassung der sekundären Geschäftsprozesse

Aus Abbildung 5.3 geht hervor, dass die beiden Kernprozesse PEP und KOVP über ihre Anforderung sowie ihr Ergebnis direkten Bezug zum Kunden haben. Bei PEP ist dies ein serienreifer Fahrzeugtyp, den ein Privatkunde ordern kann. Bei KOVP ist das Ergebnis das bestellte Fahrzeug, das der Kunde geliefert bekommt. Hierbei sind natürlich neben den Privatkunden unterschiedliche Kundentypen wie Firmenkunden, Vertriebspartner etc. denkbar. Die Kundentypen der Kernprozesse verbindet jedoch eine Eigenschaft: Sie sind als externe, sprich unternehmensfremde Kunden zu bezeichnen.

Damit die Kernprozesse eine zufriedenstellende Leistung für die externen Kunden erbringen können, sind sie jedoch auf eine Vielzahl interner Unterstützungsprozesse angewiesen. Diese werden in der Literatur auch als sekundäre Geschäftsprozesse bezeichnet[113]. Da aber auch sie den Grundprinzipien der Geschäftsprozesse folgen (siehe Kapitel 5.3), orientieren sie sich primär am Kunden. Nur sind es in diesem Fall größtenteils interne Kunden. Hieraus ergibt sich auch der Hauptunterschied zu den Kernprozessen.

Interne Kunden können Mitarbeiter, Abteilungen, Unternehmensbereiche oder auch Prozesse sein. Nach Schmelzer ist jeder interne Kunde ein Abnehmer von Teilergebnissen, der diese verwendet und weiterverarbeitet[114]. Durch die Weiterverarbeitung und anschließende Weitergabe wird der interne Kunde gleichzeitig zum Lieferanten. Oft fehlt in den Unternehmen das nötige Bewusstsein, um diese internen Kunden-Lieferanten-Beziehungen entsprechend zu pflegen. Dabei sind es gerade die sekundären Geschäftsprozesse, die einen bedeutenden Kostenfaktor darstellen[115]. Deshalb empfiehlt sich deren laufende kritische Überprüfung und auch innerhalb dieser Prozesse Ansatzpunkte für Ablaufverbesserungen zu suchen.

Folgende Übersicht listet die sekundären Geschäftsprozesse beim Kooperationspartner auf in denen Software zur Steuergerätediagnose eingesetzt wird.

[113] Vgl. Porter 1989, S. 63
[114] Vgl. Schmelzer 2004, S. 49
[115] Vgl. Schmelzer 2004, S. 58

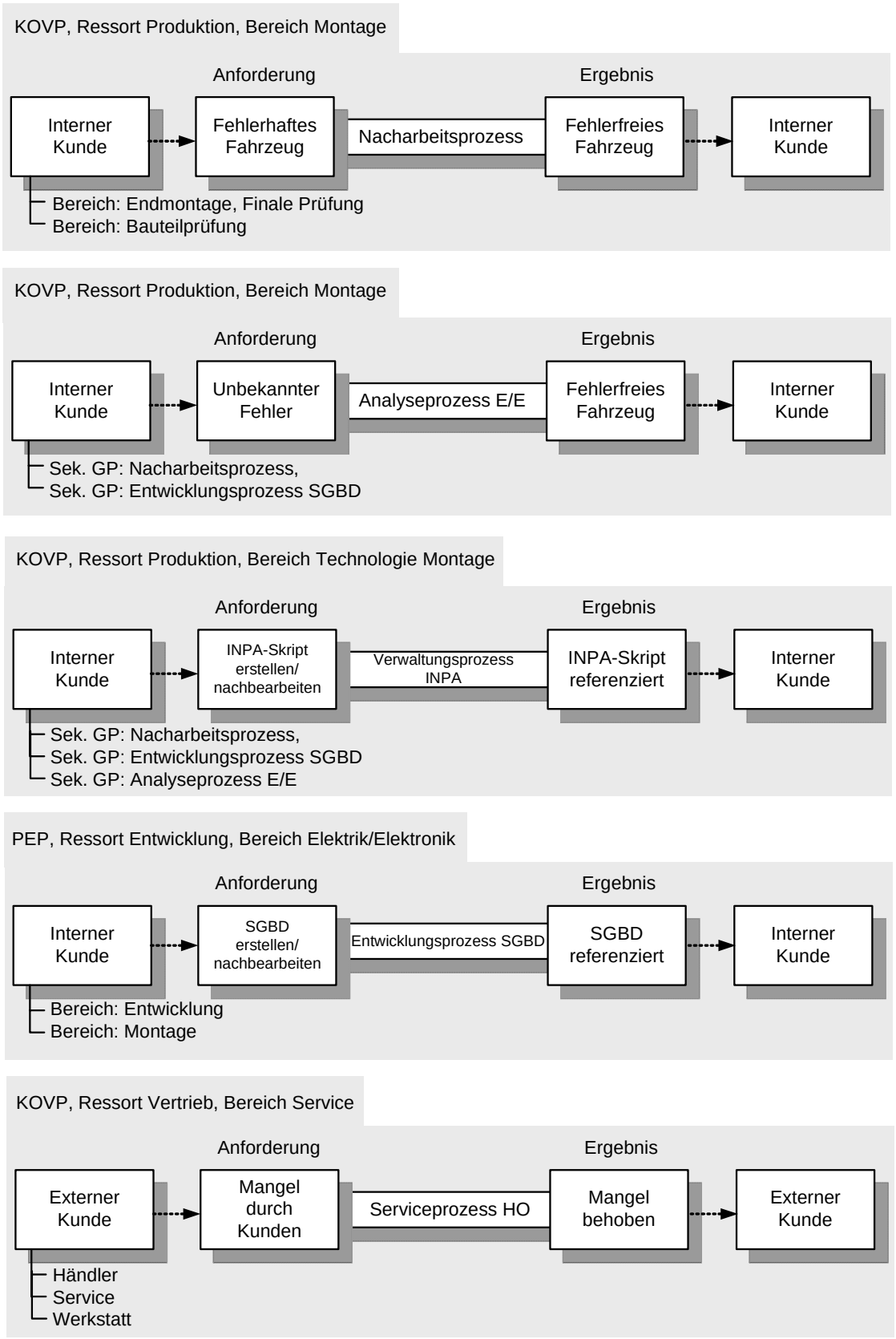


Abbildung 5.4 Sek. Geschäftsprozesse für den Einsatz von Steuergerätediagnosesoftware

Betrachtet man die Prozessbezeichnungen und die jeweiligen internen Kunden, dann zeigt sich hier deutlich, dass das Unternehmen den Schritt zur vollständigen Prozessorientierung noch nicht vollzogen hat. Mit der Einführung der Prozessorientierung versuchen Unternehmen von der Gliederung ihrer Betriebe nach Aufgabenbereichen wie Abteilungen, Ressorts, etc. wegzukommen. Das Ziel ist eine Aufgabenzuteilung rein durch Prozesse. Man verspricht sich dadurch unter anderem schneller auf Kundenwünsche bzw. sich verändernde technologische Bedingungen reagieren zu können. In der Auflistung lässt sich aber erkennen, dass die internen Kunden noch größtenteils Abteilungen (Bereiche) und nicht Prozesse sind. Auch ähneln die Prozessbezeichnungen stark den Bereichsnamen. In einem internen Schreiben über die Struktur der Firma wird explizit formuliert, dass das oberste Gliederungsprinzip des Konzerns trotz der etablierten Kernprozesse noch immer eine funktionale Gliederung nach Ressorts sei[116].

Da sekundäre Geschäftsprozesse mehr von strategischer Bedeutung sind, konnte bei der reinen Ermittlung der betroffenen Prozesse auf die Beteiligung von Facharbeitern verzichtet werden. Die Prozesse wurden im Rahmen erster Gespräche mit einem Leitungsverantwortlichen aus dem Bereich Gesamtfahrzeugprüfungsprozesse für Elektrik und Elektronik bestimmt.

Ein wichtiger Inputparameter für die Wertschöpfung im Rahmen des KOVPs ist die Montage (siehe Abbildung 5.3). Hier gibt es einige Bereiche, in dem die Nutzung der Diagnosesoftware einen großen Teil des Tagesgeschäfts ausmacht. Einer dieser Bereiche ist die Nacharbeit. Dort landen Fahrzeuge, die bei den finalen Prüfprozessen im Rahmen der Endmontage durchfallen. Falls die Nacharbeit erfolgreich verläuft, wird das jetzt fehlerfreie Fahrzeug in den Prozess der Endprüfung zurückgeschleust.

Für den Fall, dass die Mitarbeiter des Nacharbeitungsprozesses den Fehler nicht beheben können, wird das Fahrzeug in den Analyseprozess eingeschleust. Ein interner Kunde des Analyseprozesses ist folglich der sekundäre Geschäftsprozess Nacharbeitsprozess. Die dortigen Facharbeiter nützen die gleiche Diagnosesoftware (INPA) wie in der Nacharbeit um den Fehler zu analysieren. In der Analyse verbleibt das Fahrzeug in der Regel so lange, bis es einen fehlerfreien Zustand erlangt hat.

Ein weiterer interner Kunde für den Analyseprozess ist die Entwicklung. Dieser sekundäre Geschäftsprozess unterstützt allerdings die Wertschöpfungskette des anderen Kernprozesses, PEP. Ein immer wichtiger werdender Part in dem Produktentstehungsprozess (PEP) ist die Neu- und Weiterentwicklung von Steuergeräten. Die Entwicklung ist aber nicht nur Kunde für den Analyseprozess, sondern nutzt die Diagnosesoftware INPA auch selbst. Da besonders auf der mechatronischen Entwicklung im Fahrzeug ein gewaltiger Zeitdruck lastet, versucht man hier Zeit zu sparen und Fehler in Steuergeräten selbst zu diagnostizieren und zu beheben.

Wird die Fahrzeugelektronik durch die Entwicklung von neuen Steuergeräten erweitert, müssen dafür neue Diagnosemasken (Diagnoseskripte) angefertigt werden (siehe Kapitel 2.4). Dies geschieht im Rahmen des Verwaltungsprozesses der Diagnosesoftware. An dieser Stelle läuft eine Vielzahl interner Kundenaufträge ein. Falls sich Steuergeräte verändert haben, Diagnosemasken fehlerhaft sind oder erweitert werden müssen, werden diese in Form von Arbeitsaufträgen bei der Verwaltung der Diagnosesoftware abgegeben. Der Kundenkreis erstreckt sich auf alle drei bereits genannten sekundären Geschäftsprozesse. Innerhalb der Verwaltung wird die Diagnosesoftware selbst zwar selten zur Diagnose am Fahrzeug genutzt, aber mittels Simu-

[116] Vgl. FO 2002, S. 2

lation werden die Diagnosemasken getestet, bevor sie für die Nutzung durch den internen Kunden freigegeben werden.

Der zuletzt aufgelistete Prozess nimmt in seiner Eigenschaft als sekundärer Geschäftsprozess eine Sonderrolle ein. Die Leistung des Serviceprozesses wird direkt für den Käufer eines Fahrzeugs erbracht. Dieser ist in seiner Form als Kunde generell betriebsfremd. Der Serviceprozess ist daher an externe Kunden gerichtet. Der Service fällt bei dem vorliegenden Konzern unter das Ressort Vertrieb. Ein Anwendungsfall ist z. B. ein Kfz-Elektroniker, der in einer Vertragswerkstatt die Diagnosesoftware dazu nützt, um die Funktionalität der Steuergeräte zu testen oder nach dem Beheben eines Defekts die Fehlermeldung im Steuergerät zu löschen. Aufgrund der Nutzung der Diagnosesoftware durch nicht betriebsangehörige Mitarbeiter und des Kontaktes des Serviceprozesses zu externen Kunden gelten hier für die Diagnosesoftware andere Leistungsprioritäten. Es müssen z. B. verschiedene Funktionsumfänge der Diagnosesoftware zum Schutz der Fahrsicherheit und der Komplexität ausgegrenzt werden. INPA wurde für die betriebsinterne Nutzung in dem Ressort Produktion und Entwicklung optimiert (siehe Kapitel 2.4.1). Es eignet sich deshalb nicht für den Einsatz im Ressort Vertrieb. Der Serviceprozess wird deshalb bei der Ist-Erfassung nicht weiter berücksichtigt.

Die übrigen vier INPA-relevanten sekundären Geschäftsprozesse aus den Bereichen Nacharbeit, Analyse, Entwicklung und Verwaltung sollen in den folgenden Kapiteln im Detail betrachtet werden.

5.5 Erfassung der Teilprozesse

Wie in den beiden vorangegangenen Kapiteln bereits thematisiert, sind die Kernprozesse und sekundären Geschäftsprozesse eher ein strategisches Mittel des Managements. Sie dienen dazu, die langfristig ausgelegten Unternehmensziele in das operative Geschäft zu übertragen[117]. Um allerdings das tatsächliche Handeln im betrieblichen Geschehen zu definieren, werden weitere Prozesstypen als Untergruppen der Geschäftsprozesse eingeführt. Die Geschäftsprozesse untergliedern sich in Teilprozesse, Prozess- und Arbeitschritte[118].

Die Teilprozesse liefern abgestimmte Beiträge zur Wertschöpfung, aus denen sich die Ergebnisse der einzelnen Geschäftsprozesse zusammensetzen. In ihnen sind letztlich auch die Ressourcen verankert, die ein Geschäftsprozess benötigt, um die Leistung für den externen Kunden zu erbringen. Ineffiziente Teilprozessabläufe können somit das gesamte Geschäftsergebnis negativ beeinflussen. Auch wenn die einzelnen Arbeitsschritte auf dieser Ebene für das Unternehmen keine strategische Bedeutung mehr haben und die Leistungen für den externen Kunden nicht sichtbar sind, liegt es im Gesamtinteresse des Unternehmens die Teilprozesse zu optimieren[119].

Vergleichbar mit der Bezeichnungsvielfalt der strategischen Prozesse liefert die Literatur und die Praxis auch mehrere Vorschläge, um den Geschäftsprozess selbst zu unterteilen[120]. In die-

[117] Vgl. Ellis 2004, S. 5
[118] Vgl. Schmelzer 2004, S. 84
[119] Vgl. Becker 2000, S. 5
[120] Vgl. Schmelzer 2004, S. 85

ser Arbeit kommt folgende Hierarchisierung zur Anwendung: Geschäftsprozess, Teilprozess, Prozessschritt und abschließend Arbeitsschritt.

Sehr viel wichtiger als die Diskussion über die Bezeichnung erscheint aber an dieser Stelle der Detaillierungsgrad, mit dem die Teilprozesse beschrieben werden. Ein zu intensiver Detaillierungsgrad bei der Ist-Erfassung verdeckt den Blick auf bessere Alternativen[121], hemmt die Kreativität und ist mit erheblichen Kosten und Zeitaufwand verbunden[122]. Allerdings ist die Erfassung der Prozesse unabdingbar für eine Schwachstellenanalyse. Oft korreliert deren Detaillierungsgrad bis zu einem gewissen Niveau mit der Qualität der Analyse[123]. Des Weiteren bieten die Ist-Modelle besonders externen Mitarbeitern eine gute Möglichkeit, sich in die Thematik einzuarbeiten[124].

Außer Frage steht allerdings, dass es für eine Geschäftsprozessoptimierung (GPO) nicht sinnvoll ist, jeden Prozessschritt bis ins Detail oder überhaupt abzubilden. Ellis begründet dies damit, dass bei der Durchführung einer GPO bereits Einigkeit darüber besteht, dass die Ist-Prozesse ohnehin nicht optimal sind und es keinen Sinn macht, Prozesse zu beschreiben, die im Zuge der GPO geändert werden sollen[125].

Bei der Abwägung der Dokumentationstiefe kommt nach Becker vordergründig die Kompetenz des Modellierers zum Tragen. Dessen Weitsicht ist bei der Erfassung gefragt. An ihm liegt es, zu erkennen, welche Prozessschritte für die Analyse und die spätere Soll-Modellierung nötig sind. Um solche Entscheidungen fundiert treffen zu können, ist an diesem Punkt der Kontakt zu Fachexperten und qualifizierten Mitarbeitern unverzichtbar[126].

Deshalb wurden mehrere Gespräche mit Experten und Beteiligten aus den vier Bereichen geführt. Allerdings muss hierbei berücksichtigt werden, dass die Mitarbeiter in das Tagesgeschäft eingebunden sind und daher so zeitsparend wie möglich befragt werden sollten. Dies trifft besonders auf Mitarbeiter zu, die in produktionsbezogenen Prozessen tätig sind und an Takte bzw. Soll-Stückzahlen gebunden sind. Somit sind die Interviews in den Abteilungen vor Ort unter Zuhilfenahme von vorbereiteten Fragebögen durchgeführt worden.

Das Erstellen der Fragebögen wurde gestützt durch Datenmaterial aus dem Intranet, wie z. B. aus bestehenden Prozessdokumentationen. Bei einem so komplexen Aufgabengebiet wie der Diagnose von Fahrzeugelektronik gibt es jedoch eine Vielzahl verschiedener Anwendungsfälle. Diese sind in ihrer Anzahl und Variation nur unter großem Aufwand dokumentierbar. Sie finden sich deshalb kaum in bestehenden Dokumentationen wieder. Besonders für die Analyse erweisen sich Anwendungsfälle aus der Praxis von großem Wert. Infolgedessen wurde im Rahmen der Modellierung eine Prozessbegutachtung der sekundären Geschäftsprozesse vor Ort jeweils über mehrere Tage durchgeführt.

Die folgende Übersicht zeigt die Aufteilung der sekundären Geschäftsprozesse in die ermittelten Teilprozesse. Wie eingangs in diesem Kapitel erläutert, wird zur Erfassung das ARIS Toolset von IDS Scheer verwendet. Für den Einstieg mittels einer Geschäftsprozessübersicht empfiehlt Scheer die Modellierung eines Funktionsbaums.

[121] Vgl. Schmelzer 2004, S. 86
[122] Vgl. Becker 2000, S. 122
[123] Vgl. Schmelzer 2004, S. 86
[124] Vgl. Becker 2000, S. 121
[125] Vgl. Ellis 2004, S. 105
[126] Vgl. Becker 2000, S. 124

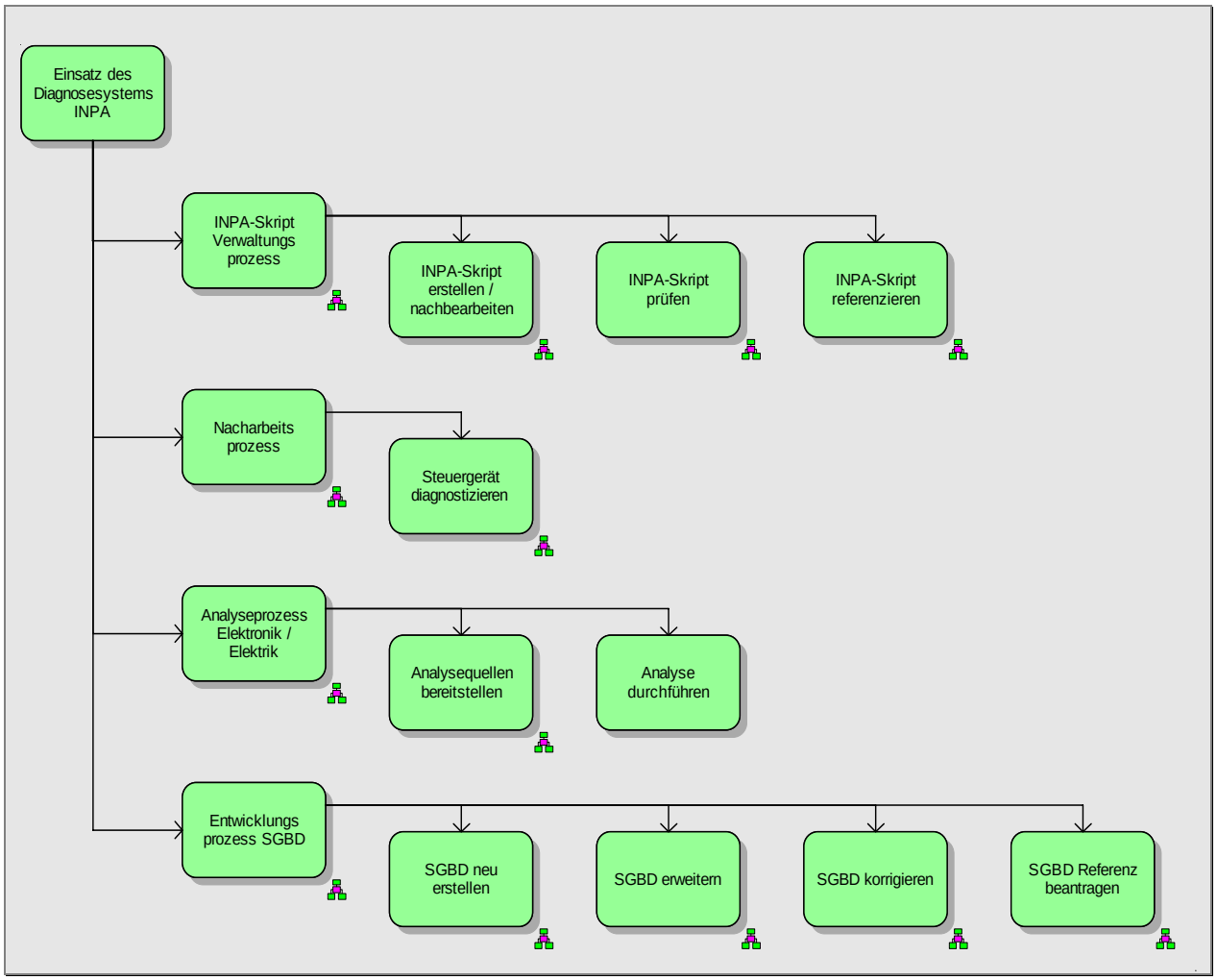


Abbildung 5.5 Einsatz des Diagnosesystems beim Kooperationspartner (Funktionsbaum)

Eine Funktion bezeichnet Scheer als Sammelbegriff für alle Arten von Prozessen, Vorgängen oder Schritten. Er räumt zwar zum besseren Verständnis der Modelle die Untergliederung in Prozesstypen ein, sieht dies aber mehr in der Freiheit des Modellierers. Bei ARIS kann der Begriff Funktion auf allen Hierarchieebenen verwendet werden[127].

Die Ursprungsfunktion des Baums bildet der "Einsatz des Diagnosesystems INPA". Durch die Struktur wird verdeutlicht, dass in dem Funktionsbaum nur Prozesse erfasst werden, an denen INPA direkt beteiligt ist. In Ausnahmefällen werden auch "INPA-fremde" Prozesse erläutert, falls diese für das Verständnis der INPA-Prozesse notwendig sind oder die INPA-Prozesse indirekt beeinflusst werden.

Von der Ursprungsfunktion weichen die vier sekundären Geschäftsprozesse Nacharbeitsprozess, Verwaltungsprozess, Analyseprozess und Entwicklungsprozess ab. Diese sind in dem Funktionsbaum (Abbildung 5.5) mit ihren zugehörigen Teilprozessen dargestellt.

Die folgenden Kapitel liefern einen Überblick über die Abläufe der vier sekundären Geschäftsprozesse. Hierbei wird zunächst ein Überblick über den Gesamtprozess gegeben, anschließend

[127] Vgl. Scheer 2001, S. 4-2

wurden dessen Teilprozesse erläutert. Die Prozessbeschreibungen enthalten eine Prozessabbildung in der EPK-Notation, eine Tabelle mit einer formalen Prozessbeschreibung sowie eine Ablaufbeschreibung. Die Abbildungen und Tabellen finden sich am Ende des jeweiligen Kapitels.

Aufgrund des Umfangs der Prozessdaten werden nur diejenigen Teilprozesse genauer erläutert, die für die spätere Analyse und Soll-Modellierung relevant sind. Eine vollständige Prozessbeschreibung ist in dem internen Fachkonzept "Prozesserfassung und Softwarearchitektur" nachzulesen[128]. Das Fachkonzept umfasst alle Prozessdaten und Modelle, die im Rahmen dieser Arbeit in Zusammenarbeit mit dem Kooperationspartner erfasst wurden.

[128] INPA-Analyse 2007

5.6 Erfassung des Nacharbeitsprozesses

Der Nacharbeitsprozess ist in das Ressort Produktion eingegliedert. Er wird angestoßen, falls ein Fahrzeug nach der Endmontage den finalen Prüflauf nicht besteht.

5.6.1 Prozessübersicht

Ist ein Fahrzeug nach den Vorgaben des Arbeitsplans montiert, beginnt man mit dem Prüfen der Bauteile. Hierbei werden die Bauteile selbst, die Zusammenbauten und die Funktionen des Fahrzeugs überprüft. Dieser abschließende Prüflauf wird "Final" genannt und derzeit durch das Prüfsystem CASCADE gesteuert.

Verläuft der Prüflauf nicht fehlerfrei, erhält das Fahrzeug den Status "Sperre" bzw. "n. i. O.". Es wird somit in den Nacharbeitsprozess eingeschleust. Im Nacharbeitsbereich wird es zunächst in einer Wartezone (Bearbeitungspuffer) gelagert. Von dort aus werden die Fahrzeuge, falls keine besondere Priorität gesetzt wurde, nach dem FIFO-Prinzip von den die Nacharbeitsmitarbeitern (Mechaniker, Elektroniker, Mechatroniker) zur Bearbeitung abgeholt.

In der Nacharbeit sind die Arbeitsplätze nahezu homogen. Sie sind unter anderem mit einem Diagnoseterminal und einem zugehörigen Funk-Diagnosestecker ausgerüstet. Der Mitarbeiter befördert das Fahrzeug aus der Wartezone an einen solchen Arbeitsplatz.

Mit dem ausgeschleusten Fahrzeug wird ein gedrucktes Fehlerprotokoll geliefert. Auf dem Fehlerprotokoll ist der CASCADE-Fehlercode vermerkt. Über die Software "Fehlercodesuche" kann zu der CASCADE-Fehlercode-ID (Identifikationsnummer) eine Fehlerbeschreibung und Verfahrensanweisung ausgegeben werden.

Falls die Fehlercodesuche für die ID keine Ausgabe liefert und der Fehler auch dem Nacharbeiter noch unbekannt ist, setzt sich dieser mit der Analyse-Abteilung oder dem Hersteller der fehlerhaften Fahrzeugkomponente in Verbindung. Bei einem Elektronikfehler kann dies z. B. der Hersteller des Steuergeräts sein. Wenn weder Analyse noch der Lieferant eine zeitnahe Verfahrensanweisung zur Fehlerbehandlung (z. B. Telefon oder E-Mail) liefern können, wird das Fahrzeug zur weiteren Behandlung in die Analyse ausgeschleust (siehe Kapitel 5.7). Größtenteils stellt die Anwendung "Fehlercodesuche" allerdings eine Fehlerbeschreibung mit Anweisung zur Fehlerbehebung bereit.

An dieser Stelle ist es für den Einsatz von INPA entscheidend, ob der Fehler eine elektronische, mechatronische oder rein mechanische Bearbeitung erfordert.

Ist der Fehlertyp ausschließlich im Bereich der Mechanik angesiedelt, wird die weitere Behandlung die INPA-Software nicht betreffen. Anwendungsfälle für einen solchen Prozess sind z. B. beschädigte Karosserieteile, lose Verschraubungen, fehlende Nieten oder diverse fehlerhafte Interieurteile, die nicht mit der Elektronik verbunden sind. Die weitere Fehlerbehebung besteht in einem Austausch der defekten Teile oder in einer Nachbesserung der Beschädigung.

Falls der Fehler aber die Elektronik oder Mechatronik des Fahrzeugs betrifft oder in ihr seine Ursache hat, wird die Diagnose der Steuergeräte mit der Diagnosesoftware INPA durchgeführt

(siehe rote Markierung in Abbildung 5.6). Auf den Prozess der mechatronischen Fehlerbearbeitung wird deshalb im folgenden Kapitel 5.6.2 näher eingegangen.

Befindet sich das Fahrzeug nach der Fehlerbearbeitung in einem fehlerfreien Zustand ("i. O."), wird es in den "Final"-Prozess zurückgeschleust. Je nach Schwere des Fehlerfalls wird dort der finale Prüfprozess nochmals vollständig oder teilweise durchgeführt.

Kann mittels der Verfahrensanweisung zur Fehlerbehebung der Fehler nicht behoben werden, wird das Fahrzeug aus der Endmontage ausgeschleust und an die Analyse übergeben.

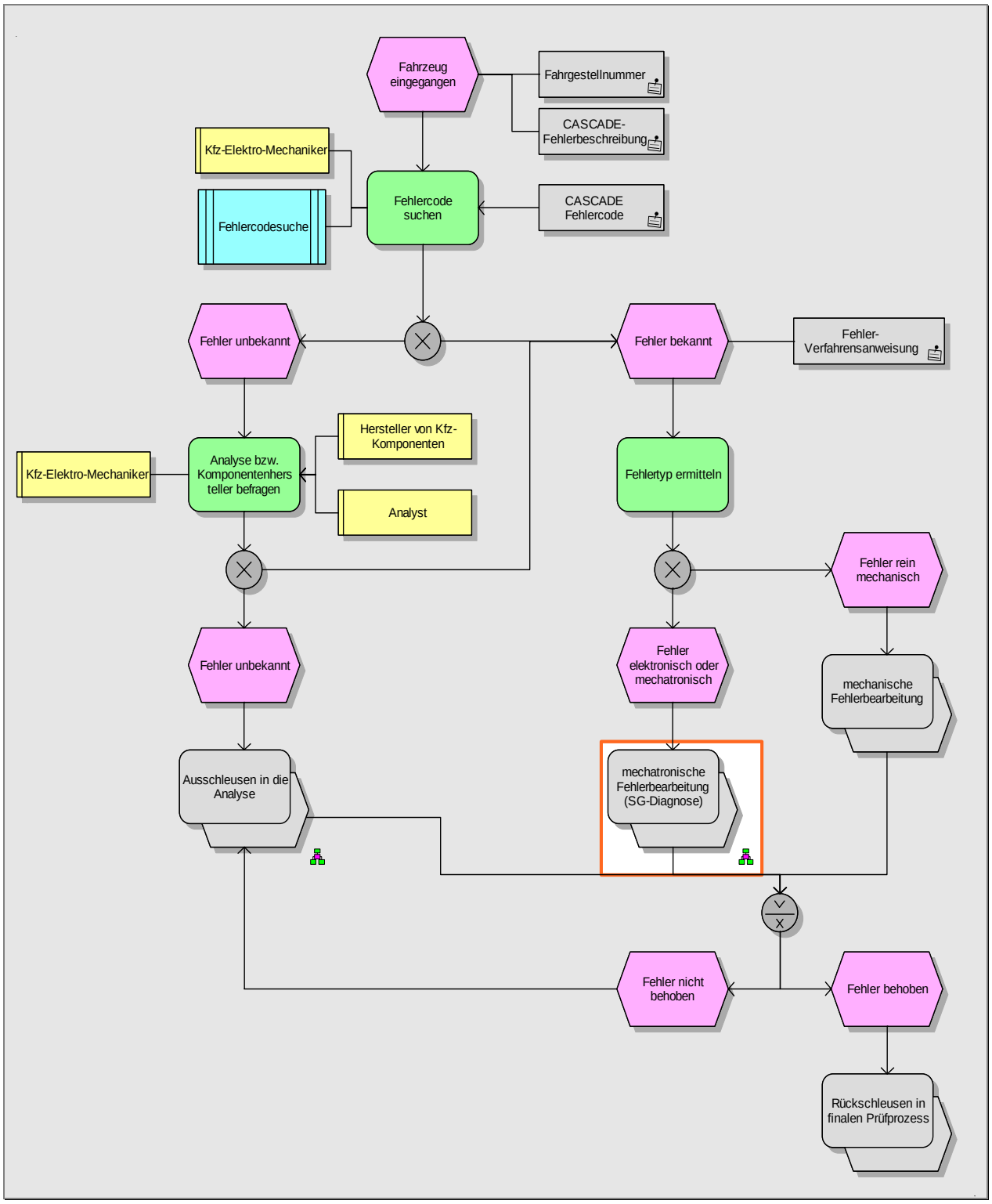


Abbildung 5.6 Sek. GP: Nacharbeitsprozess (EPK)

Name:	GP: Nacharbeitsprozess
Zuordnung:	KP: Kundenorientierter Vertriebsprozess, Ressort: Produktion, Bereich Montage
Prozess-verantwortlicher:	Kfz-Mechatroniker (Karosseriemontage des zugehörigen Werks)
Input:	Fahrzeuge, die den automatisierten finalen Prüflauf (Final) nicht bestehen, werden zur Fehlerkorrektur in die Nacharbeit ausgeschleust. Den Input für den Nacharbeitsprozess stellt somit ein Fahrzeug mit dem Status "n. i. O." und einer entsprechenden Fehlermeldung dar. Die Fehlermeldung des Finals kommt von dem dort verwendeten Prüfsystem CASCADE.
Objekte:	- Fahrgestellnummer - CASCADE Fehlercode + Fehlerbeschreibung - Verfahrensanweisung zur Fehlerbehebung - Anwendungssystem Fehlercodesuche
Ergebnisse:	- Fehler behoben (Fahrzeug wird in den finalen Prüflauf bzw. Endmontage-Prozess zurückgeschleust) - Fehler nicht behoben (Fahrzeug wird in den Analyseprozess ausgeschleust, siehe Kapitel 5.7)
Output:	- Fahrzeug mit Status "i. O." - Fahrzeug mit Status "n. i. O." und Fehlerbeschreibung aus Nacharbeit

Tabelle 5.1 Sek. GP: Nacharbeitsprozess (EPK)

5.6.2 Steuergerät diagnostizieren

Der Prozess "Steuergerät diagnostizieren" wird angestoßen, wenn in der Nacharbeit ein Fahrzeug mit einem elektronischen oder mechatronischen Fehler angeliefert wird. Der Nacharbeiter konnte mittels der Software-Fehlercodesuche eine Verfahrensanweisung abrufen, um den Fehler zu beheben.

Um zunächst auf die Fahrzeugelektronik zugreifen zu können, verbindet der Nacharbeiter das Fahrzeug mit der Diagnosestation. Dafür wird das Funkdiagnose-Interface MDA am Fahrzeug über einen OBD-Stecker angeschlossen. Bei diversen Baureihen befindet sich der Anschluss z. B. im Fahrerfußraum. Er ist mit einer eingerasteten Kappe abgedeckt.

Der Funksender (MDA am Fahrzeug) kommuniziert über eine eindeutige Kartennummer mit der dazugehörigen Empfänger-Funkkarte, die in die PC-Diagnosestation (das Diagnoseterminal) eingebaut ist.

Nach dem Einschalten der Zündung und dem Starten der CASCADE-Anwendung kann über den Basissystem-Treiber (siehe Kapitel 5.3) die Verbindung zur Fahrzeugelektronik aufgebaut werden.

Zur Bestätigung des Fehlerzustands führt der Nacharbeiter den fehlerhaften Prüfablauf aus dem Final nochmals aus. Um diesen in der Anwendung CASCADE zu laden, liest der Nacharbeiter über das Fehlerprotokoll mittels Strichcode (Infrarot-Lesestift) die Fahrgestellnummer ein. Anhand der Fahrgestellnummer wird über die Baureihe die zugehörige Prüfreihe geladen.

Sobald ein Fehler beim sequenziellen Abarbeiten der Testfälle auftritt, bricht CASCADE den Prüflauf ab. Anhand des Bereichs, in dem der Prüflauf zum Stocken kam, und anhand der Fehlerbeschreibung kann der Nacharbeiter (durch Erfahrungswerte und Hinweistexte aus der Anwendung "Fehlercodesuche") das Steuergerät bestimmen, das für den Fehler verantwortlich ist.

Um das Steuergerät den verschiedenen Diagnosetests zu unterziehen, startet der Benutzer über die CASCADE-Oberfläche die INPA-Anwendung.

Über die Fahrgestellnummer ist dem Nacharbeiter die Baureihe des Fahrzeugs bekannt. Diese muss er beim Starten der INPA-Anwendung über eine Auswahlliste bestimmen. Daraufhin listet INPA alle zur Verfügung stehenden Diagnoseskripte (siehe Kapitel 2.4.2) auf. Wie in Kapitel 2.4 erläutert, behandelt ein Diagnoseskript einen bestimmten Steuergerätetyp. Die Bezeichnung der Diagnoseskripte wird in der Liste als Kürzel angezeigt. Aufgrund von Erfahrungswerten weiß der Nacharbeiter, welches Diagnoseskript für welches Steuergerät verantwortlich ist. Durch manuelle Auswahl des Diagnoseskripts wird dieses geladen.

Ist das entsprechende Diagnoseskript geladen, kann der Nacharbeiter über verschiedene Diagnosemasken das Verhalten des Steuergeräts untersuchen. Die Abfragemöglichkeiten, die auf den einzelnen Masken geboten werden, hängen vom jeweiligen Steuergerät ab. Es gibt allerdings Grundfunktionalitäten, die für nahezu jedes Steuergerät unterstützt werden. Diese sind das Lesen des Fehlerspeichers, das Abfragen von verschiedenen Werten und das Steuern von Steuergerätfunktionen. Daraus ergeben sich drei grundsätzliche Maskentypen: Fehler, Status und Steuern. Der Umfang an Befehlen, mit dem jeder Maskentyp ausgestattet ist, ist bei jedem Diagnoseskript unterschiedlich. Er basiert auf dem Typ des Steuergeräts und dem Diagnosezweck, für den das Diagnoseskript entwickelt wurde.

Welche Funktionalität geprüft wird, hängt von dem vorliegenden Fehlerfall ab.

Auf unterschiedliche Anwendungsfälle und Diagnosetests wird in Kapitel 4 bei der Analyse des Prozesses weiter eingegangen. Dort werden verschiedene Nacharbeitsszenarien vorgestellt.

Ziel der Diagnosetätigkeit mit INPA ist es zunächst, die Fehlerursache genau einzugrenzen.

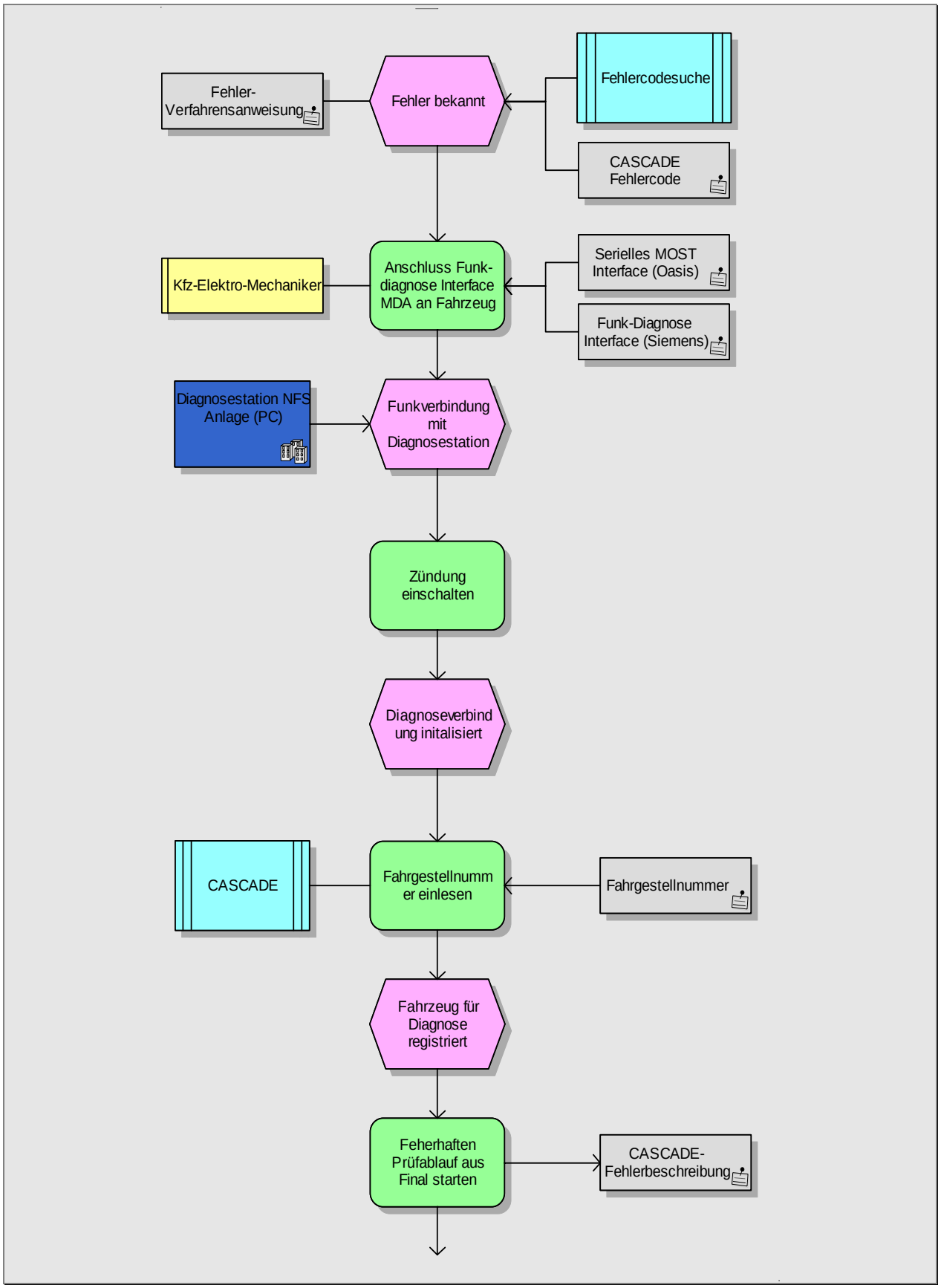
Wird als Fehlerursache ein defektes Bauteil, z. B. Kabel, Stecker, Sensor oder Regler ermittelt, wird dieses ausgetauscht. Danach wird durch manuelles Steuern der fehlerhaften Steuergerätefunktion anhand des Diagnoseskripts überprüft, ob der Fehler behoben wurde. Falls nicht, wird die Diagnosearbeit mit INPA fortgesetzt.

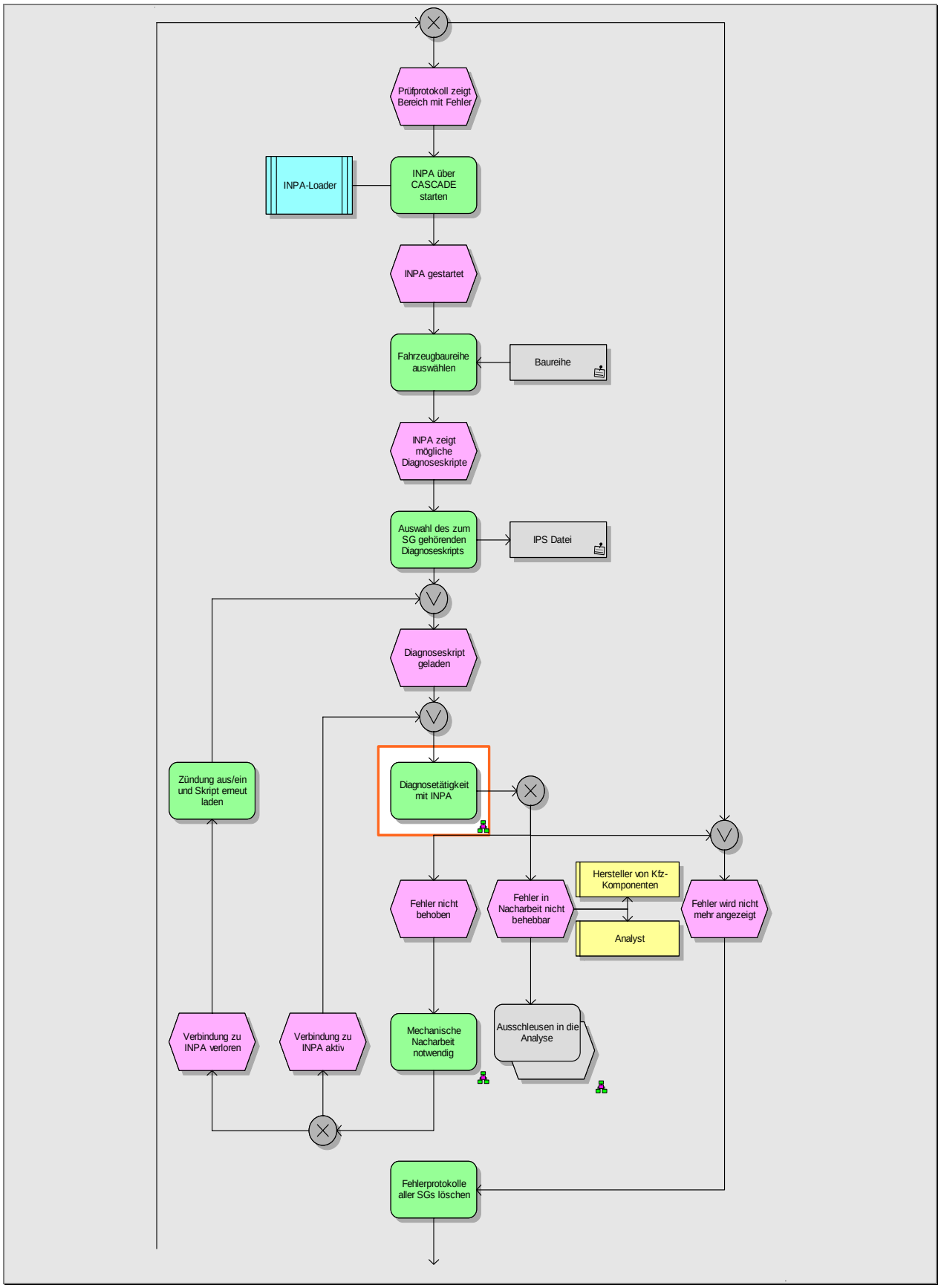
Kann die Fehlerursache nicht genau eingegrenzt werden bzw. kann der Fehler auch durch Austausch der betroffenen Elektronikteile nicht behoben werden, muss das Fahrzeug in die Analyse ausgeschleust werden. Dies geschieht zum Teil in Absprache mit dem Steuergerätelieferanten und Analysemitarbeiter. Hierfür erstellt der Nacharbeiter zusätzlich zur CACSCADE-Fehlerbeschreibung eine eigene formlose Fehlerbeschreibung. In den meisten Fällen wurde dies bereits mit einem Analysemitarbeiter persönlich abgesprochen.

Führen die elektronische Nachbearbeitung mit INPA sowie mechanische Reparaturleistungen hingegen dazu, dass der Fehler von INPA nicht mehr angezeigt wird, unterzieht der Mitarbeiter das Fahrzeug erneut dem eingangs durchgeführten CASCADE-Prüfablauf. Dazu muss er zunächst den Fehlerspeicher aller betroffenen Steuergeräte löschen. Dies ist sowohl mit INPA als auch mit CASCADE möglich.

Verläuft der Prüfablauf erfolgreich, das heißt läuft ohne Unterbrechung bis zum Ende durch, wird das Fahrzeug in das Final zurückgeschleust. Es erhält den Status "i. O.". Von dort werden die Fahrzeuge in den Vertriebsprozess weitergeleitet.

Zeigt das CASCADE-Prüfprotokoll erneut einen Fehler, beginnt der Diagnoseprozess des Steuergeräts mit diesem von neuem.


Fehler-
Verfahrensanweisung
Fehler bekannt
Fehlercodesuche
CASCADE
Fehlercode
Kfz-Elektro-Mechaniker
Anschluss Funk-
diagnose Interface
MDA an Fahrzeug
Serielles MOST
Interface (Oasis)
Funk-Diagnose
Interface (Siemens)
Diagnosestation NFS
Anlage (PC)
Funkverbindung
mit
Diagnosestation
Zündung
einschalten
Diagnoseverbind
ung initalisiert
CASCADE
Fahrgestellnumm
er einlesen
Fahrgestellnummer
Fahrzeug für
Diagnose
registriert
Feherhaften
Prüfablauf aus
Final starten
CASCADE-
Fehlerbeschreibung


Prüfprotokoll zeigt Bereich mit Fehler
INPA-Loader
INPA über CASCADE starten
INPA gestartet
Fahrzeugbaureihe auswählen
Baureihe
INPA zeigt mögliche Diagnoseskripte
Auswahl des zum SG gehörenden Diagnoseskripts
IPS Datei
Diagnoseskript geladen
Zündung aus/ein und Skript erneut laden
Diagnosetätigkeit mit INPA
Hersteller von Kfz-Komponenten
Fehler nicht behoben
Fehler in Nacharbeit nicht behebbar
Analyst
Fehler wird nicht mehr angezeigt
Verbindung zu INPA verloren
Verbindung zu INPA aktiv
Mechanische Nacharbeit notwendig
Ausschleusen in die Analyse
Fehlerprotokolle aller SGs löschen

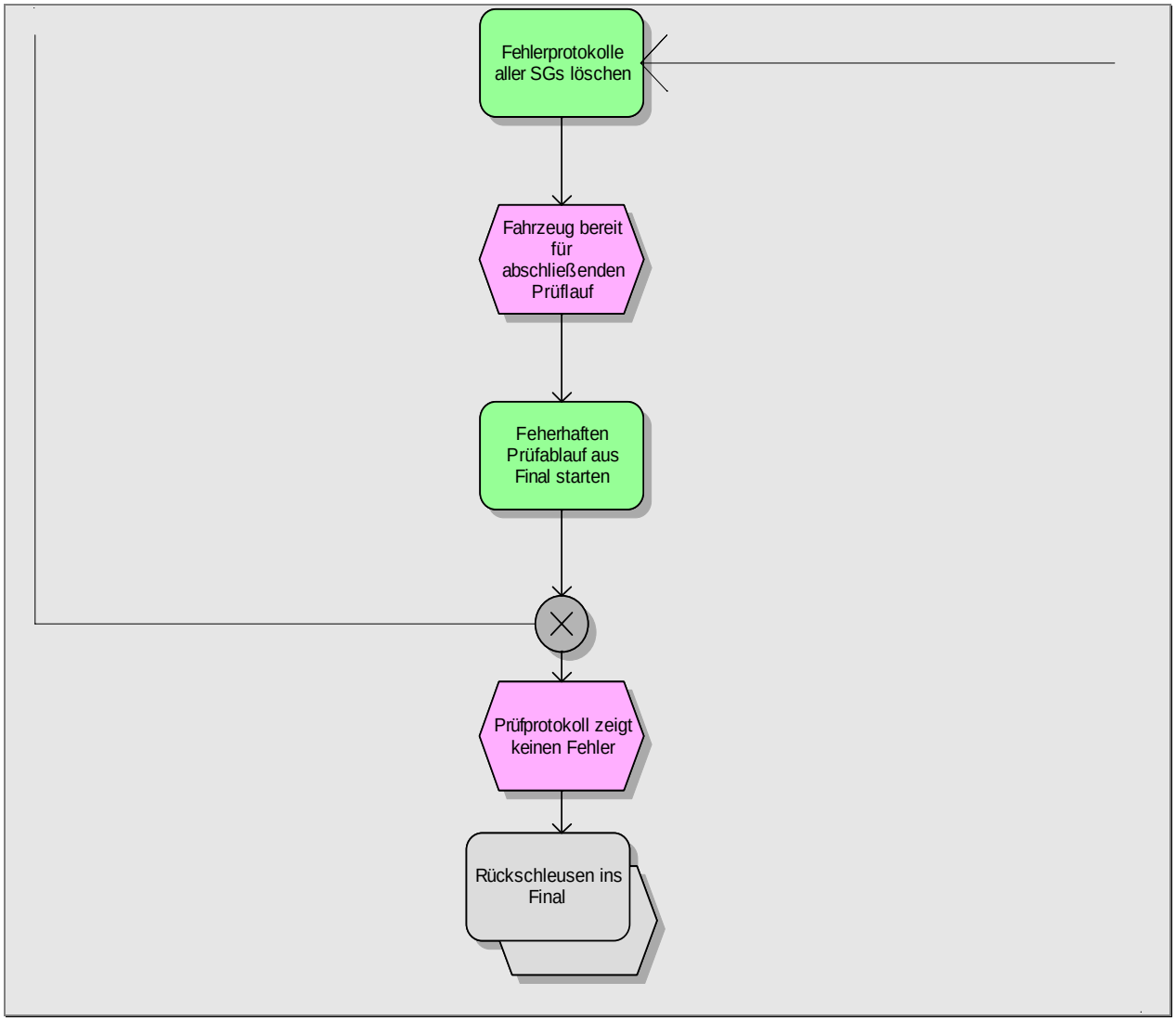


Abbildung 5.7 TP: Steuergerät diagnostizieren (EPK)

Name:	TP: Steuergerät diagnostizieren
Zuordnung:	GP: Nacharbeitsprozess
Prozess-verantwortlicher:	Kfz-Mechatroniker (Abteilung Nacharbeit)
Input:	- Fehler (durch Nacharbeiter erfasst) - Verfahrensanweisung (liegt dem Nacharbeiter vor) - Fehlertyp: elektronisch oder mechatronisch
Objekte:	- Fahrgestellnummer - CASCADE: Fehlercode + Fehlerbeschreibung - Verfahrensanweisung zur Fehlerbehebung - Anwendungssystem Fehlercodesuche

	- Anwendungssystem CASCADE - Anwendungssystem INPA-Loader - PC-Diagnosestation - Serielles MOST Interface (Oasis) - Funk-Diagnose-Interface (Siemens)
Ergebnisse:	- Fehler behoben (Fahrzeug wird in den finalen Prüflauf bzw. Endmontage-Prozess zurückgeschleust) - Fehler nicht behoben (Fahrzeug wird in den Analyseprozess ausgeschleust, Kapitel 5.7)
Output:	- Fahrzeug mit Status "i. O." - Fahrzeug mit Status "n. i. O."

Tabelle 5.2 Sek. TP: Steuergerät diagnostizieren (EPK)

5.7 Erfassung des Analyseprozesses

Der sekundäre Geschäftsprozess Analyseprozess findet sich beim Kooperationspartner in dem Ressort Produktion wieder. Er ist Teil des Geschäftsprozesses Absicherungsprozess E/E. Seinen Anstoß erhält der Analyseprozess größtenteils von dem im vorherigen Kapitel vorgestellten Nacharbeitsprozess. Mit dem Analyseprozess sollen beispielsweise Fehler untersucht werden, die im Rahmen der Nacharbeitstätigkeit nicht behoben werden konnten.

5.7.1 Prozessübersicht

Ausgangspunkt für die Analyse ist der Analyseauftrag. Dieser beinhaltet die CASCADE-Fehlerbeschreibung (siehe Kapitel 5.6.1). Der Analyseauftrag kann ergänzt werden durch eine formlose Fehlerbeschreibung, z. B. aus der Nacharbeit. Diese wird oft verfasst, wenn die Nacharbeit bei Problemfällen unmittelbar Kontakt mit der Analyse aufnimmt.

Die Fehlerbeschreibung wird in ausgedruckter Form mit dem Fahrzeug gemeinsam in der Analyse angeliefert. Ein formloses Erläuterungsschreiben kann den Analysemitarbeiter z. B. auch per E-Mail erreichen.

Um die Analyse beginnen zu können, werden nach dem Eingang des Fahrzeugs zuerst die Fahrzeugdaten ausgelesen. In diesem Schritt kommt das INPA-System zum Einsatz. Dieser Ablauf wird in Kapitel 5.7.2 näher beschrieben.

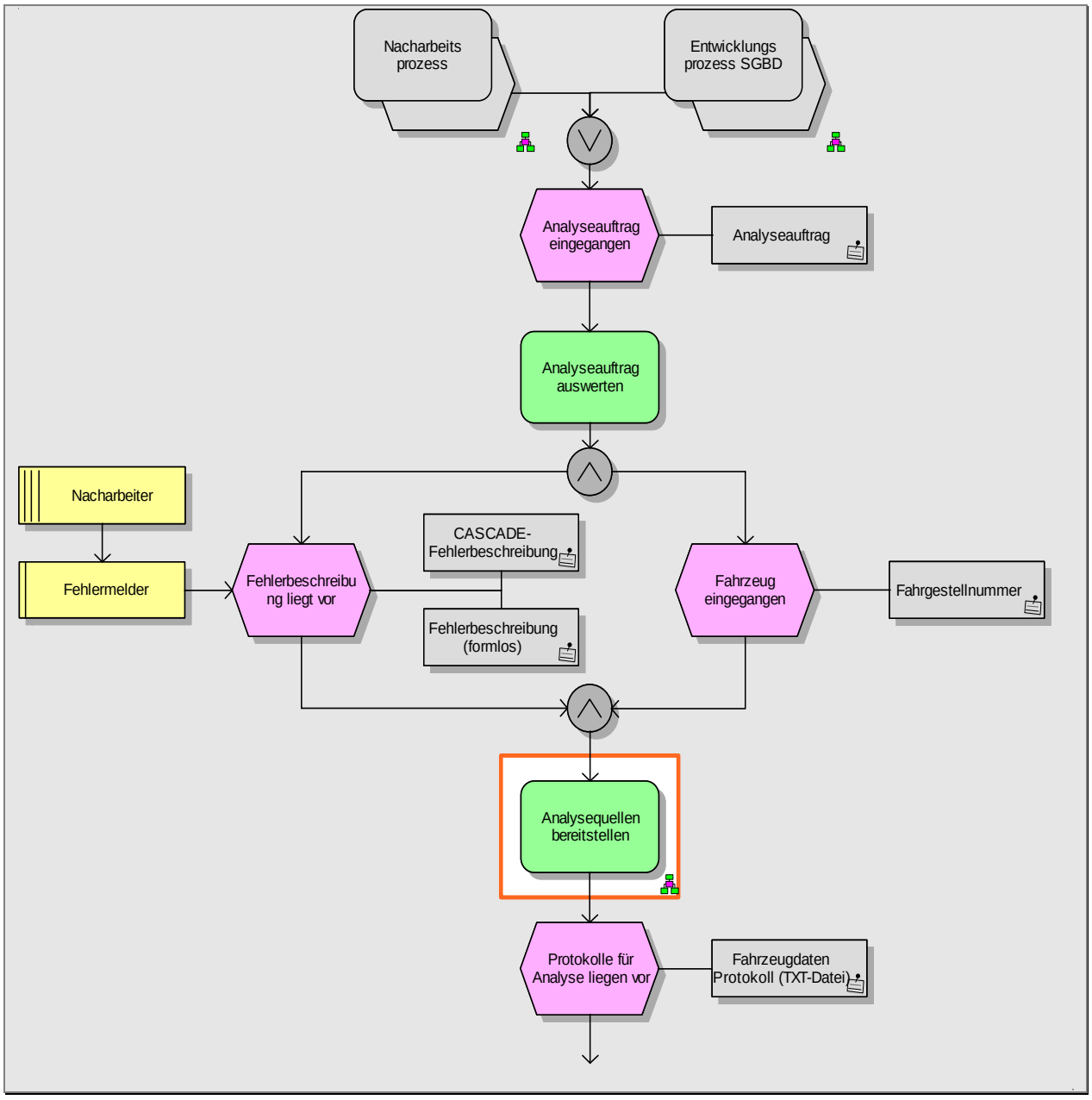
Anhand der ausgelesenen Analysequellen wird versucht, den Konfliktbereich näher einzugrenzen. Bei dieser Eingrenzung wird auch die INPA-Software verwendet, um auf das Steuergerät zuzugreifen. Speziell diese Arbeit der Analysemitarbeiter mit INPA ähnelt allerdings der der Nacharbeiter sehr stark. Auch hier werden über Diagnosemasken Werte auf dem Steuergerät gesetzt und abgerufen (Kapitel 5.6.2). Der Prozess wird folglich von den Optimierungen, die für den Nacharbeitsprozess durchgeführt werden, in gleicher Weise profitieren. Er kann deshalb bei der Ist-Beschreibung ausgespart werden.

Nach der durchgeführten Analyse verfasst der zuständige Mitarbeiter das Ergebnis in einem Analysebericht. Dieser enthält unter anderem die weitere Vorgehensweise. Es können z. B. weitere Untersuchungen mit speziellen Messgeräten erforderlich werden.

Falls die Fehlerursache erkannt werden konnte, wird eine Nacharbeitsroutine erstellt und das Fahrzeug wieder in die Nacharbeit zurückgeschleust. Die Verfahrensanweisung kann (durch die Nacharbeit) ab diesem Zeitpunkt mittels der Anwendung Fehlercodesuche abgerufen werden.

Falls die Analyse zu keinem Ergebnis führt, wird das Fahrzeug mit einer Auslieferungssperre versehen. Die weitere Vorgehensweise hierfür wird in der Prozessbeschreibung "Problemanalyse am Gesamtfahrzeug/Fahrzeugsystem"[129] beschrieben. Diese Schritte betreffen aber das INPA-System nicht mehr.

[129] siehe Hinweis Tabelle 5.3 des GP Analyseprozess Elektrik/Elektronik


Nacharbeits prozess
Entwicklungs prozess SGBD
Analyseauftrag eingegangen
Analyseauftrag
Analyseauftrag auswerten
Nacharbeiter
Fehlermelder
Fehlerbeschreibu ng liegt vor
CASCADE-Fehlerbeschreibung
Fehlerbeschreibung (formlos)
Fahrzeug eingegangen
Fahrgestellnummer
Analysequellen bereitstellen
Protokolle für Analyse liegen vor
Fahrzeugdaten Protokoll (TXT-Datei)

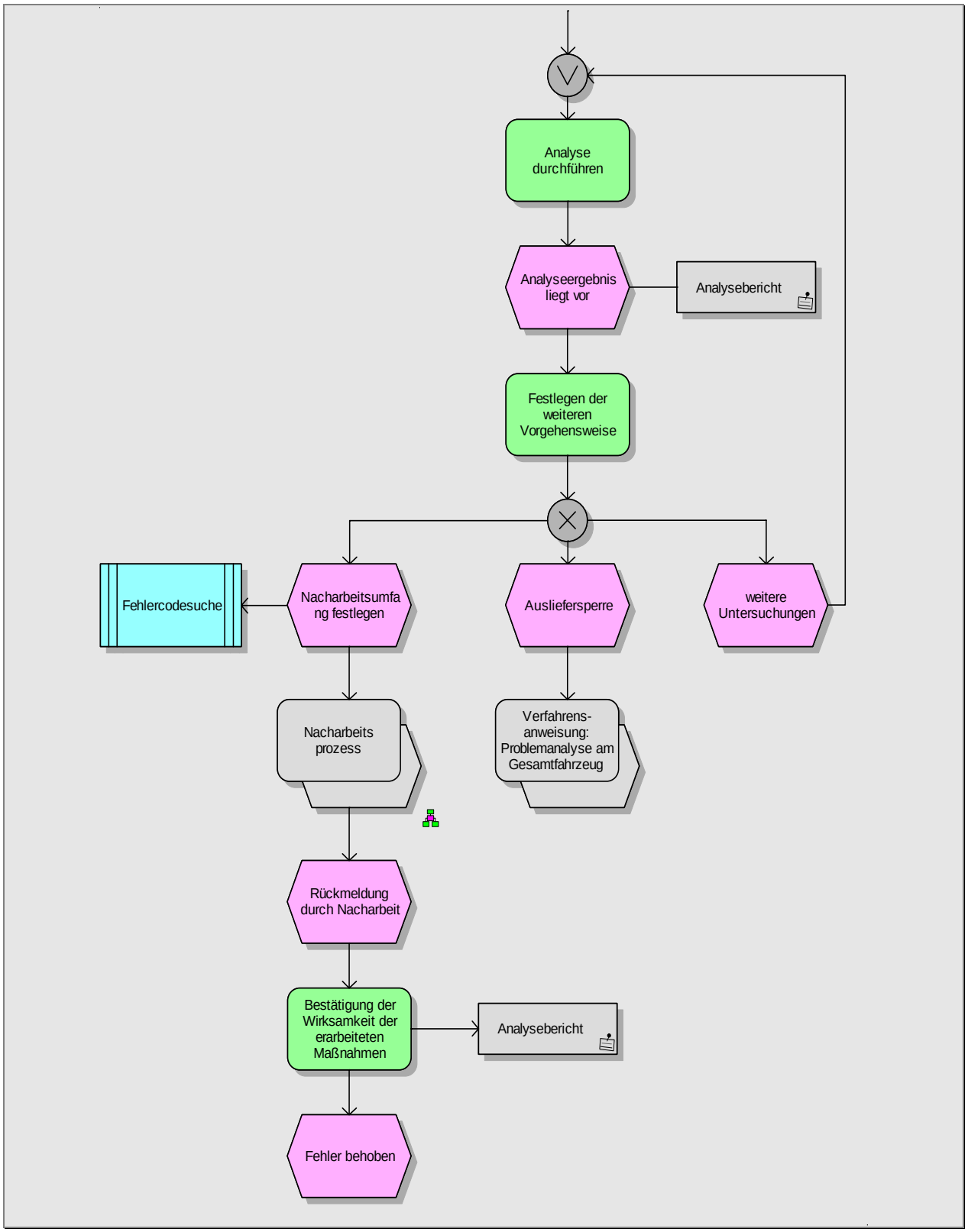


Abbildung 5.8 GP: Analyseprozess Elektrik/Elektronik

Name:	Sek. GP: Analyseprozess Elektrik/Elektronik
Zuordnung:	KP: Kundenorientierter Vertriebsprozess, Ressort: Produktion GP: Absicherungsprozess E/E *(Hinweis: Der abgebildete Analyseprozess basiert auf dem Konzept zur Verfahrensanweisung "Problemanalyse am Gesamtfahrzeug/Fahrzeugsystem"[130] Kernfertigung des Werkes Dingolfing. Die Arbeitsschritte, die nicht direkt das INPA-System betreffen, sind stark vereinfacht dargestellt.)*
Prozess-verantwortlicher:	Analysemitarbeiter
Input:	Der Anstoß kann durch den Nacharbeitsprozess oder den Entwicklungsprozess SGBD erfolgen. Die Nacharbeit in der Endmontage soll möglichst schnell erfolgen. Es soll versucht werden, das Fahrzeug zeitnah wieder in den abschließenden Fertigungsprozess einzugliedern. Fahrzeuge, bei denen dies aufgrund unbekannter Fehler oder technischer Schwierigkeiten nicht möglich ist, werden in die Analyse ausgeschleust.
Objekte:	- Fehlerbeschreibung (CASCADE oder formlos) - Analyseauftrag - Fahrgestellnummer (FGNR) - Fahrzeugdaten-Protokoll (TXT-Datei) - Analysebericht
Ergebnisse:	- Anweisung zur Nacharbeit - Auslieferungssperre - Weitere Untersuchungen
Output:	Fahrzeug mit Verfahrensanweisung zur Fehlerbehebung oder Auslieferungssperre

Tabelle 5.3 Sek. GP: Analyseprozess Elektrik/Elektronik (EPK)

5.7.2 Analysequellen bereitstellen

Um die Fahrzeugdaten bereitzustellen, muss der Analysemitarbeiter das Fahrzeug an das Diagnosegerät anschließen. Im Vergleich zur Nacharbeit ist das Diagnosegerät hierbei vorzugsweise ein Notebook und kein feststehendes Terminal, da der Mitarbeiter parallel mehrere Analysefälle bearbeitet. Die Hardwareschnittstelle zum Fahrzeug ist deshalb ein mobiles EDIC PCMCIA Interface.

Sobald die Zündung eingeschaltet wird stellt der Basissystem-Treiber die Verbindung zu den Steuergeräten des Fahrzeugs her.

Auf dem Notebook wird die Anwendung INPA manuell gestartet.

Der Mitarbeiter lädt für die Baureihe des Fahrzeugs das entsprechende Diagnoseskript. Es unterscheidet sich von den Skripten der Nacharbeit aus folgenden Gründen: Die Analyse versucht

[130] Der Quellenverweis findet sich in dem Fachkonzept "Prozessanalyse des Ablaufsystems INPA" (Quelle: INPA-Analyse 2007)

zunächst den Fehlerstatus des gesamten Fahrzeugs auszulesen und muss zusätzlich vor Beginn der Arbeit das Fahrzeug in den Analysemodus versetzen (besondere Steuergerätefunktionalität). In der Nacharbeit wird allerdings anhand der Verfahrensanweisung aus der Software-Fehlercodesuche sehr viel spezifischer vorgegangen. Die Arbeit in der Nacharbeit ist in den jeweiligen Fällen mehr auf einzelne Steuergeräte beschränkt.

Für die Analyse hingegen bietet das Diagnoseskript die umfassende Funktionalität "Fahrzeugdaten lesen". Mit dieser Funktion werden Textdateien lokal auf das Diagnosegerät (Notebook) des Mitarbeiters geschrieben.

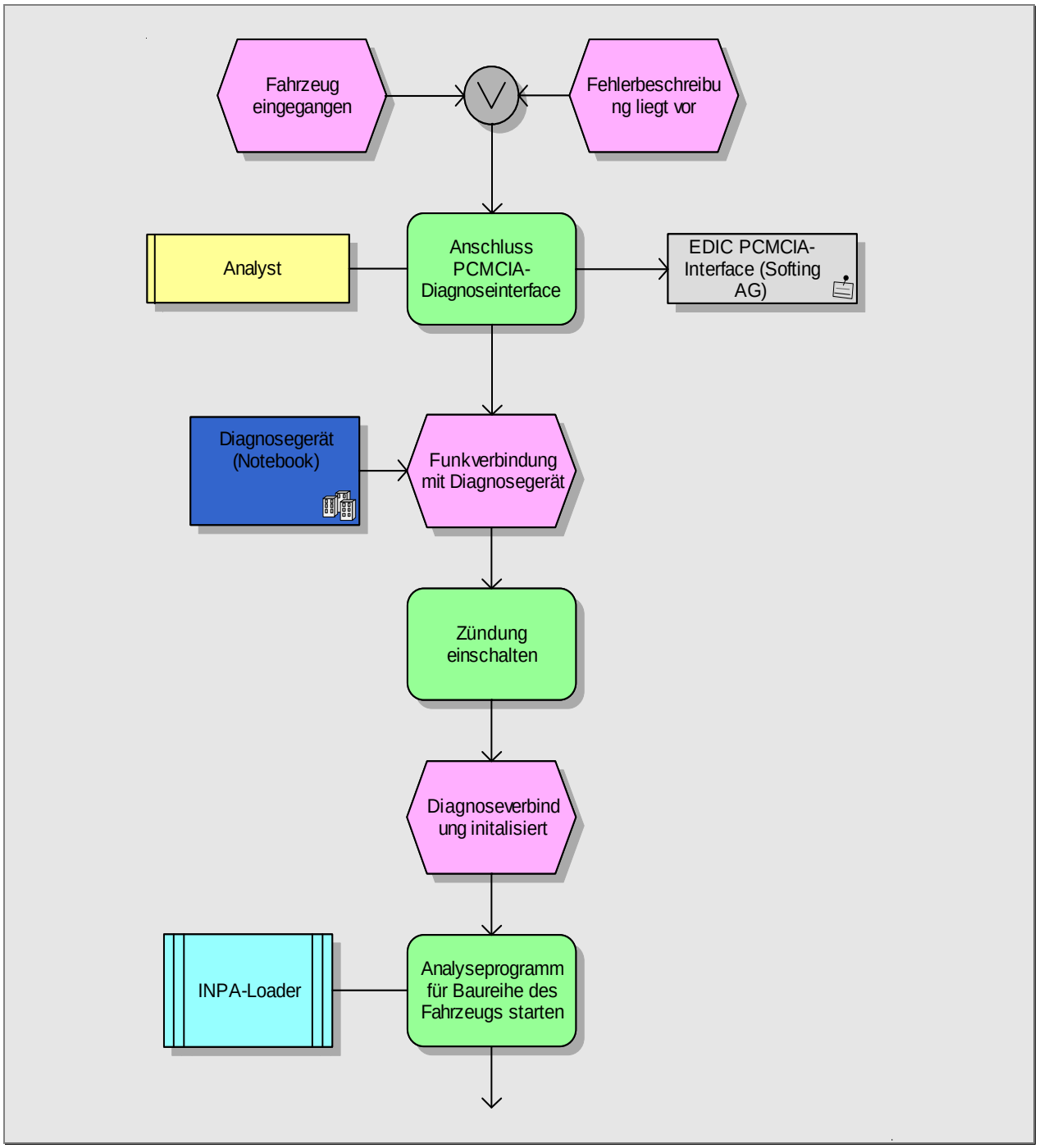
Dabei werden unter anderem folgende Werte abgefragt:

- Allgemeine Fahrzeugdaten (FGNR, Produktionsdatum, Kilometerstand etc.)
- ID-Daten aller Steuergeräte. Dadurch ist erkennbar, ob jedes Steuergerät seine Identifikationsdaten ordnungsgemäß zurückliefert.
- Programmierstatus aller Steuergeräte. Der Status zeigt an, ob das Steuergerät in Betrieb ist.
- MOST (Default/Current Registry)-Daten aller Steuergeräte. Alle auf dem MOST-Protokoll basierende Steuergeräte werden auf ihre Kommunikationsfähigkeit hin überprüft, und bei Möglichkeit werden deren derzeitige Werte abgefragt.
- Fehler-, Info- und History-Speicher aller Steuergeräte. Sie sind für den Analysten von großer Wichtigkeit, da Verhalten und Fehlerzustände der Steuergeräte erfasst werden, die eventuell zu dem Problemfall geführt haben.
- Powermanagement aller Steuergeräte (Stromversorgung, Spannung etc.)
- Getriebe-Daten der dafür verantwortlichen Steuergeräte
- Optional werden z. B. Telefon-, Fensterheber-, ACC- oder Lenkwinkelsensor-Daten gelesen.

Diese lokal gespeicherten Dateien bilden den Grundstock für die Fehleranalyse.

In den meisten der Fälle löscht der Mitarbeiter im Anschluss alle Fehlerprotokolle, um bei der Reproduktion des Fehlers im Rahmen der Analyse sofort darauf aufmerksam gemacht zu werden. Das Diagnoseskript bietet eine entsprechende Funktion, alle Fehlerprotokolle der Steuergeräte auf einmal zu löschen.

Nach dem Löschen ist das Fahrzeug bereit für die eingehende Analysetätigkeit.


Fahrzeug eingegangen
Fehlerbeschreibung liegt vor
Analyst
Anschluss PCMCIA-Diagnoseinterface
EDIC PCMCIA-Interface (Softing AG)
Diagnosegerät (Notebook)
Funkverbindung mit Diagnosegerät
Zündung einschalten
Diagnoseverbindung initalisiert
INPA-Loader
Analyseprogramm für Baureihe des Fahrzeugs starten

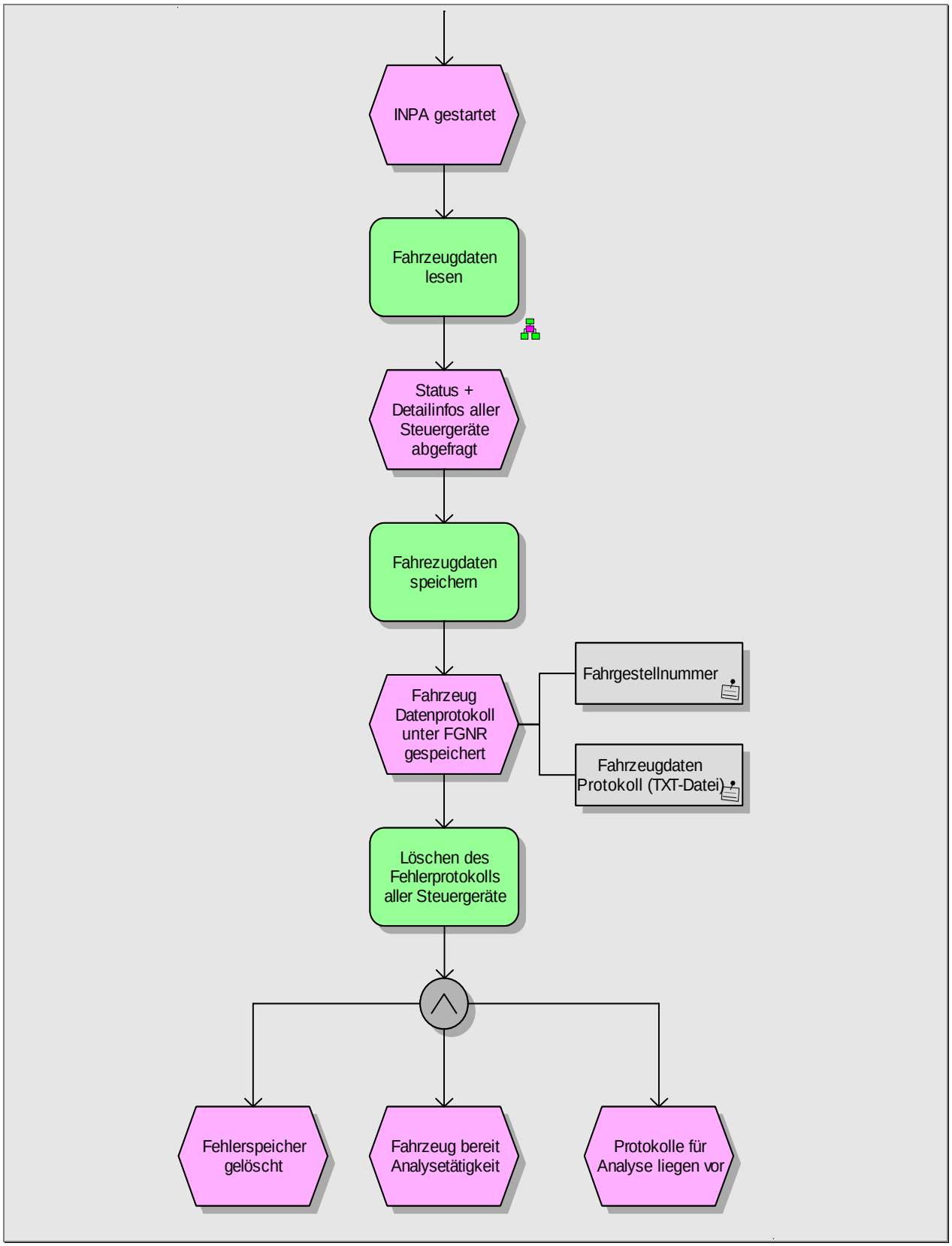


Abbildung 5.9 TP: Analysequellen bereitstellen (EPK)

Name:	TP: Analysequellen bereitstellen
Zuordnung:	GP: Analyseprozess Elektrik/Elektronik
Prozess-verantwortlicher:	Analysemitarbeiter
Input:	Der Anstoß erfolgt innerhalb des Analyseprozesses Elektrik/Elektronik. Voraussetzung für den Anlauf des Prozesses ist eine vorliegende Problembeschreibung (formlos durch Auftraggeber und/oder CASCADE-Fehlercode) im Rahmen des Analyseauftrags und die Anlieferung des betroffenen Fahrzeugs.
Objekte:	- EDIC PCMCIA Interface (Softing AG) - Diagnosegerät (Notebook) - Fahrgestellnummer - Anwendungssystem INPA-Loader
Ergebnisse:	- Fehlerspeicher gelöscht - Fahrzeug bereit für Analysetätigkeit - Fahrzeugdatenprotokoll liegt vor
Output:	Fahrzeugdatenprotokoll (TXT Datei)

Tabelle 5.4 TP: Analysequellen bereitstellen (EPK)

5.8 Erfassung des Verwaltungsprozesses

Zur Untersuchung der Fahrzeuge mit INPA im Nacharbeits- und Analyseprozess sind sogenannte Diagnoseskripte notwendig (siehe Kapitel 3.3.1). Die Verwaltung der Diagnoseskripte erfolgt beim Kooperationspartner von zentraler Stelle.

Die zuständige Abteilung ist dem Ressort Produktion zugeordnet. Sie bietet ihre Dienstleistung aber auch für die Diagnoseskripte an, die in anderen Ressorts zum Einsatz kommen. Die Kernaufgaben der Verwaltung werden im folgenden Kapitel zusammengefasst.

5.8.1 Prozessübersicht

Die Aufträge erhält die Verwaltung von der Nacharbeit, Analyse und SGBD-Entwicklung. Grundsätzlich unterscheiden sich die Aufträge danach, ob ein Diagnoseskript nachbearbeitet oder neu erstellt werden soll.

In den Abteilungen für SGBD-Entwicklung und Analyse werden zum Teil auch Diagnoseskripte selbst erstellt. Diese werden zur Prüfung bzw. Referenzierung an die Verwaltung geschickt. Die Form der Referenzierung und Prüfung ist die gleiche wie in der Verwaltung. Deshalb können sie für die Prozessbeschreibung ausgespart werden. Es ergeben sich somit für die INPA-Systemanalyse keine neuen Anforderungen.

Die Gründe für die Neuerstellung eines Diagnoseskripts liegen meistens in der Neuentwicklung eines Steuergeräts. Ein neues Steuergerät oder dessen zugehörige SGBD (siehe Kapitel 2.3) benötigen ein Diagnoseskript, mit dem sie bei der Entwicklung getestet und in der Nacharbeit untersucht werden können.

Für die Nachbearbeitung eines Diagnoseskripts gibt es allerdings die vielfältigsten Gründe. Ein möglicher Grund ist beispielsweise, dass für die Nacharbeit zusätzliche Statusabfragen oder Steuerbefehle im Rahmen ihrer Diagnosetätigkeit benötigt werden. Denkbar sind auch Fehler in bestehenden Skripten, die bei der Prüfung und Referenzierung nicht erkannt werden. Auch die Änderung einer SGBD kann zur Nachbearbeitung eines Diagnoseskripts führen. Werden z. B. Jobs (siehe Kapitel 2.3) in einer SGBD geändert, hinzugefügt oder gelöscht, so müssen diese Änderungen, falls das Diagnoseskript den Job auch benutzt, entsprechend eingearbeitet werden.

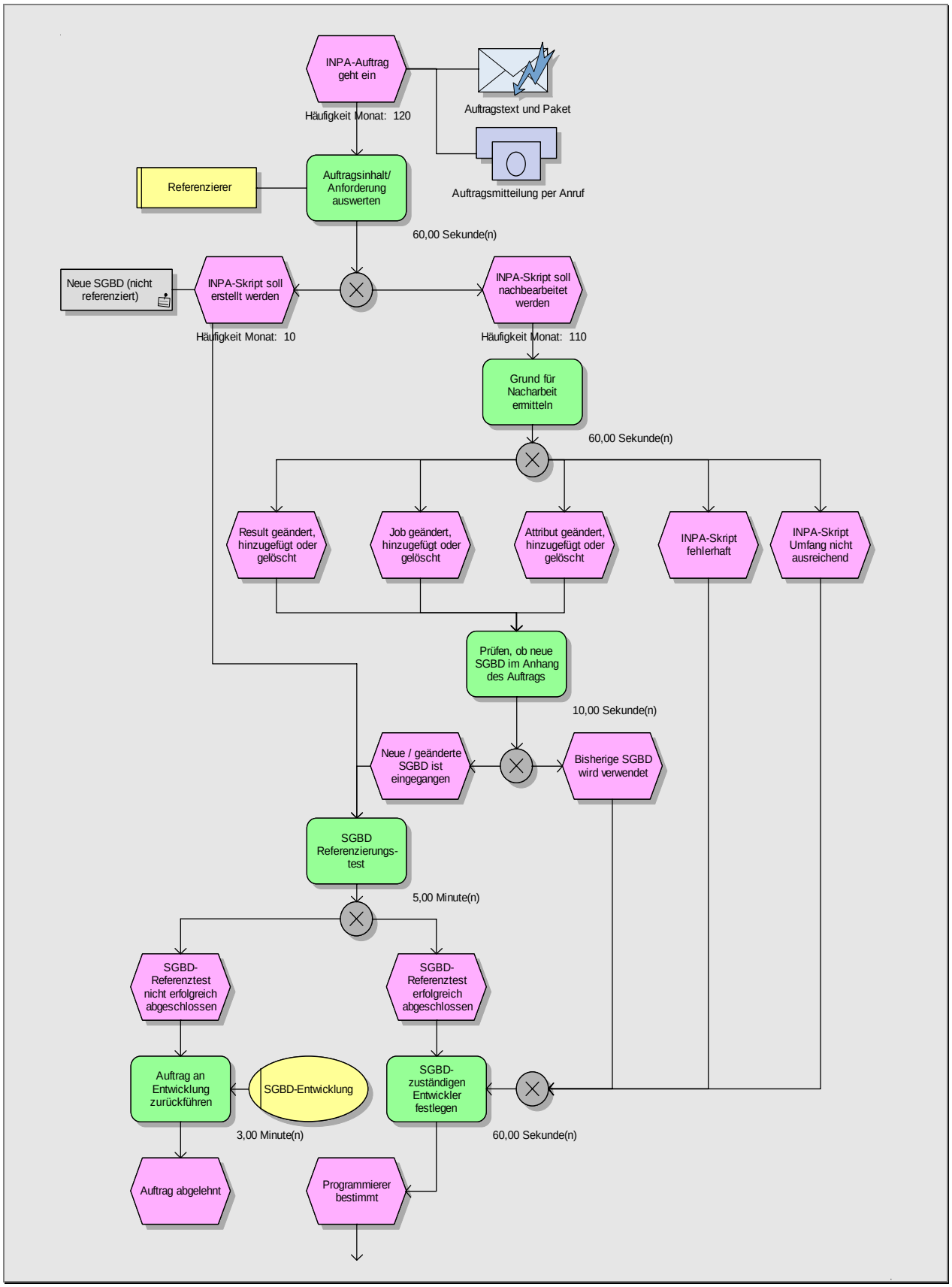
Zum größten Teil sind in der Verwaltung die Zuständigkeiten für eine SGBD und die damit zusammenhängenden Diagnoseskripte einem Programmierer zugeordnet. Die Programmierer werden teilweise auch direkt von dem Diagnostiker (z. B. einem Nacharbeiter) benachrichtigt. In den übrigen Fällen leitet der Referenzierer, der die Aufträge z. B. für Neuentwicklungen annimmt, diese an den zuständigen Programmierer weiter.

Sobald der Programmierer den Auftrag angenommen hat, werden die für INPA relevanten Teilprozesse angestoßen. Je nach Auftrag wird ein Diagnoseskript erstellt oder nachbearbeitet.

Nach der Bearbeitung wird das Diagnoseskript vom Programmierer geprüft. Die Prüfung ist eine Verknüpfung aus funktionalem und formalem Test. Nach erfolgreicher Prüfung durch den Programmierer leitet dieser das Diagnoseskript zur Referenzierung an den zuständigen Verwaltungsmitarbeiter weiter.

Der Test in der Referenzierung ist mehr formaler Natur und größtenteils automatisiert. Bei den Testabläufen kommt INPA selbst nicht zum Einsatz. Deshalb wird der Referenzierungstest in dieser Arbeit nicht näher erläutert. Eine ausführliche Dokumentation findet sich in INPA-Analyse 2007, S. 24. Ist dieser Test erfolgreich, entstehen aus den Testabläufen kompilierte Diagnoseskripte. Diese sogenannten IPO-Dateien werden auf einem internen Server abgelegt ("referenziert") und können von dort von den autorisierten Abteilungen bezogen werden. Die IPO-Dateien sind die eigentlichen ladbaren Diagnoseskripte (siehe Kapitel 2.4.3).

Die beiden Prozesse in der Diagnoseskriptverwaltung, bei denen INPA eine Schlüsselrolle einnimmt, werden in den folgenden Kapiteln weiter erläutert.


INPA-Auftrag geht ein
Auftragstext und Paket
Häufigkeit Monat: 120
Auftragsmitteilung per Anruf
Referenzierer
Auftragsinhalt/ Anforderung auswerten
60,00 Sekunde(n)
Neue SGBD (nicht referenziert)
INPA-Skript soll erstellt werden
INPA-Skript soll nachbearbeitet werden
Häufigkeit Monat: 10
Häufigkeit Monat: 110
Grund für Nacharbeit ermitteln
60,00 Sekunde(n)
Result geändert, hinzugefügt oder gelöscht
Job geändert, hinzugefügt oder gelöscht
Attribut geändert, hinzugefügt oder gelöscht
INPA-Skript fehlerhaft
INPA-Skript Umfang nicht ausreichend
Prüfen, ob neue SGBD im Anhang des Auftrags
10,00 Sekunde(n)
Neue / geänderte SGBD ist eingegangen
Bisherige SGBD wird verwendet
SGBD Referenzierungs-test
5,00 Minute(n)
SGBD-Referenztest nicht erfolgreich abgeschlossen
SGBD-Referenztest erfolgreich abgeschlossen
Auftrag an Entwicklung zurückführen
SGBD-Entwicklung
SGBD-zuständigen Entwickler festlegen
3,00 Minute(n)
60,00 Sekunde(n)
Auftrag abgelehnt
Programmierer bestimmt

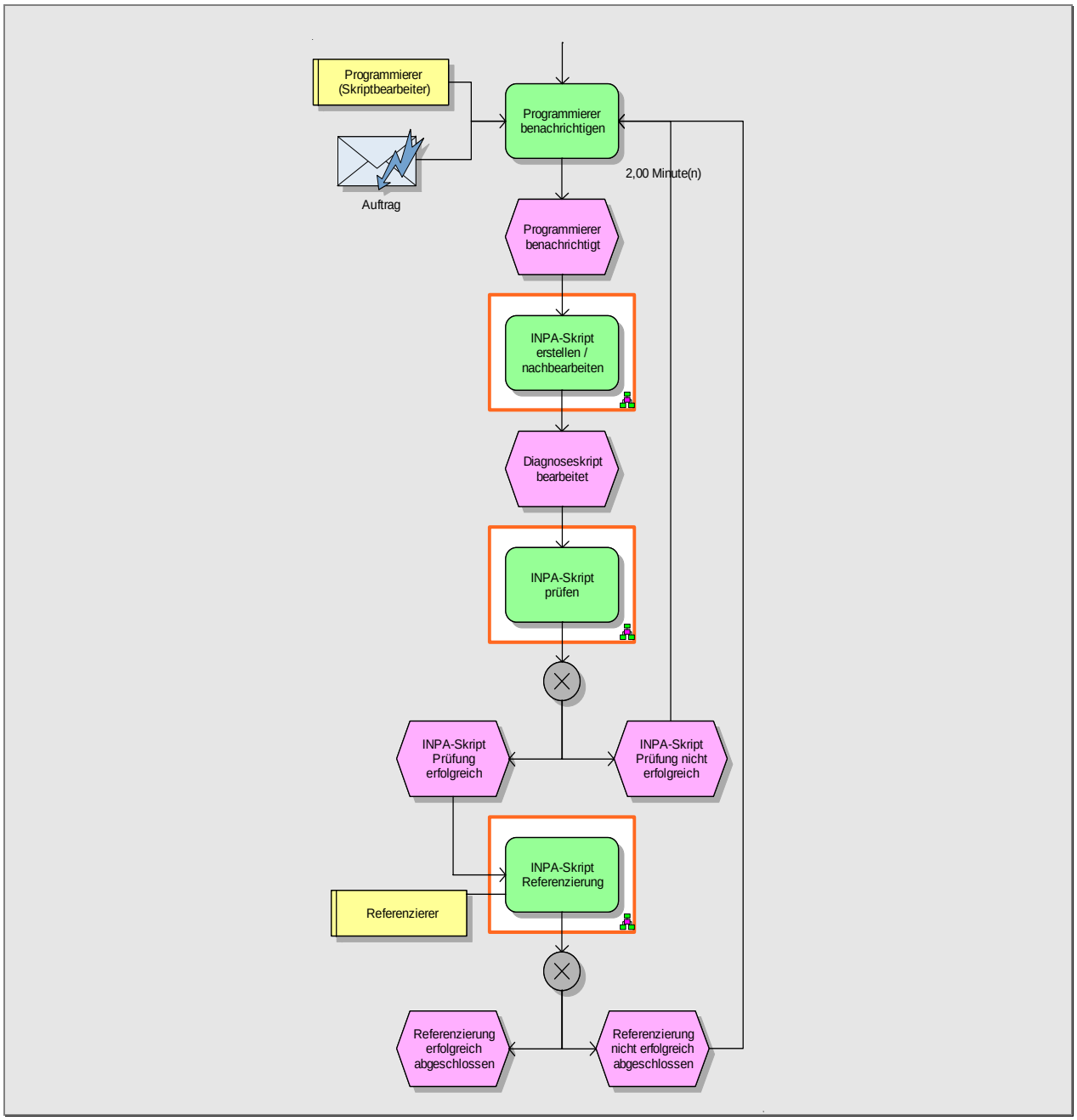


Abbildung 5.10 Sek. GP: INPA-Skript-Verwaltungsprozess (EPK)

Name:	INPA-Skript-Verwaltungsprozess
Zuordnung:	KP: Kundenorientierter Vertriebsprozess, Ressort: Produktion, Bereich: Montage
Prozess-verantwortlicher:	- Programmierer (Skriptbearbeiter) - Referenzierer
Input:	Der Auftrag zur Erstellung oder Nachbearbeitung für ein INPA-Skript kann aus folgenden Prozessen erfolgen: - Nacharbeitsprozess - Analyseprozess Elektronik / Elektrik

	- Entwicklungsprozess SGBD
Objekte:	- Auftragsmitteilung per Fax, E-Mail, Telefon etc. - Auftragstext - Paket (Erstellte SGBD, INPA-Skripte zur Referenzierung etc.)
Ergebnisse:	Referenzierung erfolgreich abgeschlossen
Output:	Referenzierte INPA-Skriptquelle (*.SRC) Datei und ladbares INPA-Skript (*.IPO)

Tabelle 5.5 Sek. GP: INPA-Skript-Verwaltungsprozess (EPK)

5.8.2 Diagnoseskript erstellen/nachbearbeiten

Bei der Bearbeitung von Diagnoseskripten werden grundsätzlich zwei Typen von Aufträgen unterschieden:

ein bestehendes Skript muss nachbearbeitet werden oder

ein Diagnoseskript muss neu erstellt werden, z. B. für eine neu eingeführte SGBD.

Im Fall der Nachbearbeitung kopiert der zuständige Programmierer die freigegebene Quelldatei (SRC-Datei) aus dem X:/REFERENZ-Verzeichnis in seine lokale INPA-Umgebung. Damit wird sichergestellt, dass Programmierer und Auftraggeber, z. B. der Mechaniker in der Nacharbeit auf dem gleichen Stand sind. Per Telefonat können noch zusätzlich die Version, der Name der Quelle und des geladenen Skripts (IPO-Datei) verglichen werden.

Falls ein Diagnoseskript neu erstellt werden muss, kann der Programmierer eine bereits bestehende Skriptquelle einer ähnlichen Baureihe kopieren oder eine Diagnoseskriptvorlage als Ausgangspunkt verwenden. Die Vorlage bietet dem Programmierer ein vorbearbeitetes Skript, in dem z. B. bereits die Funktionstasten (F1-F20) mit den üblich verwendeten Funktionen hinterlegt sind sowie diverse Standard-Diagnosemasken (Informationsmaske, Druckmaske etc.) bereits implementiert sind.

Hat der Entwickler die entsprechende Diagnoseskriptquelle in seiner Entwicklungsumgebung geöffnet, erfolgt die Ergänzung bzw. Verbesserung der Funktionalität. Dieser Prozess wird im anschließenden Modell näher erläutert.

Hat der Programmierer alle geforderten funktionellen Bearbeitungen durchgeführt, endet dieser Teilprozess. Das aktualisierte Skript wird im Rahmen des Verwaltungsprozesses an den Teilprozess "INPA-Skript prüfen" weitergegeben.

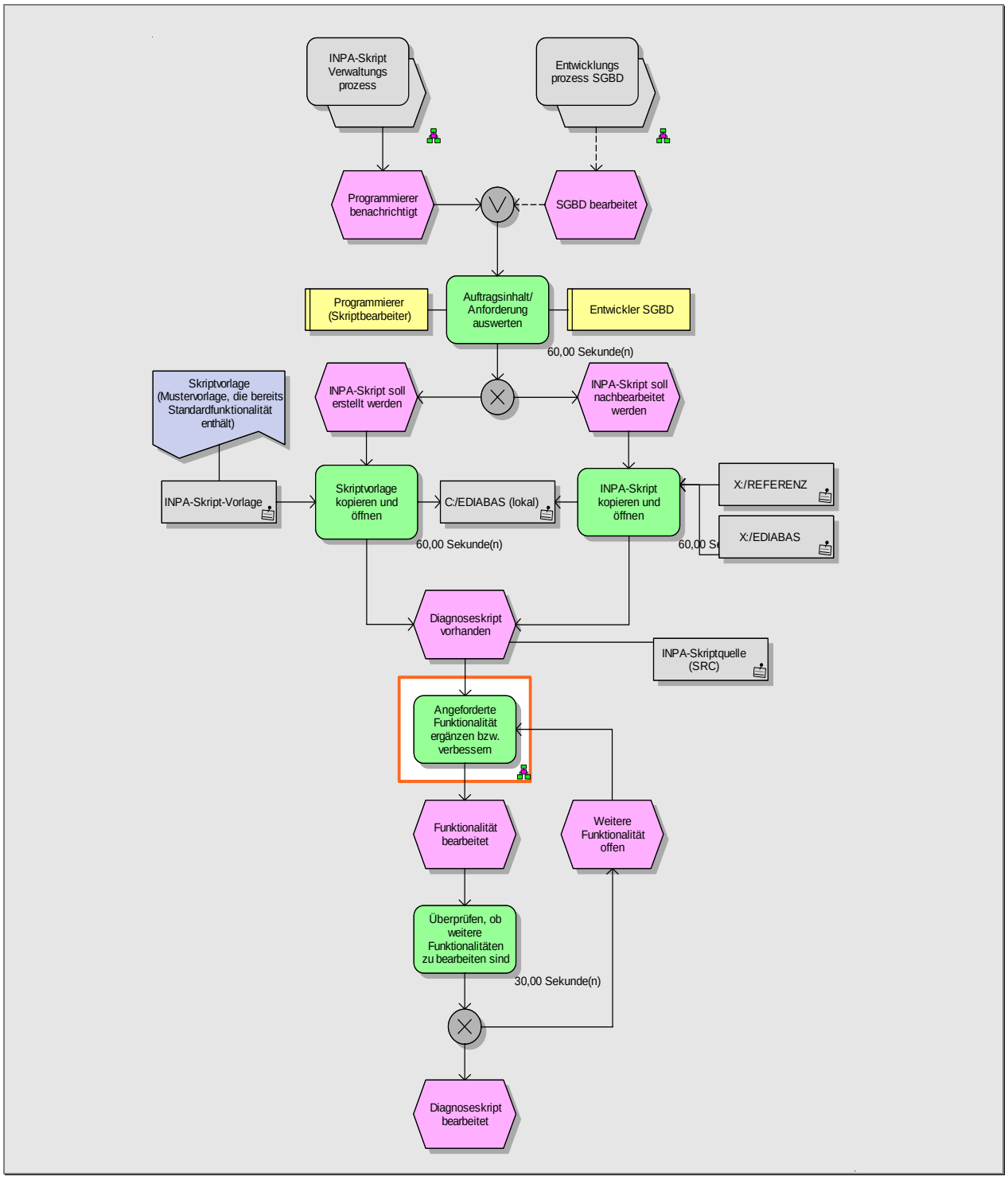


Abbildung 5.11 TP: INPA-Skript erstellen/nachbearbeiten (EPK)

Name:	Teilprozess: INPA-Skript erstellen/nachbearbeiten
Zuordnung:	GP: INPA-Skript Verwaltungsprozess
Prozess-verantwortlicher:	Programmierer (Skriptbearbeiter)

Input:	Der Anstoß erfolgt innerhalb des Verwaltungsprozesses. In der Regel geht ein Auftrag beim Referenzierer ein. Dieser beauftragt daraufhin den zuständigen Programmierer mit der Neuerstellung oder Nachbearbeitung eines INPA-Skripts. Es sind aber auch Aufträge per E-Mail oder Telefon möglich, die aufgrund eindeutiger Zuständigkeit für eine Baureihe oder SGBD direkt bei dem Programmierer aufgegeben werden.
Objekte:	- X:/REFERENZ (Interne Ablageverzeichnisse für freigegebene INPA-Skripte) - X:/EDIABAS (Interne Ablageverzeichnisse für freigegebene SGBDs) - C:/EDIABAS (Lokale Installation des EDIABAS Servers und des INPA Toolsets32) - INPA-Skriptquelle (SRC) - INPA-Skript Vorlage
Ergebnisse:	Diagnoseskript aktualisiert bzw. erstellt
Output:	INPA-Skriptquelle (*.SRC) Datei und ladbares INPA-Skript (*.IPO)

Tabelle 5.6 TP: INPA-Skript erstellen/nachbearbeiten (EPK)

5.8.3 Angeforderte Funktionalität ergänzen/verbessern

Sobald die Diagnoseskriptquelle vom Referenz-Server kopiert wurde (siehe Kapitel 5.8.2) öffnet der Programmierer die Entwicklungsumgebung. Diese besteht aus vier monolithen Anwendungssystemen (Texteditor, EDIABAS Toolset32, INPA-Loader und SGBD INPA Dienste-Oberfläche).

Um eine Steuergerätefunktionalität im Skript zu ergänzen, werden der Jobname, dessen Aufrufparameter (Attribute) und Ergebnisformate (Result-Sets) benötigt. Mit dem EDIABAS Toolset32 kann der Programmierer die genaue Bezeichnung des SGBD-Jobs sowie dessen Attribut und Result-Sets ermitteln.

Auf die Abhängigkeiten und fachlichen Hintergründe von Jobs, Attributen und Results wird bei der Prozessanalyse in Kapitel 6 noch weiter eingegangen.

Die Diagnoseskriptquelle selbst wird mit einem handelsüblichen Texteditor geöffnet.

Da keine interne Schnittstelle zwischen Texteditor und Toolset32 besteht, muss der Programmierer die Bezeichnungen von Jobs, Attributen und Result-Sets per Copy & Paste oder per Tastatureingabe in die Diagnoseskriptquelle übertragen.

Neben den inhaltlichen Anpassungen ist bei einer funktionellen Nacharbeit auch die grafische Gestaltung der Diagnoseoberfläche durchzuführen. Auf die Möglichkeiten, die INPA hierfür bietet, wird bei der Analyse anhand von Anwendungsfällen in Kapitel 6 eingegangen. Dort werden die Variationen der grafischen Objekte näher dargestellt. An dieser Stelle soll gezeigt werden, wie diese eingebunden und positioniert werden.

Da INPA über keinen grafischen Editor verfügt, erfolgt die Positionierung über den Texteditor. Der Aufbau einer Diagnoseoberfläche erfolgt auf sogenannte Lines (Zeilen), auf denen über

Stellen die Position der grafischen Objekte bestimmt wird. Die Größe der grafischen Objekte ist meistens fest vorgegeben. Der Programmierer gibt zum Erstellen eines bestimmten grafischen Objekts nur die Position ein.

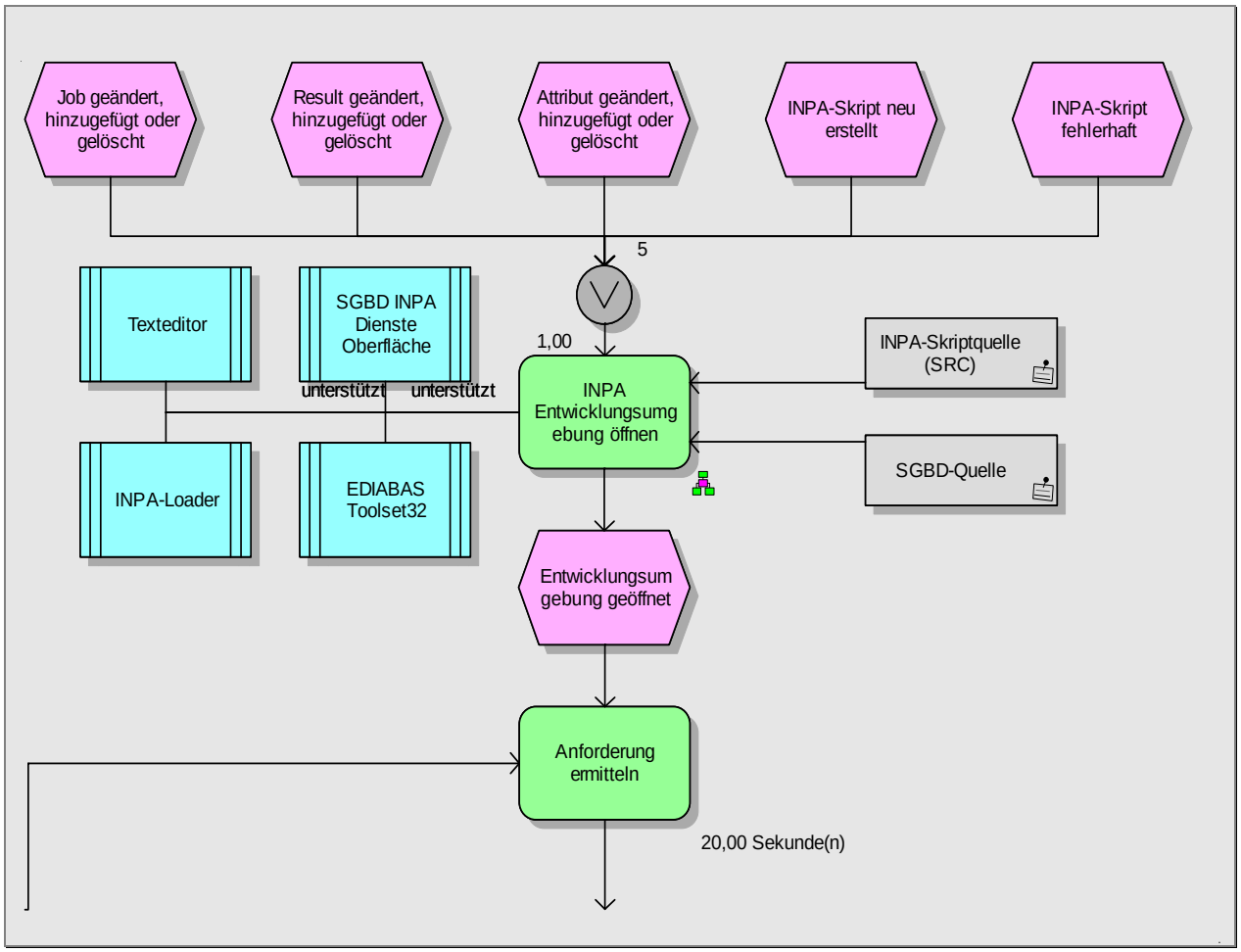
Nach der Bearbeitung der Diagnoseskriptquelle (SRC-Datei) muss diese kompiliert werden. Dies erfolgt mit der Software "SGBD INPA-Dienste Oberfläche". Sofern der Quellcode fehlerfrei kompiliert werden kann, wird eine IPO-Datei ausgegeben. Dieses ladbare Diagnoseskript kann nun mit dem INPA-Loader geöffnet werden. Der INPA-Loader ist das Anwendungssystem, mit dem auch die Nacharbeit, Analyse und Entwicklung die Diagnoseskripte startet. Der Programmierer kann in dem gestarteten Diagnoseskript die Gestaltung der einzelnen Diagnoseoberflächen überprüfen. Falls die Positionierung nicht in Ordnung ist, führt der Prozess wieder zur Bearbeitung der Quelle mit dem Texteditor zurück.

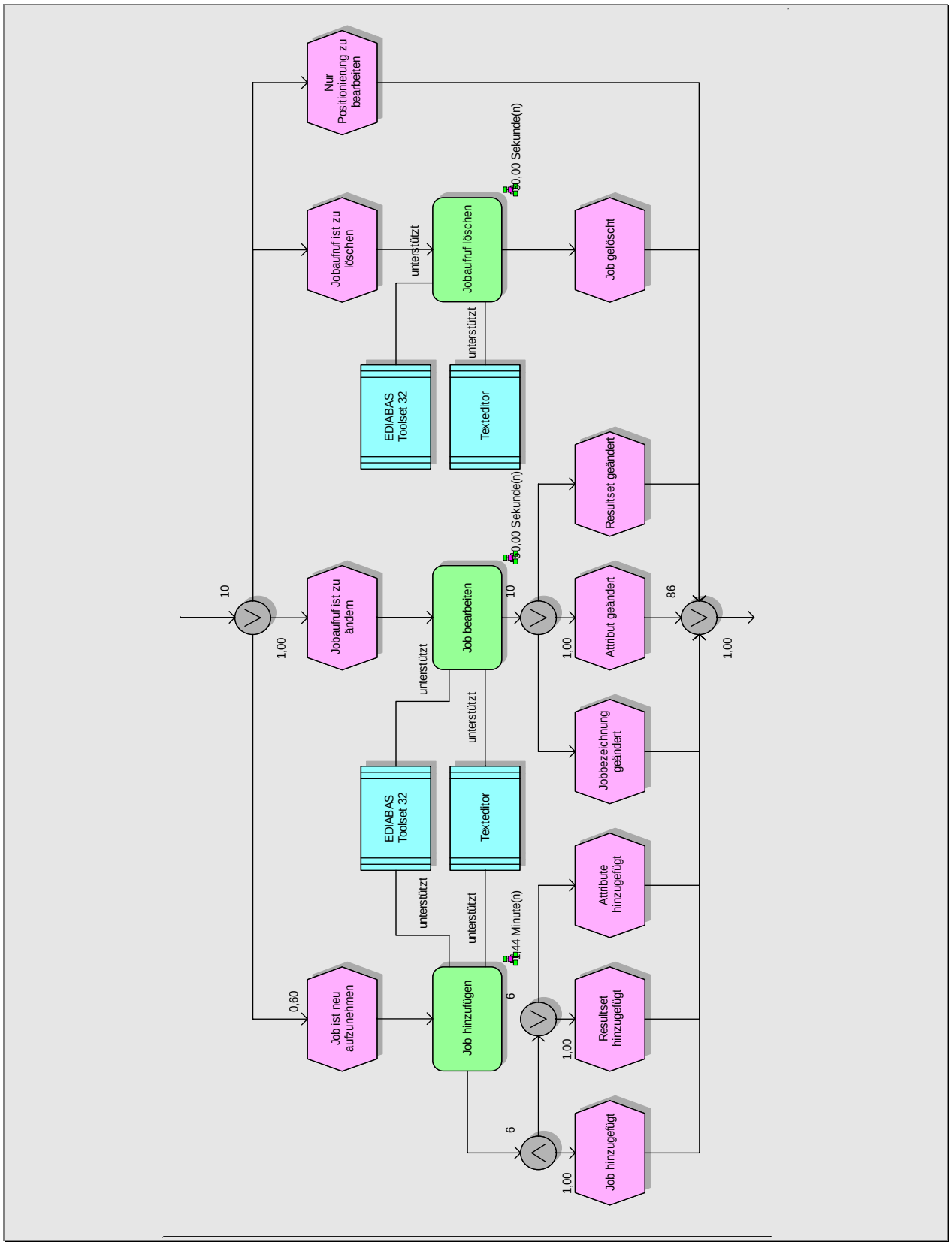
Nach Abschluss der Positionierung und erfolgreicher Kompilierung des Diagnoseskripts überprüft der Programmierer die Funktionalität der Oberfläche. Dabei ist zu erwähnen, dass während des Ablaufs der Positionierung grobe funktionale Fehler alleine durch das Kompilieren der Quelle bereits auffallen und ausgebessert werden können.

Der funktionale Test ist zum größten Teil ein optischer Test. Die Diagnoseoberfläche wird im INPA-Loader betrachtet und dort wird auch überprüft, ob sie für diverse Vorgabewerte die gewünschte Anzeige liefert. Auch hier führt der Weg wieder zurück zur Textbearbeitung der Diagnoseskriptquelle, falls funktionale Fehler zu korrigieren sind.

Sofern das Skript in einem nicht deutschsprachigen Werk eingesetzt werden soll, erfolgt nach dem erfolgreichem Abschluss der funktionalen Prüfung noch die Erweiterung bezüglich Mehrsprachigkeit. Hierzu muss pro Sprache jeweils eine zusätzliche IPO-Datei erstellt werden.

Nach Ergänzung der Mehrsprachigkeit ist die funktionale Ergänzung bzw. Verbesserung abgeschlossen und der Prozess führt wieder in den Teilprozess "INPA-Skript erstellen/nachbearbeiten" zurück. Falls hier festgestellt wird, dass noch weitere Funktionalitäten zu bearbeiten sind, beginnt der Prozess wieder von neuem.


Job geändert, hinzugefügt oder gelöscht
Result geändert, hinzugefügt oder gelöscht
Attribut geändert, hinzugefügt oder gelöscht
INPA-Skript neu erstellt
INPA-Skript fehlerhaft
5
Texteditor
SGBD INPA Dienste Oberfläche
1,00
INPA-Skriptquelle (SRC)
unterstützt
unterstützt
INPA Entwicklungsumg ebung öffnen
SGBD-Quelle
INPA-Loader
EDIABAS Toolset32
Entwicklungsum gebung geöffnet
Anforderung ermitteln
20,00 Sekunde(n)


Nur Positionierung zu bearbeiten
Jobaufruf ist zu löschen
unterstützt
Jobaufruf löschen
30,00 Sekunde(n)
Job gelöscht
unterstützt
EDIABAS Toolset 32
Texteditor
Resultset geändert
30,00 Sekunde(n)
10
Jobaufruf ist zu ändern
Job bearbeiten
10
Attribut geändert
86
1,00
1,00
1,00
unterstützt
unterstützt
Jobbezeichnung geändert
EDIABAS Toolset 32
Texteditor
Attribute hinzugefügt
unterstützt
unterstützt
144 Minute(n)
0,60
Job ist neu aufzunehmen
Job hinzufügen
6
Resultset hinzugefügt
1,00
6
Job hinzugefügt
1,00

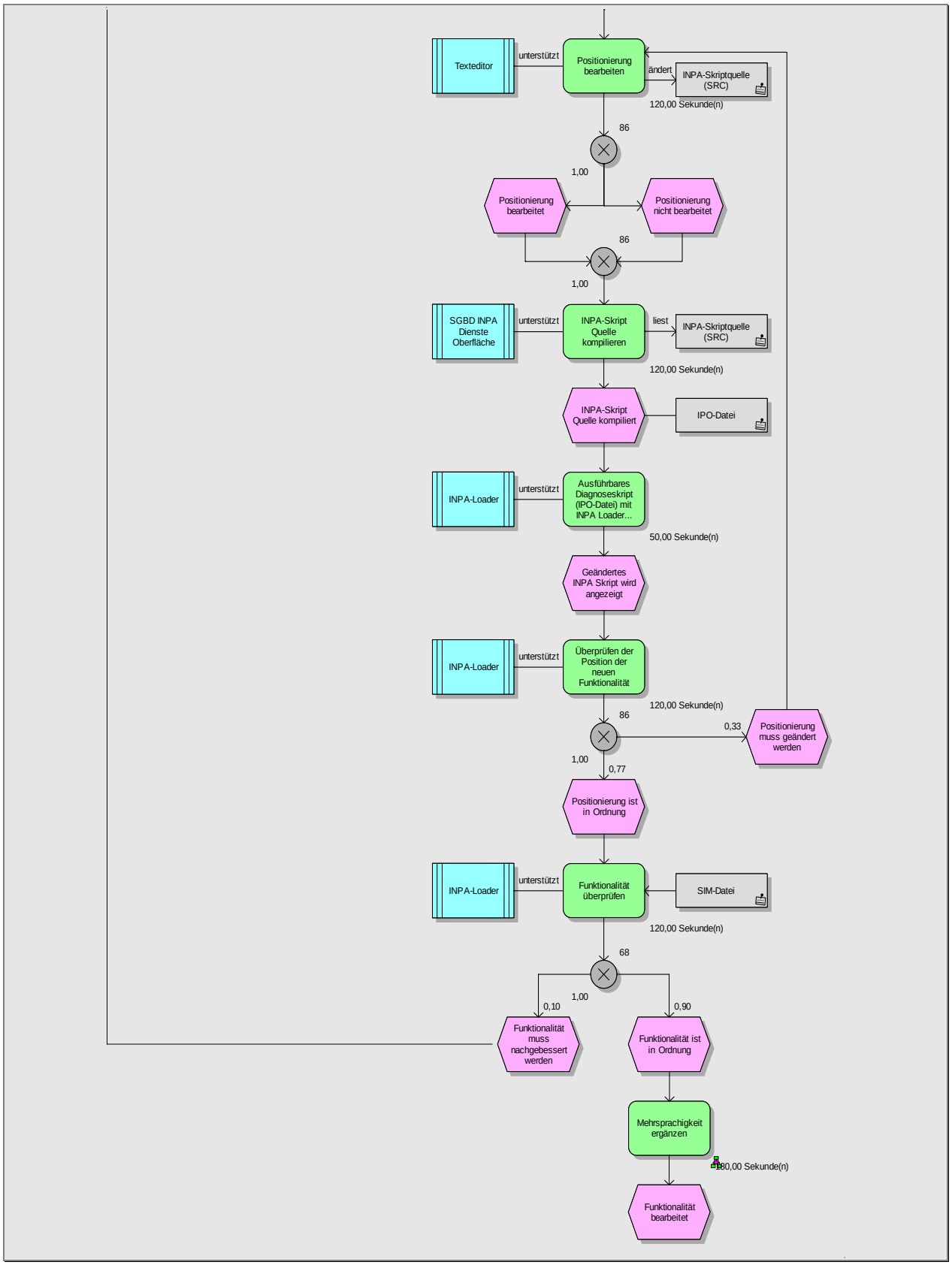


Abbildung 5.12 Arbeitsschritt: Angeforderte Funktionalität ergänzen/verbessern (EPK)

Name:	Prozess: Angeforderte Funktionalität ergänzen/verbessern
Zuordnung:	GP: INPA-Skript-Verwaltungsprozess TP: INPA-Skript erstellen/nachbearbeiten
Prozess-verantwortlicher:	Programmierer (Skriptbearbeiter)
Input:	Der Prozess wird durch den Teilprozess "INPA-Skript erstellen/nachbearbeiten" angestoßen (Kapitel 5.8.2). Der Teilprozess stellt als Input die zu bearbeitende INPA-Skriptquelle bereit.
Objekte:	- INPA-Skriptquelle (*.SRC-Datei) - Diagnoseskript / INPA-Skript (*.IPO-Datei) - Anwendungssystem Texteditor - Anwendungssystem EDIABAS Toolset32 - Anwendungssystem INPA-Loader - Anwendungssystem SGBD INPA Dienste Oberfläche
Ergebnisse:	Eine Funktionalität ergänzt bzw. verbessert
Output:	I NPA Skriptquelle (*.SRC) Datei und ladbares INPA-Skript (*.IPO)

Tabelle 5.7 Arbeitsschritt: Angeforderte Funktionalität ergänzen/verbessern (EPK)

5.9 Erfassung des Entwicklungsprozesses

Die hier abgebildete Form des SGBD-Entwicklungsprozesses stellt eine abstrahierte Version der in Standard GS 95013[131] beschriebenen Prozessabläufe dar, der die Prozesse zur Erstellung, Pflege und Absicherung von Steuergeräte-Beschreibungsdateien (SGBD) allgemein beschreibt[132]. Die Vereinfachung dient dazu, die Verwendung des INPA-Systems herauszustellen.

Aus Sicht des INPA-Systems ist bei der Entwicklung von Steuergeräten vorrangig die Verarbeitung der SGBDen (siehe Kapitel 3.2) von Belang. Im Hinblick darauf erläutert das folgende Kapitel den SGBD-Entwicklungsprozess.

5.9.1 Prozessübersicht

Die SGBD bietet die Möglichkeit, um über das Basissystem, auf das Steuergerät zuzugreifen (siehe Kapitel 2.3). Falls ein Steuergerät neu entwickelt oder dessen Funktionalität geändert wird, muss auch die zugehörige SGBD erstellt bzw. bearbeitet werden.

Aufträge für die Neuerstellung einer SGBD kommen daher meistens aus dem Fahrzeugprojekt-Bereich (Systemdesign). Die Anforderungen, SGBDen zu bearbeiten, können vielseitig aus der Nacharbeit, Analyse oder aus dem Servicebereich kommen.

[131] Vgl. SGBD 2003, GS 95013
[132] Der Quellverweis findet sich in dem Fachkonzept "Prozessanalyse des Ablaufsystems INPA" (Quelle: INPA-Analyse 2007)

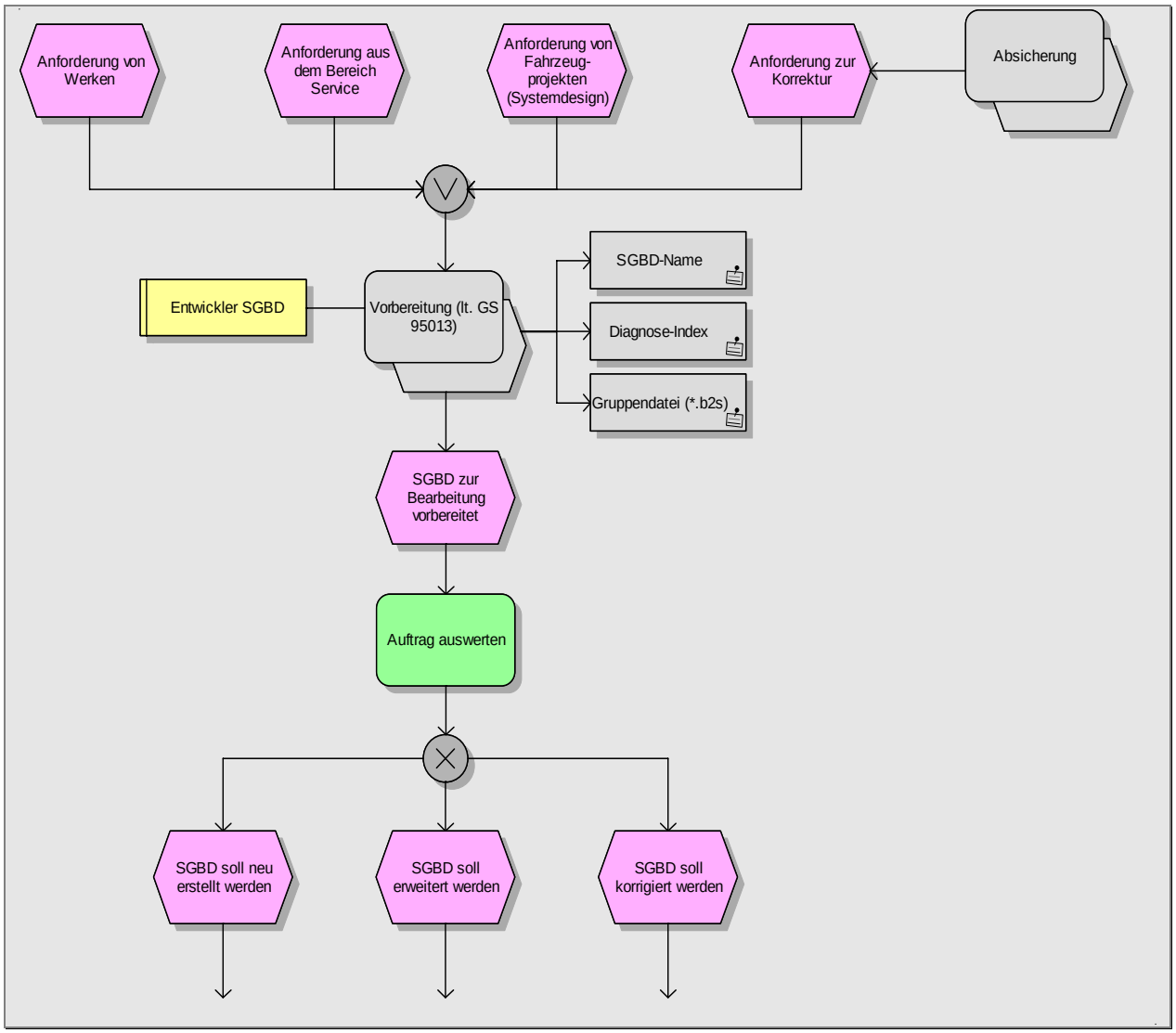
Erfolgt ein Auftrag zur Neuerstellung an die Entwicklungsabteilung, wird im Rahmen der Vorbereitungsphase zunächst der Name des Steuergeräts bestimmt. Dieser wird in Referenzlisten indiziert, um Missverständnisse auszuschließen. Aus dem Namen des Steuergeräts ergibt sich die Bezeichnung für die SGBD (siehe Kapitel 2.3).

Falls eine bestehende SGBD erweitert oder korrigiert werden soll, werden deren Quellen von dem offiziellen Referenz-Server zur Nachbearbeitung geladen. Die folgende Bearbeitung der SGBD-Datei, bei der auch das INPA-System zum Einsatz kommt, wird in Kapitel 5.9.2 beschrieben.

Nach der Bearbeitung wird die SGBD einer formalen Prüfung unterzogen. Dies erfolgt mit der Software SGBD Generator/Compiler.

Im Anschluss erfolgt die Laborprüfung der SGBD am Steuergerät selbst. Verläuft auch diese erfolgreich, beantragt der SGBD-Entwickler die Referenzierung der SGBD. Jede referenzierte SGBD durchläuft danach den Systemabsicherungsprozess (siehe GS 95013[133]).

[133] Vgl. SGBD 2003, GS 95013


Anforderung von Werken
Anforderung aus dem Bereich Service
Anforderung von Fahrzeug-projekten (Systemdesign)
Anforderung zur Korrektur
Absicherung
Entwickler SGBD
Vorbereitung (lt. GS 95013)
SGBD-Name
Diagnose-Index
Gruppendatei (*.b2s)
SGBD zur Bearbeitung vorbereitet
Auftrag auswerten
SGBD soll neu erstellt werden
SGBD soll erweitert werden
SGBD soll korrigiert werden

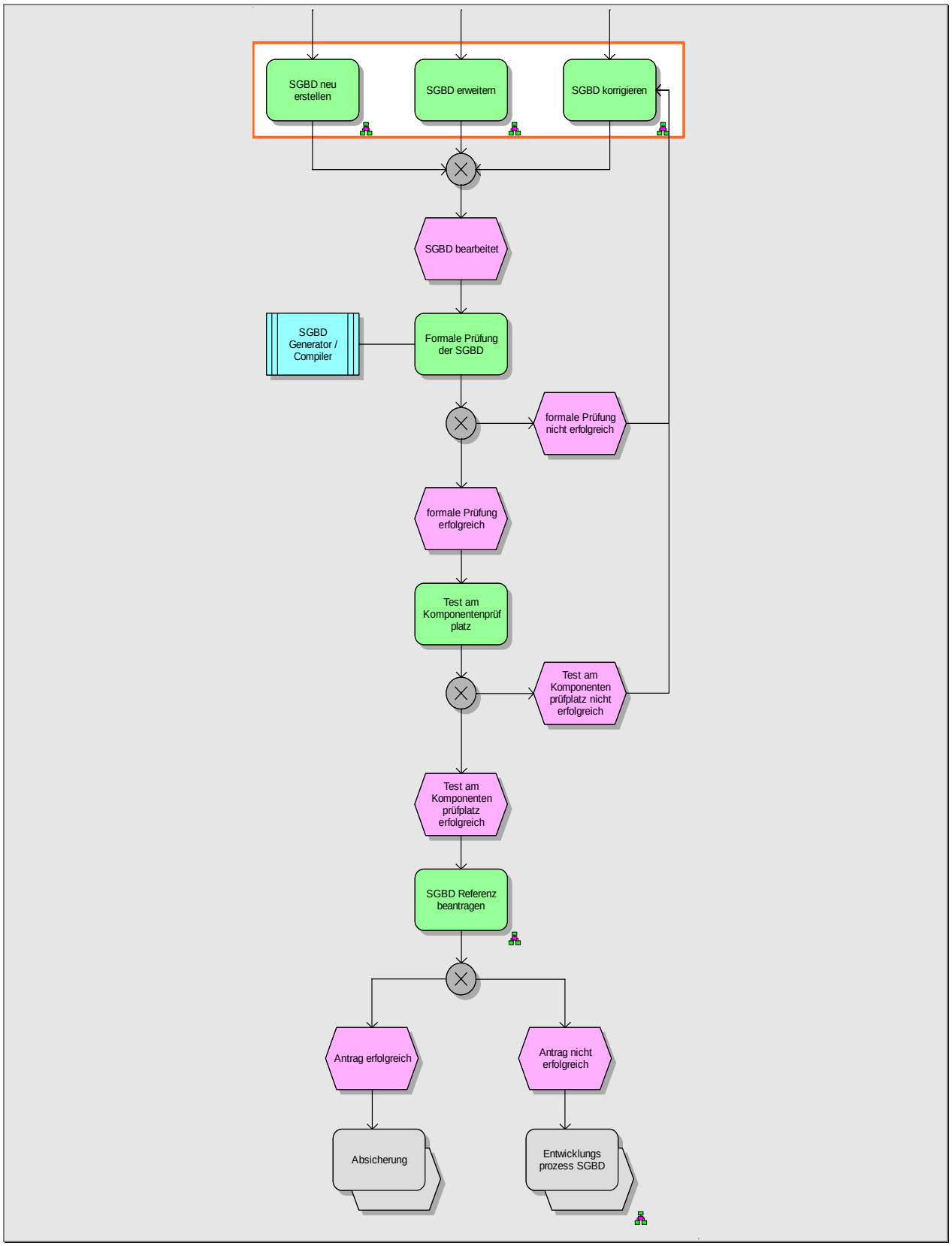


Abbildung 5.13 Sek. GP: SGBD-Entwicklungsprozess (EPK)

Name:	GP: SGBD-Entwicklungsprozess
Zuordnung:	KP: Produktentstehungsprozess (PEP), Ressort Entwicklung, Bereich Elektrik/Elektronik
Prozess-verantwortlicher:	SGBD-Entwickler
Input:	Der Anstoß erfolgt durch: - standardisierte Diagnoseaufträge - Diagnosevorlagen
Objekte:	- SGBD-Name - Diagnose-Index - Gruppendatei - Anwendungssystem SGBD Generator / Compiler
Ergebnisse:	- Referenzierte SGBD - Gruppendatei
Output:	- Ausgefüllte Diagnosevorlagen (Beschreibung von Fehlercodes, Diagnoseaufträgen, etc.) - SGBD und Gruppendatei

Tabelle 5.8 Sek. GP: SGBD-Entwicklungsprozess (EPK)

5.9.2 SGBD erstellen/erweitern/korrigieren

Soll eine SGBD neu erstellt werden, stehen dem Entwickler Mustervorlagen sowie standardisierte Bibliotheken zur Verfügung. Im Fall einer Korrektur oder Erweiterung kann er die Quelle der jeweiligen SGBD vom Referenz-Server laden.

Mittels Texteditor und dem SGBD-Generator ergänzt der SGBD-Entwickler die entsprechende Funktionalität in der SGBD.

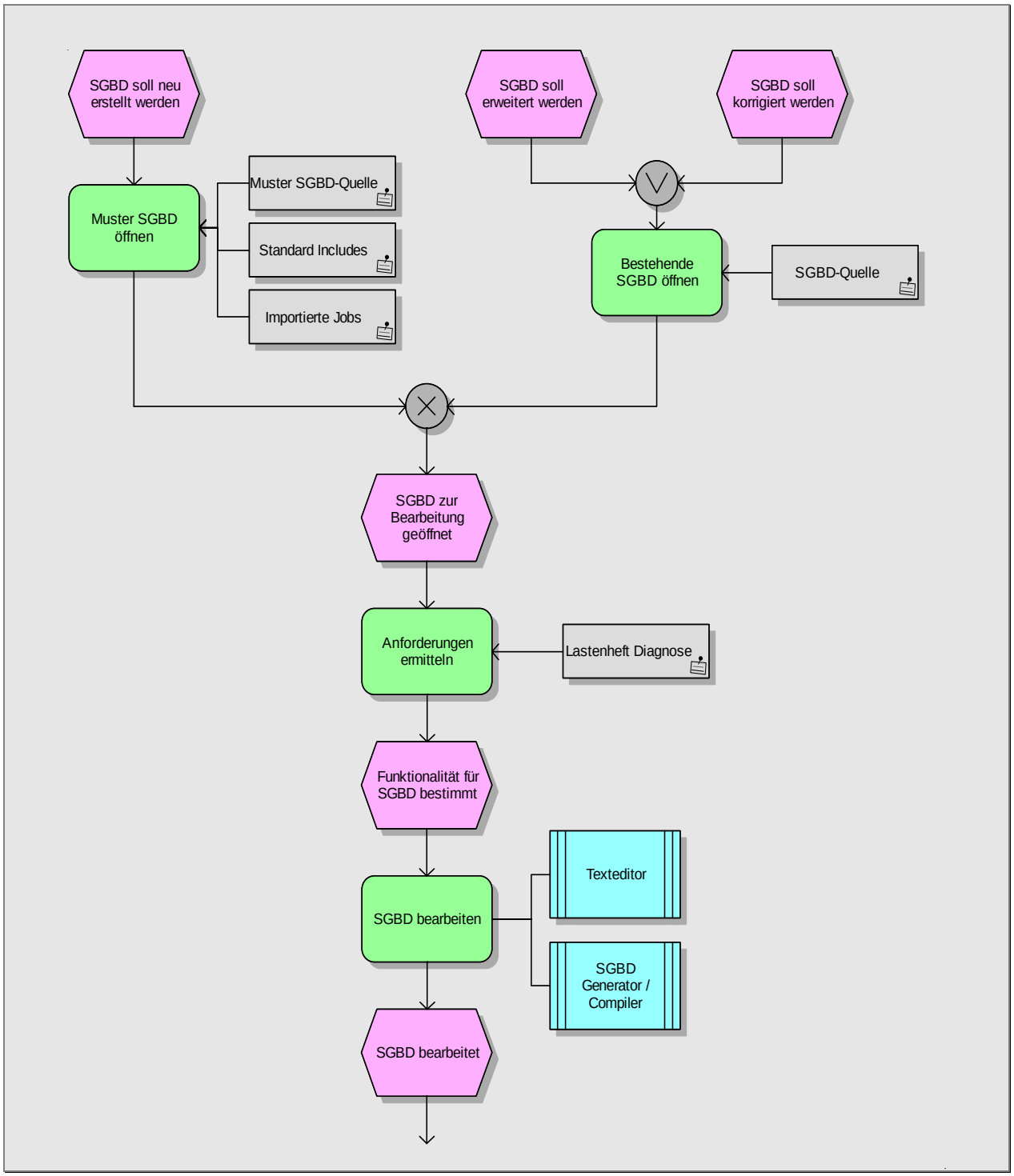
Wird eine neue SGBD erstellt, muss auch das Diagnoseskript neu erstellt werden. Dies passiert zum Teil durch die Verwaltung (siehe Kapitel 5.8) oder auch in dem Bereich der SGBD-Entwicklung selbst.

Der Ablauf der Neuerstellung von Diagnoseskripten soll deshalb an dieser Stelle nicht beschrieben werden, da er sich mit dem in Kapitel 5.8.2 beschriebenen Ablauf nahezu deckt.

Ähnliches gilt bei der Korrektur oder Erweiterung der Steuergerätefunktionalität. Ist die SGBD zu korrigieren bzw. zu erweitern, müssen auch die entsprechenden Job-Aufrufe des zugehörigen Diagnoseskripts angepasst werden. Die Abläufe zur Nachbearbeitung des Diagnoseskripts decken sich mit denen, die in Kapitel 5.8.2 und 5.8.3 beschrieben werden. Sie werden deshalb in diesem Kapitel ausgespart. Die geplanten Verbesserungen in den Referenzprozessen der Verwaltung (Kapitel 5.8) werden in diesem Bereich ihre Vorteile in gleicher Weise zur Geltung bringen. Eine ausführliche Dokumentation der SGBD-Entwicklung findet sich in INPA-Analyse 2007, S. 47-48.

Um eine geänderte Funktionalität der SGBD zu testen, bedient sich der SGBD Entwickler erneut des Diagnoseskripts. Muss die SGBD nachgebessert werden, müssen folglich die Änderungen auch im Diagnoseskript vorgenommen werden. Da die SGBD auf diese Art schrittweise erstellt und getestet wird, muss innerhalb eines Auftrags das Diagnoseskript mehrmals nachgebessert werden. Der in Kapitel 5.8.3 beschriebene Prozess wiederholt sich sehr oft.

Nach erfolgreichem Test der SGBD mittels Diagnoseskript durchläuft die SGBD die weiteren Stationen (formaler Test etc.) des SGBD-Entwicklungsprozesses (siehe Kapitel 2.6).



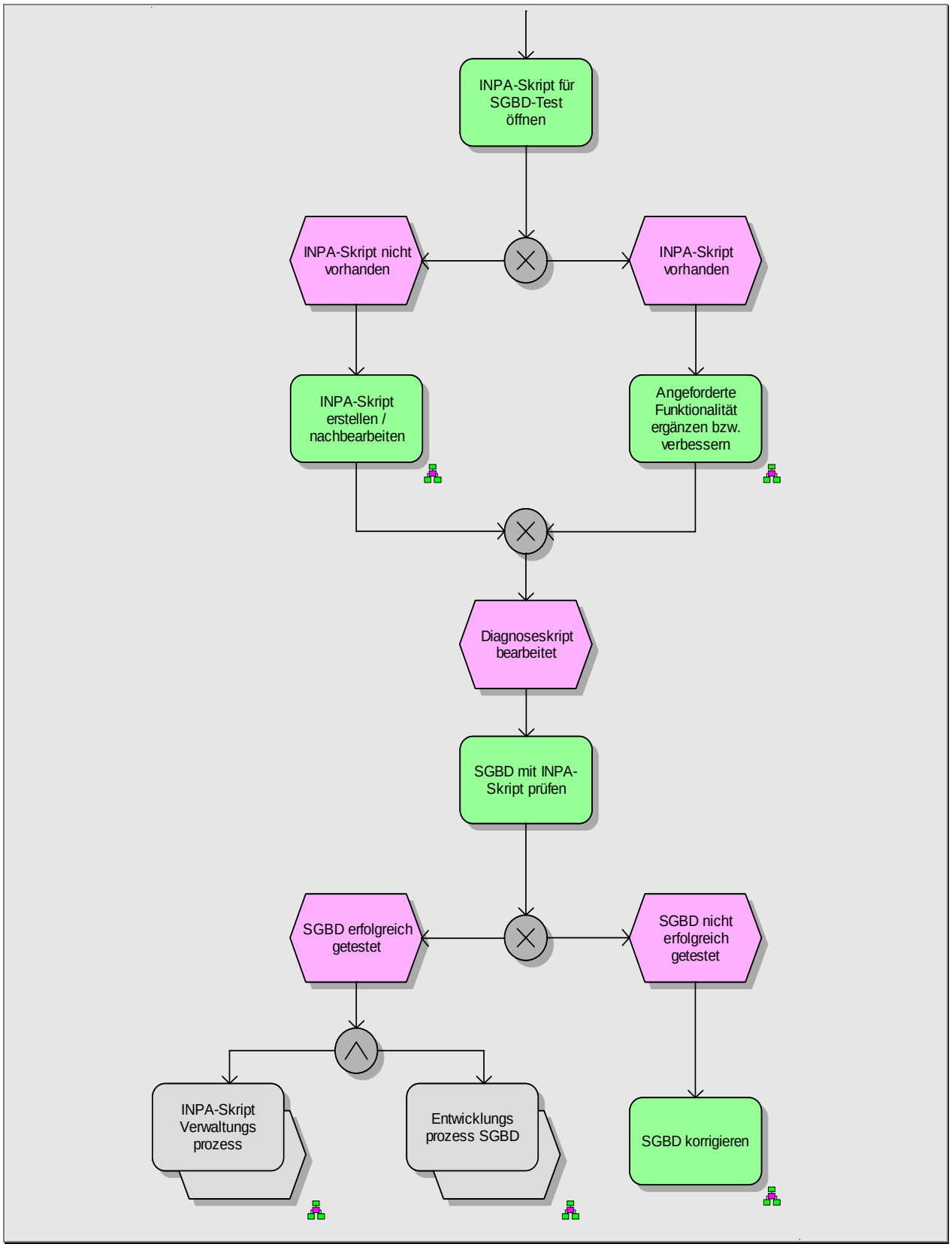


Abbildung 5.14 TP: SGBD erstellen/erweitern/korrigieren (EPK)

Name:	TP: SGBD erstellen/erweitern/korrigieren
Zuordnung:	GP: SGBD-Entwicklungsprozess
Prozess-verantwortlicher:	SGBD-Entwickler
Input:	Auftrag zur Neuerstellung, Erweiterung oder Korrektur einer SGBD. Der Anstoß erfolgt innerhalb des SGBD-Entwicklungsprozesses.
Objekte:	- Lastenheft-Diagnose - Muster-SGBD-Quelle - SGBD-Quelle einer bestehenden Datei - Standard Includes - Importierte Jobs - Anwendungssystem SGBD Generator / Compiler
Ergebnisse:	Bearbeitete SGBD (Funktionaler Test durch INPA-System)
Output:	Siehe Standard GS 95013[134]: - Neu erstellte / bearbeitete SGBD - Neu erstelltes / bearbeitetes INPA-Skript

Tabelle 5.9 TP: SGBD erstellen/erweitern/korrigieren (EPK)

[134] Vgl. SGBD 2003, GS 95013

6 Analyse der Ist-Prozesse

Um die erfassten Prozesse analysieren zu können, ist es zunächst notwendig, die Leistungsparameter zu definieren[135]. Anhand dieser Leistungsparameter können die Prozesse entsprechend bewertet werden.

Bei der Ermittlung dieser Leistungsparameter helfen verschiedene Methoden[136]. In der Praxis eignen sich nicht immer alle Methoden für ein Ist-Analyse-Szenario[137]. Eine Auswahl der bekanntesten Methoden wie Benchmarking oder Referenzanalysen wird in Kapitel 6.1 kurz vorgestellt und deren Anwendungsmöglichkeit geprüft.

Für die Auswahl eines Leistungsparameters und dessen Anwendung auf die Prozessmodelle gibt es keine einheitliche Vorgehensweise. Es ist hierbei vielmehr die "kreative Natur" des Analysten gefragt[138].

Die Leistungsparameter ergeben sich aus Ziel- und Messgrößen, um die Ist-Leistung des Prozesses zu bestimmen[139]. Sie werden deshalb innerhalb der Analyse oft auch als Kennzahlen bezeichnet. Die Parameter können sowohl subjektiver Art sein, wie z. B. bei der Bewertung der Awenderzufriedenheit oder auch durch konkrete Berechnung ermittelt werden, wie z. B. bei dem Leistungsparameter Prozesszeit. Die für diese Arbeit ausgewählten Kennzahlen werden in Kapitel 6.2 aufgelistet.

Durch die Auswertung der Prozesse anhand der Kennzahlen ergeben sich Schwachstellen. Diese sollen während der Soll-Modellierung in Kapitel 7 durch Optimierung der Prozesse beseitigt werden.

Die drei Bereiche, in denen generelle Verbesserungspotenziale vermutet werden und in denen gezielt nach Schwachstellen gesucht wird, sind bereits in Kapitel 4 wie folgt definiert worden: Funktionalität, Grafik und Mobilität.

Nach diesen Bereichen wird die Analyse gegliedert (Kapitel 6.3.2) und in den Kapiteln 6.4 bis 6.6 beschrieben. Die Erläuterungen werden um Zahlenmaterial und Anwendungsfallbeispiele ergänzt.

---

[135] Vgl. Schmelzer 2004, S. 173
[136] Vgl. Ellis 2004, S. 26-28
[137] Vgl. Becker 2000, S. 147
[138] Vgl. Becker 2000, S. 146
[139] Vgl. Becker 2000, S. 176

6.1 Anwendbarkeit der Methoden zur Prozessanalyse

6.1.1 Anwendbarkeit von Benchmarking

Ziel des Benchmarkings ist der Vergleich von Kennzahlen zwischen Unternehmen oder Unternehmensteilen[140]. Für den Vergleich werden Prozesse von Unternehmen ausgewählt, die einen Unternehmensbereich, wie z. B. die Nacharbeit in der Montage, vorbildlich beherrschen. Der Vergleich ergibt die Leistungsparameter des Vorbild-Unternehmens als Vorgabe für die eigene Analyse.

Problematisch für die Anwendung in der Praxis ist beim Benchmarking, einen passenden Vergleichspartner zu finden. Denn Unternehmen geben ihren Konkurrenten in der Regel ihre wettbewerbskritischen Daten nicht preis[141]. Gerade in einem so wirtschaftlich wichtigen Bereich wie der Kfz-Elektronik werden die Daten mit besonderer Sorgfalt gehütet. Bei den Automobilkonzernen zählt die Elektronik derzeit zu den entscheidenden Wettbewerbsfaktoren.

Ein weiterer Punkt, der die Anwendung innerhalb dieser Ist-Analyse unmöglich macht, ist die mangelnde Vergleichbarkeit der Kennzahlen. Die Nacharbeit- und Verwaltungsprozesse sind durch den Einsatz von proprietärer Software spezifisch an die betrieblichen Rahmenbedingungen des Kooperationspartners angepasst. Dies macht es sehr schwer, sich mit anderen Automobilherstellern mittels Benchmarking zu vergleichen.

Auf die Analyse mittels Benchmarking wird deshalb verzichtet.

6.1.2 Anwendbarkeit der Referenzanalyse

Ein Referenzmodell dient der allgemein gültigen Dokumentation von Best-Practice innerhalb eines abgegrenzten Problembereichs. Im Gegensatz zu einem individuellen Vergleich mit einem anderen Unternehmen (wie beim Benchmarking) sind Referenzmodelle mehr für eine Unternehmensbranche oder Fachbereiche innerhalb der Unternehmen gültig. Die Referenzmodelle mit Vorbildcharakter werden auf die eigenen Prozesse angewendet. Daraus ergeben sich Leistungsparameter mit den entsprechenden Ansprüchen für die Bewertung der eigenen Prozesse.

Für die praktische Verwendung ergeben sich hier ähnliche Probleme wie beim Benchmarking. Für die Referenzanalyse gilt es, ein Modell zu finden, das auf den vorliegenden Ist-Zustand angewendet werden kann. Referenzmodelle liegen auch eher für Standard-Geschäftsprozesse vor wie Rechnungswesen, Personalwirtschaft o. Ä.[142].

Ein Abgleich der Ist-Modelle mit einem Referenzmodell konnte nicht durchgeführt werden, da für die Problemdomäne "Nacharbeit Elektronik/Elektrik in der Fahrzeugindustrie" kein Referenzmodell existiert.

[140] Vgl. Becker 2000, S. 144
[141] Vgl. Becker 2000, S. 145
[142] Vgl. Becker 2000, S. 147

6.1.3 Anwendbarkeit von Kunden-/Anwenderbefragungen

Kundenbefragungen geben Aufschluss über die Zufriedenheit mit den bereitgestellten Prozessabläufen und deren Ergebnisse[143].

Diese Methode zur Bewertung nimmt eine gesonderte Position ein, da dabei nur Daten für die Kennzahl der Kundenzufriedenheit gewonnen werden können. Sie wird aber trotzdem in den meisten Konzepten zur Analyse aufgeführt, da sie die direkte Wertung der Prozessbeteiligten einfließen lässt.

Durch Befragung der Beteiligten wird ermittelt, an welchen Stellen der Prozess subjektiv Mängel aufweist. Die Berücksichtigung solcher Kritiken in den optimierten Prozessen erhöht besonders die Akzeptanz für Neuerungen bei Nutzern und Kunden.

Auch wird der Begriff "Kunde" im Bereich der Prozessanalyse nicht nur klassisch mit den Endkunden (Käufern) in Verbindung gebracht. In Kapitel 5.4 wurde der Begriff des internen Kunden eingeführt. Dies sind Mitarbeiter, die Ergebnisse interner Prozesse entgegennehmen und diese weiterverarbeiten., wie z. B. der Steuergeräteanalyst, der ein ausgeschleustes Fahrzeug aus der Nacharbeit entgegennimmt. Mit dieser Definition des Kunden schließt die Kundenbefragung auch Nutzer der Diagnosesoftware ein.

Die Befragung der Mitarbeiter hat zu Kritik an den Eigenschaften des Diagnosesystems und an umständlichen Arbeitsschritten geführt.

Neben dieser Kritik schafft die Kundenbefragung auch einen guten Einstieg in die Prozessanalyse. So konnten z. B. wie in Kapitel 4 erläutert, über einleitende Gespräche mit Mitarbeitern vor Ort und dem Management die drei Bereiche identifiziert werden, in denen Verbesserungspotenziale erwartet werden (Funktionalität, Grafik und Mobilität). Dies erleichtert den Einstieg in die Analyse, da der Problembereich abgegrenzt und gegliedert wird.

6.1.4 Anwendbarkeit der Schwachstellenanalyse

Ziel der Schwachstellenanalyse ist es, auf Basis der erhobenen Modelle eine möglichst vollständige, konsistente Liste aller Schwachstellen und Verbesserungspotenziale zu erstellen[144].

Der Prozess der Lokalisierung von Schwachstellen ist kaum standardisierbar und setzt vor allem Erfahrung und analytische Fähigkeiten voraus[145]. Die Beurteilung der identifizierten Schwachstellen hinsichtlich ihrer Bedeutung für das Unternehmen bzw. den Problembereich hängt stark von dem fachlichen Blickwinkel ab. Je nach Unternehmensbereich werden demnach die Schwachstellen mit unterschiedlicher Wichtigkeit wahrgenommen. Eine exakte allgemeingültige Bewertung von Schwachstellen ist folglich, laut Becker, in der Regel nicht möglich[146]. Aus dem Blickwinkel der Softwareentwicklung, sind vor allem Schwachstellen wie Systembrüche oder Medienbrüche von großer Bedeutung.

[143] Vgl. Schmelzer 2004, S. 173
[144] Vgl. Becker 2000, S. 146
[145] Vgl. Becker 2000, S. 147
[146] Vgl. Becker 2000, S. 147

Erfasst werden die Schwachstellen über Kennzahlen. Die Verwendung beider Begriffe gehen in der Prozessanalyse fließend ineinander über. Man spricht auch von der sogenannten Kennzahlenanalyse oder dem Kennzahlenkonzept[147].

In der Literatur finden sich zahlreiche Listen von Kennzahlen bzw. zu analysierende Schwachstellen für die Prozessanalyse[148]. Im folgenden Kapitel werden die in dieser Arbeit verwendeten Kennzahlen eingangs erläutert.

6.2 Auswahl von Kennzahlen zur Durchführung der Schwachstellenanalyse und Anwenderbefragung

In Kapitel 6.1 wurden die Schwachstellenanalyse und die Kunden- bzw. Anwenderbefragung als geeignete Methoden zur Prozessanalyse ausgewählt. Für das Aufdecken der Schwachstellen gibt es eine Vielzahl von Kennzahlen. Die Bandbreite reicht von technischen oder funktionalen Kennzahlen (z. B. Prozesszeit), über finanzielle Kennzahlen (z. B. Prozesskosten) bis hin zu sozialen Kennzahlen (z. B. Ergonomie). Die Auswahl erfolgt immer fallspezifisch und ist unter anderem von dem Unternehmensbereich oder den fachlichen Erwartungen abhängig (siehe Kapitel 6.1.4). Kapitel 6.2.1 bis 6.2.3 stellt die Kennzahlen vor, die für die Schwachstellenanalyse ausgewählt wurden. Die Anwenderzufriedenheit (Kapitel 6.2.4) ist eine Kennzahl sozialen Typs und dient der Auswertung der Analysemethode Anwenderbefragung.

6.2.1 Medien- und Systembruch

Unter einem Medienbruch versteht man den Wechsel eines Informationsträgers von einem Arbeitsschritt zum nächsten, obwohl in beiden ein und dieselbe Information verarbeitet wird. Es wird z. B. das Medium, das die Information trägt, durch den Arbeitsvorgang gewechselt, ohne dass dies aufgrund des Informationsgehalts notwendig wäre[149].

Durch einen Medienbruch wird der Arbeitsfluss verlangsamt oder teilweise sogar unterbrochen. Unter Umständen mindert sich auch die Qualität der Information. Außerdem weist ein Medienbruch immer die Gefahr einer hohen Fehleranfälligkeit auf.

Gleiche Risikofaktoren gelten bei Systembrüchen. Für diese Arbeit sind vor allem Anwendungssystembrüche relevant. Sie entstehen, wenn der Arbeitsablauf einen Wechsel, zwischen einem oder mehreren Programmen in aufeinanderfolgenden Arbeitsschritten erfordert. Klassisch ist hier der Wechsel zwischen zwei Programmen, um Informationen von einem in das andere Programm per Kopieren und Einfügen zu übertragen.

Diese Brüche schaffen neben den beschriebenen Risiken auch eine große Anwenderunzufriedenheit. Besonders bei einer IT-basierten Prozessoptimierung durch den Einsatz neuer Software wird großes Augenmerk auf diese Brüche gelegt. Es besteht eine gewisse Erwartungshaltung bei den Nutzern, dass die Brüche mit der neuen Software behoben werden.

[147] Vgl. Ellis 2004, S. 118-120
[148] Vgl. Staud 2001, S. 18-19
[149] Vgl. Ellis 2004, S. 30

Aus diesem Grund soll versucht werden, im Rahmen der Analyse die Anwendungssystembrüche zu identifizieren und zu beseitigen.

6.2.2 Prozessqualität

Die Messung der Prozessqualität ist ein Mittel, um Prozesse mit definierten physischen Ergebnissen zu bewerten. Bei diesen sogenannten Produktionsprozessen fällt es leicht, die Qualität des Prozesses anhand seiner Fehlerhäufigkeit zu messen: Wie viele hergestellte Autos einer Schicht sind fehlerfrei? Um die Fehleranzahl von Prozessen analytisch zu verarbeiten, gibt es viele Ansätze. Bekannte Vertreter sind Six Sigma oder FPY[150].

Die hier vorliegenden Prozesse sind zwar im Rahmen des Ressorts Fertigung angesiedelt, stellen aber im Grunde Dienstleistungsprozesse dar. Bei Dienstleistungen kann schwerer ein Prozessfehler bestimmt werden[151]. Wird z. B. ein fehlerhaftes Fahrzeug nachgebessert, läuft die Nachbesserung so lange, bis das Fahrzeug wieder fehlerfrei funktioniert. Der erfolgreiche Abschluss des Prozesses steht in der Regel außer Frage und ist so nicht aussagekräftig. Ein anderes Beispiel ist die Erstellung eines Diagnoseskripts. Auch hier kann der Erfolg schwer gemessen werden. Die Feststellung, ob die Diagnosemasken von guter oder schlechter Qualität sind, würde eine umfangreiche Evaluation jeder Maske erfordern und ist subjektiv geprägt.

Deshalb wird aufgrund mangelnder signifikanter Daten oder auch aufgrund der zu hohen Kosten für die Beschaffung dieser Daten auf die Messung der Prozessqualität bei Dienstleistungsprozessen häufig verzichtet bzw. über andere Kennzahlen kompensiert.

Mängel an der Prozessqualität zeigen sich im vorliegenden Fall nur an wenigen Stellen. So wird z. B. im Rahmen der Grafikanalyse darauf verwiesen, dass fehlende grafische Möglichkeiten des derzeitigen Systems eine exakte Eingrenzung des Fehlers bei der Diagnose erschweren (Kapitel 6.5). Allerdings findet sich dieser Mangel auch bei der Auswertung der Nutzer- und Kundenzufriedenheit. Die Qualität wird deshalb eher über diesen Leistungsparameter abgebildet.

6.2.3 Prozesszeit und -kosten

Prozesszeiten haben erheblichen Einfluss auf Effektivität, Effizienz, Reaktionsfähigkeit und Flexibilität eines Unternehmens bzw. deren Ressorts. Schmelzer erklärt deutlich, dass sich kürzere Prozesszeiten positiv auf Prozesseffektivität und Prozesseffizienz auswirken[152].

Wie bereits im ersten Kapitel beschrieben sind dies die primären Verbesserungsziele, die mit der Arbeit im Rahmen der Steuergerätediagnose verfolgt werden sollen. Durch die Steigerung der Effizienz und Effektivität bei der Diagnose von Steuergeräten wird gehofft, den vorliegenden Fehler im Fahrzeug zielgerichteter und somit schneller zu beheben.

Die Prozesszeit nimmt somit neben dem Systembruch (Kapitel 6.2.1) die zentrale Kennzahl für die Analyse und Soll-Modellierung ein. Besonders bei der Simulation in Kapitel 7 soll gezeigt werden, dass durch die Verbesserungen, z. B. im Bereich der Mobilität, die Arbeit während der Diagnose verkürzt werden kann.

---

[150] Vgl. Schmelzer 2004, S. 196-202
[151] Vgl. Ellis 2004, S. 25
[152] Vgl. Schmelzer 2004, S. 186-187

Da in verschiedenster Literatur deutlich dokumentiert wird, dass kürzere Prozesszeiten gleichzeitig Kosten reduzieren und ihre Qualität verbessern vgl. Schmelzer[153] und Ellis[154], soll auf eine umfassende Prozesskostenrechnung verzichtet werden. Es wird darauf verwiesen, dass eine Beschleunigung des Nacharbeitsprozesses, Fahrzeuge schneller wieder in den Vertriebsprozess integrieren lässt und durch diese Fahrzeuge weniger laufende Kosten, z. B. für die Lagerung, verursacht werden.

6.2.4 Anwenderzufriedenheit

Für die Beurteilung der Anwenderzufriedenheit gilt es zunächst, die Probleme, Bedürfnisse, Wünsche und Erwartungen richtig zu verstehen[155].

Um diese Anforderungen zu strukturieren, gibt es z. B. den Ansatz des Kano-Modells[156]. Hierbei werden die Anforderungen in drei Kategorien gegliedert. Es gibt Anforderungen, die der Nutzer an das Produkt stellt, die erfüllt werden müssen. Sie bleiben vom Anwender vorab unausgesprochen und sind von essenzieller Bedeutung für die Umsetzung. Weiter gibt es die Anforderungen, die nicht als selbstverständlich betrachtet werden, aber notwendig sind. Diese werden vom Nutzer meistens durch Fragen artikuliert. Als dritte Gruppe werden Wünsche und Verbesserungen genannt, die der Nutzer meist eigeninitiativ fordert.

Um einen hohen Grad an Akzeptanz für die neue Diagnosesoftware zu erreichen, wird versucht, die erste Gruppe der notwendigen "Muss"-Anforderungen vollständig zu erfüllen und auf die Erwartungen und Wünsche so gut wie möglich einzugehen.

Durch die umfassende Erfassung des Ist-Zustandes mit allen die Diagnosesoftware betreffenden Prozessen und Anwendungsfällen wird sichergestellt, dass alle notwendigen Anforderungen bekannt sind. Die neue Diagnosesoftware wird daher die elementaren Voraussetzungen für eine Diagnosesoftware mitbringen. Die erste Gruppe der Anforderungen des Nutzers werden damit erfüllt. Diese Punkte werden in der Arbeit eher marginal behandelt. Sie erscheinen als essenzielle Notwendigkeit bei der Anforderungsdefinition in Kapitel 7. Beispiele sind die Unterstützung von Basissystem-Treibern, um grundsätzlich mit dem Steuergerät kommunizieren zu können. In dieser Arbeit sollen allerdings vorrangig die Neuerungen und Verbesserungen herausgestellt werden. Die Nutzerwünsche und Kritikpunkte am bestehenden System sind deshalb für die Zufriedenheitsanalyse von größerem Interesse.

Besonders in der Verwaltung der Diagnoseskripte, wurde viel Unzufriedenheit bei der Erstellung und Bearbeitung geäußert, da die Arbeitsweisen äußerst umständlich sind. An dieser Stelle begründet sich die Unzufriedenheit als weicher Leistungsparameter auch durch andere härtere Kennzahlen. Denn ein umständliche Arbeitsweise ist auch Indikator für eine höhere Prozesszeit.

[153] Vgl. Schmelzer 2004, S. 187
[154] Vgl. Ellis 2004, S. 119
[155] Vgl. Schmelzer 2004, S. 183
[156] Vgl. Schmelzer 2004, S. 183

6.3 Durchführung der Schwachstellenanalyse und Anwenderbefragung

6.3.1 Erfassung der Schwachstellen und Anwenderurteile

Anhand der Kennzahlen Anwendungssystembruch, Prozesszeit und Prozessqualität wird versucht, die Schwachstellen in den gegenwärtigen Prozessen aufzudecken. Dafür bieten sich bei jedem Leistungsparameter verschiedene Vorgehensweisen an. Ihre Eignung hängt von vielen Faktoren ab, wie z. B. den Werkzeugen, die bei der Ist-Erfassung verwendet wurden. Einfluss nimmt auch, wie umfangreich jeweils die für eine Kennzahlenberechnung notwendigen Basisdaten sind.

Für die Analyse des Leistungsparameters Anwendungssystembruch bietet sich in dem vorliegenden Fall die Nutzung des Ist-Modellierungswerkzeugs ARIS an. In Kapitel 5.1 wurde dessen Konzept ausführlich erläutert. Es wurde dabei bereits auf die spätere Verwendung in der Analyse hingewiesen. Dies begründet sich jetzt mit der Nutzung für die Systembruchanalyse. ARIS bietet für die Suche nach Medien- und Systembrüchen ein Analysetool.

Mit dem Analysetool können Prozesse kennzahlenbasiert bewertet werden. Hierzu verarbeitet das Tool Analyseskripte. Für die Bewertung der Prozesse bietet ARIS-Analyseskripte zur Überprüfung von Organisationswechsel, Medienbrüchen, Anwendungssystembrüchen und Funktionsklassifikationen. Das Analyseergebnis kann z. B. als formatiertes Textdokument ausgegeben werden. Die Analyseskripte können individuell angepasst oder selbst erstellt werden.

Für die Analyse der Anwendungssystembrüche wurde ein Skript entwickelt, das in einem ausgewählten EPK bei allen Funktionsübergängen feststellt, ob ein Bruch der Anwendungssysteme vorliegt. Ein Bruch liegt laut Skript vor, wenn die vorhergehende und die nachfolgende Funktion nicht die gleiche Zuordnung an den genannten Typen besitzen. Das Skript wurde auf alle in Kapitel 5 erfassten Prozesse angewendet. Nach der Auswertung der Prozesse wurden die Prozesse mit Systembrüchen mit den entsprechenden Prozessverantwortlichen näher analysiert. Signifikante Brüche wurden in die Analysebeschreibung dieses Kapitels mitaufgenommen. Sie sind vor allem in der Verwaltung bei der Bearbeitung der Diagnoseskripte aufgefallen. Sie werden in den folgenden Kapitel 6.4 und 6.5 näher erläutert.

Für die Analyse der Prozesszeiten ist die Erfassung der Ist-Daten von großer Wichtigkeit. Sie müssen sorgfältig erfasst werden und vollständig sein. Nur dann kann ein Prozess aussagekräftig analysiert werden. Alle erfassten Prozessdaten wurden in die ARIS-Datenbank eingepflegt und sind in Prozessdaten Ist-System[157], dieser Dissertation beigelegt. Die Prozesszeiten haben sich bei der vorliegenden Ist-Erfassung aus Arbeitsberichten der Fachabteilungen, Erfahrungswerten, sowie direkten Messungen vor Ort ergeben. Hierfür wurden die Prozesse im Rahmen der Ist-Erfassung (Kapitel 5) mehrere Wochen intensiv begleitet und Gespräche mit den Prozessverantwortlichen geführt.

Mit dem Analysetool von ARIS können Prozesse aufgrund der eingepflegten Prozessdaten animiert und ausgewertet werden. Die Animation der ARIS-Analyse dient der dynamisierten Darstellung von Prozessabläufen und der Berechnung von zeit- und kostenorientierten Kenn-

[157] INPA-Prozessdatenerfassung 2006

zahlen. Die Animation wurde für alle hoch frequentierten Prozesse, wie die Diagnoseskriptverwaltung und die Diagnose von Steuergeräten, durchgeführt. Das Ergebnis der Animation hat zeitintensive Prozessabschnitte und die Bildung von Warteschlangen bei deren Abarbeitung aufgezeigt. An den Stellen mit solchen Auffälligkeit wurden die Prozesse intensiver überprüft. Dies war vor allem bei der Durchführung der Diagnose in der Nacharbeit und bei der Bearbeitung der Diagnoseskripte notwendig. Eine eingehende Analyse der Prozesszeit erfolgt in den Kapiteln 6.4.2 und 6.6.1.

Die Prozessqualität kann hingegen nicht ausschließlich anhand konkreter Prozessdaten bestimmt werden. Kapitel 6.2.2 verdeutlicht hierzu, dass es bei Dienstleistungsprozessen schwer fällt, die Prozessqualität zu messen. Denn bei Dienstleistungsprozessen, wie der Verwaltung oder Nacharbeit, gibt es kein konkret bewertbares Endprodukt (Leistung). Ohne die Leistung klassifizieren zu können, kann auch die Qualität des Prozesses nicht beurteilt werden. Deshalb wird auch in der Literatur die Prozessqualität häufig als Ergebnis einer Sammlung diverser Leistungsparameter beschrieben. Als mögliche Leistungsindikatoren werden Anwenderzufriedenheit, Prozessdurchlaufzeit, Wartezeit, Systembrüche oder Flexibilität genannt[158]. Deshalb erscheint es auch bei dieser Analyse sinnvoll, die Mängel in der Prozessqualität anhand der übrigen Leistungsparameter zu erarbeiten. Defizite finden sich vor allem bei der grafischen Darstellung der Bedienoberfläche (siehe Kapitel 6.5.2).

Bei der Bewertung der Anwenderzufriedenheit zeigt sich die Sonderstellung dieses Leistungsparameters. Er ist als Kennzahl mit "sozialem" Charakter definiert. Neben den finanziellen und funktionalen Kennzahlen zeigt sich die Anwenderzufriedenheit als Kennzahl mit weicher Bewertungsgrundlage[159]. Die Zufriedenheit lässt sich nicht konkret, wie z. B. bei der Zeitmessung, erfassen. Es kann somit kein genauer Grenzwert festgelegt werden, ab wann eine Optimierung notwendig ist. Vielmehr gilt es hier die verschiedenen Meinungen zusammenzutragen und gemeinsame Kritikpunkte als Konsens zu bestimmen.

Vorrangige Methode zur Analyse ist die persönliche Befragung[160]. Ob diese nun per Telefon, E-Mail, im Gespräch oder mittels anderen Medien durchgeführt wird – die Meinung des Befragten bleibt subjektiv. Da das derzeitige Diagnosesystem allerdings schon Jahrzehnte im Einsatz ist, hat die Befragung von erfahrenen Mitarbeitern allerdings profunde Hinweise für Verbesserungen geliefert. Die Befragung wurde im Rahmen der Prozessdatenerfassung durchgeführt. Es wurden Mitarbeiter in der Verwaltung, Nacharbeit, Analyse und Entwicklung nach ihrer Beurteilung, Kritik und Verbesserungsvorschlägen im Bezug auf das Ist-System befragt. Da die drei Ansatzpunkte der Arbeit Funktionalität, Grafik und Mobilität auch durch Gespräche mit dem Kooperationspartner erarbeitet wurden, sind erwartungsgemäß in allen drei Bereichen die Unzufriedenheiten konkret benannt worden (Kapitel 6.4 bis 6.6).

6.3.2 Gliederung der Analyseergebnisse

Die Analyse gliedert sich nach den Ansatzpunkten Funktionalität, Grafik und Mobilität. Die Ansatzpunkte wurden zu Beginn der Arbeit ermittelt und stellen Bereiche dar, in denen die Diagnose neuen Herausforderungen gegenübersteht (siehe Kapitel 4).

[158] Vgl. Ellis 2004, S. 25-31
[159] Vgl. Ellis 2004, S. 138
[160] Vgl. Schmelzer 2004, S. 185

Innerhalb der drei Bereiche erfolgt die Gliederung nach den Kennzahlen aus Kapitel 6.2. Die Kapitelbezeichnungen geben an, welche Kennzahlen in dem Bereich zu einer Schwachstelle geführt haben. Dies wird ergänzt durch eine grobe Beschreibung der damit kritisierten fachlichen Tätigkeit oder Funktion.

Jeder dieser Hauptkritikpunkte wird ergänzt durch Beispiele. Es werden also in der Regel Anwendungsfälle genannt, bei denen das Auftreten der Schwachstelle zu beobachten ist. Für eine detaillierte Beschreibung der betroffenen Prozesse wird auf Kapitel 5 verwiesen. Falls der Prozessbereich in Kapitel 5 noch nicht tief greifend genug behandelt wurde, wird auch der Prozessabschnitt selbst nochmals erläutert.

Neben Anwendungsfällen und Prozesserläuterung ist für die Analyse natürlich auch der Stellenwert der Tätigkeit im Hinblick auf die Gesamtabläufe in der Steuergerätediagnose von Bedeutung. Dieser wird anhand von Prozessdaten, wie Zeiten, Frequenzen, Häufigkeiten oder Stückzahlen belegt.

Nach dem Abstecken des fachlichen Rahmens, in dem sich die Tätigkeit bewegt, werden die durch die Kennzahlen erarbeiteten Schwachstellen aufgeführt. Die Schwachstellen werden durch Ausschnitte von Diagnoseskriptinhalten oder Abbildungen der Diagnoseoberfläche verdeutlicht. Diese Art der Gliederung wird auch in Kapitel 6.4 bei der Formulierung der Optimierungsvorschläge verwendet.

6.4 Analyseergebnisse zur Funktionalität

6.4.1 Anwenderzufriedenheit: Erstellen und Nachbearbeiten von Diagnoseskripten

Der Teilprozess "INPA-Skript erstellen/nachbearbeiten" nimmt bei der Nutzung des INPA-Diagnosesystems eine wichtige Rolle ein. In Kapitel 5.8.2 wurde dessen Ablauf im Rahmen der Verwaltung der Diagnoseskripte vorgestellt. Es wurde darauf verwiesen, dass ferner in der Analyse und der Entwicklung der Steuergeräte auch teilweise Diagnoseskripte selbstständig von den Mitarbeitern erstellt werden.

Durch die Ist-Erfassung wurde ermittelt, dass allein innerhalb des Geschäftsprozesses in der Verwaltung monatlich 450 Fälle zur Nachbearbeitung und 40 Fälle zur Neuerstellung von Diagnoseskripten eingehen (siehe Kapitel 6.4.1.2).

Ein typischer Anwendungsfall beim Erstellen oder Nachbearbeiten von Diagnoseskripten ist das Skript um eine Diagnosefunktionalität zu erweitern. Hierfür ist unter anderem der Einbau von Aufrufen von Steuergerätebefehlen in das Diagnoseskript notwendig.

6.4.1.1 Anwendungsfall: Diagnoseskriptfunktionalität erweitern

Der Anwendungsfall zeigt zusammenfassend einige Schwachstellen auf, die im Laufe des Kapitels 6.4 weiter analysiert werden sollen. Er eignet sich deshalb als Einstieg in die Analyse und stellt sich wie folgt dar.

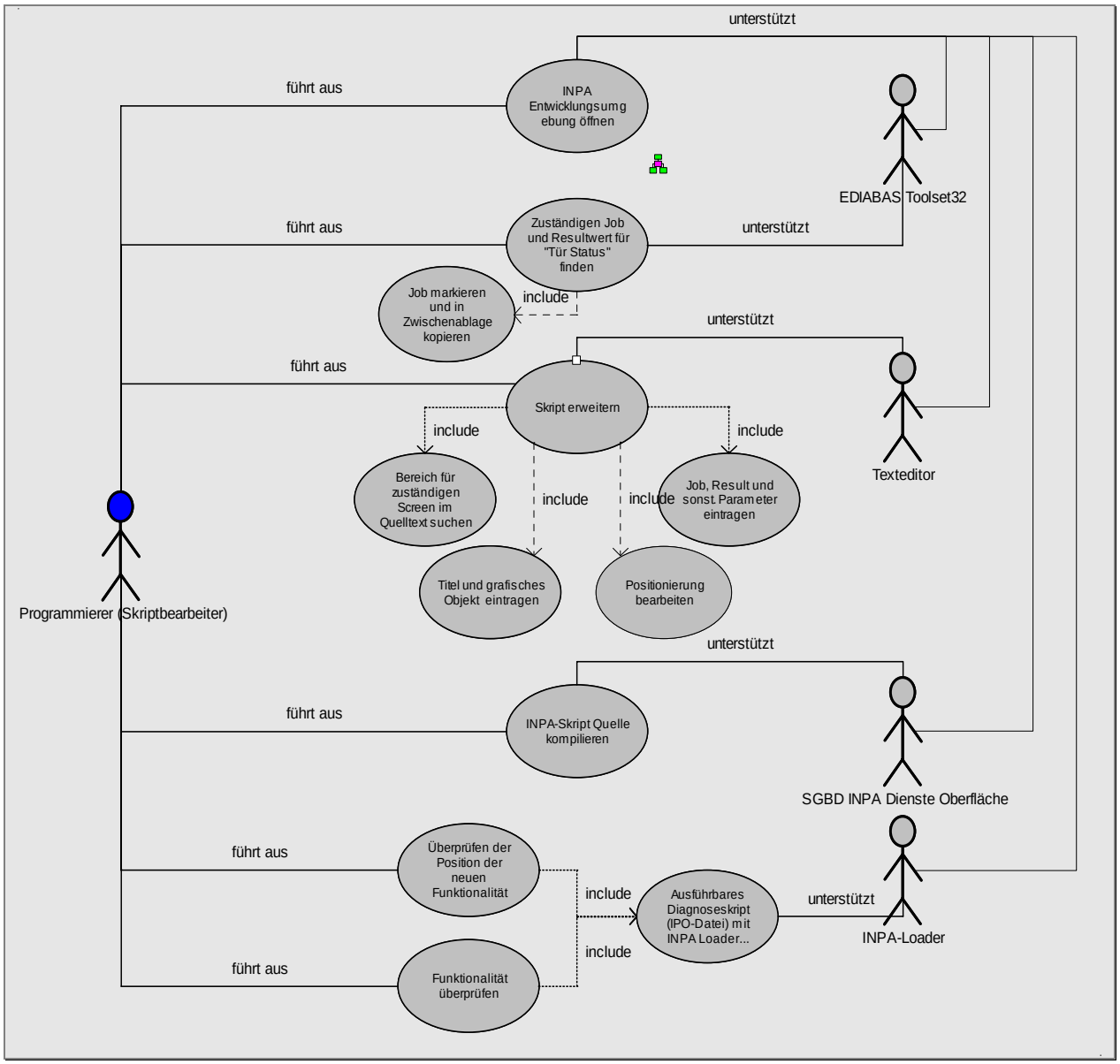


Abbildung 6.1 Funktionalität erweitern: "Tür Status" (UML-Anwendungsfalldiagramm)

Name:	Funktionalität erweitern: Tür Status
Ziel:	Bereitstellung des erweiterten Skripts
Startereignisse:	Aus der Nacharbeit geht die Forderung ein, dass ein bestehendes Diagnoseskript um eine zusätzliche Statusanzeige erweitert werden soll. GP: INPA-Skript-Verwaltungsprozess Ereignis: INPA-Skript soll nachbearbeitet werden
Endereignisse:	Die Anforderung wurde im INPA-Skript ergänzt. TP: INPA-Skript erstellen / nachbearbeiten Ereignis: INPA-Skript bearbeitet

Akteure:	**Bezeichnung**	**Eigenschaft**
	Programmierer (Skriptbearbeiter)	Verantwortlich
	Nacharbeit	Auftraggeber
	Texteditor	Anwendungssystem
	SGBD INPA Dienste Oberfläche	Anwendungssystem
	EDIABAS Toolset32	Anwendungssystem
	INPA-Loader	Anwendungssystem
Zuordnung:	GP: INPA-Skript Verwaltung Teilprozess: INPA-Skript erstellen / nachbearbeiten	

Tabelle 6.1 Anwendungsfall: Funktionalität erweitern (UML-Anwendungsfalldiagramm)

Ursprung des Anwendungsfalls ist ein Auftrag zur Skriptbearbeitung für den Programmierer. Für die Baureihe eines Fahrzeugs soll als weitere Statusinformation der Schließzustand der Tür angezeigt werden. Die Anzeige erfolgt mittels einfarbiger LED für den An-/Aus-Zustand.

Um von einem Steuergerät einen Status abzufragen, muss der entsprechende Job und der dazugehörige Ergebnistyp (Result-Set) ermittelt werden (siehe Kapitel 3.2.4). Um Job und Result-Set zu bestimmen, muss zunächst das EDIABAS Toolset32 geöffnet werden. Zunächst lädt der Programmierer die für den Funktionsbereich und die Baureihe zuständigen SGBD. Nach dem Laden kann er sich für die gesuchte Funktionalität den Jobnamen und das Result per Auswahllisten anzeigen lassen.

Der Screenshot aus dem EDIABAS Toolset32 zeigt die Auswahl des Jobs "STATUS_ISTWERTE", mit dem Result "STAT_TUER_ZU", das dazugehörige Result-Format und, wenn vorhanden, noch zusätzliche Aufrufparameter.

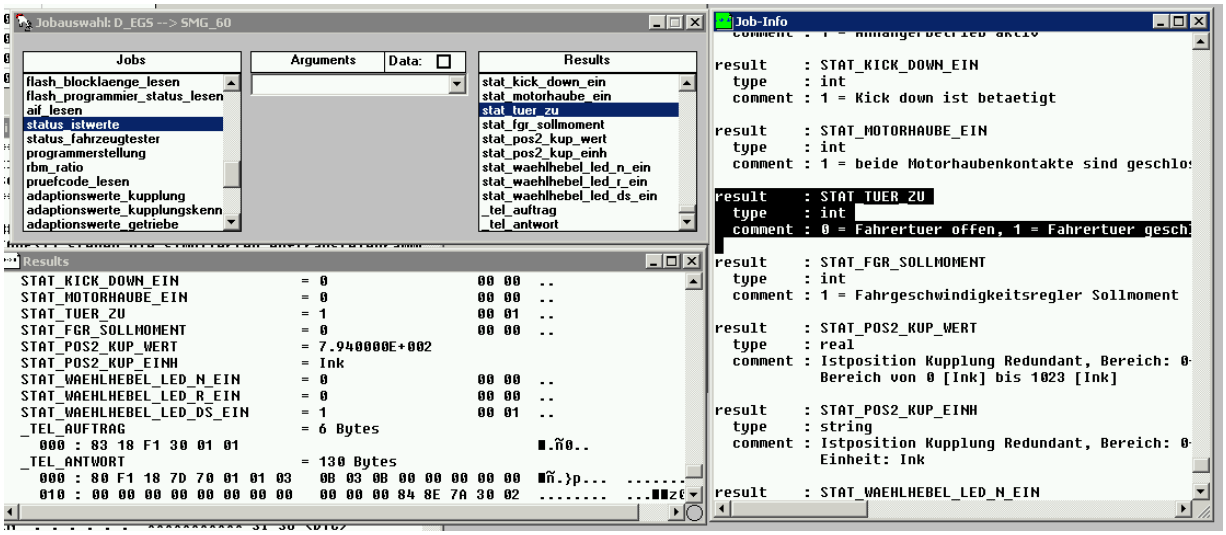


Abbildung 6.2 EDIABAS Toolset32

Um den Steuergerätebefehl in das Diagnoseskript zu übertragen, muss die Diagnoseskriptquelle vom Programmierer manuell bearbeitet werden. Die Quellen werden in einer proprietären Skriptsprache geschrieben. Die Programmiersprache trägt den Namen PABS (Prüfablaufbeschreibungssprache) und ist von der Syntax her der Programmiersprache C ähnlich.

Zur Bearbeitung wird die Diagnoseskriptquelle mit einem beliebigen ASCII-Texteditor geöffnet. Da der Status einer Tür nur zwei Zustände haben kann, eignet sich für die Ausgabe eine LED. Deshalb muss der Programmierer den Methodenaufruf INPAapiResultDigital eintragen. Die spezifischen Werte für den Jobaufruf (Jobname, Result-Set etc.) müssen per Copy & Paste übertragen werden. Es gibt keine direkte Schnittstelle zwischen Texteditor und Toolset.

```
INPAapiJob(sgbd,"STATUS_ISTWERTE","","");

// Tür
text(15,1,"@Tür@");
ergebnisDigitalAusgabe("STAT_TUER_ZU", 17, 1," @ZU@ "," @AUF@ ");

(ruft auf…)

// Ausgabe eines Ergebnisses als Digital-/LED-Wert
ergebnisDigitalAusgabe(in: string ergebnis,in: int y,in: int x,
            in: string trueText,in: string falseText)
{
  bool tmpBool;

  INPAapiResultDigital(tmpBool,ergebnis,1);
  digitalout(tmpBool,y,x,trueText,falseText);
}
```

Abbildung 6.3 Quellcode: Aufruf Steuergerätebefehl

Nach dem Ändern und Kompilieren der Quelle mittels der Anwendung "SGBD INPA Dienste Oberfläche", kann die erzeugte IPO-Datei im INPA-Loader angezeigt werden.

Die obige Skriptergänzung führt zu folgender Ausgabe auf der Oberfläche:

Abbildung 6.4 Oberflächenausschnitt Diagnosemaske für geschlossene Tür

Da auch die grafische Gestaltung per Texteingabe in der Quelle erfolgt, kann der Programmierer erst nach dem Kompilieren und Laden des kompilierten Diagnoseskripts die Ausgabe betrachten (Abbildung 6.4).

6.4.1.2 Stellenwert des Anwendungsfalls

Eine detaillierte Analyse der Prozesse, die Anwendungsfall hinterlegt sind, folgt in den weiteren Kapiteln. Da dieser Anwendungsfall eine Übersicht über die Verwaltungsarbeit mit dem Diagnosesystem gibt, wird in diesem Kapitel vorrangig der Gesamteindruck aus Sicht der Mitarbeiter analysiert. Es werden Mitarbeiter danach befragt, wie sie die Arbeit mit dem System empfinden. Auf die Vorteile von solchen sozialen Leistungsparametern, wie z. B. Awenderzufriedenheit, wurde bereits in Kapitel 6.2 verwiesen. Des Weiteren wird die Relevanz des Prozesses "INPA-Skript erstellen/nachbearbeiten" (Kapitel 5.8.2) wie folgt begründet:

```
Bereich: Kooperationspartner, Basissystementwicklung, ca. 8 Programmierer verantwort-
lich für INPA-Skript Verwaltung

Fälle zur Bearbeitung von INPA-Skripten (pro Monat): X1                  = 450
Fälle zur Neuerstellung von INPA-Skripten (pro Monat): X2                = 40
Mittlere Durchlaufzeit bei Bearbeitung von INPA-Skripten (Stunden): T1   = 1,5
Mittlere Durchlaufzeit bei Neuerstellung von INPA-Skripten (Stunden): T2 = 6
Aufkommen pro Monat (Stunden): X1*T1+X2*T2=S1                            = 915
Aufkommen je Mitarbeiter (pro Monat in Stunden): A1=S1/M1                = 114,4
Durchschnittl. Arbeitstage / Monat (lt. Personalabteilung, 2007): PI     = 19,67
Auslastung durch Prozess je Programmierer (Prozent): P1=A1/(PI*8) * 100% = 72,68

(Quelle: INPA-Prozessdatenerfassung 2006, "INPA-Skript erstellen / nachbearbeiten")
```

Abbildung 6.5 Auswertung: Arbeitsauslastung durch Neuerstellung/Bearbeitung von Diagnoseskripten

Der Prozess nimmt folglich knapp drei Viertel der Arbeitszeit jedes Diagnoseskript-Programmierers ein. Aufgrund dieser hohen Gewichtung erhält auch die Awenderzufriedenheit einen entsprechenden Stellewert in der Analyse. Die Schwachstellen, die von den Mitarbeitern vermehrt angeführt wurden, sollen mit dem Soll-Konzept verbessert werden.

Die konsolidierten Kritikpunkte werden in den folgenden Unterkapiteln kurz beschrieben. Sie werden in den weiteren Kapiteln der Analyse durch weitere Kennzahlen, z. B. für Systembrüche und Prozesszeitberechnungen, noch eingehender behandelt.

6.4.1.3 Schwachstelle: Mehrere Entwicklungsprogramme notwendig

Unzufrieden sind die Anwender vor allem damit, dass sie vier verschiedene Programme benutzen müssen. Sie müssen zum Heraussuchen des Jobs das EDIABAS Toolset32, zum Einbau einen Texteditor, zum Kompilieren die SGBD INPA Dienste Oberfläche und zum Anzeigen den INPA-Loader verwenden. Das Springen zwischen den Programmen kostet Zeit und weist ein hohes Fehlerpotential auf.

Von der Bedienung abgesehen bedeutet dies auch bei der Installation und Aktualisierung der gesamten Entwicklungsumgebung einen Mehraufwand.

6.4.1.4 Schwachstelle: Proprietäre Programmiersprache

Wie aus der Beschreibung des Anwendungsfalls hervorgeht, ist die Diagnoseskriptquelle in einem proprietären Code verfasst. Dies bedeutet, dass für die Erstellung der Diagnoseskripte eine systemeigene C-nahe Programmiersprache entwickelt wurde.

Für neue Programmierer ist es deshalb sehr mühsam, sich in die Erstellung der Skripte einzuarbeiten. Des Weiteren bietet die proprietäre Programmiersprache nicht den Umfang der Programmiersprache C. Folglich müssen verschiedene Dynamiken sehr umständlich programmiert werden. Im Bereich der Grafik können verschiedene Defizite auch nicht per Workarounds (provisorische Lösungen in der Programmierung) kompensiert werden. Konkrete Beispiele hierfür folgen in Kapitel 6.5.

6.4.1.5 Schwachstelle: Fehlende Schnittstellen

Des Weiteren wurden umständliche Prozessschritte bemängelt. Es gibt keine direkten Import/Export-Schnittstellen zwischen den Programmen. Jobbezeichnungen, Typen von Results etc. müssen markiert, kopiert und in die Skriptquelle eingefügt werden. Dieser Vorgang ist sehr fehleranfällig. Außerdem fallen die Fehler erst später nach dem Kompilieren beim Laden des Diagnoseskripts auf.

6.4.1.6 Schwachstelle: Editieren per Texteditor

Unzufriedenheit schafft zudem die generelle Arbeit im Texteditor. Die Positionierung von grafischen Elementen über die Eingabe von Zahlen ist sehr zeitaufwendig. Besonders wenn Objekte im Nachhinein hinzugefügt werden, müssen die Positionszahlen aller nebenstehenden Objekte erneut angepasst werden. Auch diese Problematik wird ausführlich in Kapitel 6.5 behandelt.

6.4.2 Systembruch und Prozesszeit: Job bearbeiten/hinzufügen

Im Anwendungsfall von Kapitel 6.4.1 wurde erläutert, dass ein Diagnoseskript seine Funktionalität über den Einbau von Steuergerätebefehlen (Jobs) erlangt. Im Rahmen des Prozesses "Angeforderte Funktionalität ergänzen bzw. verbessern" werden diese Jobs dem Skript neu hinzugefügt, bearbeitet oder gelöscht. Dieser Prozess wurde bereits in Kapitel 5.8.3 vorgestellt.

Für die Analyse in diesem Kapitel soll repräsentativ der Arbeitsschritt "Job hinzufügen" betrachtet werden.

6.4.2.1 Arbeitsschritt: Job hinzufügen

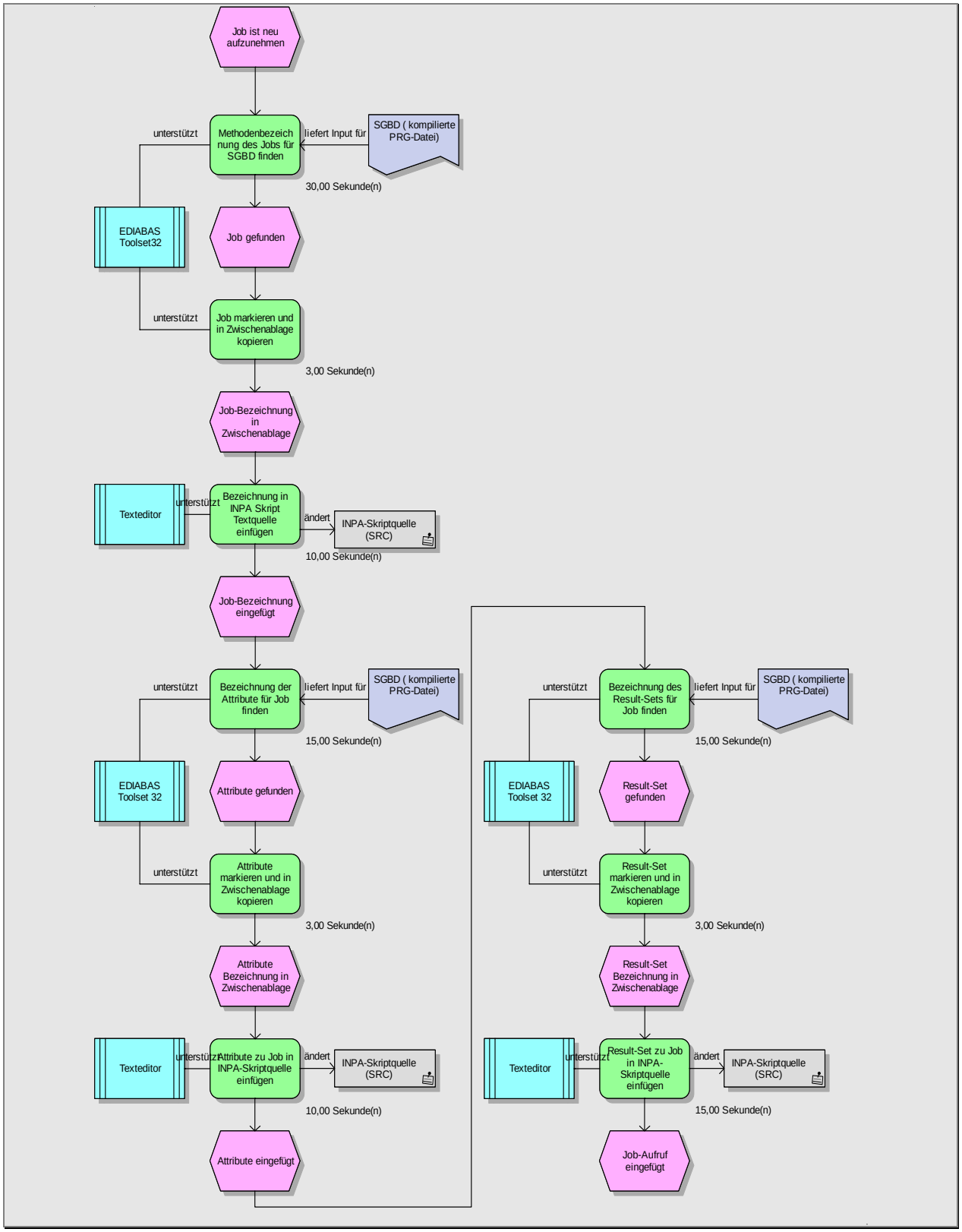


Abbildung 6.6 Arbeitsschritt: Job hinzufügen (EPK)

Name:	Arbeitsschritt: Job hinzufügen
Zuordnung:	GP: INPA-Skript Verwaltungsprozess TP: INPA-Skript erstellen/nachbearbeiten P: Angeforderte Funktionalität ergänzen/verbessern
Prozess-verantwortlicher:	Programmierer (Skriptbearbeiter)
Input:	Der Prozess wird durch den Prozess "Angeforderte Funktionalität ergänzen/verbessern" angestoßen (Kapitel 5.8.2). Der Prozess stellt als Input die zu bearbeitende INPA-Skriptquelle bereit.
Objekte:	- INPA-Skriptquelle (*.SRC-Datei) - SGBD - Anwendungssystem Texteditor - Anwendungssystem EDIABAS Toolset32
Ergebnisse:	Job-Aufruf hinzugefügt
Output:	Erweiterte INPA-Skriptquelle (*.SRC)-Datei

Tabelle 6.2 Arbeitsschritt: Job hinzufügen (EPK)

Soll ein Diagnoseskript um einen Jobaufruf erweitert werden, muss der Programmierer zunächst die genaue Bezeichnung des Jobs ermitteln. Im Anwendungsfall aus Kapitel 6.4.1.1 sollte z. B. der Schließzustand der Fahrertür ausgegeben werden.

Hierfür startet der Programmierer die Anwendung EDIABAS Toolset32. In der Anwendung öffnet er die SGBD, die für die Fahrzeugreihe und den funktionalen Bereich im Fahrzeug zuständig ist. Nach dem Laden öffnet sich eine Fensteranordnung, wie in Abbildung 6.2 dargestellt.

Innerhalb eines Fensters werden alle Jobnamen aufgelistet. Findet der Programmierer den entsprechenden Job, kann er ihn im Fenster markieren und in die Zwischenablage kopieren.

Daraufhin öffnet er mit einem Texteditor das Diagnoseskript und trägt den Jobnamen dort ein. Für einen vollständigen Aufruf eines Jobs benötigt INPA noch die Angabe diverser Aufrufparameter und den Eintrag des Rückgabewerts (Result-Set).

Beides muss der Programmierer wieder aus dem EDIABAS Toolset32 beziehen. Er wählt dazu in dem Fenster den Job aus und löst damit das Laden der zugehörigen Attribute und Result-Set Informationen aus.

Daraufhin muss der Programmierer wieder jedes einzelne Attribut markieren, kopieren und in die Diagnoseskriptquelle einfügen.

Wichtig für die Ausgabe des Jobaufrufs ist zudem der Typ des Rückgabewerts sowie dessen Positionierung.

Auf die Positionierung wird in Kapitel 6.5 im Bereich Grafik eingegangen. Der Typ des Results bestimmt, welche Art von Ergebnis geliefert wird: numerisch, Text oder ein boolescher Wert (wahr/falsch). Den Typ liest der Programmierer aus dem EDIABAS Toolset32 aus und tippt daraufhin den richtigen Methodenaufruf für Job und Result im Texteditor ein. Ein Script-Beispiel ist in 6.3.1.1 angegeben. Der Programmierer muss selbst entscheiden und den richti-

gen Methodenaufruf für den zugehörigen Resulttyp wählen, anderfalls kommt es beim Ausführen des Diagnoseskripts zu Laufzeitfehlern.

6.4.2.2 Stellenwert des Arbeitsschritts

Die Schwachstellenanalyse anhand von ARIS hat ergeben, dass der Prozess "Job hinzufügen" Anwendungssystembrüche aufweist. Die Arbeitsweise beim Hinzufügen eines Jobs ist ähnlich der Arbeitsweise, einen Job zu bearbeiten. Deshalb hat die Analyse auch einen Systembruch bei dem Arbeitsschritt "Job bearbeiten" ergeben. Die Umfänge werden in Kapitel 6.4.2.3 beschrieben.

Da Anwendungssystembrüche meistens eine umständliche Arbeitsweise erfordern, wurde auch die Prozesszeit analysiert (siehe Kapitel 6.4.2.4).

Da das Bearbeiten und Hinzufügen von Jobs eine der Kerntätigkeiten der Programmierer ist, weisen beide Prozesse zusammen eine große Häufigkeit auf.

```
Bereich: Kooperationspartner, Basissystementwicklung

Fälle zur Bearbeitung von INPA-Skripten (pro Monat): X1                      = 450
Fälle zur Neuerstellung von INPA-Skripten (pro Monat): X2                    = 40
Wahrscheinlichkeit bei Skript-Bearbeitung, dass Job zu löschen ist: X1W1     = 0,1
Wahrscheinlichkeit bei Skript-Bearbeitung, dass Job zu bearbeiten ist: X1W2  = 0,6
Wahrscheinlichkeit bei Skript-Bearbeitung,
       dass Gestaltung (Positionierung) zu verändern ist: X1W3               = 0,05
Wahrscheinlichkeit bei Skript-Bearbeitung, dass Job hinzuzufügen ist: X1W4   = 0,25
Fälle bei denen "Job bearbeiten" oder "Job hinzufügen"
       durchlaufen wird: X1W2*X1+X1W4*X1+X2=S1                               = 423
Anteil an allen Fällen, bei denen einer der beiden Prozesse
       durchlaufen wird (Prozent): P1=S1/(X1+X2)*100 %                       = 84,6

(Quelle: INPA-Prozessdatenerfassung 2006, Prozessdaten "Angeforderte Funktionalität
ergänzen bzw. verbessern")
```

Abbildung 6.7 Auswertung: Häufigkeit der Tätigkeit "Job bearbeiten" und "Job hinzufügen"

6.4.2.3 Schwachstelle: Informationsübertragung per Kopieren/Einfügen

Der Analysebericht der ARIS-Auswertung hat ergeben, dass bei über 60 % aller Funktionsübergänge Systembrüche festgestellt wurden (siehe Quelle INPA-Analyseberichte 2007, Bericht "Job hinzufügen").

Folgende Abbildung stellt einen dieser Systembrüche grafisch dar:

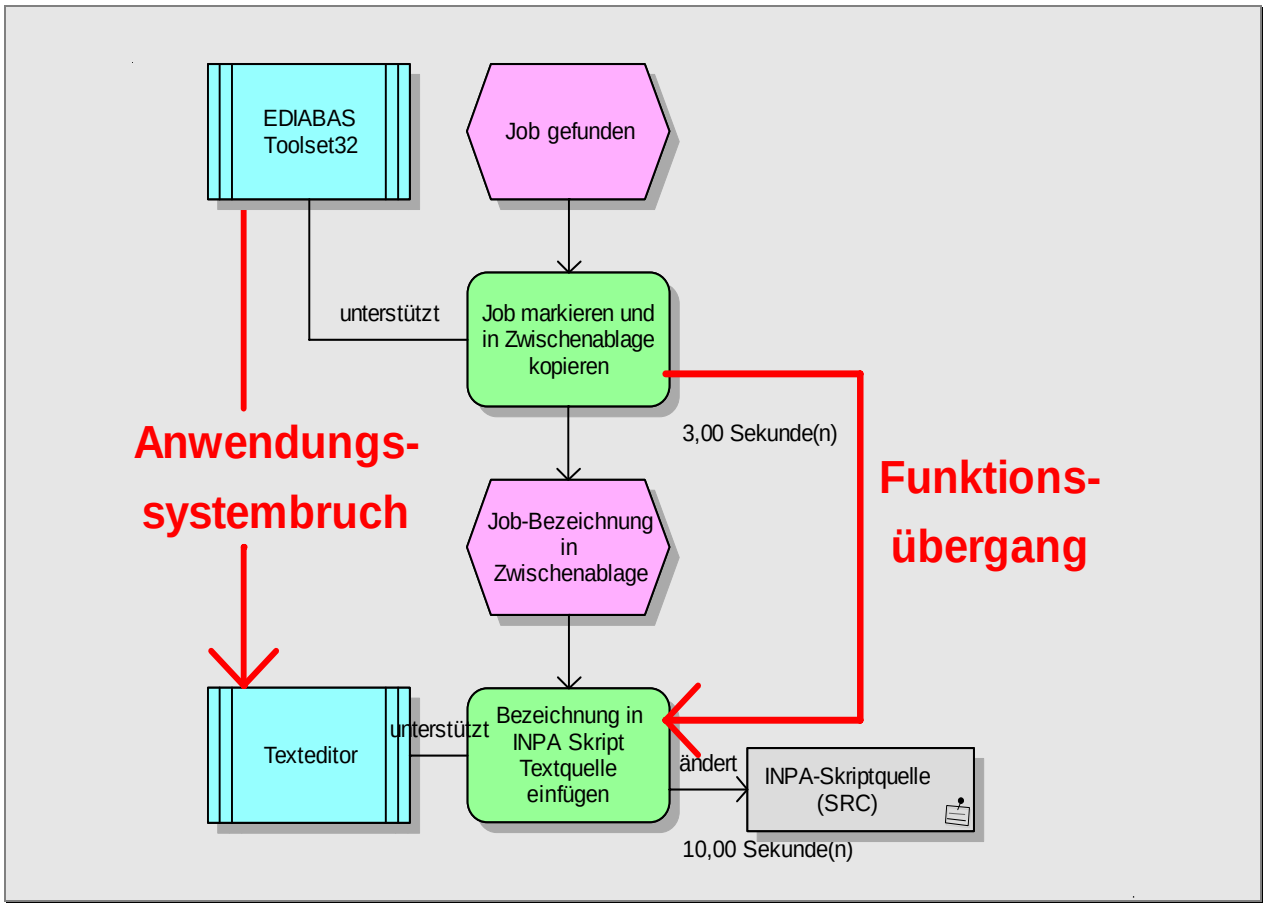


Abbildung 6.8 Prozessausschnitt: Job hinzufügen (Systembruchanalyse)

Der Anwender ist gezwungen für den Einbau von Diagnosefunktionalität permanent zwischen den Applikationen EDIABAS Toolset32 und dem Texteditor zu wechseln. Diese Art der Informationsübertragung (per Copy & Paste) ist äußerst fehleranfällig und sorgt auch für Unmut bei der Benutzung.

Kritik in diese Richtung wurde auch schon im vorherigen Kapitel bei der Anwenderunzufriedenheit bezüglich der Vielfalt an benötigten Programmen dokumentiert (siehe Kapitel 6.4.1.3 und 6.4.1.5).

Für die Soll-Modellierung gilt es, diese Systembrüche aufzulösen. Der vollständige von ARIS erzeugte Analysebericht ist in der Quelle INPA-Analyseberichte 2007 erfasst.

6.4.2.4 Schwachstelle: Zeitaufwendige Informationsübertragung

Neben der Komplexität bei der Abarbeitung des Prozesses, beanspruchen die Systembrüche außerdem Prozesszeit.

Der Arbeitsschritt "Job hinzufügen" beinhaltet rein die Informationsübertragung der Job-, Rückgabewert- und Attributbezeichnung in das Diagnoseskript. Weitere Arbeitsschritte wie Kompilieren, grafische Bearbeitung und Test sind darin nicht enthalten. Folgende Daten zeigen, welchen Anteil er an dem Prozess "Angeforderte Funktionalität ergänzen bzw. verbessern" annimmt.

```
Bereich: Kooperationspartner, Basissystementwicklung, ca. 8 Programmierer verantwort-
lich für INPA-Skript Verwaltung

Mittlere Durchlaufzeit Prozessschritt "Job hinzufügen" (Minuten): T1  = 1:44
Mittlere Durchlaufzeit Prozess "Angeforderte Funktionalität
      ergänzen bzw. verbessern" (Minuten): T2                        = 16:50
Anteil an Prozess (Prozent): P1 = T1/T2 * 100 %                      = 10,30

(Quelle: INPA-Prozessdatenerfassung 2006, Prozessdaten "INPA-Skript erstellen / nach-
bearbeiten" und "Job hinzufügen")
```

Abbildung 6.9 Auswertung: Anteil der Tätigkeit "Job hinzufügen" an Prozesszeit

Da eine reine "motorische" Informationsübertragung erfolgt, belegt der Arbeitschritt mit über 10 % einen hohen Anteil an der Gesamtbearbeitungszeit. Für die Soll-Modellierung wird es deshalb vor allem wichtig sein, die Mittlere Durchlaufzeit (MDZ) von einer Minute und 44 Sekunden zu reduzieren.

6.5 Analyseergebnisse zur Grafik

In Kapitel 6.4 wurde die Diagnosesoftware INPA aus Sicht ihrer Funktionalität analysiert. Es wurden hierbei vor allem Schwachstellen aufseiten der Programmierung (Verwaltung) der einzelnen Diagnoseskripte festgestellt.

In diesem Kapitel werden die grafischen Möglichkeiten der Diagnosesoftware untersucht. Im Geschäftsprozess Verwaltung und Nacharbeit wurden Schwachstellen ermittelt.

6.5.1 Systembruch und Prozesszeit: Positionierung und Gestaltung grafischer Ausgaben

In Kapitel 6.4.2 wurde das Hinzufügen und Bearbeiten von Jobs in den Diagnoseskripten analysiert. Es wurde der Vorgang bewertet, wie die Jobaufrufe in die Diagnoseskripte integriert werden.

Im Anschluss an den Einbau des Jobs muss dessen grafische Ausgabe aufbereitet werden. Mit den dafür notwendigen Schritten befasst sich dieses Kapitel.

6.5.1.1 Anwendungsfall: Anzeige Status Tür

Für das INPA-System gibt es einige fest definierte Objekte, um die Ergebnisse der Steuergeräte auszugeben. Beispiele sind ein einfarbiges LED oder eine zweifarbige Balkendarstellung. In dem Anwendungsfall von Kapitel 6.4.1 sollte eine LED den Status der Fahrertür offen oder geschlossen anzeigen.

Folgende Abbildung zeigt die bestehende Ansicht, um die die Statusanzeige erweitert werden soll:

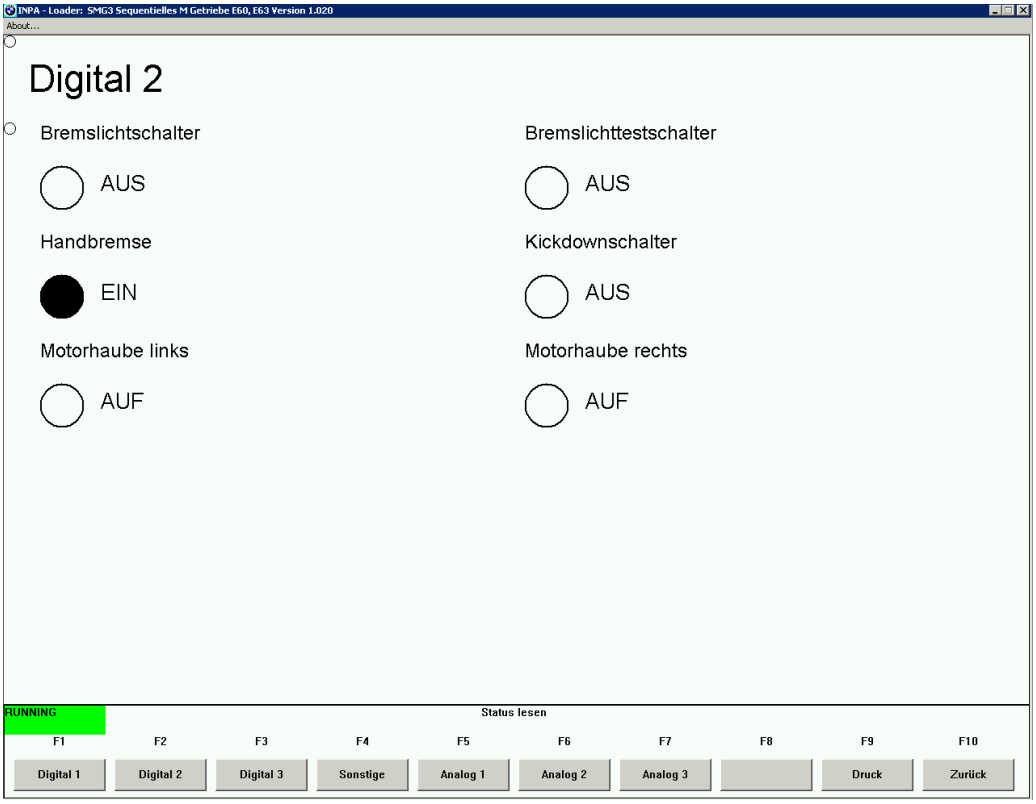


Abbildung 6.10 Diagnosemaske "Digital 2" des Diagnoseskripts SMG_60

Die zusätzliche LED soll in die erste Zeile direkt unter der Überschrift eingetragen werden. Hierfür ist es zunächst notwendig, das Ergebnis der Statusabfrage STAT_TUER_ZU in der Diagnoseskriptquelle einzutragen.

Die Erweiterung erfolgt per Texteditor auf folgende Weise:

```
        // Tür
        text(0,1,"Tür");
        ergebnisDigitalAusgabe("STAT_TUER_ZU", 2, 1," ZU   "," AUF  ");
```

Abbildung 6.11 Quellcode: Anzeige Türstatus

Die farbliche Markierung der Zahlen gibt die Position auf dem Ausgabemonitor an. Allerdings muss zusätzlich die Positionierung aller anderen LEDs nachgebessert werden. Denn die LED für die Bremslichtschalter rutscht durch das neue Element von Zeile 2 in Zeile 7.

Vorher:

```
        // Bremslichtschalter
        text(0,1,"Bremslichtschalter");
        ergebnisDigitalAusgabe("STAT_BREMSLICHTSCHALTER_EIN", 2, 1," EIN     ",
             " AUS  ");
```

Abbildung 6.12 Quellcode: Positionierung Bremslichtschalter

Nachher:

```
        // Bremslichtschalter
        text(5,1,"Bremslichtschalter");
        ergebnisDigitalAusgabe("STAT_BREMSLICHTSCHALTER_EIN", 7, 1," EIN     ",
             " AUS  ");
```

Abbildung 6.13 Quellcode: Veränderte Positionierung Bremslichtschalter

Auf diese Weise muss mit allen weiteren LEDs auf der Seite manuell per Tastatureingabe verfahren werden.

6.5.1.2 Anwendungsfall: Anzeige Getriebeschema

Ein weiterer Anwendungsfall, der die grafische Gestaltung mit INPA verdeutlicht, ist die Abbildung eines Getriebeschemas. Es soll ein Abbild der Schaltsteuerung des Schalthebels dargestellt werden. Das Bild enthält für die Diagnosearbeit die Adaptionswerte und die Istwerte bei der Schaltung.

Folgende Ausgabe zeigt ein simuliertes Beispiel:

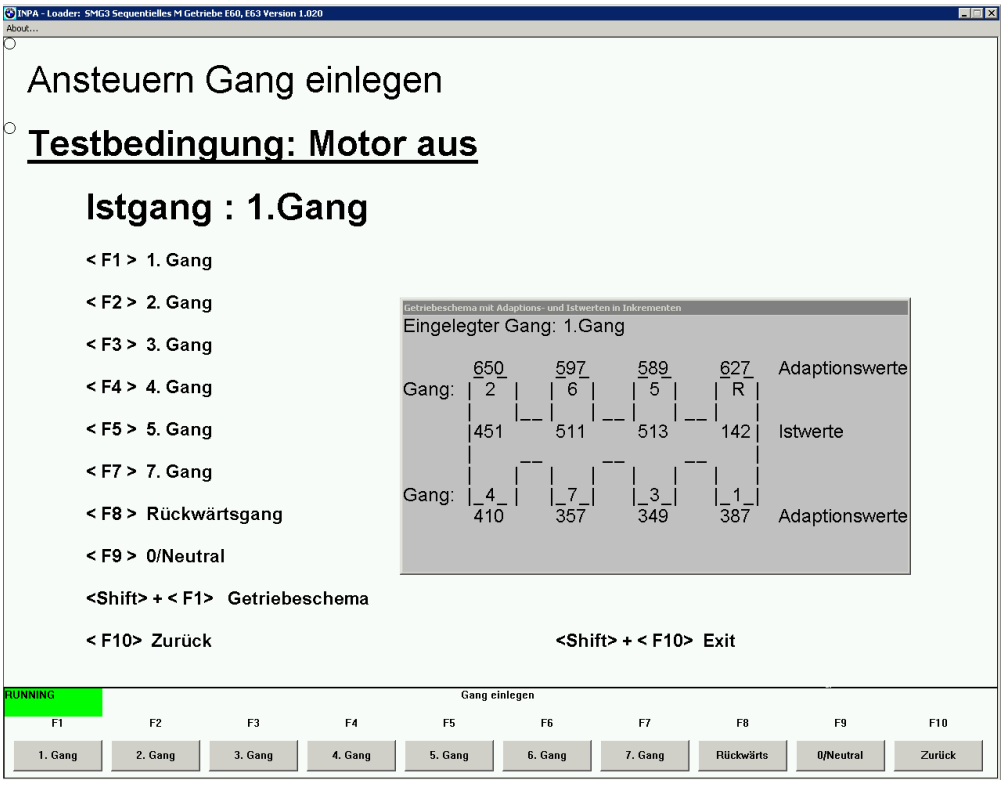


Abbildung 6.14 Diagnosemaske "Ansteuern Gang einlegen" des Diagnoseskripts SMG_60

Da die Auswahl an Zeichenobjekten sehr begrenzt ist und auch das Zeichnen von Linien mit INPA nicht möglich ist, muss der Programmierer dieses Bild per ASCII-Zeichen abbilden.

Ein Auszug der Programmierung stellt sich wie folgt dar:

```
      userboxftextout(3,"_",  2,x_offset+23,0,0);
      userboxftextout(3,"_",  2,x_offset+25,0,0);
      (…)
      userboxftextout(3," |", 3,x_offset+ 1,0,0);
      userboxftextout(3,"2",  3,x_offset+ 3,0,0);
      userboxftextout(3," |", 3,x_offset+ 5,0,0);
      //Beschriftung:-----------------------------------------------
            userboxftextout(3,"Gang:",          3,          0,0,0);
            userboxftextout(3,"Adaptionswerte", 2,x_offset+28,0,0);
            userboxftextout(3,"Istwerte",       5,x_offset+28,0,0);
            userboxftextout(3,"Gang:",          8,          0,0,0);
            userboxftextout(3,"Adaptionswerte", 9,x_offset+28,0,0);
```

Abbildung 6.15 Quellcode: Anzeige Getriebeschema

Die Positionierung jedes ASCII-Zeichen muss vom Programmierer einzeln und manuell eingegeben werden (siehe Zahlenmarkierung in Abbildung 6.15).

Um die Tätigkeit nach der Eingabe der ASCII-Zeichen zu erläutern, wird der Anwendungsfall in den zugehörigen Prozessausschnitt eingegliedert. Kapitel 6.5.1.3 stellt diesen Ausschnitt dar.

6.5.1.3 Prozessausschnitt: Grafik positionieren

Die folgende Abbildung 6.16 zeigt einen Ausschnitt aus dem in Kapitel 5.8.3 vorgestellten Teilprozess "Angeforderte Funktionalität ergänzen/verbessern". Der Ausschnitt verdeutlicht die Arbeit, die der Programmierer nach der Änderung der Diagnoseskriptquelle durchführen muss, um das Ergebnis auf der Diagnoseoberfläche betrachten zu können.

Der Programmierer muss zunächst das geänderte Skript kompilieren. Dies geschieht mit der Anwendung "SGBD INPA Dienste Oberfläche". Danach kann er die Ausgabe in der Anwendung INPA-Loader betrachten.

Ist die Positionierung noch fehlerhaft, muss der Programmierer wieder zurück zum Texteditor und dort die entsprechende Stelle korrigieren. Nach dem Speichern der Quelle ist das erneute Kompilieren der Quelle erforderlich.

In den meisten Fällen muss dieser Prozessabschnitt mehrfach wiederholt werden. Denn ein Gebilde, wie z. B. der Getriebeschaltweg, kann auf diese komplexe Art der Programmierung kaum durchgängig programmiert werden.

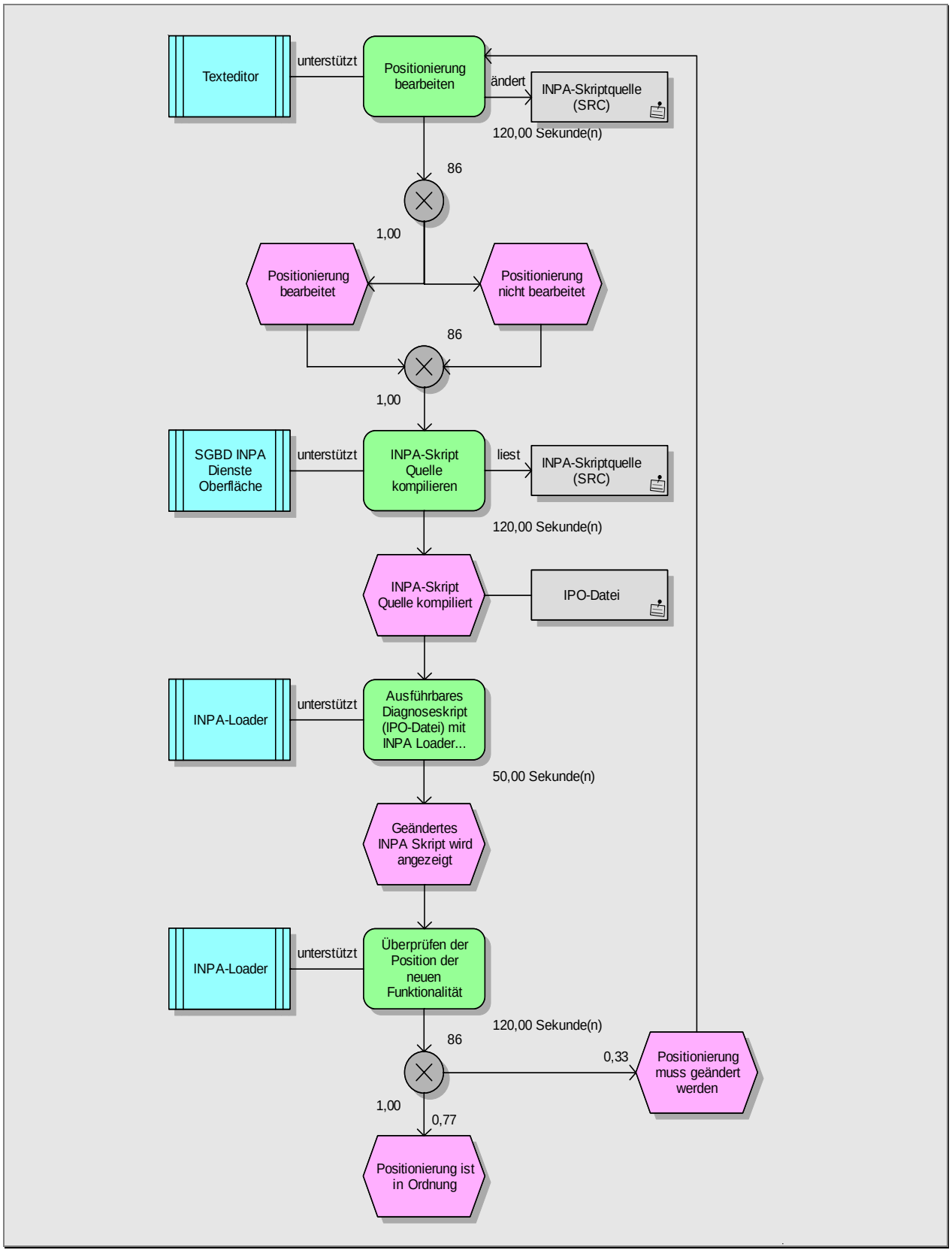


Abbildung 6.16 Prozessausschnitt: Grafik positionieren (EPK)

6.5.1.4 Stellenwert der Anwendungsfälle

Der Prozess fiel zum einen durch die automatisierte Systembruchanalyse von ARIS auf. Hier gingen aus den Auswertungsberichten deutliche Systembrüche hervor.

Zum anderen wurde bereits während der Ist-Erfassung des Prozesses klar, dass an dieser Stelle Optimierungsbedarf besteht. In Gesprächen wiesen die Programmierer betont auf die umständliche Arbeitsweise hin.

Des Weiteren haben die Zeiterfassungen gezeigt, dass die Positionierungstätigkeit einen großen Anteil an der Gesamtarbeitszeit in Anspruch nimmt.

Auf die Wichtigkeit des Prozesses wurde schon in Kapitel 6.3.1.2 hingewiesen. Es sei an dieser Stelle nur mehr vermerkt, dass jeder der 450 Fälle zur Bearbeitung von Diagnoseskripten pro Monat und jeder der monatlich 40 Fälle zur Neuerstellung von Diagnoseskripten auch eine grafische Bearbeitung erforderlich macht. Es wird dabei der in Kapitel 6.5.1.3 abgebildete Prozessausschnitt mindestens einmal durchlaufen. Mit Ausnahme eines kleinen Teils, bei dem bei der Skript-Bearbeitung der Job nur zu löschen ist (ca. 10 %).

Der Prozessabschnitt nimmt somit eine wichtige Rolle bei der täglichen Arbeit des Diagnoseskript-Programmierers ein. Die folgenden Schwachstellen sollen deshalb durch ein verbessertes Konzept aufgelöst werden.

6.5.1.5 Schwachstelle: Programmwechsel für grafische Positionierung

Die Schwachstellenanalyse des Prozesses "Angeforderte Funktionalität ergänzen bzw. verbessern" hat ergeben, dass auch hier Anwendungssystembrüche bei knapp 50 % aller Funktionsübergänge vorliegen. Diese befinden sich alle in dem aufgezeigten Prozessausschnitt.

Der Übergang vom Texteditor zur SGBD INPA Dienste Oberfläche und anschließend zum INPA-Loader sowie die Rückführung zum Texteditor bei Nachbesserungen schaffen eine unruhige und fehleranfällige Arbeitsweise bei der Programmierung.

6.5.1.6 Schwachstelle: Zeitaufwendige Positionskontrolle und Nachbesserung

Folgende Berechnung zeigt die Bedeutung, die der aufgezeigte Abschnitt für den gesamten Prozess hat.

```
Bereich: Kooperationspartner, Basissystementwicklung

Mittlere Durchlaufzeit Prozessschritt "Grafik positionieren" (Minuten): T1  = 9:05
Mittlere Durchlaufzeit Prozess "Angeforderte Funktionalität
      ergänzen bzw. verbessern" (Minuten): T2                               = 16:50
Anteil an Prozess (Prozent): P1 = T1/T2 * 100 %                             = 53,96
Faktor der Erhöhung von MDZ, beim Anwendungsfall
      Getriebeschema (Kapitel 6.5.1.2): F1 = ((T2+T1*4)/T2 – 1)             = 2,1

(Quelle: INPA-Prozessdatenerfassung 2006, Prozessdaten "INPA-Skript erstellen / nach-
bearbeiten" und "Job hinzufügen")
```

Abbildung 6.17 Auswertung: Anteil der Tätigkeit "Grafik positionieren" an der Prozesszeit

Damit nimmt der Prozessabschnitt beim einmaligen Durchlaufen bereits über 50 % der Gesamtzeit ein. Bei der Ist-Erfassung wurde allerdings ermittelt, dass bei ca. einem Drittel aller Fälle eine Nachbesserung der Positionierung erforderlich ist.

Bei so komplexen Fällen wie der Abbildung des Getriebeschemas wurden ca. fünf Positionierungsdurchläufe geschätzt. In diesem Fall erhöht der Prozessausschnitt die Gesamtdurchlaufzeit um das Zweifache.

6.5.2 Prozessqualität und Anwenderzufriedenheit: Ausgabe auf der Diagnosemaske

Dieses Kapitel beschäftigt sich mit der Einsatzfähigkeit der derzeitigen HMI-Software beim Kooperationspartner vor Ort. Es bewertet die Bedienbarkeit unter dem Aspekt der grafischen Möglichkeiten.

6.5.2.1 Stellenwert der Anwendungsfälle

Folgende Berechnung zeigt welche Bedeutung die HMI-Software allein für das Tagesgeschäft in der Nacharbeit hat.

```
Bereich: Kooperationspartner, Kfz-produzierendes Werk (Nacharbeit)
- Nacharbeitsdatum zwischen 01.11.2006 und 30.11.2006 23:59:59
- Nacharbeitsdatum zwischen 01.10.2006 und 31.10.2006 23:59:59

Nacharbeitsfälle (pro Monat): X1                              = 2565
Nacharbeitsfälle, rein mechanisch (pro Monat): X2             = 636
Nacharbeitsfälle, elektronisch/mechatronisch (pro Monat): X3  = 1929
Nacharbeitsfälle, mechatronisch, Einsatz INPA (pro Monat): X4 = 979
Nacharbeitsfälle, Einsatz INPA, die länger als 48h
      Status n. i. O. haben (pro Monat): X4                   = 205
Bearbeitungsdauer Nacharbeit,
      elektronisch/mechatronisch (pro Monat, Minuten): T1     = 147853
Durchschnittliche Bearbeitungsdauer Nacharbeitsfall,
      elektronisch/mechatronisch (Minuten): D1                = 76,7
Anteil, mechatronisch, Einsatz INPA: A1 = X1/X4 * 100 %       = 50,8

(Quelle: INPA-Prozessdatenerfassung 2006, Prozessdaten "Montagebericht, Nacharbeit,
TD-47")
```

Abbildung 6.18 Auswertung: Anteil der Einsätze des Ist-Systems in der Nacharbeit

Die Auswertung der Montageberichte hat ergeben, dass die Hälfte aller mechatronischen Nacharbeitsfälle die Arbeit mit INPA erfordert. Dies umfasst einen Arbeitseinsatz mit der HMI-Software von über 1200 Stunden pro Monat allein in der Nacharbeit im Werk Dingolfing.

Wie in Kapitel 5.3 erläutert, findet das System Einsatz in den Geschäftsprozessen Nacharbeit, Analyse und Steuergeräteentwicklung. An dieser Stelle sei erwähnt, obwohl der vorrangige Einsatzort die Nacharbeit ist, dass sich die Schwachstellen aus dem grafischen Umfeld auch in der Analyse und Steuergeräteentwicklung wiederfinden.

6.5.2.2 Anwendungsfall: Diagnose Adaptionswerte

In Kapitel 6.4.1.2 wurde der Aufwand beschrieben, ein Getriebeschema auf einer Diagnosemaske abzubilden.

Das Schema liefert eine Anzeige der Schaltung mit Adaptionswerten[161] und Istwerten. Die Diagnosemaske dient dazu, die Adaption der Schaltung zu konfigurieren. In verschiedenen Fällen wird bei der Endmontage-Prüfung (Final) festgestellt, dass die Schaltung nicht korrekt abgestimmt ist. Daraufhin wird das Fahrzeug in die Nacharbeit zur Korrektur des Fehlers ausgeschleust. Die Nacharbeiter nutzen daraufhin die Diagnosemaske zur Neuadaption der Schaltwerte.

6.5.2.3 Schwachstelle: Farbliche Darstellung und Aktualisierung

Trotz der aufwendigen Art mit der die Abbildung des Getriebeschemas programmiert werden muss, erfüllt sie nicht die Erwartungen der Diagnosemitarbeiter.

Innerhalb der manuell gezeichneten Grafik ist es nicht möglich, nur die Werte zu erneuern. Bei der Erneuerung nur eines Werts muss die gesamte Gangdarstellung neu gezeichnet werden. Je nach Aktualisierungsrate kommt es daraufhin zum Flackern der Ansicht. Da das Abrufen der Werte einige Millisekunden dauert, wird in dieser Zeit nur ein weißer Bildschirm dargestellt. Die Anpassung der Adaptionswerte ist trotz der automatisierten Adaption ein Vorgang, der sich über mehrere Minuten erstreckt. Während dieser Zeit muss der Mitarbeiter die Istwerte permanent im Auge behalten. Für einen Benutzer, der die Anzeige länger betrachten muss, ist das Flackern sehr unangenehm und es erschwert die Konzentration.

Ein weiteres Defizit bei der Darstellung ist die fehlende Möglichkeit, die Werte farbig auszugeben. Besonders bei der Überschreitung von Schwellwerten wäre eine Markierung in der sonst üblichen roten Signalfarbe sehr hilfreich. Da dies in der herkömmlichen Form der grafischen Ausgabe nicht möglich ist, muss der Mitarbeiter in eine andere Ansicht wechseln. Der Maskenwechsel behindert den Arbeitsfluss. Er stellt zum anderen auch ein Qualitätsrisiko dar, da der Mitarbeiter durch die Maske nicht auf die Überschreitung von Schwellwerten hingewiesen wird.

6.5.2.4 Anwendungsfall: Anzeige der integrierten Heiz- und Klimaanlage (IHKA)

Typische Fälle für die Nacharbeit sind auch Fehlfunktionen in der Komfortelektronik. Ein Beispiel ist die Temperaturregulierung durch die Heiz- und Klimaanlage. Da hier die Arbeit mehrerer Steuergeräte zusammenfließt, ist die Fehleranfälligkeit entsprechend höher. Das zentrale Steuergerät für die Analyse ist die IHKA (integrierte Heiz- und Klimaanlage).

Der Diagnosemitarbeiter versucht über die folgenden Masken zu ermitteln, warum sich die Temperatur im Fahrzeug durch die Bedienanlage nicht erhöhen lässt.

[161] Adaptionswerte bestimmen z. B. in einem Automatikgetriebe den Zeitpunkt, zu dem in den nächst höheren bzw. niedrigeren Gang geschaltet wird.

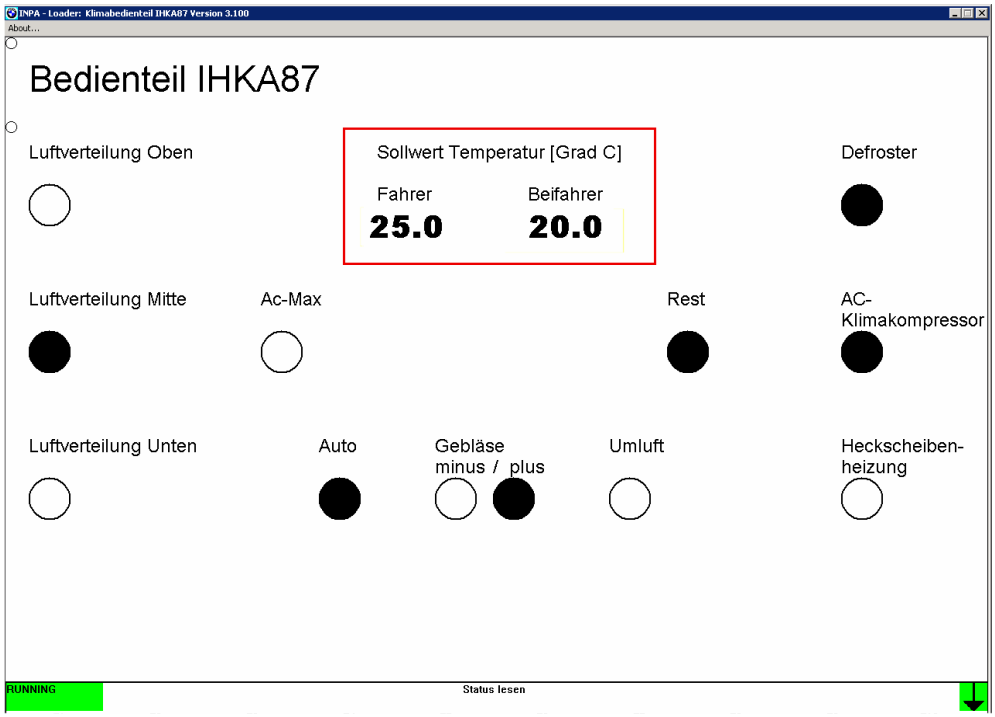


Abbildung 6.19 Diagnosemaske "Bedienteil" des Diagnoseskripts IHKA87

Die Maske zeigt, dass am Bedienteil eine Solltemperatur eingestellt ist. Dies wird durch die rot markierte Sollwertanzeige von 25.0 für Fahrer und 20.0 für den Beifahrer symbolisiert. Es ist auf der Maske nur erkennbar welche Temperatur am Bedienteil eingestellt wurde. Es ist nicht erkennbar, ob die am Bedienteil eingestellte Temperatur vom Steuergerät übernommen wurde. Das Übernehmen bedeutet in diesem Zusammenhang, dass das Steuergerät auch tatsächlich die Temperatureinstellung vornimmt, die der Benutzer wünscht.

Um diese Informationen abzufragen, muss der Mitarbeiter in eine andere Maske wechseln. Folgende Abbildung zeigt, dass das Steuergerät auf die eingestellte Temperatur nicht reagiert. Die rote Markierung verweist auf eine Sollwert-Einstellung von jeweils 0 Grad für Fahrer und Beifahrer.

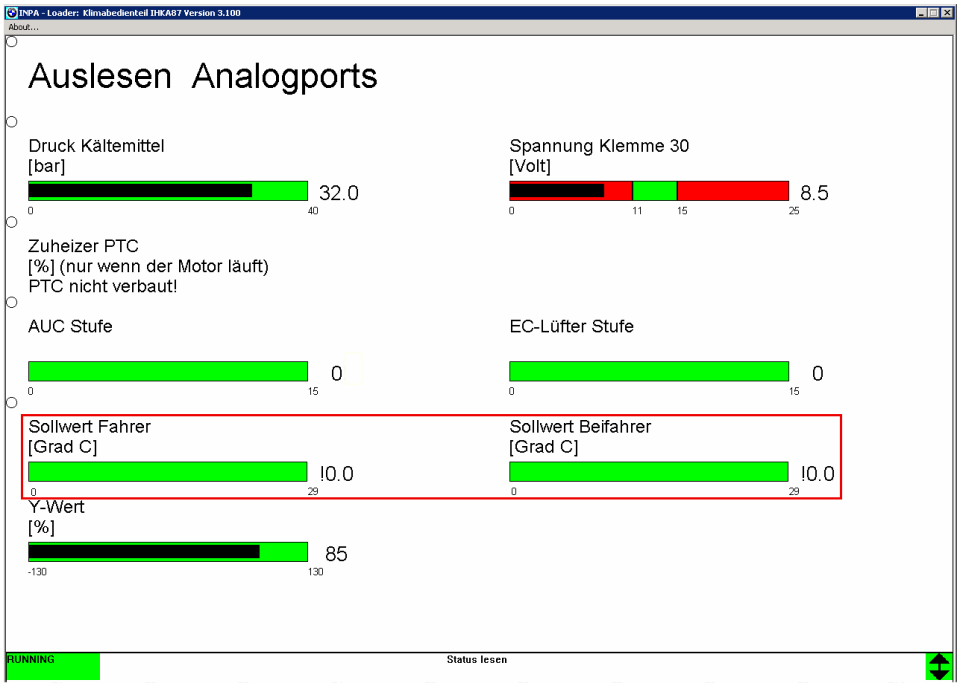


Abbildung 6.20 Diagnosemaske "Analogports (Teil 1)" des Diagnoseskripts IHKA87

Beim weiteren Durchblättern der Steuergerätewerte wird dem Diagnostiker angezeigt, dass die Temperatur im Innenraum 85 Grad beträgt, also weit über der Norm liegt und sich eigentlich schon im nicht mehr messbaren Bereich befindet, der bei 70 Grad beginnt (siehe rote Markierung folgende Abbildung).

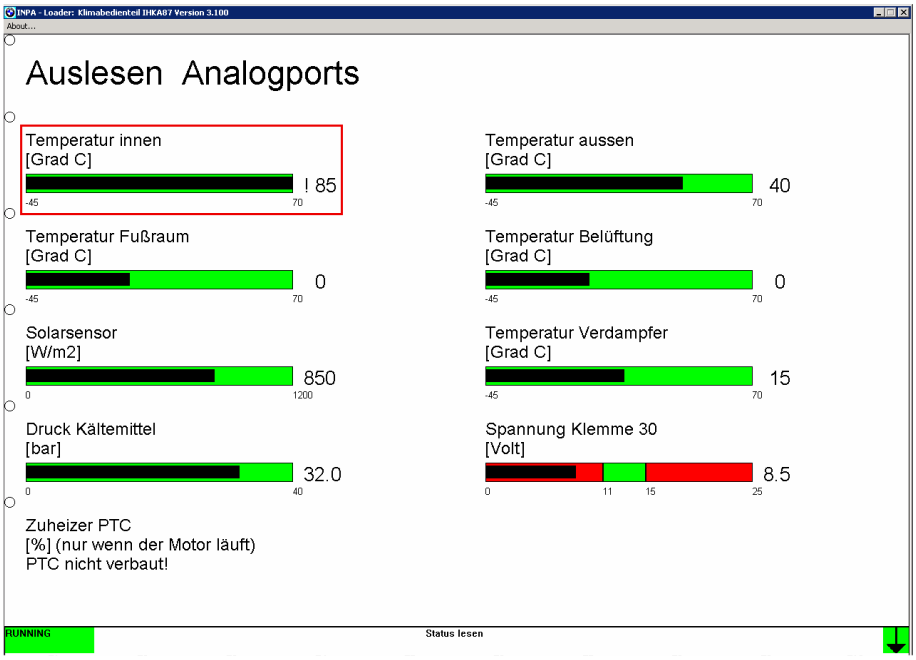


Abbildung 6.21 Diagnosemaske "Analogports (Teil 2)" des Diagnoseskripts IHKA87

Da sich die gefühlte Temperatur im Innenraum keinesfalls in diesem Rahmen bewegt, kann der Diagnosemitarbeiter daraus schließen, dass der Sensor, zur Abnahme der Innentemperatur, defekt ist. Ein Austausch des Sensors korrigiert den Wert. Aufgrund der irrtümlich gemessenen

überhöhten Innentemperatur hat das Steuergerät auf keine Einstellung zur Temperaturerhöhung mehr reagiert. Nach dem Austausch arbeitet das System wieder einwandfrei.

6.5.2.5 Schwachstelle: Realitätsnahe Gestaltung der Diagnosemaske

Der Anwendungsfall in Kapitel 6.5.2.4 zeigt die drei Standardelemente auf, die bei INPA zur Visualisierung der Steuergerätedaten genutzt werden. Das sind Text, LED (An-/Aus-Schalter in Kreisform) und der Regler mit Min-, Max- und Optimal-Wert-Anzeige.

Allerdings bietet das System keine Variation der Objekte hinsichtlich ihrer Größe, Ausrichtung, Farbgestaltung oder Ähnliches. Das Freihandzeichnen von Diagnosemasken ist nicht möglich. Nur der Umweg über die Programmierung von ASCII-Zeichenausgaben wird angeboten (siehe Anwendungsfall: Getriebeschema, Kapitel 6.4.1.2). Zudem ist es auch nicht möglich, Hintergrundbilder oder (animierte) Grafiken einzubinden.

Dies beeinträchtigt die Aussagekraft der Diagnosemaske. Teilweise wird versucht, wie bei der Abbildung des Heiz-Klima-Bedienteils, durch die Anordnung der LEDs zu symbolisieren, an welcher Stelle auf der Armatur sich der zuständige Knopf bzw. das zuständige Lämpchen befindet (siehe Abbildung 6.22). Dies ist aber nur bedingt möglich und nimmt viel Platz auf der Diagnosemaske ein.

Da sich der Diagnosemitarbeiter zunächst auf jeder Maske zurechtfinden muss, wird der Diagnoseprozess massiv verlangsamt. Durch die Einbindung einer schematischen Grafik mittels der die Werte abgebildet werden, kann die Orientierungsphase des Nutzers beschleunigt werden.

Abbildung 6.22 Bedienelement IHKA

Der Nutzer erkennt die Armaturen, wie sie im Fahrzeug verbaut sind, sofort wieder. Er findet so den gesuchten Wert sehr viel schneller. Außerdem kann der Platz für unnötige Beschriftungen anderweitig verwendet werden, da das Bild die Erläuterung übernimmt. Der gewonnene Platz kann so z. B. für weitere schematische Abbildungen genutzt werden. Zur Anzeige der Sensorwerte der Klimaanlage könnte z. B. ein Karosseriequerschnitt abgebildet werden.

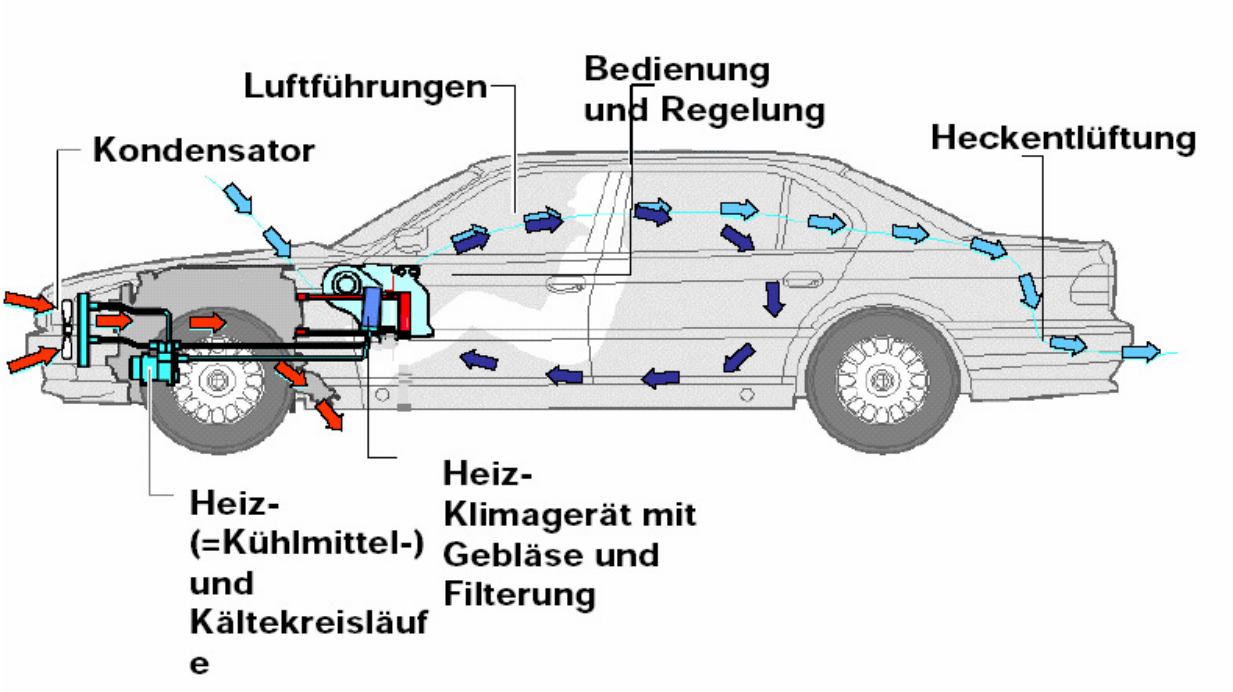


Abbildung 6.23 Karosseriequerschnitt mit Luftströmungsabbildung

Im vorliegenden Fall hätten die beiden Abbildungen dem Diagnostiker sofort gezeigt, dass am Bedienelement die Temperatur zwar entsprechend eingestellt ist, aber das Steuergerät für den Fahrzeuginnenraum eine unrealistisch hohe Temperatur misst. Der Aufruf von mehreren Diagnosemasken wäre somit unnötig.

6.5.2.6 Anwendungsfall: Anzeige Park-Distance-Control-Abstände

Besonders bei den neuen Technologien, die durch den Vormarsch der Steuergeräte im Fahrzeug Einzug erhalten haben, lässt sich erkennen, dass die Visualisierungsmöglichkeiten der derzeitigen HMI-Software nicht mehr den heutigen Anforderungen entsprechen.

Ein Beispiel ist PDC (Park Distance Control). 2002 wurde hierfür das erste Diagnoseskript erstellt. Bei PDC berechnen Ultraschallsensoren an der Rück- bzw. Vorderseite des Fahrzeugs die Position möglicher Hindernisse. Durch akustische Signale, die auf diese Hindernisse aufmerksam machen, soll das Einparken erleichtert werden.

Folgende Abbildung zeigt die Abstände, die die Sensoren zu dem nächsten Hindernis ermittelt haben:

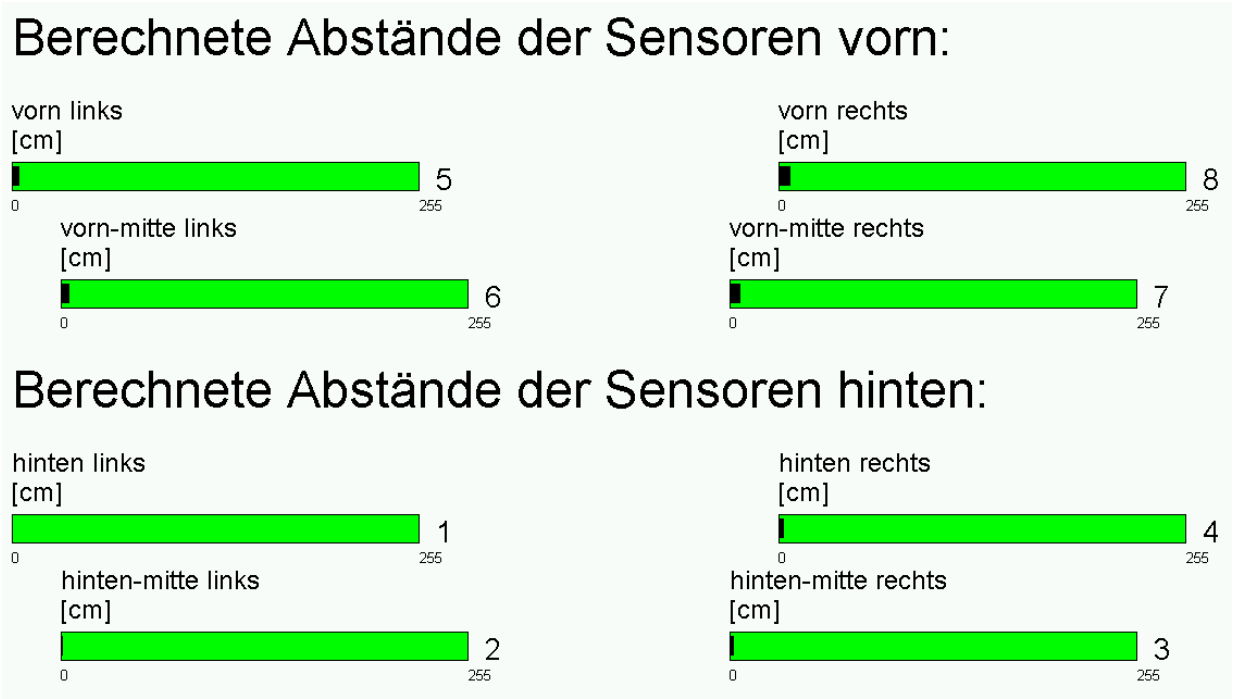


Abbildung 6.24 Diagnosemaske "Abstände" des Diagnoseskripts PDC_E65

Da PDC noch eine recht junge Technologie ist, müssen die Sensoren teilweise nach den finalen Prüfabläufen noch fein justiert werden. Dies erfolgt in der Nacharbeit.

6.5.2.7 Schwachstelle: Darstellung animierter Grafiken

Wie bereits bei den vorherigen Anwendungsfällen erwähnt, fehlt bei INPA die Möglichkeit, schematische Hintergrundbilder einzufügen und diese mit Animationen zu überlagern.

Im vorliegenden Fall würde es die Arbeit des Nacharbeiters erleichtern, wenn die Werte in Form einer grafischen Umsetzung dargestellt würden.

Abbildung 6.25 Animierte Grafik zur Anzeige PDC-Abständen

Wie in der Abbildung dargestellt, könnte für ein Fahrzeug die Entfernung zum Hindernis per hervorgehobene animierte Linien und Zahlenwerte abgebildet werden. Eine rote Linie könnte ferner das ermittelte Hindernis symbolisieren.

Durch diese Art der Darstellung könnte auch der Anwender schneller die Aussage der Diagnosemaske erfassen. Auch wäre diese Art der Darstellung platzsparender.

6.6 Analyseergebnisse zur Mobilität

In Kapitel 6.5 wurde die hohe Anzahl an mechatronischen Fehlern genannt, die während des gesamten Fertigungsprozesses auftreten können. Fehlerhafte Fahrzeuge sollen allerdings so schnell wie möglich wieder in den Vertriebsprozess eingeschleust werden.

Deshalb ist es notwendig, dass die Fehler schnell diagnostiziert und behoben werden. Besonders in der Nacharbeit herrscht deshalb großer Zeitdruck.

Der Arbeitsfluss des Nacharbeiters sollte daher nicht durch sperrige oder sogar immobile Diagnosegeräte beeinträchtigt werden. Deshalb werden in diesem Kapitel die von der HMI-Software betroffenen Prozesse hinsichtlich des Faktors Mobilität bewertet.

6.6.1 Anwenderzufriedenheit und Prozesszeit: Erreichbarkeit und Transport des Diagnosegeräts

Die Eigenschaft Mobilität wurde als eine der drei elementaren Anforderungen für eine verbesserte HMI-Diagnosesoftware definiert (siehe Kapitel 4). Der Punkt wurde verstärkt durch die Mitarbeiter des Kooperationspartners hervorgehoben.

Es gibt eine Vielzahl von Anwendungsfällen aus der täglichen Diagnosetätigkeit, bei denen ein höherer Mobilitätsfaktor der HMI-Software von Vorteil wäre. Ein paar Beispiele werden im Folgenden vorgestellt. Des Weiteren gibt es Ideen für die Erschließung neuer Diagnosemöglichkeiten in der Fertigung und im Service beim Kunden, die durch erhöhte Mobilität möglich wären. Auf diese soll im Ausblick der Arbeit abschließend eingegangen werden (siehe Kapitel 12).

6.6.1.1 Anwendungsfall: Fehlerhafte Funktionalität der Heizung im Hintersitzbereich

Bei einer Luxusbaureihe des Kooperationspartners lässt sich die Temperatur der Sitzheizung separat für den linken und rechten Rücksitz regeln. Das Steuergerät, die Aktoren (Heizgerät) und Sensoren (Temperatur Messgerät) hierfür befinden sich unter dem Rücksitz.

Bei einer fehlerhaften Heiztätigkeit muss der Nacharbeiter im Rücksitzbereich der Fahrerkabine die komplette Sitz- und Rückenlehnenverkleidung entfernen.

Sobald die Verkleidung entfernt ist, kann der Nacharbeiter die technischen Geräte für die Diagnose in Augenschein nehmen. Um den Fehler eingrenzen zu können, analysiert er nach der Freilegung über die Diagnosesoftware die einzelnen Funktionen des Steuergeräts.

Das Diagnoseterminal lässt sich jedoch nicht so positionieren, dass der Diagnostiker vom Fahrzeuginnenraum darauf zugreifen könnte. Deshalb muss der Nacharbeiter die Fahrerkabine verlassen, um am Diagnoseterminal eine Funktion auszulösen. Zur Überprüfung des ausgelösten Befehls muss er wieder die Fahrerkabine betreten. Je nach Schwierigkeitsgrad der Fehlerursache muss dieser Vorgang teilweise mehr als zehn Mal wiederholt werden. Bei dreitürigen Baureihen nimmt dieser Vorgang sehr viel Zeit in Anspruch. Da die Fahrzeuge aus dem finalen Test kommen und kurz vor dem Vertrieb stehen, darf das Interieur bei dem permanenten Ein-

und Aussteigen nicht beschädigt werden. Eine Beschädigung, würde einen Austausch und erneute Prüfsequenzen zur Folge haben.

6.6.1.2 Anwendungsfall: Defektes Steuergerät Schließanlage Kofferraum

Ein ähnliches Szenario zeigt sich bei einer nicht funktionsfähigen Schließanlage im Kofferraumbereich. Um das Steuergerät und dessen Schließmechatronik physisch zu erreichen, muss je nach Baureihe und Fehlerfall die Verkleidung im Innenraum des Kofferraums, des Deckels oder sogar die Heckstoßstange entfernt werden.

Nach dem Freilegen der zu untersuchenden Komponenten beginnt der Nacharbeiter mit der Diagnose. Aufgrund der steigenden Anzahl von elektronischen Geräten im Fahrzeug müssen diese sehr platzsparend im Kraftfahrzeug verbaut werden und sind deshalb meist (für den Menschen) nur schwer zugänglich. Deshalb befindet sich der Diagnostiker teilweise liegend unter dem Fahrzeug, wenn er die Schließanlage untersucht.

Es ergibt sich hierbei allerdings die gleiche Notwendigkeit wie im Anwendungsfall "Fehlerhafte Funktionalität der Heizung im Hintersitzbereich". Die Steuergerätebefehle müssen manuell am Diagnoseterminal ausgeführt werden. Das Terminal lässt sich zwar auf Rollen in seiner Position drehen, es besteht aber keine Möglichkeit für den Nacharbeiter liegend das Terminal zu bedienen. Die Werte am Monitor können auch nur schwer aus dieser Position abgelesen werden.

Der Nacharbeiter muss folglich für jeden Funktionsaufruf oder für das Wechseln der Ansicht seine Diagnoseposition unter dem Fahrzeug oder am Kofferraum verlassen, um die Diagnosesoftware zu bedienen.

Da das Terminal per Maus und Tastatur bedient wird, muss hierfür zum Teil auch die Schutzkleidung (wie Sicherheitshandschuhe) abgenommen werden.

6.6.1.3 Stellenwert der Anwendungsfälle

In modernen Fahrzeugen ist die Anzahl von Steuergeräten auf bereits über 80 gestiegen. Diese interagieren mit hunderten von verbauten Aktoren und Sensoren. Es ist deshalb kaum möglich all diese technischen Komponenten an einem für den Diagnostiker leicht zugänglichen Platz zu positionieren. Daher gibt es eine Vielzahl von weiteren Anwendungsfällen, bei denen der Nacharbeiter nicht direkt am Steuergerät vor Ort arbeiten und gleichzeitig die Diagnosesoftware bedienen kann.

Kritik daran wurde allerdings in diversen Anwenderbefragungen laut. Das subjektive Empfinden, dass dieser Vorgang den Arbeitsfluss behindere, ist sehr stark ausgeprägt. Die Mobilität scheint also zum einen für die Anwenderzufriedenheit von großer Wichtigkeit zu sein. Zum anderen lässt sich über Prozesszeitberechnungen leicht zeigen, dass der gesparte Gang zwischen Steuergerät und Diagnoseterminal eine deutliche Verkürzung bringt.

In Kapitel 6.4 wurde bereits gezeigt, dass der Anteil aller mechatronischen Nacharbeitsfälle, bei denen INPA genutzt wird, mit 50,8 % von 1929 Fällen pro Monat sehr hoch liegt. Als weiteres Argument für die Wichtigkeit dieser Analyse erscheint an dieser Stelle die Verweilzeit des Fahrzeugs in der Nacharbeit bis zur Korrektur des Fehlers.

Durchschnittlich sind es in einem Werk ca. 200 Fahrzeuge pro Monat, die eine Bearbeitung mit INPA erfordern und in der Nacharbeit mehr als 48 Stunden verbleiben müssen, bis ihr Fehler

behoben ist (siehe Kapitel 6.5.2). Dies liegt zum einen an der Komplexität der Nacharbeitsfälle, zum anderen an der langen Bearbeitungsdauer mit INPA, weshalb fehlerhafte Fahrzeuge nach der Ausschleusung nicht sofort bearbeitet werden können. Die ausgeschleusten Fahrzeuge werden somit erst verspätet in den Vertriebsprozess zurückgeschleust. Sie verursachen also indirekt Kosten durch Lagerung und Verzögerung des Verkaufs.

Deshalb soll anhand des Faktors Mobilität versucht werden, eine entsprechende Verkürzung der Prozesszeit zu erreichen und der Pufferbildung entgegen gewirkt werden. Im Hinblick auf die Mobilität wurden folgende Schwachstellen an der derzeitigen HMI-Software INPA aufgedeckt.

6.6.1.4 Schwachstelle: Lauffähigkeit auf mobilen Betriebssystemen

Laut Spezifikation wurde die HMI-Software INPA für den Betrieb auf einem PC 80386 mit 32 MB Arbeitsspeicher und Windows 95 konzipiert[162]. Die unterstützten Betriebssysteme sind also Windows 95, NT und höhere Versionen. INPA ist somit auf keinem mobilen Betriebssystem lauffähig.

Um trotzdem einen Mobilitätsfaktor zu erreichen, der höher ist als bei sperrigen und kaum beweglichen Diagnoseterminals, wurde der Einsatz von Notebooks untersucht. Der erwartete Vorteil wurde als nur gering eingestuft. Bei den Anwendungsfällen ergab sich kaum genug Platz, dass der Nacharbeiter selbst frei arbeiten konnte. Auch ein Notebook wurde deshalb als zu platzraubend eingestuft.

Die Forderung nach der Unterstützung eines mobilen Betriebssystems durch die Diagnosesoftware, ergibt sich aus der Nutzung von etwa handflächengroßen mobilen Endgeräten. Erst ab dieser Größe würde das Diagnosegerät den Nacharbeiter bei der Arbeit am Steuergerät nicht mehr behindern.

6.6.1.5 Schwachstelle: Nicht skalierbare Oberfläche

Selbst wenn die Diagnosesoftware INPA ein mobiles Betriebssystem unterstützen würde, wäre der mobile Einsatz nur unter Umständen möglich, denn die Oberfläche von INPA ist nicht frei skalierbar. Dies bedeutet, dass die in Kapitel 6.4 und 6.5 vorgestellten grafischen Objekte (LED, Regler etc.) weder vergrößert noch verkleinert werden könnten. Die Masken werden für die Nutzung auf einem üblichen Desktop-Monitor mit einer Auflösung von 1280x1024 erstellt. Eine Anzeige auf einem PDA würde folglich dazuführen, dass die Beschriftungen, die Zahlenwerte und teilweise die Regler nicht mehr erkennbar wären.

Da aber trotz des mobilen Einsatzes weiterhin die Nutzung auf dem Diagnoseterminal notwendig ist, wurde neben der mobilen Lauffähigkeit die fehlende Skalierbarkeit der Oberfläche bemängelt. Die Skalierbarkeit soll ohne merklichen Verlust der Bildqualität erfolgen.

6.6.1.6 Schwachstelle: Bedienung mit Tastatur erforderlich

Eine weitere Schwachstelle von INPA für eine mobile Nutzung ist die Notwendigkeit einer Tastatur. INPA verfügt weder über Schieberegler noch über Rädchen. Typisch für mobile Endgeräte ist die Nutzung per Touchscreen. Eine physische Tastatur würde den Transportkomfort beeinträchtigen. Deshalb sollte die Diagnosemaske ohne die Eingabe von Werten auskommen.

[162] Vgl. INPA 2003, S. 8

Um z. B. die Temperatur der Heiz-Klima-Automatik zu steuern, könnte ein Balken auf einer Leiste per Touchscreen verschoben werden. Dies ist bei der derzeitigen Diagnosesoftware nicht möglich. Der Wert muss noch per Eingabe über die Tastatur gesetzt werden. Solche Eingaben unterbrechen den Bedienfluss und erhöhen die Prozesszeit. Im Hinblick auf eine mobile Nutzung sind folglich die Möglichkeiten der Werteingabe derzeit nicht ausreichend.

7 Ansatzpunkte zur Prozessoptimierung (Soll-Modellierung) und Definition der daraus folgenden Anforderungen

Aufbauend auf der Analyse der Ist-Prozesse erfolgt nun die Soll-Modellierung. Das Ziel ist hier, verbesserte Prozesse (Soll-Prozesse) zu bestimmen. Sie bilden ein optimiertes Abbild der gegenwärtigen Prozesse[163].

Zur Optimierung von Prozessen bieten Praxis und Theorie viele bekannte Methoden. Beispiele sind Business Process Reengineering, kontinuierlicher Verbesserungsprozess, Total Cycle Time oder Change Management. Im ersten Kapitel 7.1 wird zunächst erläutert welche Methode sich am besten für diese Arbeit eignet.

Durch Unterstützung der ausgewählten Methode werden Optimierungsansätze definiert (Kapitel 7.2). Die Ansätze sollen die in Kapitel 6 aufgedeckten Schwachstellen auflösen. Um den Bezug zu der durchgeführten Analyse herzustellen, werden die Ansätze anhand der Analysekennzahlen aus Kapitel 6.2 gegliedert.

Um die positive Wirkung der Optimierungen theoretisch zu belegen, werden die Soll-Prozesse simuliert. Durch den Vergleich der Simulationsdaten mit den Ist-Daten wird gezeigt, dass eine praktische Umsetzung sinnvoll ist. Dies geschieht in Kapitel 7.3

Für die praktische Umsetzung der Soll-Prozesse ist eine entsprechende Diagnosesoftware erforderlich. Sie muss allen Anforderungen gerecht werden, damit die Prozesse optimiert durchgeführt werden können. Um die geeignete Software hierfür zu finden wird abschließend in diesem Kapitel ein Anforderungskatalog aus den Soll-Prozessanforderungen erstellt. Er bildet die Basis für eine Analyse der auf dem Markt angebotenen Software.

[163] Vgl. Becker 2000, S. 153

7.1 Einsatz der Prozessverbesserung als Optimierungsmethode

Trotz der Vielfalt an Konzepten unterscheidet man grundsätzlich zwei Arten der Prozessoptimierung: die Prozesserneuerung und die Prozessverbesserung[164]. Die bekannteste Methode der Prozesserneuerung ist das Business Process Reengineering (BPR). Häufig angewandte Methoden der Prozessverbesserung sind Total Cycle Time (TCT), kontinuierlicher Verbesserungsprozess (KVP oder auch KAIZEN) und Six Sigma[165].

In der Praxis schließen sich die Methoden nicht gegenseitig als Lösung aus, sondern sie ergänzen sich. Wegen ihrer Wechselwirkung ist es wichtig, diese Methoden nicht isoliert einzusetzen. Zur Erreichung eines optimalen Ergebnisses können die Methoden gleichzeitig angewandt werden bzw. kann von Fall zu Fall entschieden werden, wann welche Methode eingesetzt wird[166].

Bei BPR handelt es sich allerdings um einen ungleich radikaleren Ansatz, als bei den anderen genannten Methoden. BPR bedeutet, bestehende Strukturen radikal in Frage zu stellen und gegebenenfalls auf der "grünen Wiese" neu anzufangen[167]. Laut Ellis schrecken deshalb viele Unternehmen davor zurück. Dabei spielen die zweifellos vorhandenen Risiken eines solch rigorosen Ansatzes eine Rolle. Darüber hinaus kann der finanzielle Aufwand immense Ausmaße annehmen. Auch soziale Aspekte sind bei der Entscheidung gegen BPR oft von Bedeutung. So sind Mitarbeiter kaum bereit, vorhandene Besitzstände, Strukturen und Abläufe grundsätzlich in Frage zu stellen. Die Abwehrhaltung ist besonders ausgeprägt, wenn der eigene Arbeitsplatz oder der eigene Einflussbereich gefährdet sind[168]. Deshalb erfreut sich der wenig radikalere Ansatz durch Prozessverbesserung größerer Beliebtheit[169].

Besonders bei der Einführung einer neuen Software ist man auf bereitwillige Kooperation angewiesen. Einige Schwachstellen, die in Kapitel 6 aufgedeckt wurden, wurden bei der Mitarbeiterbefragung genannt, als man nach Schwachstellen fragte. Die Zustimmung, die man durch das Abstellen dieser Schwachstellen bekommt, soll nicht durch eine zu radikale Herangehensweise beeinträchtigt werden. Deshalb wird die Optimierung über eine punktuelle Prozessverbesserung vorgezogen.

Wie erwähnt, können die Verbesserungsmethoden auch als Mix angewandt werden. Das Ziel von TCT, KVP und Six Sigma ist es, die Prozessleistung zu steigern. Die Verbesserungen wirken sich in allen Methoden positiv auf die Anwenderzufriedenheit und Qualität aus. Sie legen den Schwerpunkt auf die gezielte Beseitigung von Schwachstellen[170].

Alle drei Methoden gehen hierbei nach dem Plan-Do-Check-Act (PDCA)-Zyklus vor: Problem erkennen/erfassen, Problem analysieren, Ursache beseitigen, Leistung prüfen und Lösung ein-

[164] Vgl. Schmelzer 2004, S. 247
[165] Vgl. Schmelzer 2004, S. 249
[166] Vgl. Schmelzer 2004, S. 250-251
[167] Vgl. Ellis 2004, S. 8
[168] Vgl. Schmelzer 2004, S. 252
[169] Vgl. Ellis 2004, S. 8
[170] Vgl. Schmelzer 2004, S. 254

führen[171]. Durch die Struktur der Arbeit ist diese Vorgehensweise sichergestellt. Dieses Kapitels beschäftigt sich mit der Beseitigung der Ursache und dessen Überprüfung (per Simulation).

Hierbei stützt sich die vorliegende Arbeit vorrangig auf die Ansätze aus TCT und KVP. Der Schwerpunkt von Six Sigma ist die Lösung komplexer Prozessprobleme, mit der hohe finanzielle Einsparungen angestrebt werden[172]. Die Optimierung ist allerdings ausschließlich auf objektiv messbare Daten gestützt. Grundlagen sind hier Pareto-Analyse oder diverse Histogramme aus Messsystemen. Da bei dem vorliegenden Konzept jedoch auch subjektive bzw. soziale Leistungsparameter zum Tragen kommen (siehe Kapitel 6.2), erscheint eine Mischung der Ansätze aus TCT und KVP als besser geeignet.

Bei TCT wird das Hauptaugenmerk auf die Prozesszeit gelegt. Die Prozesszeit soll verkürzt werden, um dadurch die Prozesskosten zu senken und dabei zumindest die Qualität des Prozesses erhalten[173]. Hierfür sieht TCT den Abbau von Barrieren vor. Zwei dieser Barrieren wurden vermehrt durch die Analyse aufgedeckt. Zum einen sind diese die Sachbarrieren in Form von nicht konsistenten Informationen. Sie treten z. B. bei Medienbrüchen bzw. Anwendungssystembrüchen auf. Es handelt sich dabei um Informationen, die nicht durchgängig in den notwendigen Anwendungen zur Verfügung stehen und so im Prozessverlauf umkopiert werden müssen. Zum anderen sollen bei TCT Prozessbarrieren aufgehoben werden, die sich in komplexen und für den Nutzer nicht transparenten Abläufe wiederfinden. Ein Beispiel hierfür ist das aufwendige Positionieren von Grafiken mit der bestehenden HMI-Software. Das Auflösen von Schwachstellen wie Systembrüchen oder zu langen Prozesszeiten wird deshalb durch die TCT-Methode gestützt.

Die Methode KVP konzentriert sich auf das Auflösen des anderen Typs von gefundenen Schwachstellen, nämlich der Anwenderzufriedenheit[174]. Bei KAIZEN bzw. KVP steht der Kunde im Mittelpunkt. Hierbei ist sowohl der interne als auch externe Kunde als Zielgruppe definiert. Es soll dessen Zufriedenheit gesteigert werden. Hierbei fordert KVP, dass bei der Schwachstellenanalyse die Mitarbeiter ihr Wissen sowie ihre Wünsche und Vorstellungen miteinbringen. Wie in Kapitel 6.2 erläutert, war dies ein wichtiger Bestandteil der Analyse. Die Auflösung der Schwachstellen, die aus der Kennzahl Anwenderzufriedenheit gewonnen wurden, deckt sich demzufolge vollkommen mit der Vorgabe des KVP.

Im nächsten Kapitel werden deshalb Verbesserungsansätze definiert, um die Schwachstellen im Bereich der Anwenderzufriedenheit, Prozesszeiten/-kosten und Anwendungssystembrüche aufzulösen.

[171] Vgl. Schmelzer 2004, S. 255
[172] Vgl. Schmelzer 2004, S. 270-278
[173] Vgl. Schmelzer 2004, S. 258-259
[174] Vgl. Schmelzer 2004, S. 262-269

7.2 Bestimmung der Verbesserungsansätze und Simulation der Verbesserungen (Soll-Prozesse)

In Kapitel 7.1 wurde erläutert, dass im vorliegenden Fall kein radikaler Erneuerungsansatz zur Optimierung gewählt werden soll. Vielmehr wird versucht die Optimierung durch punktuelle Prozessverbesserung zu erwirken.

Als Verbesserungsansatz wurde die Reduktion der Prozesszeit (Teil der Methode TCT), die Auflösung von Barrieren bzw. Brüchen (Teil der Methode TCT) und die Steigerung der Anwenderzufriedenheit (Teil der Methode KVP) priorisiert. Diese Schwachstellen wurden in Kapitel 6 über die gleichnamigen Analysekennzahlen ermittelt. Die Gliederung der Verbesserungsansätze orientiert sich an diesen Analysekennzahlen.

7.2.1 Auflösung der Systembrüche

Bei der Verwaltung der Diagnoseskripte konnten einige Anwendungssystembrüche ermittelt werden. Besonders beim Bearbeiten und Hinzufügen von Steuergerätebefehlen (Jobs) trat eine starke Häufung dieser Brüche auf. Über 60 % aller Funktionsübergänge führen zu einem Systembruch (siehe Kapitel 6.4.2.3).

Dies liegt hauptsächlich an der notwendigen Datenübertragung per Kopieren und Einfügen zwischen dem EDIABAS Toolset32 und dem Texteditor. Um den Systembruch aufzuheben, wäre es notwendig, die Zugriffsdaten für das Steuergerät (Jobs etc.) direkt im Texteditor (Bearbeitungsprogramm für Diagnoseskripts) abrufbar zu machen.

Der Prozess, wie in Kapitel 6.4.2.1 dargestellt, soll wie folgt verändert werden:

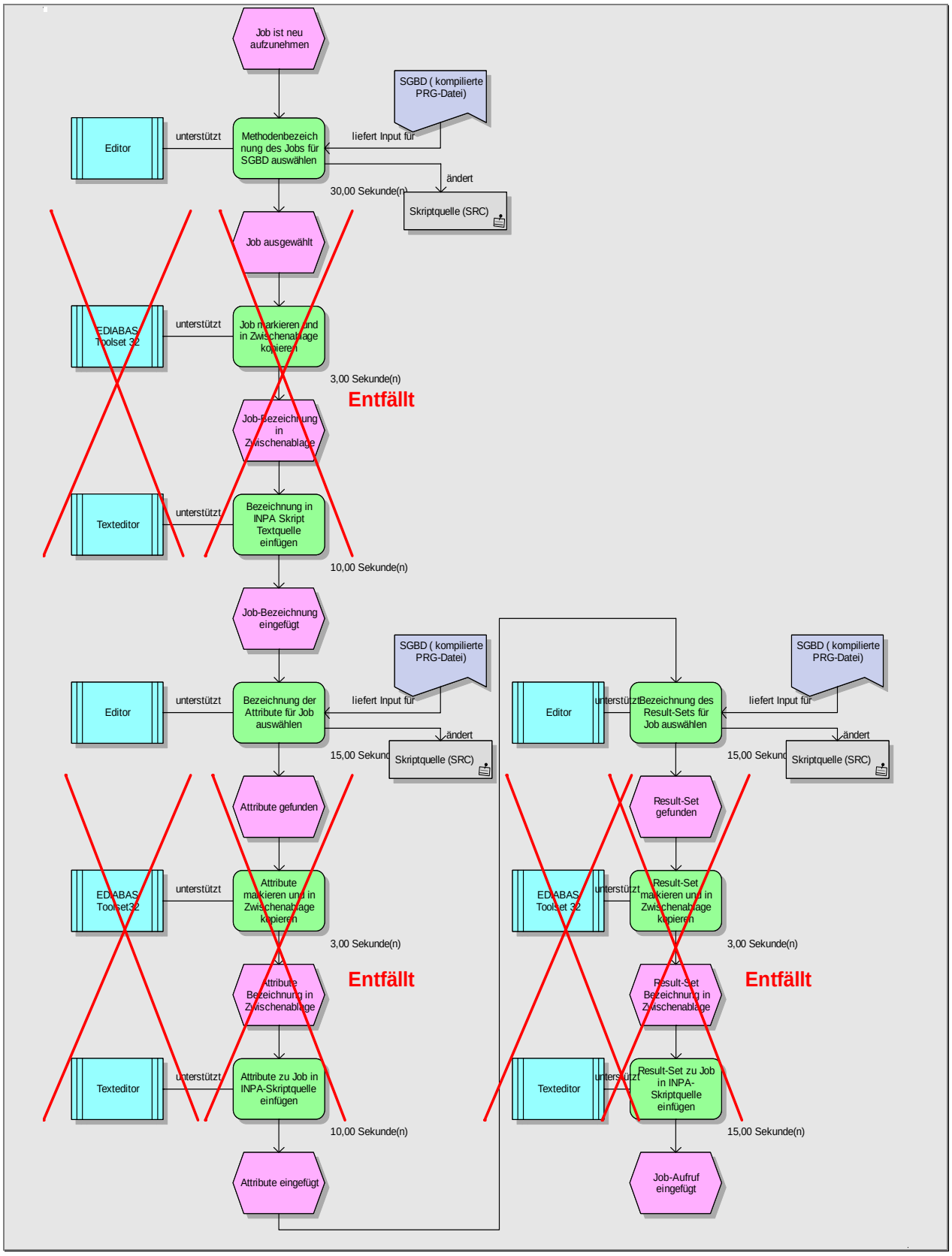


Abbildung 7.1 Soll-Arbeitsschritt: Job hinzufügen (EPK)

Name:	Soll-Arbeitsschritt: Job hinzufügen
Zuordnung:	GP: Skript Verwaltungsprozess TP: Skript erstellen/nachbearbeiten P: Angeforderte Funktionalität ergänzen/verbessern
Prozess-verantwortlicher:	Programmierer (Skriptbearbeiter)
Input:	Der Prozess wird durch den Prozess "Angeforderte Funktionalität ergänzen/verbessern" angestoßen (Kapitel 5.8.2). Der Prozess stellt als Input die zu bearbeitende Skriptquelle bereit.
Objekte:	- Skriptquelle (*.SRC-Datei) - SGBD - ~~Anwendungssystem Texteditor~~ (entfällt für neuen Editor) - ~~Anwendungssystem EDIABAS Toolset32~~ (entfällt für neuen Editor) - Anwendungssystem Editor
Ergebnisse:	Job-Aufruf hinzugefügt
Output:	Erweiterte Skriptquelle (*.SRC) Datei

Tabelle 7.1 Soll-Arbeitsschritt: Job hinzufügen (EPK)

Durch die Optimierung wird nur noch ein Anwendungssystem benötigt. Diese Anwendung wird schlicht als Editor bezeichnet, da sie repräsentativ für die neu einzuführende Software steht. Der Editor bietet die Möglichkeit, direkt die Steuergerätezugriffsdaten auszuwählen. Durch das Auswählen werden sie in das Diagnoseskript eingetragen.

Durch die Reduktion auf ein Anwendungssystem werden folglich die Systembrüche innerhalb dieses Arbeitsschritts komplett aufgelöst. Belegt wird dies durch die Simulation des Soll-Prozesses.

Die Simulation zeigt, dass sich die Anzahl der durchzuführenden Tätigkeiten von neun auf drei reduziert hat. Es ist somit auch eine Reduktion der Prozessdurchlaufzeit zu erwarten. Im Konzept TCT wird diese Folge als fast zwangsläufig bei der Auflösung von Brüchen angesehen. Deshalb wird bei TCT als erster Schritt immer versucht, solche Barrieren aufzuheben.

Des Weiteren zeigt der Analysebericht, dass bei den verbleibenden Funktionsübergängen (Abfolge der Tätigkeiten) die Zahl der Anwendungssystembrüche auf 0 und folglich auch die Rate auf 0 % gesunken ist. Der Bericht ist in der Quelle INPA-Analyseberichte 2007, Bericht "Job hinzufügen (Soll)", hinterlegt.

Das gleiche Ergebnis stellte sich durch die Einführung des Editors beim Soll-Arbeitsschritt "Job bearbeiten" ein. Auch hier konnte die Zahl der Anwendungssystembrüche auf 0 reduziert werden (siehe Quelle INPA-Analyseberichte 2007, Bericht "Job bearbeiten (Soll)").

Innerhalb der Verwaltungstätigkeit der Diagnoseskripte wurden noch weitere Systembrüche festgestellt. Um die Funktionalität eines Diagnoseskripts zu überprüfen, z. B. nachdem ein Steuergerätebefehl hinzugefügt wurde, muss das Skript kompiliert werden. Erst nachdem es kompiliert wurde, kann es geladen werden. Die Kompilierung erfolgt mit der Anwendung "SGBD INPA Dienste Oberfläche" (siehe Kapitel 6.5.2.4) und kann im INPA-Loader angezeigt werden. Falls im Anschluss noch Änderungen durchzuführen sind, erfolgen diese wieder

im Editor. Auch hier entstehen durch den Wechsel der Anwendungen drei bis vier Anwendungssystembrüche (siehe Quelle INPA-Analyseberichte 2007, Bericht "Angeforderte Funktionalität ergänzen bzw. verbessern"). Die Brüche konnten dadurch aufgelöst werden, dass das Diagnoseskript nicht mehr kompiliert werden muss, sondern direkt interpretiert werden kann. Die technischen Hintergründe werden in Kapitel 7.3 bei der Definition der softwaretechnischen Anforderungen noch genauer erläutert.

Auch weitere kleinere Systembrüche können durch Anforderungen an den Editor aufgelöst werden. Die Anforderungen werden in Kapitel 7.3 dargelegt. Der Prozess "Angeforderte Funktionalität ergänzen bzw. verbessern" konnte damit komplett von Systembrüchen befreit werden (siehe Quelle INPA-Analyseberichte 2007, Bericht "Angeforderte Funktionalität ergänzen bzw. verbessern (Soll)").

In den übrigen Geschäftsprozessen wurden durch die Analyse keine schwerwiegenden Systembrüche ermittelt.

7.2.2 Reduktion der Prozesszeit

7.2.2.1 Bereich Grafik

Die Analyse des Prozesses "Angeforderte Funktionalität ergänzen bzw. verbessern" hat ergeben, dass die grafische Positionierung über 50 % der gesamten Prozessdurchlaufzeit beansprucht (siehe Kapitel 6.5.1.6). Bei komplizierten grafischen Gestaltungen wird dieser Prozessabschnitt noch öfter durchlaufen und nimmt einen noch größeren Anteil ein. In Kapitel 6.5.1 wurde der Positionierungsvorgang aufgezeigt, um ein Getriebeschema abzubilden. Die Analyse hat ergeben, dass bei diesem Anwendungsfall die grafische Positionierung die Gesamtdurchlaufzeit verdoppelt.

Der Prozessausschnitt wurde in Kapitel 6.5.1.3 abgebildet und erläutert. Die derzeitige Arbeitsweise sieht vor, dass die Positionierung der Objekte, wie z. B. ASCII-Zeichen oder vordefinierte Objekte, wie z. B. LEDs, per Zahleneingabe erfolgt. Wie im Anwendungsfall erläutert, trägt der Benutzer mittels eines Texteditors die Werte in die Diagnoseskriptquelle ein. Danach muss die Quelle mit der Anwendung "SGBD INPA Dienste Oberfläche" kompiliert werden. Das erstellte Kompilat (Diagnoseskript) kann mittels der Anwendung "INPA-Loader" ausgeführt werden. Entspricht die grafische Gestaltung nicht den Vorstellungen des Programmierers, müssen die entsprechenden Textstellen in der Quelle korrigiert werden. Nach dem Speichern der Quelle ist das erneute Kompilieren der Quelle erforderlich.

Die textbasierte Bearbeitung des Diagnoseskripts ist sehr zeitaufwendig. Die Ausgabe von Objekten muss zunächst über das Eingeben des entsprechenden Funktionsaufrufs programmiert werden. Als Nächstes erfolgt die Positionseingabe per Zahlenwert. Die Werte bestimmen die Zeile und das Zeichen innerhalb der Zeile, in der das Objekt ausgegeben werden soll. Falls ein zusätzlicher Steuergerätebefehl in ein bestehendes Skript eingefügt werden muss, kann es sein, dass für das neue Objekt bestehende Objekte verschoben werden müssen. Dies bedeutet, dass für alle zu verschiebenden Objekte neue Angaben von Zeilen und Zeichenwerten eingetragen werden müssen.

Ein grafischer Editor würde die aufwendige textbasierte Positionierung überflüssig machen. Könnten die grafischen Objekte wie in einem üblichen grafischen Editor gezeichnet, in ihrer

Größe geändert und verschoben werden, wäre die Eingabe von Positionswerten nicht mehr notwendig.

Eine weitere zeitliche Verkürzung des Prozessabschnitts würde die Auflösung der Notwendigkeit von Kompilaten bringen. Ein Kompilat ist das Ergebnis der Übersetzung der Diagnoseskriptquelle in Maschinencode. Dieser Maschinencode kann vom INPA-Loader gelesen und ausgeführt werden. Für INPA ist die Bildung des Kompilats folglich eine strukturelle Notwendigkeit (siehe Kapitel 5.4.1). Technisch gibt es Möglichkeiten, auf den Vorgang des Kompilierens zu verzichten und den Quellcode zur Laufzeit direkt zu interpretieren. Unter Nutzung dieser Möglichkeit würde das Kompilieren entfallen. Unter Nutzung eines grafischen Editors und des Interpreter-Konzepts würde sich der Prozessabschnitt wie folgt verändern:

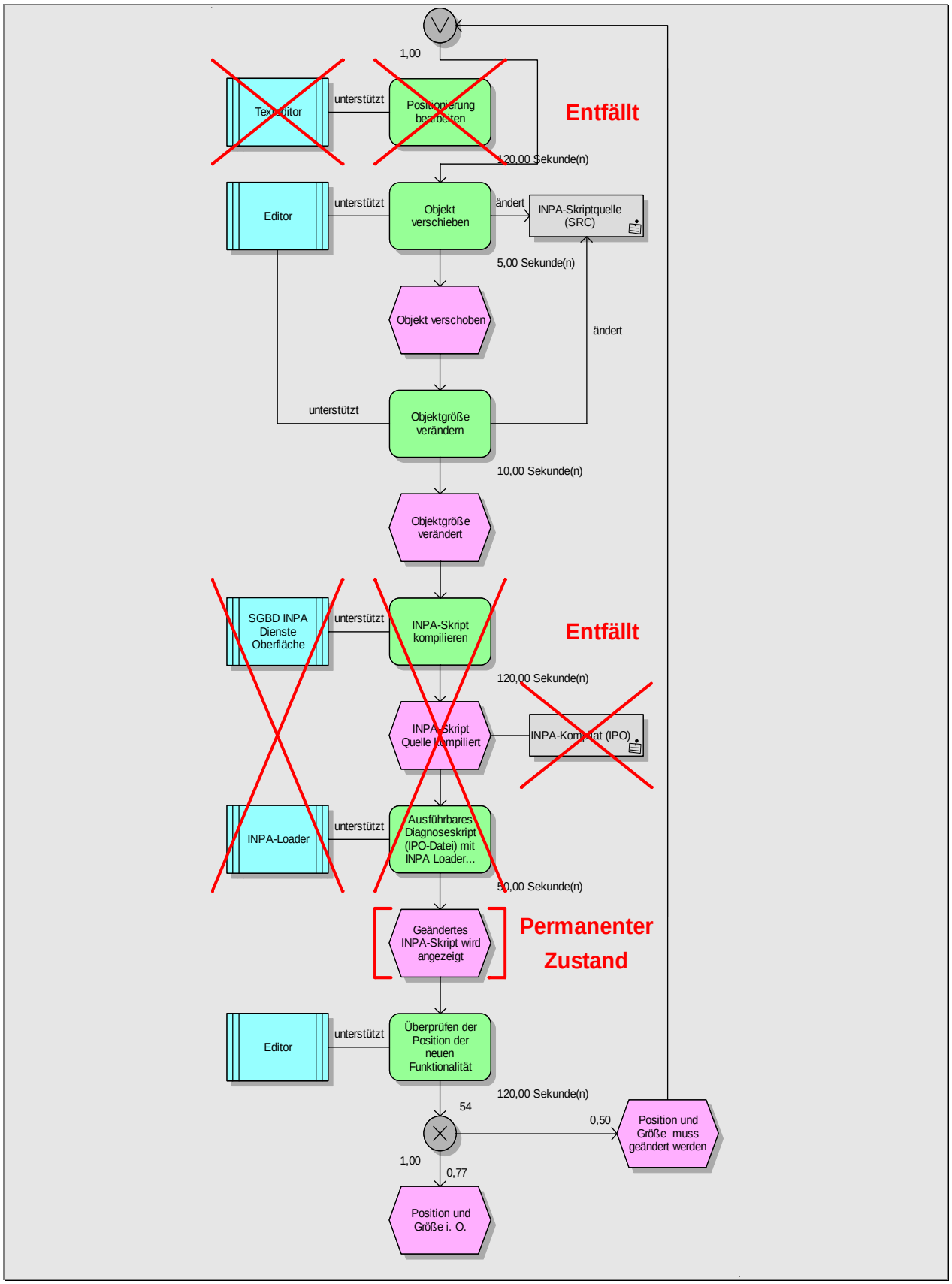


Abbildung 7.2 Soll-Prozessausschnitt: Grafik positionieren (EPK)

Durch die beiden Optimierungen verkürzt sich der Prozess lediglich auf die Arbeitsschritte Objekt verschieben und Objektgröße verändern. Bei einem grafischen Editor erfolgt dies üblicherweise per Mausbewegung direkt am Objekt selbst. Es entfällt die Notwendigkeit, die Diagnoseskriptquelle mittels der "SGBD INPA Dienste Oberfläche" in das INPA-Kompilat umzuwandeln. Auch das anschließende Betrachten mittels INPA-Loader entfällt, denn im Editor kann das geänderte Skript jederzeit betrachtet werden. Das Ereignis "Geändertes INPA-Skript wird angezeigt" wird zu einem permanenten Zustand während der grafischen Bearbeitung. Folglich kann auch die Positionierung direkt im Editor geprüft werden.

Durch die Prozesszeitverkürzung ergibt sich auch der Vorteil, dass weitere Anwendungssystembrüche aufgelöst werden. Die Anwendungen "SGBD INPA Dienste Oberfläche" und INPA-Loader entfallen.

Die Simulation des Soll-Prozesses hat folgende Zeitersparnis ergeben:

```
Bereich: Kooperationspartner, Basissystementwicklung E/E (inkl. Schätzwerte)

Mittlere Ist-Durchlaufzeit Prozessschritt
       "Grafik positionieren" (Minuten): I1                        = 9:05
Mittlere Ist-Durchlaufzeit Prozessschritt
       "Grafik positionieren" ohne Kompilierung (Minuten): IOK1    = 2:50
Anteil an Kompilierung an Ist-Durchlaufzeit
       Prozessschritt: A1=IOK1/I1 * 100%                           = 31,2
Geschätzte Soll-Durchlaufzeit Prozessschritt
       "Grafik positionieren" (Minuten): S1                        = 3:22
Geschätzte Reduktion beim Prozessschritt
       "Grafik positionieren" (Prozent): R1= S1/I1 * 100%          = 37,1

(Quelle: INPA-Prozessdatenerfassung 2006, Prozessdaten "Angeforderte Funktionalität
ergänzen bzw.  verbessern")
```

Abbildung 7.3 Auswertung: Reduktion bei Soll-Simulation "Grafik positionieren"

Die Verbesserung bringt demnach eine sichere Einsparung von ca. 2 Minuten 50 Sekunden durch Wegfall des Kompilierens. Für den gesamten Prozessabschnitt "Grafik positionieren" wird durch den Soll-Prozess eine geschätzte Ersparnis von 37 % vermutet. Die Optimierungen werden deshalb in den Anforderungskatalog aufgenommen.

7.2.2.2 Bereich Mobilität

Im Bereich Mobilität wurde die schlechte Erreichbarkeit des Diagnosegeräts kritisiert. Während der Nacharbeit am Fahrzeug muss der Kfz-Mechatroniker zur Diagnose das Steuergerät selbst und dessen elektronische Komponenten physisch erreichen. Gleichzeitig muss er aber auch Befehle über das Diagnosegerät an das Steuergerät schicken.

In Kapitel 6.6 wurden Anwendungsfälle beschrieben, in denen der Nacharbeiter Steuergerät und Diagnosegerät nicht gleichzeitig erreichen kann. Dies begründet sich zum einen dadurch, dass die Steuergeräte teilweise an sehr schwer zugänglichen Stellen verbaut sind, zum anderen durch die Immobilität des derzeitigen Diagnosegeräts. Ersteres muss als gegeben hingenommen werden. Durch die erwartete Zunahme der Steuergeräteanzahl in den nächsten Jahren ist in diesem Bereich eher noch eine Verschärfung der Situation zu erwarten. Auf die Immobilität kann allerdings im Rahmen der heutigen technischen Potenziale reagiert werden.

Neben Notebooks, die immer noch als zu sperrig und hinderlich für die Diagnosetätigkeit empfunden wurden, bietet der Markt seit einigen Jahren leistungsfähige Handgeräte an.

Sie sind durch ihre Größe vom Nacharbeiter leicht in der Arbeitskleidung verstaubar und behindern so dessen Bewegungsfreiheit nicht. Der Verbesserungsansatz besteht folglich darin, dass die Diagnosesoftware sowohl auf einem Desktop-Betriebssystem wie auch auf einem mobilen Betriebssystem lauffähig ist. INPA erfüllt diese Anforderung nicht.

Der Weg vom Steuergerät zum Diagnoseterminal kann unterschiedlich lang und schwierig sein. Dies hängt davon ab, wo das Steuergerät verbaut ist. Es gibt Geräte, die nur vom Bodenbereich des Fahrzeugs oder nur vom Innern des Fahrzeugs aus erreichbar sind. Das Gerät kann aber auch, wie in den eher seltenen Fällen, direkt zugänglich sein. Die Prozesszeitersparnis lässt sich deshalb kaum als aussagekräftiger Durchschnitt ermitteln. Sie ist aber dennoch offensichtlich. Der Nacharbeiter spart sich mit einem portablen Diagnosegerät den kompletten Weg zwischen Steuergerät und Diagnoseterminal. Die Zeitersparnis soll nicht als Durchschnitt, sondern deshalb anhand eines konkreten Anwendungsfalls belegt werden.

Als Beispiel dient der Anwendungsfall aus Kapitel 6.6.1. Er beschreibt eine Nacharbeit an einer defekten Rücksitzheizung. Der Nacharbeiter muss sich für die Diagnose im Rücksitzbereich des Fahrzeugs befinden. Dort muss die Rücksitzverkleidung entfernt werden, um die elektronischen Komponenten zu erreichen. Das Aus- und Einsteigen muss mit größter Sorgfalt geschehen, damit das Interieur nicht beschädigt wird. Für den Anwendungsfall haben sich folgende Daten ergeben.

```
Bereich: Kooperationspartner, Kfz-produzierendes Werk (Nacharbeit)

Gesamte Bearbeitungsdauer Anwendungsfall (Minuten): G1            = 24
Mittlere Prozesszeit, Arbeitsschritt: Weg zum Diagnoseterminal
      inkl. Aussteigen (Sekunden): M1                             = 26
Mittlere Prozesszeit, Arbeitsschritt: Weg zum Steuergerät
      inkl. Einsteigen (Sekunden): M2                             = 22
Anzahl Durchführung des Arbeitschritts:
      Steuergerätebefehl ausführen: X1                            = 5
Gesamte mittlere Prozesszeit für Wege M1 und M2: G2               = 48
Anteil der Wegezeit an der gesamten
      Bearbeitungsdauer (Prozent): A1 = G2*X1/(G1*60) * 100 %     = 16,7
Nacharbeitsfälle, mechatronisch, Einsatz INPA (pro Monat): X4     = 979
Anteil der INPA-betreffenden Nacharbeitsfälle mit
      Steuergeräten im Interieurbereich (Schätzwert in %): A2     = 40 %
Arbeitszeit für Wege M1 und M2
      (pro Monat in Stunden): Z1=G2/3600*X1*X4*A2/100             = 26

(Quelle: INPA-Prozessdatenerfassung 2006, Prozessdaten "Montagebericht, Nacharbeit,
TD-47")
```

Abbildung 7.4 Auswertung: Zeiten für Ein- und Aussteigen

Die Bearbeitungszeit des vorliegenden Anwendungsfalls würde sich um ca. 17 % verkürzen. Nimmt man den Fall als repräsentativ für alle Nacharbeitsfälle an, die Steuergeräte aus dem Interieurbereich betreffen, würde sich durch den Einsatz eines mobilen Diagnosegeräts auf einen Monat verteilt eine Zeitersparnis von ca. 3 Manntagen ergeben.

7.2.3 Steigerung der Anwenderzufriedenheit

Die Optimierung der Anwenderzufriedenheit lässt sich nicht wie andere, z. B. zeitlich basierte, Leistungsparameter konkret messen. Dennoch rechtfertigt sie sich als Kennzahl mit "sozialem" Charakter, besonders bei der Einführung einer neuen Software.

Beim Einsatz neuer Software und der daraus resultierenden Änderung der Arbeitsabläufe ist die Akzeptanz der Mitarbeiter sehr wichtig. Die Integrationsleistung der Mitarbeiter ist mitentscheidend für den Erfolg der Prozessverbesserung. Deshalb ist es empfehlenswert, neben den "Muss"-Anforderungen der Anwender (siehe Kapitel 6.2.4) auch deren Wünsche und Verbesserungsvorschläge zu prüfen und bei Möglichkeit umzusetzen.

Die Verbesserungsansätze, die sich aus der Anwenderunzufriedenheit ergeben, stellen sich wie folgt dar.

7.2.3.1 Bereich Funktionalität

In der Analyse ist bei der Mitarbeiterbefragung an mehreren Stellen die Vielzahl der notwendigen Anwendungen für die Entwicklung der Diagnoseskripte kritisiert worden: eine Anwendung zum Bearbeiten des Quellcodes, eine Anwendung zum Erfassen der benötigten Steuergerätedaten, eine Anwendung zum Kompilieren und eine Anwendung zum Ausführen der Maske.

Neben den nachgewiesenen Mängeln, wie Systembrüchen und hohem Zeitaufwand, wird dieser Umstand auch durch die INPA-zuständigen Programmierer kritisch betrachtet. Eine Reduktion auf eine Entwicklungsumgebung würde den Arbeitsfluss für den Programmierer der Diagnosemasken stark vereinfachen, weniger fehleranfällig machen und angenehmer gestalten. Letzteres würde die Akzeptanz für die neue Diagnosesoftware stärken.

Deshalb besteht die Optimierungsanforderung nicht nur in einem grafischen Editor (siehe 7.2.2.1), sondern auch in einer Entwicklungsumgebung, in der die Diagnosemaske während der Erstellung so anzeigt wird, wie sie auf der HMI-Software in der Nacharbeit später dargestellt wird. Auf diese Weise kann die Funktionalität unmittelbar während der Entwicklung getestet werden.

Um komplexere Diagnoseabläufe zu programmieren, muss weiterhin die Möglichkeit bestehen, den Quellcode nicht nur grafisch, sondern auch manuell zu editieren. Der Quellcode sollte nicht in einer proprietären Programmiersprache verfasst sein (siehe Kapitel 6.4.1.4). Die Verwendung eines weitverbreiteten Standards reduziert die Einarbeitungszeit und lässt das mühsame Erlernen einer Pseudo-Programmiersprache entfallen.

Ein weiterer Verbesserungsansatz zielt auf die Anzahl der zu verwaltenden Dateien für eine Diagnosemaske ab. Im jetzigen Diagnosesystem ist es notwendig, für jede Maske mindestens vier Dateien zu archivieren. Das sind der Quellcode (SRC), das Kompilat (IPO), ein notwendiges Zwischenprodukt des Kompilierens (IPS) und eine Datei, die die auszugebenden Textstellen in den verschiedenen Sprachen enthält (TXT). Soll allerdings dann eine Maske in einer anderen Sprache ausgeführt werden, kann der Wechsel nicht zur Laufzeit erfolgen, sondern es müssen pro Sprache eine weitere IPS- und IPO-Datei erstellt werden. Für eine Diagnosemaske in Englisch und Deutsch müssen folglich sechs Dateien verwaltet werden. Dies erzeugt natürlich Kosten bezüglich des Speicherverbrauchs, bedeutet aber vor allem für die Verwaltung einen enormen Zeitaufwand. Der Referenzierer (zuständige Verwalter der Dateien) muss auch bei minimaler Bearbeitung der Diagnosemaske alle Dateien erneuern. Als Verbesserungsansatz wird gefordert, die Anzahl auf eine einzige Quelle zu reduzieren. Die Quelle soll auch direkt ausgeführt werden können, sodass keine Kompilate benötigt werden. Diese Datei wird ergänzt durch eine Sprachdatei, aus der zur Laufzeit die Übersetzungen gelesen werden und so die Sprache zur Laufzeit geändert werden kann.

7.2.3.2 Bereich Grafik

Von den Anwendern vor Ort wurde vor allem die nicht zweckmäßige Gestaltung der Diagnoseoberfläche bemängelt. Im Detail umfasste die Kritik die geringe Farbtiefe, die Zeichenelemente und die optisch sichtbare Aktualisierung.

Es können nur die drei Elementarfarben abgebildet werden, ergänzt durch Schwarz und Grün sowie Weiß als Hintergrundfarbe. Die Verbesserung soll mindestens in einer 16 Bit Farbskala bestehen um Bilder und Elemente realitätsnah abbilden zu können. Die Beschränkung sollte eher durch die Hardware (das Diagnosegerät) erfolgen und nicht durch die Software derartig eingeschränkt sein.

Zur umfassenden realistischen Darstellung von Bedienelementen, wie der in Kapitel 6.5.2.4 dargestellten Klimabedienoberfläche ist es erforderlich, dass Hintergrundbilder dargestellt werden können. Diese sollen auch für Dynamiken und Interaktionen, wie z. B. das optische Abbilden von An-/Aus-Knöpfen, genutzt werden können.

Das Funktion Freihandzeichnen im grafischen Editors soll die Darstellung abrunden. Elemente, die nicht als Bilder zur Verfügung stehen, können über die standardmäßigen grafischen Werkzeuge Kreis, Linie, Rechteck etc. gezeichnet werden.

Auf diese Weise wird dem Diagnostiker vor Ort eine Oberfläche geboten, auf der er sich sofort zurechtfindet. Er muss nicht erst durch Lesen der Elementbeschreibungen erfassen, welches Objekt welchen Status anzeigt, sondern sieht die Anzeigefläche auf die Art vor sich, wie er sie vom Fahrzeug her kennt.

Weitere Unzufriedenheit schafft das Aktualisierungskonzept von INPA. Jede Maske muss für die Erneuerung von einzelnen Werten komplett neu gezeichnet werden. Da die Abfrage der Werte vom Steuergerät auch ihre Zeit benötigt, wird das Diagnosebild kurzzeitig weiß, bevor es neu dargestellt wird. Da die Erneuerungsrate für eine genaue Diagnosetätigkeit sehr hoch eingestellt ist (Millisekundenbereich) flackert die Diagnosemaske permanent. Für den Benutzer, der diese während der kompletten Diagnosetätigkeit im Auge behalten muss, ist dies sehr anstrengend. Ein Bild, in dem nur die geänderten Werte erneuert würden, würde das Betrachten der Maske angenehmer gestalten.

7.2.3.3 Bereich Mobilität

Die Verbesserung der Prozesse im Bereich Mobilität war Gegenstand von Kapitel 7.2.2.2. Sie schafft neben der Verkürzung der Prozesszeit für den Nacharbeiter auch eine größere Arbeitszufriedenheit, wenn die Arbeit am Steuergerät nicht permanent durch den Weg zum Diagnoseterminal unterbrochen werden muss.

Ergänzt werden kann die Verbesserung der Prozesse noch durch die Auflösung der Notwendigkeit der Tastatur. Das bisherige System INAP muss über eine Tastatur gesteuert werden. Es müssen z. B. Werte für Regler noch über die Eingabe von Zahlen bestimmt werden. Die physische Tastatur beeinträchtigt die Mobilität, weshalb deren Verwendung nicht in Frage kommt. Eine Soft-Tastatur, die von mobilen Betriebssystemen auf der Anzeigefläche abgebildet wird, ist für Nacharbeiter mit Schutzhandschuhen schwer zu bedienen. Deshalb sollen Werte über entsprechend große Regler auf dem Touchscreen einstellbar sein.

7.3 Anforderungsdefinition zur softwaretechnischen Umsetzung der Verbesserungen

Um Anforderungen in der IT zu definieren, stehen inzwischen viele Techniken zur Verfügung. Es haben sich hierfür Requirements Engineering, Change Management, Pflichtenheft oder das Prototyping als Möglichkeiten entwickelt, um den vielfältigen Aufgaben gerecht zu werden[175].

Aber unabhängig davon, mit welcher Technik Anforderungen erschlossen werden – die aus einem Projekt gewonnene Software kann nur so gut sein, wie die Anforderungen formuliert sind[176]. Dabei ist es wichtig, dass man sich bei deren Formulierung auf das Wesentliche konzentriert und sich nicht in Nebensächlichkeiten verliert. Was aber neben der Ausformulierung genauso wichtig ist, ist die Struktur der Anforderungen.

Nach Rupp gibt es eine Vielzahl unterschiedlicher Klassifikationen, um Anforderungen zu gliedern[177]. Eine der meistverbreiteten Gliederungsaspekte sind rechtliche Verbindlichkeit, Priorität, Art und Detailstufe. Je nachdem, welche und wie viele unterschiedliche Lesergruppen (Entwickler, Projektleiter, Manager etc.) sich mit den Anforderungen beschäftigen, bieten sich die einzelnen Gliederungsarten als besser oder schlechter an[178].

Im vorliegenden Fall soll die Anforderungsdefinition der Entwicklung bedarfsgerechter Software dienen. Basierend auf den Anforderungen soll zunächst eine Marktanalyse durchgeführt werden. Bei negativem Untersuchungsergebnis wird der Anforderungskatalog weiter als Basis eines Pflichtenhefts für die Neuentwicklung einer geeigneten Software verwendet.

Die Zielgruppe, an die sich der Anforderungskatalog richtet, sind also vornehmlich Entwickler und Beschaffer von Software. Aus diesem Grunde erschien eine Gliederung nach der Art geeignet zu sein. Es können hiermit die Anforderungen grob nach den Architekturschichten einer Software gegliedert werden. Dies ist für eine spätere Implementierung von Vorteil und erleichtert auch, die auf dem Markt befindliche Software nach ihren Fähigkeiten zu klassifizieren[179]. Aus der Softwarearchitektur ergibt sich folgende Anforderungsklassifikation: Schnittstellen, technische und funktionale Anforderungen.

Das Kapitel listet Anforderungen auf, die für eine HMI-Software in der Kfz-Diagnosetechnik beim Kooperationspartner elementar sind (Kapitel 2.2) und die zur Umsetzung der Verbesserungsansätze dienen (Kapitel 7.2).

7.3.1 Schnittstellen-Anforderungen

Innerhalb dieses Anforderungsbereichs werden die notwendigen Softwareschnittstellen erörtert. Softwareschnittstellen stellen die logischen Berührungspunkte zwischen autarken Softwaresystemen dar. Über die Berührungspunkte werden auf definierte Weise Informationen, z. B. Steuergerätedaten, oder Kommandos, z. B. Steuergerätebefehle, ausgetauscht.

[175] Vgl. Versteegen 2001, S. 6 19
[176] Vgl. Versteegen 2004, S. 39
[177] Vgl. Rupp 2002, S. 159
[178] Vgl. Rupp 2002, S. 160
[179] Vgl. Rupp 2002, S. 171

7.3.1.1 Unterstützung von proprietären Basissystemen

Wie in Kapitel 2.2 dargestellt, erfolgt die Kommunikation der HMI-Software mit dem Fahrzeug über das Basissystem. Im vorliegenden Fall ist dieses Basissystem eine proprietäre Entwicklung (siehe Kapitel 3.3). Um diese Kommunikationsschnittstelle nützen zu können, muss der eigens dafür entwickelte Treiber in die HMI-Software integriert werden können. Dies stellt sich als elementare Anforderung an die Diagnosesoftware, um überhaupt im Rahmen dieser Arbeit eingesetzt werden zu können.

Um das Diagnosegerät mit dem Fahrzeug zu verbinden, ist eine Hardwareschnittstelle notwendig[180]. Die Diagnosesoftware muss die Hardware unterstützen. Da die Hardwareunterstützung durch das Basissystem geregelt wird (siehe Kapitel 2.2), muss diesbezüglich keine weitere Anforderung an die HMI-Software gestellt werden.

7.3.1.2 Import der Steuergeräteinformationen in die Entwicklungsumgebung

Um die Systembrüche in den Verwaltungsprozessen aufzulösen, ist der Aufruf der Steuergeräteinformationen innerhalb der Entwicklungsumgebung ein entscheidender Verbesserungsansatz. In Kapitel 7.2.1 wurde als Verbesserungsansatz ein Editor beschrieben, in dem die Daten für den Zugriff auf das Steuergerät direkt ausgewählt werden können. Für den Zugriff muss der Name des Steuergerätebefehls, eventuell der Aufrufparameter und die Ergebnisbeschreibung angegeben werden. Die genauen technischen Hintergründe wurden bereits bei der Beschreibung des Basissystems in Kapitel 3.2.4 erörtert.

Die notwendigen Steuergeräteinformationen sollen auf der Editoroberfläche in einer Form dargestellt werden, sodass der Benutzer die Zugriffsdaten unmittelbar in die Diagnosemaske einbinden kann, ohne dass zusätzliche Programme notwendig werden. Dadurch würden die Systembrüche in den Prozessen "Job bearbeiten" und "Job hinzufügen" aufgelöst.

7.3.2 Technische Anforderungen

Diese Anforderungsart beschreibt technische Grundvoraussetzungen. Die technischen Anforderungen werden oft auch als nicht funktionale Anforderungen deklariert[181]. Sie stellen die Basis für die funktionellen Forderungen dar. Sie werden deshalb in ihrer Wichtigkeit teilweise über die funktionellen Ansprüche gestellt[182].

Die technischen Anforderungen werden deklariert, wenn die Software z. B. in einem speziellen technischen Umfeld einsatzfähig sein muss[183]. Für die HMI-Software in dieser Arbeit wäre das beispielsweise der mobile Einsatz.

7.3.2.1 Betriebssystemübergreifende Lauffähigkeit der HMI-Software (Mobilität)

Aus technischer Sicht muss die HMI-Software auf zwei verschiedenen Typen von Endgeräten lauffähig sein, auf Diagnoseterminals (gewöhnlichen Arbeitsplatzrechnern) und mobilen Handgeräten (PDAs).

[180] Beispiel für eine Verbindungshardware ist ein OBD-Diagnosestecker, siehe Kapitel 2.2
[181] Vgl. Rupp 2002, S. 172
[182] Vgl. Rupp 2002, S. 264
[183] Vgl. Rupp 2002, S. 173

Da bei Arbeitsplatzrechnern beim Kooperationspartner sowie bei vielen anderen Automobilherstellern ausschließlich Microsoft Betriebssysteme eingesetzt werden, muss die Desktop-Version der HMI-Software eine notwendige Kompatibilität zu XP kompatibel sein. Aus Kostengründen für die Verwaltung der HMI-Software soll nach Möglichkeit nur eine Anwendung nötig sein, die auf mobilen und Desktop-Betriebssystemen lauffähig ist. Zumindest wird aus Gründen der Benutzerfreundlichkeit eine HMI-Software benötigt, die sich in Aufbau und Bedienung von mobil zu Desktop kaum unterscheidet. Damit sollen zusätzliche Aufwendungen bei der Handhabung, Einarbeitung und Schulung gering gehalten werden.

Durch die Unterstützung der Lauffähigkeit auf einem mobilen und auf einem Desktop-Betriebssystem kann eine entsprechende Verkürzung der Prozesszeit bei der Diagnosedurchführung in der Nacharbeit erwirkt werden (siehe Kapitel 7.2.2.2).

7.3.2.2 Stand-alone-Lauffähigkeit der HMI-Software

Neben der Unterstützung eines mobilen Betriebssystems wird in Bezug auf die Lauffähigkeit noch eine andere technische Anforderung gestellt. Die HMI-Software soll im Stand-alone-Betrieb bedienbar sein. Dieser Anspruch leitet sicht nicht aus der Optimierung ab, sondern ergibt sich als Notwendigkeit aufgrund des Einsatzes in der Automobilindustrie. Er wird von dem derzeitigen System bereits erfüllt.

Der Begriff stand-alone (allein stehend, allein operierend) wird in der IT weitläufig in den Bereichen Hardware und Software verwendet. Dementsprechend unspezifiziert ist die eigentliche Bedeutung des Begriffs und sie bestimmt sich mehr aus dem Kontext heraus, in dem der Begriff verwendet wird. Im ursprünglichen Sinn sollte die Eigenschaft stand-alone trennen zwischen den Programmen, die vom Benutzer direkt (unabhängig von anderen Programmen) angestoßen werden können, und denen, die in andere Programme integriert sind. Im Applikationsbereich bedeutet Stand-alone-Betrieb, dass eine Anwendung eigenständig, nur mit lokalen Ressourcen lauffähig ist, ohne Anbindung an ein anderes Gerät oder Netzwerk. Dies kommt sinngemäß der Verwendung in diesem Anforderungskatalog sehr nahe.

Mit dem Punkt Stand-alone-Betrieb wird die Forderung aufgestellt, dass die HMI-Software ohne Verbindung zu Servern bzw. irgendeiner anderen Netzwerkkomponente diagnosefähig ist. Die HMI-Software muss die komplette Funktionsfähigkeit, die für die Diagnose benötigt wird, integriert haben. Demnach müssen die Zugriffsinformationen auf das Steuergerät und die Treiber für die Schnittstellenunterstützung (siehe Kapitel 7.3.2.1) lokal auf dem Diagnosegerät installiert sein.

7.3.2.3 Ausgabe in hoher Bildqualität

Im Rahmen der Analyse wurde die Möglichkeit gefordert, Bedienelemente realitätsgetreu abzubilden. In diesem Bereich wurde in erster Linie kritisiert, dass nur fünf Farben mit einer festen Hintergrundfarbe genutzt werden können.

Auf weitere grafische Notwendigkeiten, um die Diagnosemasken realistisch zu gestalten, wird in den funktionalen Anforderungskapiteln 7.3.3 eingegangen. Die hier definierte technische Forderung nach höherer Farbtiefe und Auflösung soll dafür die Voraussetzung bieten.

Das menschliche Auge kann etwa 400.000 verschiedene Farben wahrnehmen. Diese Zahl ergibt sich aus der wahrnehmbaren Anzahl an Farbtönen (ca. 130) und an Farbsättigungen (ebenfalls ca. 130) und aus den wahrnehmbaren Helligkeitswerten (zwischen 15 und 25).

Bei der Darstellung mit 16 Bit High Color (65.536 Farben) kann es deshalb zu sichtbaren Abstufungen für den Betrachter kommen.

True Color mit einer Farbtiefe von 24 Bit, also mit 2^{24} = 16.777.216 Farben, ist hingegen wirklichkeitsgetreu. True Color reicht aus, um eine für das menschliche Auge realistische Farbdarstellung zu ermöglichen. Es basiert auf den drei Grundfarben (Primärfarben) Rot, Grün und Blau, mit jeweils 256 Farbtönen (Helligkeitsstufen 0 bis 255) pro Grundfarbe. Kombiniert man alle Farbtöne der drei Grundfarben miteinander, dann ergibt dies 256 × 256 × 256 ~ 16,8 Millionen Farben. Im Softwarebereich werden diese Farben mittels des RGB-Modells unterstützt, in dem für (R)ot, (G)rün und (B)lau jeweils eine der 256 Farben angegeben werden kann.

Für die HMI-Software und für die Entwicklungsumgebung ergibt sich somit die Anforderung das RGB-Modell zu unterstützen. Obwohl die meisten mobilen Endgeräte diese Farbtiefe noch nicht unterstützen (bei PDA derzeit High Color üblich), sollte die Software aus Sicht eines langjährigen Einsatzes diese Möglichkeit nicht beschränken.

Gleiches gilt für die Auflösung. Das Grafikformat sollte so gewählt werden, dass die Hardware die Auflösung nicht beschränkt. Einige Entwicklungsumgebungen speichern Grafiken in der festen Auflösung, mit der sie am Monitor dargestellt werden. Hier wäre eine Speicherung der Grafik in relativer Größe von Vorteil. Diese Forderung wird im Rahmen der Skalierbarkeit der Oberfläche noch weiter spezifiziert.

7.3.2.4 Unterstützung von Mehrsprachigkeit

Da jeder deutsche Automobilhersteller, so auch der Kooperationspartner, weltweit über Produktionsstätten verfügt (siehe Kapitel 1.1) ist es notwendig, für die Darstellung und den Entwurf der Masken Mehrsprachigkeit zu unterstützen.

Um Diagnosemasken werksübergreifend einsetzen zu können, ist standardmäßig zumindest eine Übersetzung in Englisch und in die jeweilige Landessprache notwendig. Mit der Unterstützung verschiedener Übersetzungen ist folglich auch deren Schriftsatz erforderlich. Durch Einsatz im asiatischen oder osteuropäischen Raum werden hierfür zusätzliche Zeichensätze benötigt. Deshalb wird eine Unterstützung des Unicodes als notwendig erachtet.

Unicode ist ein internationaler Standard für Zeichen und Schriftkulturen. Unicode strebt eine möglichst vollständige weltweite Erfassung aller bekannten Zeichen aus gegenwärtigen und vergangenen Schriftkulturen an. In der Version Unicode 4.0 sind bereits 96.382 verschiedene Zeichen erfasst. Damit sind bereits fast alle noch gesprochenen Sprachen erfasst[184].

Für die Verwendung der Mehrsprachigkeit wird eine Umschaltung zur Laufzeit benötigt. Die Unterstützung darf nicht dazu führen, dass mehrere Versionen der gleichen Diagnosemasken erstellt werden müssen. Dies wird bei den funktionalen Anforderungen an die Entwicklungsumgebung in Kapitel 7.3.3 noch näher beschrieben.

7.3.2.5 Verwendung von Standards

Die abschließende technische Anforderung des Katalogs ist auch Teil einer Tendenz in der gesamten Industrie. Mit der Einführung von softwaregestützter Fertigung vor gut 30 Jahren entstanden sehr viele Insellösungen durch firmeneigene (proprietäre) Systeme[185]. Mit Zunahme

[184] Vgl. Microsoft 2003, S. 43
[185] Vgl. Schneider 2004, S. 688

der Rechnerleistung und Vernetzungsmöglichkeiten sollen die Einzellösungen nun in einem System zusammengeführt werden. Die Qualität von Auswertungen soll damit verbessert und die Verwaltung erleichtert werden. Dies ist durch viele verschiedene firmenspezifische Datenformate oder in sich geschlossene Software schwer und teilweise überhaupt nicht möglich. Dieser Umstand verursacht bei vielen Unternehmen einen hohen Kostenfaktor, wenn versucht wird, die Einzellösungen zusammenzuführen. Deshalb gewinnt die Forderung nach offenen und internationalen Standards bei Neueinführungen immer mehr an Gewicht.

Kann z. B. der Editor in ein standardisiertes Entwicklungsframework integriert werden, das beim Fahrzeughersteller bereits eingesetzt wird, so fällt zum einen die softwaretechnische Verwaltung leichter. Zum anderen werden sich die Programmierer schneller mit der Funktionalität des Editors vertraut machen, da sie die Entwicklungsumgebung an sich bereits kennen.

Ferner sollen vor allem die Diagnosemasken in einem gängigen Grafikformat gespeichert werden. Dies trägt zu deren Wiederverwendbarkeit und möglicher Nutzung in anderen Systemen bei. Die Verwendung verbreiteter Standards z. B. bei dem Format der Diagnosemasken würde abermals die Einarbeitungszeit der Programmierer in das neue System verkürzen. Ist der Quellcode nämlich in einer für den Programmierer bekannten Sprache (Format) gespeichert, entfällt das Erlernen einer proprietären Sprache. Diese würde auch die Zufriedenheit der Programmierer steigern (siehe Kapitel 7.2.3.1)

7.3.3 Funktionale Anforderungen an die HMI-Software

Funktionale Anforderungen beschreiben Aktionen, die von der Software ausgeführt werden, und Interaktionen, die dem Anwender ermöglicht werden[186]. Sie bilden den fachlichen Funktionsumfang, der zur Laufzeit zur Verfügung gestellt wird.

In diesem Kapitel werden die funktionalen Anforderungen beschrieben, die an die HMI-Software gestellt werden. Die HMI-Software dient zum Darstellen der Diagnosemasken und somit zur Durchführung der Diagnosetätigkeit am Fahrzeug.

7.3.3.1 Interaktion mit Benutzer

Um den Informationsfluss zwischen Mensch und Maschine anzustoßen und zu erhalten, muss die HMI-Software Dynamiken und Interaktion unterstützen. Das Interagieren mit dem Benutzer ist deshalb elementare Funktionalität jeder HMI-Software.

Diese Anforderung wird deshalb von dem derzeit eingesetzten System bereits unterstützt. In Kapitel 2.2 wurde im Zusammenhang mit den Erläuterungen der Grundlagen die technischen Hintergründe der Dynamik und Interaktion bereits besprochen. Sie seien an dieser Stelle nur aufgrund ihrer essenziellen Bedeutung kurz angeführt.

Die Interaktion beschreibt im Allgemeinen "die wechselseitige Beeinflussung zweier, voneinander weitgehend unabhängiger […] Funktionseinheiten"[187].

Dies bedeutet, dass z. B. ein Diagnostiker, der am Fahrzeug das rechte Fernlicht testen will, den zugehörigen Knopf auf der Diagnosemaske betätigt. Die Diagnosesoftware reagiert nun und gibt den Befehl an das Fahrzeug weiter. Das Licht wird aktiviert – der Zustand des Lichts

[186] Vgl. Rupp 2002, S. 173
[187] Charwat 1992, S. 227

(am Fahrzeug) wurde beeinflusst. Das Steuergerät hingegen gibt das Ergebnis der Aktivierung an die Diagnosesoftware zurück. Dadurch beeinflusst das Steuergerät rückwirkend die Werte der Diagnosesoftware (Wechselwirkung zwischen Diagnosesoftware und Steuergerät).

Die Dynamik steht in der Diagnosesoftware nun dafür, dass der geänderte Wert (Status des Lichts) auch auf der Diagnoseoberfläche in der Diagnosemaske angezeigt wird. Dynamische oder aktive Oberflächen sind folglich Darstellungen, die "ihre Gestalt ändern können"[188], in der Diagnosetechnik z. B. entsprechend dem An-/Aus-Status des Lichts am Fahrzeug.

Nur durch die Interaktion mit dem Benutzer und dynamisierten Oberflächen kann mit der HMI-Software sinnvoll eine Diagnose durchgeführt werden.

7.3.3.2 Diagnoseoberfläche unterstützt Maskensystem

Ein weiterer konzeptioneller Kernpunkt für den Einsatz der HMI-Software in der Automobilindustrie ist die flexible Gestaltungsmöglichkeit der Oberfläche.

In der Nacharbeit muss die Diagnose am kompletten Fahrzeug durchgeführt werden, das bedeutet, dass Diagnoseoberflächen für die verschiedensten Steuergeräte zur Verfügung gestellt werden müssen.

Hierfür empfiehlt sich das Konzept von Diagnosemasken. Die HMI-Software bietet den Rahmen, um einzelne Masken zu laden. Die Masken können individuell für jedes Steuergerät erstellt werden. Das Erstellen erfolgt in der Entwicklungsumgebung (siehe Kapitel 7.3.4).

Es erscheint aufgrund der steigenden Zahl an Steuergeräten nicht möglich, für jeden Typ oder Funktionsbereich eine eigene Diagnosesoftware zu verwenden. Einarbeitungszeit der Nacharbeiter und Verwaltungskosten würden mit jedem Steuergerät enorm steigen.

7.3.3.3 Freie Skalierbarkeit der Oberfläche

Eine Anforderung, die durch das aktuelle System noch nicht unterstützt wird, ist die Skalierfähigkeit der Masken.

Wie in Kapitel 7.3.2 bereits erläutert, sollen die Masken über die HMI-Software gleichzeitig auf mobilen und Desktop-Betriebssystemen genutzt werden (siehe Kapitel 7.3.2.1). Allerdings nur wenn die Diagnosemasken frei von Qualitätsverlust auf beliebige Größe skalierbar sind, kann man sie in gleicher Form auf handflächengroßen Displays und Monitoren darstellen. Dadurch wird verhindert, sie für den mobilen und den Desktop-Einsatz zusätzlich bearbeiten und mehrere Versionen der Maske erstellen zu müssen.

Die Handhabung der Quelldatei als Einzelstück (Unikat) wird in Kapitel 7.3.4.2 als Anforderung festgelegt. Dies ist notwendig, um den Aufwand für die Verwaltung der Masken (Speicherung) nicht durch die Anforderung der Mobilität zu verdoppeln. Aufgrund der hohen Zahl an Masken bedingt durch Baureihe, Steuergeräte etc. wäre das nicht vertretbar.

Aus softwaretechnischer Sicht bedeutet das Skalieren von Masken, die dargestellten Objekte beliebig zu vergrößern bzw. verkleinern zu können. Die verlustfreie Skalierung bedeutet, dass das Objekt bei diesem Vorgang nicht an Darstellungsqualität verliert. Dadurch kann ein grafisches Objekt einer Maske in einer beliebigen Größe dargestellt werden und bleibt so für den Benutzer, egal auf welchem Display, klar erkennbar. Der Benutzer kann z. B. Anzeigewerte

[188] Langmann 1994, S. 110

oder Regler verlustfrei auf eine gewünschte Größe bringen, um sie so auch auf einem kleinen Display noch deutlich zu erkennen.

7.3.4 Funktionale Anforderungen an die Entwicklungsumgebung

Die Entwicklungsumgebung (Editor) dient zur Erstellung der Diagnosemasken, die mittels der HMI-Software dargestellt werden. Im Folgenden wird beschrieben, welchen funktionellen Umfang die Entwicklungsumgebung dem Benutzer bieten muss.

7.3.4.1 Kompilierung nicht erforderlich und WYSIWYG-Unterstützung

In der Prozessanalyse wurde die aufwendige Art kritisiert mit der die grafischen Positionierungen durchgeführt werden müssen. Auf den Vorgang des Kompilierens soll verzichtet und der Quellcode der Maske zur Laufzeit direkt interpretiert werden.

Das Kompilieren bedeutet, dass das auszuführende Programm in einer Quellsprache vorliegt und für die spätere Ausführung in ein semantisches Äquivalent in einer Zielsprache umgewandelt wird. Üblicherweise handelt es sich dabei um die Übersetzung einer Programmiersprache (lesbar) in eine Maschinensprache (nicht lesbarer Bytecode). Das Übersetzen wird als Kompilierung bezeichnet. Daraus ergibt sich für den Prozess der grafischen Gestaltung der Nachteil, dass alles, was programmiert wurde, zunächst kompiliert und gestartet werden muss, bevor die grafische Gestaltung tatsächlich überprüft werden kann.

Abhilfe würde hier eine WYSIWYG-Entwicklungsumgebung schaffen. Bei echtem WYSIWYG (Akronym für "What You See Is What You Get") wird ein Dokument während der Bearbeitung am Bildschirm genauso angezeigt, wie es bei der Ausgabe am Endgerät aussieht[189]. Die Maske soll also während des Entwickelns in der Entwicklungsumgebung schon so vorliegen, wie sie später in der HMI-Software ausgeführt werden soll. Modelle, die ein Kompilieren oder eine andere Form der Nachbearbeitung der Maske erfordern, verfolgen somit nicht das WYSIWYG-Konzept.

Die Anforderung geht folglich dahin, dass eine reine Interpretation der Masken zur Laufzeit erfolgt. Die HMI-Software arbeitet demnach als Interpreter der Maske, um diese zu starten.

Ein Interpreter ist eine Software, die einen Programm-Quellcode (Maske) im Gegensatz zu Compilern nicht in eine auf dem System direkt ausführbare Datei umwandelt, sondern den Quellcode einliest, analysiert und direkt ausführt. Die Analyse des Quellcodes erfolgt also zur Laufzeit des Programms. Diese Interpretation kann auch die Entwicklungsumgebung während der grafischen Erstellung der Masken durchführen. Damit wird das WYSIWYG-Konzept verfolgt. Der Programmierer spart sich die zusätzlichen Schritte des Kompilierens (Senkung der Prozesszeit) und die dafür notwendigen Programme (Auflösung des Anwendungssystembruchs).

Der Nachteil der Interpretersprachen ist die im Vergleich zu kompilierten Programmen deutlich langsamere Ausführungsgeschwindigkeit. Reine Interpreter lesen und analysieren den Quellcode eines Skripts und führen dann die entsprechenden Aktionen aus. Dies ist im Vergleich zu Compilersprachen, bei denen das Programm vor seiner Ausführung in Maschinenco-

[189] Vgl. Charwat 1992, S. 468

de übersetzt wird, sehr zeitaufwendig. Da es sich bei der Diagnose nicht um umfangreiche Algorithmen, Berechnungen und das Abarbeiten von großen Datenmengen handelt, sondern lediglich Steuergerätebefehle ausgeführt werden, kann dieser Nachteil durch derzeitig üblich 400 MHz PDA-Prozessoren bereits kompensiert werden (siehe Kapitel 11).

7.3.4.2 Redundanzfreies Erstellen der Masken ("Unikat")

Aus den technischen Anforderungen, wie Mehrsprachigkeit und betriebssystemübergreifende Lauffähigkeit, ergibt sich bei vielen Systemen das Problem einer notwendigen Mehrfachspeicherung der Masken.

Die Mehrsprachigkeit wird bei diesen Systemen nicht dynamisch zur Laufzeit umgesetzt, sondern die Masken müssen beim Erstellen separat in der jeweiligen Sprache gespeichert werden.

Auf ähnliche Weise ist die Kompatibilität zu verschiedenen Betriebssystemen gelöst. Die Diagnosemaske muss für jedes Betriebssystem speziell angepasst werden und folglich auch in mehreren Varianten gespeichert werden.

Dies kann dazu führen, dass eine Diagnosemaske in zwei mal zwei Sprachen (z. B. Englisch und Deutsch für Windows XP und Windows Mobile 2003), also vier Varianten, gehalten werden muss. Die zusätzlich daraus entstehenden Kosten und der Verwaltungsaufwand sollen bei einem neuen System vermieden werden. Es wird daher eine nicht redundante (überflüssig vorhanden sein) Speicherung der Masken gefordert. Der Begriff beschreibt in der Informatik, dass Daten oder Informationen nur einmal gehalten bzw. gespeichert werden. Die Masken werden demzufolge, egal ob für deutsche oder fremdsprachige Anzeige und für mobile und Desktop-Betriebssysteme, nur einmal erstellt und verwaltet. Die Maske ist somit eindeutig bzw. ein Unikat.

7.3.4.3 Grafisches Erstellen der Masken

Das Erstellen der Diagnosemasken mittels der Entwicklungsumgebung soll grafisch gestützt sein. An sich wird in Kapitel 7.3.4.1 diese Anforderung bereits implizit genannt. Dort wird beschrieben, dass die Masken während des Erstellens bereits das Aussehen und die Funktionalität haben sollen, wie sie sie beim späteren Einsatz mittels HMI-Software haben. Die Entwicklungsumgebung muss eine grafische Ansicht bieten, in der die Diagnosemasken dargestellt werden.

Anhand dieser Ansicht in der Editoroberfläche soll die Maske erstellt werden. Die Forderung besteht in einer typischen Bedienweise ähnlich wie in gängigen Zeichenprogrammen. Auf den Umfang wird im folgenden Kapitel 7.3.4.4 noch näher eingegangen.

Neben der grafischen Art des Erstellens muss für komplexe Diagnosemasken, in denen z. B. Steuergerätedaten noch aufbereitet werden müssen, bevor sie angezeigt werden können, auch eine Eingabemöglichkeit für Quellcode bestehen.

Optimalerweise generiert der grafische Editor Quellcode, der vom Programmierer noch nachbearbeitet werden kann. Auf diese Weise kann der Programmierer die Auswirkung seiner Änderung direkt auf der grafischen Ansicht verfolgen.

Die grafische Gestaltungsmöglichkeit würde an vielen Stellen in den bestehenden Prozessen eine Verkürzung der Prozesszeiten zur Folge haben (siehe Kapitel 7.2.2.1), sowie eine erhebli-

che Steigerung der Anwenderzufriedenheit beim Erstellen der Diagnosemasken in der Verwaltung (siehe Kapitel 7.3.2.1).

7.3.4.4 Umfassende grafische Gestaltungsfunktionalität

Durch den grafischen Editor ergibt sich ein großes Potenzial für Verbesserungen. Eines davon ist die Zuweisung der Steuergerätebefehle zu einem grafischen Objekt. In Kapitel 7.3.1.2 wurde die Notwendigkeit erläutert, die Steuergerätedaten direkt in den Editor zu integrieren (Import). Mittels der weiteren Forderungen nach einem grafischen Editor können diese beiden Forderungen in einer weiteren Forderung zusammengefasst werden: die Entwicklungsumgebung soll die Möglichkeit bieten, die Zugriffsdaten auf das Steuergerät direkt einem funktionalen Objekt, wie z. B. Button (Knopf), zuzuweisen. Durch das Drücken des Knopfs wird der Befehl an das Steuergerät geschickt.

Ein weiterer Ansatzpunkt ist die realitätsgetreue Darstellung der Masken zu verbessern (Kapitel 7.2.3.2). Dafür muss es möglich gemacht werden, statische oder dynamische Hintergrundbilder einzubinden. Dadurch kann dem Benutzer, z. B. durch das Abbild einer Armatur, auf einen Blick gezeigt werden, welche Werte dargestellt werden.

Um komplexere Grafiken abzubilden, wie z. B. Schaltkreise elektronischer Komponenten, muss eine Art Freihandzeichnen ermöglicht werden. Der Umfang soll einem handelsüblichen Editor entsprechen. Es sollen Linien, Rechtecke, Kreise etc. dargestellt werden können, die mit Linien- und Füllfarben versehen werden können (siehe Kapitel 7.3.2.3).

Um eigenerstellte Formen in anderen Masken wieder verwenden zu können, soll die Möglichkeit von Vorlagen bzw. Bibliotheken geboten werden. Diese Bibliotheken sollen beliebig durch den Benutzer erweiterbar sein. Dadurch kann das Erstellen von Masken, falls Abbildungen, wie z. B. LED, Regler, Diagramme etc., häufiger genutzt werden, beschleunigt werden.

7.3.4.5 Unterstützung von Upload/Download

Diese Forderung kommt dem Wunsch nach mehr Benutzerfreundlichkeit vonseiten der Anwender nach. Um Masken aus der Verwaltung an die Einsatzorte, z. B. die Nacharbeit, zu verteilen, nutzt der Kooperationspartner Referenzserver (siehe Kapitel 5.8.1). Den dort abgelegten Versionsstand muss der Programmierer zur Bearbeitung nutzen. Im gleichen Zuge muss der Diagnostiker (Nacharbeiter, Analyst etc.) sicherstellen, dass er bei seiner Arbeit den aktuellsten Stand nützt.

Eine Upload/Download-Funktion innerhalb der Entwicklungsumgebung würde es erleichtern, immer die aktuelle Diagnoseskriptquelle zur Bearbeitung vorliegen zu haben. Sowie nach der Bearbeitung die Quelle und das kompilierte Skript auf den Server spielen zu können.

Falls sich diese Möglichkeit bietet, wäre eine ähnliche Downloadfunktion zur Aktualisierung des kompilierten Diagnoseskripts innerhalb der HMI-Software auch sinnvoll und würde die Benutzerfreundlichkeit erhöhen.

7.3.5 Zusammenfassung der Anforderungen

Die erarbeiteten Anforderungen gelten als Grundlage für die Analyse des Softwareangebots (Kapitel 8). Sie werden deshalb abschließend zusammengefasst dargestellt. Auch um das Ergebnis der Marktanalyse darzulegen, wird diese Auflistung genutzt. Es werden für die jeweils

auf dem Markt befindlichen Produkte die erfüllten bzw. nicht erfüllten Anforderungen genannt (siehe Kapitel 8.4).

- **Schnittstellen-Anforderungen**
  - Unterstützung von proprietären Basissystemen
  - Import der Steuergeräteinformationen in die Entwicklungsumgebung
- **Technische Anforderungen**
  - Betriebssystemübergreifende Lauffähigkeit der HMI-Software (Mobilität)
  - Stand-alone-Lauffähigkeit der HMI-Software
  - Ausgabe in hoher Bildqualität
  - Unterstützung von Mehrsprachigkeit
  - Verwendung von Standards
- **Funktionale Anforderungen an die HMI-Software**
  - Interaktion mit Benutzer
  - Diagnoseoberfläche unterstützt Maskensystem
  - Freie Skalierbarkeit der Oberfläche
- **Funktionale Anforderungen an die Entwicklungsumgebung**
  - Kompilierung nicht erforderlich und WYSIWYG-Unterstützung
  - Redundanzfreies Erstellen der Masken ("Unikat")
  - Grafisches Erstellen der Masken
  - Umfassende grafische Gestaltungsfunktionalität
  - Unterstützung von Upload/Download

8 Analyse des Marktangebots in Bezug auf die definierten Anforderungen

Der Anforderungskatalog aus Kapitel 7 stellt einerseits sicher, dass das gesuchte Diagnosesystem die elementaren Voraussetzungen für den Einsatz beim Kooperationspartner erfüllt. Darüber hinaus definiert sind im Katalog die Anforderungen definiert, die erfüllt sein müssen, um die ermittelten Verbesserungsansätze aus der Soll-Modellierung zu realisieren.

In Kapitel 8 wird nun überprüft, ob der Anspruch des Anforderungskatalogs durch das Angebot auf dem Markt gedeckt werden kann. Im Rahmen dieser Arbeit wird nur eine Kurzanalyse durchgeführt, da der Kooperationspartner bereits seit Jahren Marktangebote für verbesserte Diagnosesoftware einholt. Die Angebote konnten allerdings bisher nicht den Anforderungen des Kooperationspartners gerecht werden.

Die Kapitel 8.1 bis 8.3 dienen dazu, die auf dem Markt befindliche und für diese Arbeit relevante Diagnosesoftware zu erheben. Um das umfassende Angebot an Diagnosesystemen zu beurteilen, werden zunächst in Kapitel 8.1 die Kriterien festgelegt, die eine Software erfüllen muss, um für eine nähere Betrachtung in Frage zu kommen. Darüber hinaus werden die Quellen definiert, die für die Suche benutzt werden.

Falls Forschungsprojekte und Softwareangebote die in Kapitel 8.1 genannten allgemeinen Kriterien erfüllen, werden sie in Kapitel 8.2 und 8.3 näher betrachtet.

Danach werden die gefundenen Lösungen in Kapitel 8.4 mit den Anforderungen verglichen. Abschließend wird in Kapitel 8.5 das Ergebnis der Marktuntersuchung dargelegt.

8.1 Kriterien für die Analyse

Bereits vor einigen Jahren hat das Marktangebot an Standardsoftware in den meisten Branchen eine Größenordnung erreicht, die nur mit Hilfe aktueller Nachschlagewerke überschaubar

bleibt[190]. Schon zur Jahrtausendwende war die "Dynamik des Softwaremarkts [...] kaum mehr zu überbieten"[191].

Deshalb ist es nahezu unmöglich, eine aktuelle weltweite Liste aller Anbieter von Diagnosesoftware zu erfassen. Selbst wenn dies gelingen würde, könnte sie durch die Schnelllebigkeit des Markts am nächsten Tag schon wieder veraltet sein. Es ist daher notwendig, die Suche zielorientiert durchzuführen und Einschränkungen festzulegen.

Um die auf dem Softwaremarkt befindlichen Angebote zu erfassen, wurde ein dreiteiliges Rechercheverfahren angewendet[192]. In Phase 1 wird der Suchbereich thematisch abgegrenzt. In Phase 2 werden über die gewählten Medien die Informationen beschafft. In der letzten Phase 3 werden dann die Informationen ausgewertet.

Zu Phase 1: Die gröbste Abgrenzung ergibt sich aus der Themenstellung der Arbeit. Es wird ausschließlich Diagnosesoftware für elektronische Kfz-Steuergeräte betrachtet. Die Software muss eine interaktionsfähige und dynamisierte Diagnoseoberfläche bieten. Für die detaillierte Abgrenzung wird der Anforderungskatalog (Kapitel 7.3) herangezogen, durch dessen Ansprüche an Funktionalität, grafische Ausgabe und Mobilität die in Frage kommende Diagnosesoftware weiter eingegrenzt wird.

Neben den technischen Einschränkungen erscheint auch eine Abgrenzung des zu untersuchenden Markts als gerechtfertigt. Nach einem Bericht des VDA im Jahr 2006 stammen 21 % aller weltweit produzierten Fahrzeuge von deutschen Herstellern[193]. Folglich gehört mehr als jedes fünfte Auto weltweit einer in Deutschland beheimateten Konzernmarke an. Für eine aussagekräftige Marktanalyse scheint es deshalb durchaus sinnvoll sich auf den deutschen Softwaremarkt zu beschränken – zumal der Verlag moderne industrie (vmi) allein für diesen Markt über 1.600 Automobilzulieferer verzeichnet.

Zu Phase 2: Der Verweis auf Quellen, wie den Branchenkatalog des VMI, führt zu der zweiten Phase der der Marktuntersuchung. In der Phase werden die Medien bestimmt, mit deren Hilfe die Diagnosesoftware gefunden werden soll. Aus technischer Sicht lassen sich Medien in vier Kategorien aufteilen[194].

Mit dem ersten Typ den primären Medien, wird jede Form der Konversation zweier Menschen ohne technische Hilfsmittel bezeichnet. In persönlichen Gesprächen wurden z. B. auf Messen, wie beispielsweise der SPS-IPC, der Systems und der Hannover Industrie-Messe, Informationen über Neuerungen und derzeitige Angebote eingeholt. Auch im Gespräch mit dem Einkauf des Kooperationspartners wurden Kontakte ermittelt, von denen Softwareangebote eingeholt wurden.

Von den sekundären Medien, den Printmedien, wurden Fachzeitschriften der Automobilbranche und Dienstleister-Kataloge ausgewertet. Beispiele sind der bereits genannte Automobil-Zulieferer-Katalog, der Automatisierungs-Atlas des SPS Magazins und Zeitschriften wie Automobil-Produktion.

Der größte Teil des Suchergebnisses konnte jedoch aus den tertiären und vor allem quartären Medien, dem Internet, gewonnen werden. Tertiäre Medien erfordern sowohl bei der Herstel-

[190] Vgl. Grupp 1999, S. 30
[191] Grupp 1999, S. 30
[192] Vgl. Landwehr 1977, S. 9-16
[193] Vgl. VDA 2006, S. 36]
[194] Vgl. Ludes 2003, S. 64

lung als auch beim Empfang technische Einrichtungen wie Film, Radio und Fernsehen. Wird für die Kommunikation ein Computer benötigt, spricht man von quartären Medien. Die Recherche erfolgte über branchennahe bis hin zu allgemeinen Suchmaschinen. Beispiele hierfür, sind Softguide, VDA, Automobil Produktion und globale Suchmaschinen wie Wikipedia und Google. Aufgrund der beschriebenen Schnelllebigkeit sind Suchmaschinen oft die einzige Möglichkeit, um auf aktuelle Angebote zu stoßen. Des Weiteren wurden Internetseiten von Anbietern besucht, die in Fachzeitschriften beworben wurden.

Das umfangreiche Ergebnis der Suche wurde durch die beiden Kernbereiche des Anforderungskatalogs Schnittstellen und technische Anforderungen an die Software eingegrenzt. Eine Auswahl der für eine intensivere Betrachtung ausgewählten Angebote wird im Anschluss in den Kapiteln 8.2 und 8.3 geliefert. Die Auflistung gliedert sich in Ansätze aus Forschungsprojekten und kommerzielle auf dem Markt angebotene Software, wobei der Schwerpunkt der Analyse auf bereits auf dem Markt verfügbarer Software liegt.

Zu Phase 3: In der letzten Recherchephase werden die gefundenen Softwareprodukte fundiert bewertet. Dies wird in Kapitel 8.4 durchgeführt und das Ergebnis der Auswertung abschließend in Kapitel 8.5 erläutert.

8.2 Ansätze aus Forschungsprojekten

Die Suche nach Forschungsprojekten im Bereich der Prüfung und Diagnose von elektronischen Steuergeräten ergibt ein breites Spektrum an Hochschulen und Universitäten. Eine Vielzahl von Fakultäten, Instituten oder Lehrstühlen setzen sich ganz spezifisch mit dieser Thematik auseinander. So wurde z. B. 2004 an der Universität Stuttgart der in Deutschland erste Lehrstuhl für Kraftfahrzeugmechatronik eingerichtet. Forschungsansätze zur Verbesserung der Diagnose von elektronischen Steuergeräten in der Automobilindustrie wurden an der TU Berlin, Universität Braunschweig, Universität Kassel, TU München, Universität Paderborn, Hochschule Reutlingen und der Universität Stuttgart ausfindig gemacht.

Viele der Forschungsprojekte werden in Kooperation mit Industrieunternehmen durchgeführt und unterliegen Geheimhaltungsauflagen. Das Forschungsprojekt "Entwicklung von Software für die Diagnose von Kfz-Steuergeräten in der Automobilproduktion" der Steinbeis Stiftung und der Hochschule Reutlingen wird z. B. in Zusammenarbeit mit T-Systems durchgeführt. Das Lizenz- und Informationsrecht liegt hier bei der T-Systems, Niederlassung Leinfelden.

Die Forschungsansätze der öffentlichen Projekte setzen an den unterschiedlichsten Stellen in der Diagnose an. Es wird versucht, der Diagnoseproblematik z. B. über Ursachen-Wirkungs-Diagramme oder modellbasierte Verhaltensanalyse zu begegnen. Viele Lösungen haben ihren Verbesserungsansatz auch in speziellen Funktionsbereichen des Fahrzeugs, wie z. B. der Motor-Abgas oder Komfortelektronik. Ein allgemeiner Ansatz für die Diagnose von Steuergeräten, der seinen Schwerpunkt auf den HMI-Bereich bzw. die Diagnoseoberfläche legt, wurde allerdings nicht gefunden.

Die auf Funktionsbereiche spezialisierten Lösungen wurden nicht näher betrachtet. Im Folgenden werden zwei allgemeine Lösungsansätze kurz dargestellt.

8.2.1 Universität Paderborn

An der Universität Paderborn wurde an der Fakultät für Elektrotechnik, Informatik und Mathematik ein Projekt ausfindig gemacht, dass die Diagnose von Steuergeräten mit zum Inhalt hat.

Das Forschungsprojekt wird vom Institut s-lab (Software Quality Lab) durchgeführt, das zunächst als erste wissenschaftliche Einrichtung der Fakultät für Elektrotechnik, Informatik und Mathematik der Universität Paderborn errichtet wurde. Die Geschäftsführung wird durch einen Beirat aus Mitgliedern der Universität und den assoziierten Partnern beraten.

Das relevante Projekt trägt den Namen "Software-Qualitätssicherung in automotiven Steuergeräte-Netzwerken" und wird von Prof. Dr. Hans Kleine-Büning geleitet.

Das Projekt beschäftigt sich mit dem Problem, dass die Vernetzung der Steuergeräte in Automobilen zu einer neuen Komplexitätsstufe in der Entwicklung automotiver Software geführt hat. Vernetzte Steuergeräte stellen neue Anforderungen an die Qualitätssicherung bei der Entwicklung von Software für Steuergeräte. In einem gemeinsamen Projekt mit der dSPACE GmbH entwickelt das s-lab Methoden für die Software-Qualitätssicherung in automotiven Steuergeräte-Netzwerken. Im Rahmen des Projekts wurde ein Beitrag zur Diagnose in verteilten Systemen geleistet.

S-lab stellt einen "neuartigen Diagnoseansatz" vor[195]. Der Ansatz beruht darauf, dass ein Fehler in einem heutigen mechatronischen System aufgrund des hohen Vernetzungsgrads, nicht direkt, sondern nur über seine Auswirkungen (Symptome) beobachtet werden kann. Deshalb ist die Diagnose, also die Lokalisierung der eigentlichen Ursache ("first principle"), in solchen verteilten automotiven Systemen eine große Herausforderung[196].

Der Beitrag behandelt das Erkennen von Symptomen und die Identifizierung von Fehlern. Dabei wird durch den Einsatz der Modellkompilation ein neues Diagnoseverfahren für automotive Systeme konstruiert[197].

Idee des Modellkompilationsansatzes ist es, eine Klassifikationsfunktion zu erstellen, die mittels beobachteter Werte des Systemverhaltens ein Element aus einer Fehlermenge zum Ergebnis hat. Dafür ist Wissen über den Kausalzusammenhang von Symptomen und Fehlern notwendig. Dieses Wissen wird in Verhaltensmodelle (Ursachen-Wirkungs-Modelle) mit hohem Detaillierungsgrad eingepflegt[198].

Voraussetzung für die Anwendung der Modellkompilation ist, dass das zu diagnostizierende System durch ein ausführbares Verhaltensmodell repräsentiert werden kann[199]. Das System ist deshalb nicht zur Diagnose der kompletten Bandbreite an möglichen Elektronikfehlern geeignet. Es hat vielmehr seinen Schwerpunkt in Fehlern, die eine verzweigte Ursachenkette über mehrere Steuergeräte vorweist und die aufgrund ihrer Häufigkeit und Ähnlichkeit über Verhaltensmodelle abgebildet werden können. Durch den Einsatz von Verhaltensmodellen wird auch versucht, den Fehler automatisiert zu erarbeiten. Eine manuelle Fehlersuche durch den Dia-

[195] Balzer 2007, S. 243
[196] Vgl. Balzer 2007, S. 244
[197] Vgl. Balzer 2007, S. 245
[198] Vgl. Balzer 2007, S. 245
[199] Vgl. Balzer 2007, S. 247

gnostiker ist nicht vorgesehen. Deshalb setzt sich der Beitrag auch wenig mit der Problematik der Gestaltung der Benutzeroberfläche auseinander.

8.2.2 TU München

Ein weiteres Projekt, in dem die Diagnose von Steuergeräten behandelt wurde ist INDIA - Intelligente Diagnose in der industriellen Anwendung. Das Projekt wurde an der TU München am Institut für Informatik durchgeführt. Die wissenschaftliche Leitung hatte Prof. Dr. Peter Struss. Auch hier wurde die enge Kooperation mit einem Betrieb aus der Industrie gewählt. Für die Schaffung neuer Diagnosesysteme arbeitet die TU mit der Robert Bosch GmbH zusammen.

Ein Gegenstand des Projekts INDIA ist die intelligente Diagnose bei mechatronischen Systemen. Ziel dieses Teilbereichs ist es, modellbasierte Methoden der Steuergerätediagnose am Kraftfahrzeug so weit voranzutreiben, dass ein praktischer Einsatz bei den Aufgaben in der Werkstatt ohne zusätzliche Grundlagenforschung möglich wird[200].

Über ähnliche Ansätze auf der Grundlage von Verhaltensmodellen, wie im Beispiel der Universität Paderborn, geht das Projekt der TU München noch einen Schritt hinaus und nimmt auch Autorensysteme für Fehlersuchanleitungen und Fehler-Möglichkeiten und Einfluss-Analyse (kurz FMEA) ins Konzept auf.

Für das modellbasierte Werkzeug zur Diagnose und Fehleranalyse von Fahrzeug-Subsystemen erfasst das Projekt einen Einsatzbereich, der sich teilweise mit den Aufgaben des Geschäftsprozesses Nacharbeit (siehe Kapitel 5.6) deckt. Das Projekt beschreibt hier die Online-Diagnose. Darunter wird eine Folge geeigneter Tests und Messungen durch den Diagnostiker verstanden. Damit wird die Fehlerursache allmählich so weit eingegrenzt, dass gewöhnlich durch Ersetzen einer defekten Komponente der Fehler behoben werden kann. Gewonnen, werden die Informationen für die Eingrenzung entweder aus manuell vorgenommenen Messungen, Beobachtungen (etwa Spannungs- und Widerstandsmessungen in der Elektrik), Messungen mittels eines angeschlossenen Testers (hier Diagnose- bzw. HMI-Software) oder interner Fehlerprotokolle des Steuergeräts[201].

Das beschriebene Szenario ähnelt sehr der Ausgangssituation dieser Dissertation. Darüber hinaus werden die Aufgaben der Diagnose ähnlich erfasst. Auch das INDIA-Projekt sieht das Ziel der Diagnose offenkundig darin, die Komponenten zu identifizieren, deren Fehlverhalten die beobachteten störenden Effekte, das heißt die Symptome, verursacht haben könnten. Allerdings führt die Analyse der Aufgabenstellung beim INDIA-Projekt in eine andere Richtung. Im Projekt wird aus dem Ziel, die fehlerhafte Komponente zu finden, der Schluss gezogen, dass wiederum das Wissen über das Verhalten und eventuelle Fehlverhalten der vorhandenen Komponenten grundlegend ist. Ebenso sind es die Kenntnis der Struktur des Subsystems und die Verhaltensvorhersage, die es gestattet, das im Modell implizierte Verhalten mit dem tatsächlich beobachteten Verhalten in Beziehung zu setzen. Es wird weiter vorausgesetzt, dass normalerweise die Fehlerursachen nicht in einem Schritt ermittelt werden kann. Das hat zur Folge, dass die Erzeugung geeigneter Tests für die Gewinnung zusätzlicher Information eine weitere Teilaufgabe darstellt. Deren Natur besteht darin, Aktionen vorzuschlagen, die mit Sicherheit oder zumindest gewisser Wahrscheinlichkeit Unterschiede in den Auswirkungen verschiedener (Fehl-)Verhalten beobachtbar machen. Also erfordert auch dieser Schritt eine Form der Verhal-

[200] Vgl. INDIA 2001, S. 11
[201] Vgl. INDIA 2001, S. 37

tensvorhersage und darüber hinaus Wissen über die zur Realisierung notwendigen Aktionen und den damit verbundenen Zeit- und Kostenaufwand[202].

Aus dieser Analyse wird im Projekt ein Prototyp entwickelt, dem folgende technische Architektur zu Grunde liegt: Elementar für die Lösung ist eine "Bibliothek von Verhaltensmodellen der verwendeten Komponententypen und eine Strukturbeschreibung ("Netzliste") des betrachteten technischen Systems. Aus diesen beiden Elementen wird dessen Verhaltensmodell konstruiert (Systemmodell, "device model"), was durch eine Softwarekomponente, die Modellgenerierung ("Model composition") automatisch geschieht"[203].

Der Ansatz des INDIA-Projekts setzt sich ab diesem Zeitpunkt folglich weniger mit der Testersoftware (also HMI-Software) auseinander, sondern damit, ein Autorensystem, ein Vorhersagemodell und eine Anleitung für die Diagnosetätigkeit zu liefern. Der Ansatz der vorliegenden Dissertation setzt im Prinzip einen Schritt früher an. Während die HMI-Software im INDIA Projekt dazu benutzt wird, um die Rohdaten für die Verhaltensmodelle zu beschaffen und sie deshalb lediglich als Mittel zum Zweck gesehen und nicht eingehender behandelt wird, ist sie in dieser Arbeit, Ansatzpunkt für Verbesserungen.

Im folgenden Kapitel soll nun der Beitrag behandelt werden, den der kommerzielle Softwaremarkt für den Problembereich einer zeitgenössischen Diagnoseoberfläche leistet

8.3 Angebote auf dem Softwaremarkt

Das Vorhaben, die auf dem Markt befindliche Diagnosesoftware zu erheben, dient dem Ziel, eine Software zu finden, die allen gestellten Anforderungen entspricht. Eine detaillierte Beschreibung jedes einzelnen Produkts, dass die geforderte Kernfunktionalität nicht erfüllt, ist aus diesem Grunde nicht lohnenswert.

Deshalb wird in dieser Arbeit nicht jedes gefundene Produkt erläutert, sondern es werden repräsentative Vertreter aufgeführt. Dadurch können Stärken und Schwächen besser herausgestellt werden und die Analyse verliert sich nicht in der Erläuterung diverser Softwareangebote, die im Grunde die gleiche Funktionalität aufweisen.

Die Auswertung der folgenden Vertreter erfolgt in Kapitel 8.4.

8.3.1 Diagnostic Pocket PC Scan Tool

Es finden sich auf dem Markt eine Reihe von Anbietern, die statt des Konzepts von frei erstellbaren Diagnosemasken eine starre Diagnoseoberfläche aufweisen.

Auf dieser vom Hersteller vorgegebenen Oberfläche werden die wichtigsten Statusanzeigen und Befehle zur Diagnose ausgegeben. Diese Art von Diagnosesystem besteht also nur aus der HMI-Software. Eine Entwicklungsumgebung zum Erstellen der Diagnosemasken wird nicht angeboten.

[202] Vgl. INDIA 2001, S. 38
[203] INDIA 2001, S. 39

Die HMI-Software wird auf dem Endgerät, üblicherweise einem Desktop-PC, installiert und ans Fahrzeug angeschlossen. Mit dem Aufkommen leistungsstärkerer PDAs haben einige Anbieter ihre Software auf mobile Betriebssysteme portiert, wie z. B. der Anbieter EASE Simulation Inc. sein Produkt Diagnostic Pocket PC Scan.

Das Produkt ist vorrangig ein Kfz-Diagnosesystem zur Messung von auftretenden Fehlfunktionen bei der Motorsteuerung. Es ist darauf spezialisiert, Auskunft über Abgas- und Emissionswerte und die damit in Verbindung stehenden Betriebsdaten zu geben.

Produkte wie das Pocket PC Scan Tool sind also darauf ausgerichtet Daten aus spezifischen Funktionsbereichen des Fahrzeugs anzuzeigen.

Die Anbieter liefern meist auch verschiedene Softwareversionen für die einzelnen Typen von Endgeräten. Für die Nutzung auf dem PDA und dem Notebook müssen also separate Produkte erworben bzw. installiert werden. Diese Art von Software stellt eine kostengünstige Lösung zur Fahrzeugdiagnose dar (ca. 300 €). Die Zielgruppen sind Privatkunden, die die Software unter anderem zur Kfz-Selbstkontrolle nutzen.

8.3.2 Diagra D Scan Tool

Eine etwas flexiblere Lösung stellt die preislich nächsthöherliegende Gruppe der Diagnosesoftware dar. Der Vertreter Diagra D, der Firma RA Consulting, ist eine Diagnosesoftware, die auf Arbeitsplatzrechnern und auf PDAs lauffähig ist. Der Preis von Diagra D liegt hier bereits bei über 650 € für eine Lizenz.

Die Diagnosesoftware ist bei diesem Produkt ein Teil der 3-teiligen Produktfamilie Diagra D. Diagra D liefert neben der Standard-Diagnosesoftware noch eine Werkstatt-Diagnose-Funktion und erweiterte Entwicklerfunktionen.

Der Bereich Werkstatt-Diagnose ist eine kundenspezifische Anpassung von Diagra D zum Auslesen der Fehlerspeicher und zur Durchführung spezieller Tests. Er ist nicht frei als Standard erhältlich, sondern wird kundenspezifisch angepasst. Auch die erweiterten Entwicklerfunktionen werden nur speziell für geschulte Fahrzeugingenieure angeboten. Sie erfordern fundiertes Wissen im Bereich der Fahrzeugelektronik.

Im Vergleich zu Produkten wie dem Pocket PC Scan Tool bietet Diagra D grafisch weitaus bessere Möglichkeiten. Steuergerätedaten können mittels der Diagnosesoftware ausgelesen werden und in einem speziellen Messtool Diagra M ausgewertet werden. Diagra M bietet die Möglichkeit, die erfassten Werte als Balkendiagramme, Graphen oder in Tabellen auszugeben. Die Auswertungen können vom Nutzer angepasst werden.

Mittels eines Erweiterungsmoduls ist es möglich, mit einem kostenfreien Editor Testabläufe zu generieren und einzubinden. Dieser bezieht sich allerdings nur auf einen beschränkten Funktionsbereich (z. B. Abgaskontrolle) im Fahrzeug. Für die Testabläufe wird keine grafische Gestaltungsmöglichkeit geboten.

Die Oberfläche der Diagnosesoftware ist auch bei diesem Produkt grundsätzlich durch den Hersteller vorgegeben. Dies trifft auch auf die mobile Erweiterung Diagra D Scan-CE zu. Die CE-Erweiterung steht für die Kompatibilität mit dem Microsoft Betriebssystem CE, womit es z. B. auf Tablet PCs lauffähig ist.

Zusammenfassend kann festgehalten werden, dass Diagra D Teil des Diagra MCD Toolkit ist, mit dem die Firma RA Consulting ein umfassendes Diagnosesystem für die Arbeit mit elektronischen Steuergeräten anbietet. Einen besonderen Vorteil sieht der Hersteller in den flexiblen Auswertungsmöglichkeiten. Der Datenaustausch zwischen den Teilbereichen Messung (Diagra M), Kalibrierung (Diagra C) und Diagnose (Diagra D) ermöglicht dies.

8.3.3 eye2m

Auch in der Steuergerätediagnose findet sich die verbreitete Client/Server-Architektur in diversen Lösungen wieder. Deshalb soll an dieser Stelle auch ein Vertreter dieser Technologie vorgestellt werden. In einem Bericht der Zeitschrift Elektronik Industrie, November 2004, beschreibt die Firma Sevenstax, wie sie zusammen mit der IMS GmbH dieses Konzept mit dem Produkt eye2m umgesetzt haben.

Kern bei dieser Architektur ist das eye2m-Modul. Es dient zur Kommunikation mit dem Steuergerät und unterstützt dabei – wie die bereits vorgestellten Diagnosetools – die dafür notwendigen Steuergeräteprotokolle. Der Unterschied zu den vorherigen Konzepten besteht darin, dass die der Steuergeräte entnommenen Informationen nicht direkt auf der Oberfläche der HMI-Software angezeigt werden, sondern zunächst an den eye2m-Server geschickt werden.

Die Übertragung erfolgt dabei per SOAP. SOAP ist ein XML-Format und dient als Datencontainer. Die Steuergerätedaten können vom eye2m-Server über eine WebService-Schnittstelle abgefragt werden.

Die Nutzer fragen die Diagnoseinformationen mit einem gängigen Internetbrowser vom eye2m-Server ab. Hierfür ist in die eye2m-Architektur noch ein Webserver mitaufgenommen. Dieser fragt über den beschriebenen WebService die Daten vom eye2m-Server ab und liefert sie in aufbereiteter Form an den Internetbrowser weiter. Die Daten werden im HTML-Format übermittelt und per HTTP übertragen.

Inzwischen gibt es auch für PDAs und Smartphones Internetbrowser die das HTML-Format fast vollständig unterstützen, wodurch sevenstax den mobilen Aspekt abdecken will.

Die Diagnoseoberfläche bei eye2m ist die Ausgabe auf dem Browser. Die Oberfläche bezeichnet sevenstax als eye2m-Portal. Das Portal kann für jeden Kunden nur durch den Hersteller spezifisch angepasst werden. Es gibt für die Diagnoseoberfläche auch bei dieser Lösung keinen Editor. Auch ist das HTML-Format grundsätzlich nicht frei skalierbar. Ansichten die auf dem PDA-Monitor angezeigt werden sollen, müssen also für diese Displaygröße speziell angepasst und gewartet werden.

Sevenstax baut bei seiner Lösung auf ein schlankes, kostengünstiges und leicht erweiterbares System zur Steuergerätediagnose. Durch die eye2m-Serverstruktur soll dem kontinuierlichen Anstieg der Elektronikfunktionalität im Fahrzeug begegnet werden. Die Diagnoseoberfläche kann von zentraler Stelle, dem eye2m-Server, verwaltet werden. Es müssen keine einzelnen Diagnosemasken an die Diagnosegeräte verteilt werden. Auf dem Diagnosegerät genügt die Installation des Browsers. Auf dem zentralen eye2m-Server können neue Diagnosemasken somit zeitsparend eingepflegt werden.

8.3.4 CANDela

Auf der Suche nach Software, die einen größeren Benutzerkomfort bezüglich der eigenen Gestaltung der Diagnoseoberflächen anbieten, trifft man auf Tools wie CANDela. Dieses Diagnosesystem der Vector Informatik GmbH, liefert eine umfassende Lösung mit eigener Entwicklungsumgebung und HMI-Software-Komponente.

CANdela steht für "CAN diagnostic environment for lean applications" und bedeutet im Deutschen sinngemäß CAN-Diagnose-Umgebung mit einsatzorientierten Einzelprogrammen. CAN steht hier für das unterstützte Kommunikationsprotokoll mit dem Fahrzeug (siehe Kapitel 2.1.4).

Es ist ein ganzheitlicher Ansatz zur Durchführung der Diagnose von Steuergeräten. Die Zielgruppe sind Fahrzeughersteller und Zulieferer von Steuergeräten. Die Diagnosesoftware ist spezialisiert darauf in allen Entwicklungsschritten eines Steuergeräts eingesetzt zu werden. Es ist nicht erst die Diagnose von Interesse, wenn das Steuergerät im Fahrzeug verbaut ist, sondern die Software unterstützt bereits bei der Entwicklung des Steuergeräts. Die Vector Informatik GmbH will damit den gesamten Entwicklungsprozess von Steuergeräten bis zum Einbau grundsätzlich verbessern.

Den Kern des Konzepts bildet eine Entwicklungsumgebung, die HMI-Software und eine gemeinsame Datenbasis. Die Datenbasis ist eine auf XML basierende Datei. Die Struktur und die Beschaffenheit der Daten ist proprietär. Durch die Speicherung in Klartext im XML-Format, können diese jedoch auch mit einem normalen Texteditor gelesen und bearbeitet werden.

Die XML-Dateien werden von der Entwicklungsumgebung CANDelaStudio erzeugt. Da CANDela den gesamten Entwicklungsprozess von Steuergeräten beschleunigen will, ermöglicht die Entwicklungsumgebung neben der Gestaltung der eigentlichen Diagnosemasken auch das Erstellen von Lasten- und Pflichtenheften für die Entwicklung.

Für das Erstellen der Diagnoseabläufe im firmeneigenen Format benötigt das CANDelaStudio den Import der Steuergerätedateien. Als Import-Format von Steuergeräten unterstützt das Tool unter anderem bereits das neue ODX-2-Format von ASAM und ist kompatibel zu dem nach ASAM definierten MCD-Diagnosesystem (siehe 2.2).

Vonseiten Vectors wird ausdrücklich auf die einfache Handhabung der Entwicklungsumgebung hingewiesen sowie auf die automatisierbare Übersetzung von Diagnosemasken. Die XML-Dateien können mehrsprachig gehalten werden. Immer wiederkehrende Textbausteine können von der Umgebung automatisch übersetzt werden. Dies ist auch in fernöstliche Sprachen, wie z. B. japanisch, möglich.

Zusätzlich bietet CANdito eine eigenentwickelte Skriptsprache (mit speziellen Diagnosebefehlen) an, um auch komplexere Abläufe von Diagnosefunktionen einfach erstellen zu können.

Um die Diagnoseabläufe durchzuführen, wird die HMI-Software CANDito bereitgestellt. CANDito kann die mit dem Studio erstellte XML-Datei auf ihrer Oberfläche abbilden. Die generelle Darstellung der Diagnoseabläufe und deren Daten erfolgt textbasiert. Für die Erfassung einzelner Messdaten, wie z. B. Spannungs- oder Temperaturanzeigen, bietet CANDito jedoch auch eine grafische Visualisierung. Messdaten können mit Balkenanzeigen, Graphen oder Gradienten visualisiert werden. Auch das Starten von Diagnoseskripten kann optisch in Form von Buttons (Knöpfen) hinterlegt werden.

Vektor bietet für andere Produkte bereits eine mobile Visualisierung an. Allerdings noch nicht für das CANDela-System. Die HMI-Software CANDito ist somit nur auf Desktop-Betriebssystemen von Microsoft wie XP oder 2000 lauffähig.

Generell geht der CANDela-Ansatz weit über den einer Diagnosesoftware hinaus und versucht beim Kfz-Hersteller durch die Optimierung des Entwicklungsprozesses die Zahl von fehlerbedingten Ausbauten von Steuergeräten zu verringern. Die Innovationen zielen bei der Software mehr auf den Bereich der Ablauforganisation ab als auf grafische Darstellung oder Mobilität.

8.3.5 DTS und LabView

Ein System, das seine Vorzüge hingegen sehr viel mehr in der grafischen Komponente sieht, stellt das Diagnose Toolset (DTS) der Softing AG kombiniert mit LabView von National Instruments dar.

Die Abkürzung LabVIEW steht für "Laboratory Virtual Instrument Engineering Workbench" und bezeichnet eine grafische Entwicklungsumgebung für Mess-, Prüf- und Automatisierungstechnik. Das Produkt wurde erstmals 1986 veröffentlicht und ist seither kontinuierlich weiterentwickelt worden. Aktuell ist es in einer Version für Windows und Unix-Betriebssysteme verfügbar. LabVIEW bietet die Möglichkeit, Programme rein auf grafische Weise zu erstellen, ohne den Programmquellcode als Text formulieren zu müssen. Gekoppelt mit dem Werkzeug DTS für die Steuergerätetechnik soll dem Benutzer ein Weg eröffnet werden, Diagnoseoberflächen einfach und flexibel zu erstellen.

In LabVIEW werden die erstellten Programme als virtuelle Instrumente bezeichnet (VIs), da Oberfläche und Funktion eines realen Werkzeugs, wie z. B. eines Spannungsanzeigers, digital nachgebildet werden. Die Ablauffunktionalität erhält ein VI aus einem Blockdiagramm (siehe folgende Abbildung):

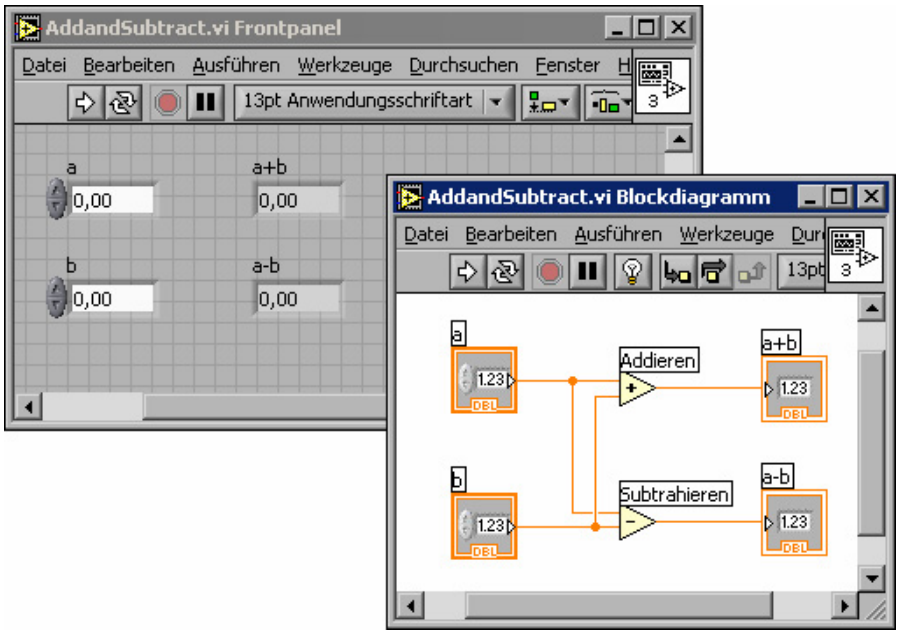


Abbildung 8.1 Blockdiagramm in LabView

In der Abbildung zeigt das obere Bild die Oberfläche, die der Benutzer später angezeigt bekommt. In diesem Fall zwei Zahleneingabefelder und zwei Ausgabefelder. Das untere Bild zeigt die Funktionalität in Form eines Blockdiagramms. Wenn Eingabefelder mit Zahlen gefüllt werden, werden sie in einem Feld addiert ausgegeben und in dem anderen Feld entsprechend subtrahiert. Die Abarbeitungs- oder Programmlogik bei LabVIEW ist also mehr datenflussorientiert. Deshalb eignet sich LabVIEW durch die Flussorientierung vor allem für die Automatisierung von Diagnoseabläufen.

VIs müssen nach der Erstellung, bevor sie genutzt werden können, kompiliert werden. Die für die Visualisierung geschaffenen Masken liegen also nicht in Klartext vor, sondern in einem für den Menschen nicht lesbaren Format (Maschinencode).

Die Oberfläche bei VIs kann zur Laufzeit nicht ohne größeren Aufwand skaliert werden. Die Skalierung der Oberfläche ist auch nicht für die gesamte Oberfläche möglich, sondern nur für spezielle Objekte, wie z. B. Graphen. Bei der Skalierung bleibt des Weiteren die Größe der Schriftart erhalten. Vis, die auf kleineren Monitoren wie PDAs dargestellt werden sollen, müssen folglich nachbearbeitet werden.

LabVIEW ist grundsätzlich auf mobilen Betriebssystemen wie Palm OS oder Pocket PC lauffähig. Es wird hierfür eine eigene HMI-Software für PDAs bereitgestellt.

Ein großer Vorteil von LabVIEW ergibt sich aus der Wiederverwendbarkeit der einzelnen VIs. Jedes VI kann in einem anderen VI als sogenanntes Sub-VI eingebunden werden. Für die Verwaltung der Diagnosemasken ergibt sich daraus der Vorteil, dass für die verschiedenen Steuergeräte oder Kommunikationsprotokolle spezielle VIs konzipiert werden können, die das Erstellen von Prüfmasken somit erleichtern.

Dieses Konzept hat sich die Softing AG für ihr Produkt Diagnose Tool Set (DTS) zunutze gemacht. Das DTS beinhaltet unter anderem ein Laufzeitsystem, um mit Steuergeräten zu kommunizieren.

Um auf die Steuergerätedaten innerhalb von LabView zugreifen zu können, bietet die Softing AG für das DTS eine spezielle Schnittstelle an. Die Schnittstelle stellt sich in Form einer VI-Bibliothek (Library) dar. Es wird damit eine Sammlung von VIs zur Verfügung gestellt, mit der ein schneller und unkomplizierter Zugriff auf die Steuergerätedaten und Funktionen erfolgen kann. Der Programmierer von Diagnosemasken kann durch die DTS-Erweiterung die Diagnoseoberfläche unabhängig von Kommunikationsprotokollen oder Steuergerätetypen erstellen.

Durch die Kombination von DTS und LabVIEW entsteht ein sehr flexibles Diagnosesystem, dass eine leicht zu bedienende grafische Entwicklungsumgebung und HMI-Software bietet, die auch auf mobilen Betriebssystemen lauffähig ist.

8.4 Vergleich der Anforderungen mit dem bestehenden Softwareangebot

In diesem Kapitel wird die gebotene Funktionalität der recherchierten Produkte mit dem Anforderungskatalog verglichen werden. Eine Zusammenfassung der Anforderungen und deren

Erfüllung bzw. Nicht-Erfüllung durch die soeben vorgestellte Software findet sich am Ende dieses Kapitels in einer tabellarischen Darstellung.

Die Einführung eines neuen Diagnosesystems stellt für Unternehmen eine kostenintensive und ressourcenbelastende Aufgabe dar. Der Aufbau entsprechender Projektteams, die Integration in die IT oder die Schulungen der Mitarbeiter für die neue Software sind nur einige Beispiele hierfür. Deshalb soll neue Diagnosesoftware möglichst viele Einsatzbereiche abdecken. Um dies zu gewährleisten, ist eine Anforderung des Katalogs, eine flexibel gestaltbare Oberfläche der HMI-Software. Durch eine Art Maskensystem soll die Diagnose für jeden Steuergerätetyp möglich werden (Kapitel 7.3.3.2). Ein System, das durch seine Oberfläche auf einen Diagnosebereich im Fahrzeug beschränkt ist und eben keine freie Oberflächengestaltung ermöglicht, wird dem Aufwand einer werksweiten Neueinführung in einem Automobilgroßkonzern nicht gerecht. Softwareangebote wie einfache Scan Tools oder auch weiterführende Systeme wie Diagra D erfüllen deshalb den geforderten Anspruch aufgrund ihrer vordefinierten Diagnoseoberfläche nicht.

Obwohl die derzeitige HMI-Software des Kooperationspartners nur auf Windows NT und XP-Betriebssystemen lauffähig ist, kann sie durch die Installation auf Notebooks trotzdem mobil eingesetzt werden. Der Mobilitätsgrad erwies sich zwar im Rahmen der Analyse als nicht hoch genug (siehe Kapitel 6.6), lässt aber trotzdem den Transport des Diagnosegeräts, z. B. mit dem Fahrzeug, zu. Dieser Transport ist vor allem in den Bereichen Steuergeräteentwicklung und Analyse wichtig. Um Steuergeräte zu analysieren oder Neuentwicklungen zu testen, können z. B. Testfahrten notwendig sein. Diese werden zum Teil auf dem Werksgelände, aber auch außerhalb durchgeführt. Selbst auf dem Werksgelände kann nicht immer sichergestellt werden, dass das interne Firmennetz erreichbar ist. Nicht nur aus technischen Gründen, sondern auch aus Sicherheitsaspekten ist es nicht erwünscht, das gesamte Werksgelände mit Netzwerkzugängen abzudecken. Besonders bei Funknetzen stößt man hier auf massive Sicherheitsbedenken. Deshalb ergibt sich die Anforderung, dass die HMI-Software eigenständig lauffähig sein soll (Stand-alone-Betrieb, siehe Abschnitt 7.3.2.2). Client/Server-Architekturen, wie bei eye2m, kommen deshalb als Lösung derzeit nicht in Frage. Es kann nicht sichergestellt werden, dass an dem Ort der Diagnose ein Zugriff auf das Firmennetz möglich ist. Bei eye2m benötigt die HMI-Software in Form eines üblichen Internetbrowsers allerdings eine permanente Verbindung zum Server. Die Lösung kann deshalb beim Kooperationspartner nicht eingesetzt werden.

Diagnosesysteme wie CANDela erfüllen hingegen den Anspruch der Stand-alone-Fähigkeit. Auch im Gegensatz zu Software wie dem Scan Tool oder Diagra D stellt CANDela für den Bereich der Oberflächengestaltung eine weitaus anpassungsfähigere Lösung dar.

Mittels einer Entwicklungsumgebung wird eine Datenbasis geschaffen, die für die Darstellung der Diagnoseoberfläche als Grundlage dient. Das System erfüllt hierbei sogar die Anforderung einer zentralen, nicht redundanten Speicherung der Datenbasis (siehe Kapitel 7.3.4.2). Auch im Rahmen einer mehrsprachigen Verwendung werden die Daten für die Diagnoseoberfläche in der zentralen Datenbasis nur einmalig gespeichert. Folglich führt der Einsatz von mehreren Sprachen nicht zu mehreren Versionen der gleichen Diagnosemaske. Mit der Speicherung der zentralen Datenbasis im XML-Format wird auch die Forderung nach der Verwendung von Standards erfüllt.

Ein weiterer Kernaspekt des Anforderungskatalogs neben der freien Gestaltung der Oberfläche, ist der technische Anspruch, auf mobilen Betriebssystemen lauffähig zu sein (siehe Kapitel 7.3.2.1). Diesem Anspruch kann die Vektor Informatik GmbH mit dem Produkt CANDela nicht gerecht werden. Sie bietet für andere Produkte zwar mobile Anwendungen an, aber die HMI-Software, mit der die CANDela-Diagnoseoberfläche dargestellt wird, ist nur auf einem Desktop-Betriebssystem von Microsoft lauffähig.

Neben dem Aspekt der Mobilität hat der Einsatz eines neuen Diagnosesystems auch das Ziel die Prozesszeit in der Verwaltung zu reduzieren. Bei Konzepten die eine Kompilierung erforderlich machen, ergibt sich immer der zusätzliche Zeitaufwand, dass die Diagnosemaske vor dem Freigabetest noch übersetzt und ausgeführt werden muss. Der zeitliche Aufwand erhöht sich beim Kompilierungskonzept noch mehr, wenn die erzeugte Anwendung auf mobilen und Desktop-Betriebssystemen lauffähig sein soll. Außerdem steigert sich auch der Verwaltungsaufwand, da neben der Quelle auch noch das Kompilat archiviert werden muss. Hier wurde ein Verbesserungspotenzial durch den Einsatz des WYSIWYG-Prinzips aufgedeckt (siehe Kapitel 7.3.4.1).

LabView basiert auf der Kompilierungstechnik. Dies ist ein Grund, warum die Lösung aus DTS und LabView nicht optimal erscheint.

Einen zweiten Grund stellt die notwendige Mehrfachspeicherung dar. Im Vergleich zu anderen Diagnosesystemen ist bei LabView hingegen eine umfangreiche Unterstützung mobiler Endgeräte gegeben. Für LabView gibt es eigenständige Module, die die Entwicklung nicht nur für Endgeräte mit dem Microsoft Pocket-PC-Betriebsystem ermöglichen, sondern auch die Nutzung auf dem Palm OS gestatten. Allerdings erfordert bei LabView die Nutzung für mobile und Desktop-Betriebssysteme eine Mehrfachspeicherung. Das zugrunde liegende Betriebssystem muss beim Erstellen eines Programms mit LabView festgelegt werden. Die Diagnoseoberflächen müssen deshalb einmal für das mobile und das Desktop-Betriebssystem verwaltet werden. Dies erhöht gerade im Hinblick auf die steigende Steuergerätezahl den Aufwand bei der Verwaltung der Diagnosemasken enorm. Das LabView Konzept erfüllt aus diesen Gründen den Anspruch des Anforderungskatalogs nicht (siehe Kapitel 7.3.4.2).

Bevor in Kapitel 8.5 ein Resümee bezüglich des Vergleichs gezogen wird, wird die Erfüllung der einzelnen Anforderungspunkte in einer Tabelle zusammengefasst. In der folgenden Tabelle symbolisieren die Kontrollkästchen, ob eine Eigenschaften erfüllt (☑) oder nicht erfüllt (☒) ist. Ein leeres Kästchen bedeutet, dass über die Anforderungen bezüglich des Produkts keine Aussage (☐) getroffen werden kann.

Anforderung	Diagnostic Pocket PC Scan Tool	Diagra D Scan Tool	eye2m	CANDela	DTS und LabView
Schnittstellen-Anforderungen					
Unterstützung des proprietären Basissystems des Kooperationspartners	☒	☒	☒	☒	☒
Import der Steuergeräteinformationen in die Entwicklungsumgebung	☐	☒	☑	☑	☑
Technische Anforderungen					
Betriebssystemübergreifende (mobile) Lauffähigkeit der HMI-Software	☑	☑	☑	☑	☑
Stand-alone-Lauffähigkeit der HMI-Software	☑	☑	☒	☑	☑
Ausgabe in hoher Bildqualität	☒	☑	☑	☑	☑
Unterstützung von Mehrsprachigkeit	☒	☑	☒	☑	☑
Verwendung von Standards bezüglich des Grafik- bzw. Skriptformats	☒	☒	☑	☒	☒
Funktionale Anforderungen an die HMI-Software	HMI vorhanden	HMI vorhanden	HMI vorhanden	HMI vorhanden	HMI vorhanden
Interaktion mit Benutzer	☑	☑	☑	☑	☑
Diagnoseoberfläche unterstützt Maskensystem	☒	(☑)	☑	☑	☑
Freie Skalierbarkeit der Oberfläche	☒	☒	☒	☒	☒
Funktionale Anforderungen an die Entwicklungsumgebung	Editor nicht vorhanden	eingeschr. editierbar	eingeschr. editierbar	Editor vorhanden	Editor vorhanden
Kompilierung nicht erforderlich und WYSIWYG-Unterstützung	☐	☒	☑	☑	☒
Redundanzfreies Erstellen der Masken ("Unikat")	☐	☒	☑	☑	☒
Grafisches Erstellen der Masken	☐	☒	☒	(☑)	☑
Umfassende grafische Gestaltungsfunktionalität	☐	☒	☒	☒	☑
Unterstützung von Upload/Download	☐	☒	☑	☑	☑

Tabelle 8.1 Vergleich Softwareangebot mit Anforderungskatalog

8.5 Ergebnis

Wie aus Tabelle 8.1 hervorgeht, wurde kein Angebot am Softwaremarkt gefunden, das die Ansprüche des Anforderungskatalogs vollständig abdeckt. Gleiches gilt für die Ansätze der Forschungsprojekte.

Was positiv am Marktangebot auffällt, ist die Unterstützung mobiler Endgeräte. Auch wird immer mehr versucht, die Einsatzflexibilität zu erhöhen, indem eine Entwicklungsumgebung zur Gestaltung der Oberflächen in das Konzept eingebunden wird. Die Entwicklungsumgebung unterstützt auch zunehmend eine grafische Gestaltungsweise.

Allerdings ist zu erkennen, dass der mobile Einsatz der HMI-Software nicht als fester Bestandteil des Kernkonzepts realisiert wurde, sondern mehr als Zusatzfeature angeboten wird. Die Folge daraus ist, dass die mobilen Oberflächen zusätzlich erstellt werden müssen. Bei Großkonzernen der Automobilindustrie mit vielen verschiedenen Baureihen wäre diese Doppeltbelastung nicht vertretbar.

Auch die Verwendung offizieller Standards findet sich nur vereinzelt in Produkten. Besonders aber in den letzten Jahren verstärkt sich in der Industrie das Bewusstsein, Systeme bereichsübergreifend vernetzen zu müssen. Dies ist bei proprietären Firmenlösungen (sogenannte Insellösungen) ohne standardisierte Schnittstellen und Formate um ein Vielfaches zeit- und kostenaufwendiger oder teilweise überhaupt nicht möglich. Als Folge daraus wird die Abneigung gegenüber Insellösungen oder proprietären Formaten immer größer.

Da die bestehenden Konzepte die Ansprüche nicht erfüllen, ist die Überlegung in Betracht zu ziehen, eine am Markt existierende Software zu erweitern. Hierbei darf allerdings die zentrale Rolle des Grafikformats nicht unterschätzt werden. Wie bei jeder grafischen Ausgabe ist die Verarbeitung des Formats tief und an vielen Stellen in der Software verankert. Das Umstellen des Grafikformats würde einen brachialen Umbau der Entwicklungsumgebung und der HMI-Software erfordern. Das Lesen und die Verarbeitung des Diagnoseskripts (Grafikformats) müsste für beide Softwarekomponenten neu programmiert werden.

Eine Umstellung in Richtung WYSIWYG-Entwicklungsumgebung, Auflösung des Kompilierungskonzepts, redundanzfreie Speicherung der Diagnosemaske oder frei skalierbare Oberfläche wäre jedoch ohne Änderung des Grafikformats nicht möglich.

Die Kritiken an den bestehenden Systemen könnte folglich nicht ohne Änderungen an dieser zentralen Stelle aufgelöst werden. Da eine solche Änderung einen derartig massiven Umbau eines fertigen Softwareprodukts zur Folge hätte und auch teilweise der Grundidee des bestehenden Konzepts widersprechen würde, erscheint die Anpassung einer am Markt gefunden Lösung nicht sinnvoll.

Es wird deshalb selbst eine neue Diagnosesoftware auf Basis des Anforderungskatalogs entwickelt.

9 Entwurf einer neuen Diagnosesoftware

Im Softwareentwicklungsprozess wird die Entwurfsphase eingeleitet, wenn die Anforderungen vorliegen und softwaretechnisch umgesetzt werden sollen[204]. Die Anforderungen wurden bereits in Kapitel 7.3 definiert. Das Ergebnis von Kapitels 8 ist, dass eine Neuentwicklung angestrebt wird.

Mit dem Entwurf wird ein Diagnosesystem konstruiert, das alle geforderten Ansprüche des Anforderungskatalogs erfüllt. Um diese Aufgabe zu bewältigen, ist besonders bei diesem Schritt der Softwareentwicklung Kreativität und handwerkliches Geschick gefragt[205].

Das Ergebnis des Entwurfs soll in diesem Kapitel präsentiert werden.

Um Systeme mit hoher Komplexität zu beschreiben, bietet es sich an, die Beschreibung in mehrere Abstraktionsebenen zu gliedern. Üblicherweise wird hierbei ein Basiskonzept sowie eine Makro- und Mikroarchitektur unterschieden[206].

Das Basiskonzept wird als Einstieg kurz erläutert. Es beschreibt allgemein die Funktionalität, ohne auf architektonische Details einzugehen. In Kapitel 9.1 werden hierfür die Kernfunktionen des Systems aufgelistet, die dann in Kapitel 9.2 in die Makroarchitektur überführt werden.

Mit der Makroarchitektur wird die grundlegende Strukturierung eines Systems dargelegt. Es sollen damit die einzelnen Softwarekomponenten und deren Zusammenhänge beschrieben werden[207]. Die Erläuterungen werden durch eine Makroarchitekturabbildung ergänzt, die alle Kernmodule des Diagnosesystems zusammenführt.

Mit der Darstellung der Makroarchitektur werden in Kapitel 9.3 auch deren Basistechnologien aufgeführt. Hierbei wird begründet, warum die einzelnen Technologien ausgewählt wurden und welche Potenziale sie bieten.

Aufbauend auf den Basistechnologien wird mit der Mikro- bzw. technischen Architektur die innere Struktur und Funktionsfähigkeit der einzelnen Komponenten wiedergegeben. Daraus

[204] Vgl. Thaller 2000, S. 50
[205] Vgl. Forbrig 2004, S. 31
[206] Vgl. Dunkel 2003, S. 11
[207] Vgl. Dunkel 2003, S. 12-14

kann direkt die Implementierung abgeleitet werden. Die Mikroarchitektur ist deshalb erst Inhalt des Kapitels 10, das sich mit der Umsetzung des Entwurfs beschäftigt (Kapitel 10).

9.1 Konzept

Das in dieser Arbeit konzipierte System bietet eine Softwarelösung zur Diagnose elektronischer Steuergeräte in der Automobilindustrie. Das System kombiniert die Vorteile mobiler Nutzung, grafischer Gestaltung und Anzeige flexibler Diagnoseoberflächen.

Hierfür wurde das Konzept in die zwei Kernelemente HMI-Software und zugehörige Entwicklungsumgebung gegliedert. Die HMI-Software stellt die Diagnoseoberfläche dar und wird als Viewer bezeichnet. Sie dient zur Durchführung der eigentlichen Diagnosetätigkeit am Fahrzeug. Mit der Entwicklungsumgebung können Diagnosemasken erstellt werden. Für die Entwicklungsumgebung wird innerhalb des Konzepts auch das Synonym Editor verwendet.

Die mit dem Editor erstellten Diagnosemasken können mit dem Viewer angezeigt werden. Die Diagnoseoberfläche des Viewers erhält so über die Maske ihre Funktionalität bzw. die Diagnosefähigkeit. Die Masken können eine große Bandbreite an grafischen Elementen mit hoher Farbtiefe und Auflösung beinhalten. Die gestalterischen Möglichkeiten reichen von Freihandzeichnungen bis hin zu dynamisierten Grafiken. Die Diagnosemasken werden im Diagnoseskript gespeichert.

Durch qualitätsverlustfreie Skalierung der Diagnoseoberfläche und betriebssystemübergreifenden Lauffähigkeit des Viewers kann das System auf mobilen Endgeräten und Diagnoseterminals parallel genutzt werden. Eine zusätzliche Überarbeitung der Diagnosemasken für den mobilen Einsatz ist nicht erforderlich.

Das Diagnoseskript muss folglich für die mobile Nutzung nicht mehrfach (redundant) gespeichert werden. Gleiches gilt auch bei Nutzung der Mehrsprachigkeit. Die Sprachumschaltung erfolgt zur Laufzeit aus separaten Sprachdateien. Unabhängig von der Anzahl zusätzlicher Fremdsprachen bleibt das ursprüngliche Diagnoseskript unberührt.

Ergänzt wird das Erstellen der Diagnoseskripte durch eine Up- und Download-Funktionalität. Aus dem Editor heraus können die Skripte in einem Server-Verzeichnis gespeichert werden und von diesem auch wieder geladen werden.

Editor und Viewer orientieren sich an verbreiteten Standards. Beide sind auf Basis von etablierten Plattformen zur Softwareentwicklung erstellt (siehe Kapitel 10.2 und 10.3). Auch das Grafikformat stützt sich auf einen offenen Vektorgrafikstandard (siehe Kapitel 10.1).

Des Weiteren verfügen Editor und Viewer über eine flexibel einstellbare Schnittstelle zur Integration eines Diagnosebasissystems. Es können folglich neben dem proprietären Basissystem des Kooperationspartners weitere Kommunikationssysteme von Kfz-Herstellern eingebunden werden.

Aus der Schnittstelle des Editors lässt sich noch ein weiterer Nutzen ziehen. Über die Schnittstelle können die Steuergerätedaten des Basissystems in den Editor importiert werden. Dadurch stehen die Steuergerätedaten innerhalb des Editors zur Verfügung. Der Programmierer kann somit bei der Erstellung der Diagnosemasken die notwendigen Steuergerätedaten im Editor

ablesen und sie direkt in das Diagnoseskript eintragen. Er muss die Steuergerätedaten nicht über ein zusätzliches Programm abrufen.

Das genaue Zusammenspiel zwischen Editor, Basissystem und Viewer wird im folgenden Kapitel mit der Architekturbeschreibung verdeutlicht.

9.2 Architektur

Mit der nachfolgenden Architekturabbildung werden die einzelnen Funktionen aus dem Konzept (Kapitel 9.1) in ein Gesamtsystem überführt. Dabei wird dargestellt, welche Softwarekomponenten in dem System verankert sind und in welcher Wechselwirkung sie zueinander stehen. Es zeigt wie die einzelnen Komponenten im Gesamtkonzept des Diagnosesystems interagieren. Diese Inhalte werden in der sogenannten Makroarchitektur erfasst.

Die Makroarchitektur kann grundsätzlich in informeller Form abgebildet werden, sollte aber folgende Richtlinien berücksichtigen[208]. Neben den bereits benannten Inhalten wie Softwarekomponenten und deren Zusammenhänge sollte auch die physische Verteilung miteinfließen. Dies beinhaltet Datenspeicher, wie z. B. Dateien, Datenbanken oder Bibliotheken, von denen gelesen bzw. auf die geschrieben wird.

Des Weiteren soll zur Steigerung der Verständlichkeit die Darstellungsform an bereits bestehende bzw. thematisch angrenzende Schemata angeglichen werden. Die abgebildete Makroarchitektur schließt sich deshalb an die Darstellungsweise der Diagnosesysteme aus Kapitel 2.2 und Kapitel 3 an.

[208] Vgl. Dunkel 2003, S. 13

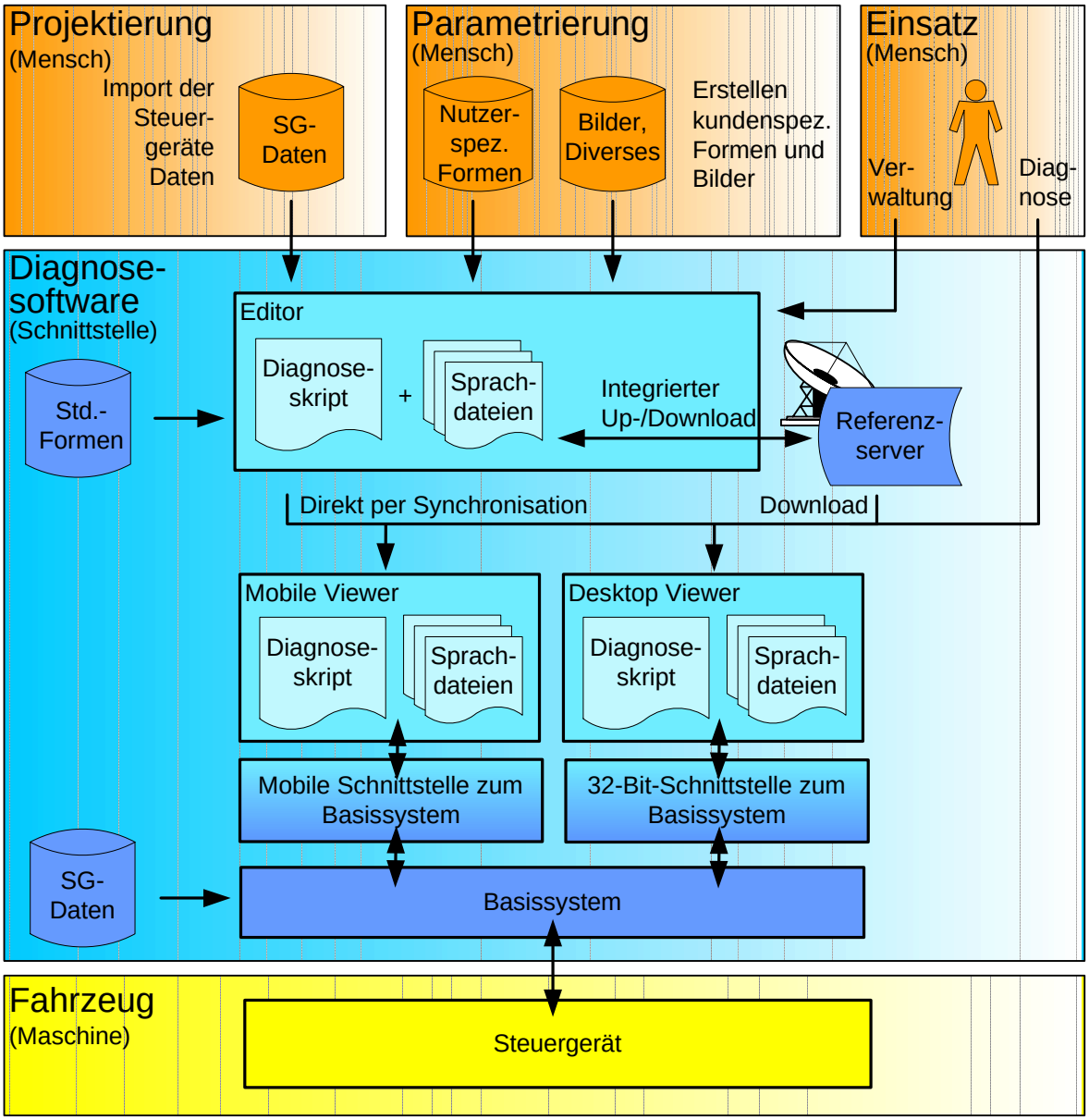


Abbildung 9.1 Makroarchitektur des neuen Diagnosesystems

Vergleicht man die Abbildung mit dem Schema des Ist-Systems (siehe Kapitel 3.3.1), dann fällt an zentraler Stelle ein grundlegender Unterschied auf. Das neue System kommt mit einer Diagnosedatei (dem Diagnoseskript) aus. Bei dem Ist-System muss hingegen die Diagnoseskriptquelle und das kompilierte ausführbare Diagnoseskript archiviert werden. In Kapitel 3.3.2 wurde bereits darauf hingewiesen, dass die Begriffe im Ist-System nicht korrekt verwendet werden. Der Ausdruck Skript ist im Zusammenhang mit einer Kompilierung nicht korrekt. Man spricht dabei eher von einer Quelle und einem Kompilat. In dem neuen System ist der Begriff Skript allerdings gerechtfertigt, denn es wird der Quellcode in der Form, in der er erstellt wird, auch ausgeführt. Eine Kompilierung ist nicht erforderlich.

Auch bei Verwendung der Mehrsprachigkeit bleibt das Konzept der redundanzfreien Speicherung des Diagnoseskripts erhalten. Die Übersetzungen werden in separaten Dateien gehalten. Innerhalb des Diagnoseskripts findet sich nur ein Verweis auf die Textstelle. Zur Laufzeit lädt

der Viewer über den Verweis den Text aus der Sprachdatei in der jeweiligen Sprache. Soll eine Diagnosemaske also in einer Fremdsprache dargestellt werden, so muss nur eine Textdatei mit den Übersetzungen in der entsprechenden Sprache erstellt und dem Diagnoseskript beigelegt werden. Die Sprache kann dann im Viewer zur Laufzeit ausgewählt werden.

Um die Diagnoseskripte anzufertigen, stellt das System einen Editor zur Verfügung. Für den Editor wurde eine flexible und funktionell leicht nachrüstbare Plattform gewählt. Die Flexibilität beim Erstellen der Skripte ergibt sich aus einem erweiterbaren Bibliothekenkonzept. In dem System werden Bibliotheken dazu verwendet um Steuergerätedaten sowie Standard- und nutzerspezifische Diagnoseformen einzubinden. Die Bibliotheken werden in Abbildung 9.1 mit dem charakteristischen zylindrischen Datenträgersymbol eines Standardflussdiagramms dargestellt.

Die wichtigste Bibliothek im Kontext der Diagnose ist die der Steuergerätedaten. Diese beinhalten die Zugriffsinformationen auf das Steuergerät. Über eine offene Schnittstelle können der Name des Steuergerätebefehls, Attribute und Ergebniswerte, die zum Aufruf eines Steuergerätebefehls notwendig sind (siehe Kapitel 3.2), in den Editor importiert werden. Die Daten sind von Baureihe zu Baureihe und von Automobilhersteller zu Automobilhersteller, unterschiedlich und müssen somit für jeden Nutzer des Diagnosesystems (Kundenprojekt) neu bereitgestellt (projektiert) werden. Innerhalb des Editors können die Steuergerätedaten per Auswahllisten selektiert werden. Über die Auswahl kann grafischen und/oder textbasierten Objekten (Diagnoseformen) Steuergerätefunktionalität zugewiesen werden.

Die Objekte werden über zwei weitere Bibliotheken integriert. Zum einen bietet das System von Haus aus eine Sammlung von standardisierten Formen zur Diagnose. Da sind Objekte, die üblicherweise in der Diagnose benutzt werden. Beispiele sind LED (An-/Aus-Anzeigen), Schalter, Gradienten oder schlichte Zahlenwert- bzw. Textausgaben. Die Bibliothek der Standardformen ist fester Bestandteil der Editorsoftware und wird an jeden Nutzer mitausgeliefert.

Das System kann durch eine beliebige Anzahl nutzerspezifischer Objekte erweitert werden. Dieser Bereich der Parametrierung kann auch durch den Nutzer selbst erfolgen. Das Grafikformat und die Programmiersprache für die Verarbeitungslogik der Masken sind in einem offenen lizenzkostenfreien Standard definiert. In diesem Format können beliebig grafische funktionale Diagnoseformen geschaffen werden und in die Diagnosemasken eingebettet werden. Ergänzt wird die Bibliothek der nutzerspezifischen Diagnoseformen durch Hintergrundbilder, Grafiken oder Stylesheets (Formatierungsvorlagen). Auf die umfassenden Möglichkeiten des Grafikformats wird in Kapitel 9.3.1 näher eingegangen.

Nach dem Erstellen der Diagnoseskripte und der zugehörigen Sprachdateien können diese über einen integrierten Upload-Mechanismus auf einen Server kopiert werden. Die Übertragung ist per FTP möglich. Hierfür muss in den Editor-Einstellungen der jeweilige Pfad angegeben werden. Über den gleichen Pfad können die Dateien auch heruntergeladen werden. Des Weiteren besteht auch die Möglichkeit, über Software von Drittanbietern die Dateien direkt auf das Diagnoseendgerät zu übertragen (Synchronisation von Endgeräten).

Zur Durchführung der Diagnosetätigkeit bietet das System einen mobil lauffähigen Viewer sowie einen Viewer für Desktop-PCs (Diagnoseterminals) an. Der mobile Viewer trägt den Namen "Mobile Viewer", die Desktop-Version den Namen "Desktop Viewer". Die Viewer-Software muss zusammen mit einer Laufzeitumgebung auf dem Endgerät installiert werden. Die Plattform, auf der der Viewer und die Laufzeitumgebung basieren, wird in Kapitel 9.3.3 näher erläutert. Unabhängig vom zu Grunde liegenden Endgerät kann das Diagnoseskript in

gleicher Form auf dem Mobile und Desktop Viewer verarbeitet werden. Es wird folglich das Konzept der redundanzfreien Skriptspeicherung auch für den mobilen Einsatz nicht gebrochen. Durch qualitätsverlustfreie Skalierbarkeit können die Diagnosemasken auf handflächengroßen PDAs und 21" Terminalmonitoren in gleicher Qualität abgebildet werden.

Zur Kommunikation mit dem Fahrzeug wird über eine offene Schnittstelle das proprietäre Basissystem des Kooperationspartners integriert. Welche softwaretechnischen Komponenten hierfür notwendig sind, wird im Rahmen der Beschreibung der Mikroarchitektur Beschreibung in Kapitel 10.2 erläutert.

9.3 Basistechnologien der Architektur

Mit dem Konzept und der Architektur wurden die zentralen Komponenten des Diagnosesystems vorgestellt. Eine wichtige Rolle nehmen hierbei der Editor und der Viewer ein sowie das Diagnoseskript.

Die Makroarchitektur (Kapitel 9.2) richtet grundlegende Anforderungen an diese Komponenten. Um den Anforderungen gerecht zu werden, müssen entsprechende technische Möglichkeiten gefunden werden. So stellen z. B. die Skalierung und die grafische Gestaltungsweise Anforderungen an das Grafikformat des Diagnoseskripts. In Kapitel 9.3.1 wird im Hinblick darauf das zentrale Grafikformat für das neue System bestimmt. Ferner muss eine Plattform gefunden werden, mit der der Viewer auf mobilen Geräten und auf Terminalanlagen lauffähig ist (siehe Kapitel 9.3.3). Der Editor muss leicht erweiterbar sein, um die Bibliotheken der Steuergerätedaten oder nutzerspezifische Diagnoseformen integrieren zu können. Auch hierfür muss eine Umgebung gewählt werden, die die technischen Grundlagen dafür bietet (siehe Kapitel 9.3.2).

Mit der Auswahl dieser Basistechnologien setzen sich die folgenden Unterkapitel auseinander. Sie liefern Argumente für die gewählten Technologien und erläutern in diesem Zusammenhang deren Hintergründe. Basierend auf diesen Entscheidungen kann die Architektur weiter verfeinert und umgesetzt werden (siehe Kapitel 10).

9.3.1 Bestimmung des zentralen Grafikformats

9.3.1.1 Entscheidung für Vektorgrafik

Eine Vektorgrafik beschreibt ein Bild durch mathematische Funktionen in einem 2- oder 3-dimensionalen Koordinatensystem. Durch die in dem Koordinatensystem eingetragenen Vektoren entstehen Linien, Kurven oder Flächen. Um z. B. eine Linie darzustellen, werden in der Vektorgrafik nur vier Funktionswerte gespeichert: Anfangspunkt, Endpunkt, Farbe und Strichstärke[209].

Bei Rastergrafiken oder Pixelgrafiken (z. B. BMP oder JPEG) hingegen werden Informationen für jeden erfassten Punkt (Pixel) des gesamten Bildes gespeichert[210]. Bei einer Linie würde dies bedeuten, dass nicht nur der Anfangs- und Endpunkt, sondern jeder Punkt auf der Linie

[209] Vgl. Holzinger 2000, S. 142-143
[210] Vgl. Henning 2003, S. 276

gespeichert wird. Die Linie wird folglich durch eine Verknüpfung gleichgroßer Punkte, einem sogenannten Raster, dargestellt. Da die Punkte eine fixe Größe haben und die Linie folglich aus einer endlichen Zahl dieser Punkte besteht, werden bei einer bestimmten Vergrößerungsstärke die Rasterpunkte für das Auge sichtbar werden. Das Bild wird ab einer bestimmten Vergrößerung für den Betrachter unscharf. Bei Verkleinerungen kommt es zu Unschärfen, da die Punkte ineinander verschwimmen. Indessen werden bei der Vektorgrafik keine einzelnen Rasterpunkte eines Objekts gespeichert, sondern nur Funktionswerte, um das Objekt zu erstellen. Dadurch kann jedes Objekt für die Darstellung an beliebige Vergrößerungs-/Verkleinerungsstufen angepasst und ohne Unschärfen gezeichnet werden.

Vektorgrafiken haben daher den Vorteil, dass sie im Gegensatz zu Rastergrafiken ohne Qualitätsverlust stufenlos skaliert werden können. Außerdem können die Elemente einer Vektorgrafik über ihre wenigen Funktionswerte nachträglich leicht modifiziert werden, wohingegen bei einer Rastergrafik jeder einzelne Pixelwert des Elements verändert werden muss. Der erforderliche Speicherbedarf liegt im Durchschnitt auch nur bei einem Zehntel von dem einer vergleichbaren Rastergrafik[211]. Die Vektorgrafik erscheint deshalb als gute Basis für ein Grafikformat, das auf variablen Monitorgrößen ausgegeben werden soll.

Zur Abbildung der Vektorgrafiken wurde der Vertreter SVG gewählt. Mit den Erläuterungen der in den Folgekapiteln werden weitere Vorteile genannt, die die Entscheidung für eine Vektorgrafik bzw. für SVG begünstigen.

9.3.1.2 Einführung in die Vektorgrafik SVG

Scalable Vector Graphics (SVG) ist eine Sprache zur Beschreibung von zweidimensionalen Vektorgrafiken mittels XML.

XML steht für "Extended Markup Language" und ist eine Beschreibungssprache für Dokumente und Daten. XML wurde von einer Arbeitsgruppe des World Wide Web Consortiums (W3C) entwickelt, um Daten bzw. Dokumente jeglicher Art strukturiert abzubilden[212]. XML hat sich heute in sehr vielen Anwendungsbereichen etabliert. Datenbanken, Protokolle, Grafikformate oder Werkzeuge wie Microsoft Office oder OpenOffice sind nur einige Beispiele davon[213].

Auch wenn der Karriereweg des SVG-Formats nicht so zielstrebig war, wie der von XML, hat es sich in Bereichen etabliert, die für die Arbeit von Interesse sind. Mit der Empfehlung zum Standard im September 2001 durch W3C haben Hersteller von Internetbrowsern und mobilen Endgeräten begonnen, SVG nativ, also ohne erforderliche Installation, zu unterstützen[214].

Durch seine Eignung für mobile Endgeräte weist es folglich einige grundsätzliche Eigenschaften auf, die für den Einsatz in dieser Arbeit von entscheidendem Vorteil sind. Im folgenden Kapitel werden deshalb die relevanten Merkmale von SVG beschrieben.

9.3.1.3 Gründe für die Auswahl von SVG

Das SVG-Format zeichnet sich durch folgende Eigenschaften aus, die zur Entscheidung für SVG geführt haben.

---

[211] Vgl. Adam 2002, S. 14
[212] Vgl. Henning 2003, S. 418
[213] Vgl. W3C-Std 2006 (Online-Quelle)
[214] Vgl. Wenz 2006, S. 19

- **Skalierbarkeit**

Durch die Unterstützung des skalierbaren Vektorgrafikformats kann eine SVG-Abbildung in gleicher Qualität auf jedes beliebige Format vergrößert werden, ohne dass die gespeicherten Informationen in der Datei zunehmen müssen. Es können hochkomplexe Darstellungen mit geringer Dateigröße gespeichert werden. Dieser Vorteil wirkt sich aber auch in die andere Richtung aus. SVG-Grafiken können praktisch auf beliebig kleine Monitorgrößen, etwa Handy oder PDA, ohne Qualitätsverlust verkleinert werden, und das ohne den Dateiinhalt dafür ändern zu müssen. Damit ist auch die Basis gesetzt, um Bildausschnitte verlustfrei zu zoomen (vergrößern bzw. verkleinern). Auf einem kleinen Monitor kann folglich ein relevanter Teilbereich gezoomt werden, bis er für den Benutzer in einer lesbaren Größe erscheint.

- **Lesbarkeit der Quelle**

Die SVG-Dateien werden textbasiert mittels der XML-Sprache gespeichert. Dies hat den Vorteil, dass die Datei mit einem gewöhnlichen Texteditor geöffnet werden kann und vom Menschen lesbar ist.

- **Platzsparende Speicherung**

Durch das Vektorformat ergibt sich der bereits erwähnte Vorteil der geringeren Dateigröße. Eine SVG-Datei benötigt bei vergleichbarer Bildqualität nur ein Dreizehntel des Speicherplatzes einer JPEG-Datei[215]. Zudem kann die textbasierte Datei noch durch handelsübliche Komprimierungsprogramme auf ein Viertel ihrer Größe reduziert werden. Diese Komprimierung wird bei SVG durch die Spezifikation explizit unterstützt und gefördert. Das komprimierte Format hat die Endung SVGZ und kann von einigen SVG-Viewern direkt gelesen werden.

- **Kompatibilität zu anderen Standards**

Durch die Speicherung in XML-Format ergibt sich die Möglichkeit der Kombination mit anderen XML-basierten Technologien. SVG ist z. B. voll kompatibel zu XHTML und weitgehend zu HTML. Das sind zwei wichtige Formate für die Darstellung von Internetseiten. Diese Integrierbarkeit und Konformität von SVG-Code zu anderen Standards sind grundlegende Unterschiede gegenüber anderen Konkurrenten[216].

- **Umfassende grafische Möglichkeiten**

SVG verfügt zusätzlich über ein breites Spektrum an Animations- und Transformationsmöglichkeiten[217]. Die Darstellung von hochauflösenden Schattenverläufen oder das Schwenken, Zerren oder Zoomen von Grafikobjekten, wie Rechtecken oder Kreisen, wird möglich.

Mit der Unterstützung des RGB-Farbmodells unterstützt SVG darüber hinaus die True Color-Darstellung von über 16 Millionen Farben[218].

[215] Vgl. Henning 2003, S. 276
[216] Vgl. W3C-Spec 2003 (Online-Quelle), Kapitel 1
[217] Vgl. W3C-Spec 2003 (Online-Quelle), Kapitel 13
[218] Vgl. Adam 2002, S. 40

- **Unterstützung von Interaktivität durch SVG**

Eine weitere Eigenschaft des SVG-Formats, die für den Einsatz in der Diagnose als unabdingbar festgelegt wurde, ist die Unterstützung von Interaktivität und Dynamiken (siehe Kapitel 7.3.3.1).

Um interaktive Aktionen und Dynamiken verarbeiten zu können, unterstützt SVG eine Skriptsprache. Der Code für die Skriptsprache kann neben dem SVG-Code in die SVG-Quelle mit eingetragen werden. Die unterstützte Skriptsprache ist ECMAScript.

ECMAScript ist eine Standardisierung der Skriptsprache JavaScript durch die ECMA Organisation. Sie sind in Syntax und Funktionsumfang nahezu identisch[219]. JavaScript ist eine objektorientierte Skriptsprache, die von der Firma Netscape entwickelt wurde. Unter den Skriptsprachen hat JavaScript weltweit die größte Verbreitung gefunden. Durch das Zusammenwirken von ECMAScript und SVG kann auf nahezu jede (Benutzer-)Aktion reagiert werden. Dies beginnt mit dem Laden einer SVG-Datei, dem Verändern der Größe (Zoomen), dem Scrollen (Auf-/Abwärts Rollen) des Dokuments bis hin zu einem pixelgenauen Klick auf dem Monitor.

Wie die Interaktionsfähigkeit erfüllen die umfassenden grafischen Gestaltungsmöglichkeiten, die Kompatibilität zu Standards und die Skalierbarkeit direkt Forderungen des Anforderungskatalogs aus Kapitel 7.3. Dies hat entscheidend zur Auswahl von SVG beigetragen. In den Kapiteln 10 und 11, bei der Umsetzung und Bewertung des Diagnosesystems, wird die Erfüllung der einzelnen Punkte des Anforderungskatalogs durch das Grafikformat noch deutlicher hervorgehoben.

An dieser Stelle sei erwähnt, dass es noch weitere Vertreter von Vektorgrafikformaten gibt. Einer der bekanntesten ist SWF (Shock Wave Flash) der Firma Macromedia. Da SVG den Bedürfnissen dieser Arbeit vollstens entspricht, wird auf einen umfassenden Vergleich mit den übrigen Vektorgrafikvertretern verzichtet. Bezüglich SWF ist zu erwähnen, dass es im Gegensatz zu SVG keine lesbare Quelldatei bietet. Die Dateien sind binär gespeichert. Das Format ist demnach auch nicht XML-basiert, sondern proprietär. Ebenso ist die Skriptssprache zur Unterstützung der Interaktivität eine proprietäre Entwicklung und nur angelehnt an ECMA- bzw. JavaScript[220]. Der Standardisierungsgrad ist mit dem von SVG nicht zu vergleichen. Ähnlich verhält es sich bei anderen Vertretern. Deshalb fiel die Entscheidung zugunsten von SVG aus.

9.3.1.4 Auswahl der passenden SVG-Spezifikation

Mit der Empfehlung zum Standard von SVG 1.0 im Jahr 2001 steigerten sich die Anfragen aus der Wirtschaft nach der Darstellung von Vektorgrafik auch auf leistungsschwächeren mobilen Endgeräten. Deshalb hat es sich die SVG-Arbeitsgruppe zu einer gemeinschaftlichen Aufgabe gemacht, eine Spezifikation zu erstellen, die für mobile Geräte bestimmt ist.

Da sich mobile Geräte aufgrund der Prozessorgeschwindigkeit, Speichergröße und Farbunterstützung sehr viel stärker als normale Arbeitsplatzrechner unterscheiden, war es nötig, zwei Profile zu definieren.

Mit dem Profil SVG Tiny soll die große Gruppe der Handy-Nutzung abgedeckt werden. Das Profil SVG Basic hingegen stützt sich bereits auf eine höhere Prozessorleistung und mehr Ar-

[219] Vgl. Henning 2003, S. 390
[220] Vgl. Ricken 2005, S. 19

beitsspeicher und soll die Gruppe der PDAs, z. B. Palms und Pocket PCs, ansprechen. Beide Profile zusammen ergeben den SVG Mobile Standard.

Der Tiny Standard weist allerdings in den Bereichen der Animationen, Interaktivität und der Farbverarbeitung einige Defizite auf. So unterstützt er z. B. keine Farbverläufe. Diese sind allerdings für die Gestaltung von Gradienten in der Diagnose hilfreich bzw. notwendig. Es können damit bei Temperaturreglern durch die schrittweise Veränderung der Farbe ein Temperaturanstieg oder -abfall simuliert werden. Auch die Interaktivität ist ein elementarer Bestandteil für eine HMI-Software in der Diagnose. Diese wird bei SVG per Einbindung des ECMAScripts realisiert. Der SVG Tiny Standard unterstützt allerdings in seiner ursprünglichen Definition das Element Skript nicht.

Für die vorliegende Arbeit ergibt sich folglich die Anforderung, dass nur die Basic- oder die Hauptspezifikation von SVG verwendet werden kann.

9.3.1.5 Integration der SVG-Spezifikation

Die Internetseite "SVG Implementations" der W3C-Homepage liefert einen nahezu vollständigen und aktuellen Überblick aller auf dem Markt verfügbaren und geplanten SVG-Implementierungen[221]. Diese Implementierungen können genutzt werden, um das SVG-Format in Softwaresysteme zu integrieren.

Für das Diagnosesystem wird zum einen eine Implementierung zur Anzeige im Viewer benötigt. Zum anderen muss das SVG-Format in den Editor integriert werden, um die Diagnoseskripte grafisch zu bearbeiten.

Für beide Anforderungen bietet der Markt grundsätzlich eine Vielzahl an Lösungen. Es finden sich aktuell zwanzig Anzeige- und über zwanzig Bearbeitungswerkzeuge auf dem offiziellen SVG-Internetauftritt[222]. Durch die Einschränkungen, die sich für den Einsatz im neuen Diagnosesystem ergeben, reduziert sich die Zahl allerdings drastisch. Besonders die Kriterien, dass die Implementierung auf mobilen und Desktop-Betriebsystemen lauffähig sein muss, und die Forderung nach der Basic bzw. Hauptspezifikation, reduzieren die Auswahl für den Viewer auf einen Anbieter.

Das eSVG-Plug-in von Intesis ist auf dem mobilen Microsoft Betriebssystem Pocket PC und auf dem Microsoft PC-Betriebssystem 2000, NT und XP lauffähig. Des Weiteren werden die Anforderungen der Interaktivität erfüllt. Durch die eSVG Script Engine kann das Plug-in das notwendige ECMAScript verarbeiten. Ein Großteil der SVG-Basic-Spezifikation wird unterstützt, was auch Farbverläufe und Animationen miteinschließt[223]. Das Plug-in erfüllt somit alle Anforderungen um im Viewer des Diagnosesystems eingesetzt zu werden.

Neben dem Viewer muss SVG auch im Editor verarbeitet werden. Da der Editor nicht die Anforderung erfüllen muss, auf mobilen Systemen lauffähig zu sein, ist das zur Verfügung stehende Angebot an Implementierungen größer.

Batik ist ein Java-basiertes Toolkit (Werkzeugsammlung) der Apache Foundation zur Integration in Programme, die SVG-Grafiken anzeigen, generieren oder bearbeiten wollen. Es ist die derzeit vollständigste Java-Implementierung der SVG-Hauptspezifikation. Sie unterstützt alle

[221] Vgl. SVGI 2007 (Online-Quelle)
[222] Vgl. SVGI 2007 (Online-Quelle), Stand 03.01.2008
[223] Vgl. eSVG Plug-in 2007 (Online-Quelle)

statischen Elemente und verfügt über eine Script Engine für ECMAScript, um Interaktivität zu gewährleisten. Im Bereich der Animationen soll die fast vollständig unterstützte Spezifikation noch weiter ausgebaut werden. Das Toolkit steht zur kostenfreien Nutzung zur Verfügung[224].

Seit die erste offizielle Spezifikation des SVG-Standards 2001 auf den Weg gebracht wurde, ist der Markt an SVG-Editierwerkzeugen enorm gewachsen. Das Bearbeiten von SVG-Grafiken ist in dieser Arbeit allerdings nur Mittel zum Zweck. Der Schwerpunkt der Editorentwicklung liegt vielmehr darin, eine Entwicklungsumgebung zu schaffen, die auf flexible Weise Diagnosemasken erstellen und bearbeiten kann. Deshalb sei an dieser Stelle nur auf die Vielfalt der auf dem Markt befindlichen SVG-Werkzeuge hingewiesen. Für das Verarbeiten des SVG-Formats wird in dieser Arbeit eine kostenfreie Implementierung, basierend auf dem Batik Toolkit, genutzt. Diese wird in den Editor integriert und für die Verarbeitung der Diagnosemasken entsprechend erweitert. Auf die dahinterliegende Programmierleistung wird in Kapitel 10 näher eingegangen.

Neben dem Bearbeiten der SVG-Grafiken muss der Editor noch weitere Eigenschaften erfüllen, um den Anforderungen der Arbeit gerecht zu werden. Die Plattform des Editors, mit der dies bewerkstelligt werden soll, wird nun im folgenden Kapitel vorgestellt.

9.3.2 Bestimmung der Editor-Plattform

Für den Einsatz im Diagnosesystem muss der Editor die zentrale Aufgabe erfüllen, Diagnosemasken zu verarbeiten. Dafür werden der Editorumgebung verschiedenste Fähigkeiten abverlangt.

Im Zusammenhang mit dem Grafikformat wurden im letzen Kapitel bereits das Erzeugen, Anzeigen und Bearbeiten von SVG genannt. Ergänzt wird diese Forderung durch das Einbinden der Steuergerätedaten und deren Zuweisung zu grafischen Objekten (Diagnoseformen). Hierbei wird auch die flexible Erweiterung durch nutzerspezifische Bibliotheken gefordert. Darüber hinaus besteht der Anspruch, keine proprietäre Lösung zu schaffen, sondern sich offiziellen und verbreiteten Standards zu bedienen.

Deshalb muss eine Umgebung gewählt werden, in die das SVG-Format integriert werden kann, unter dem Aspekt ihrer Funktionalität flexibel erweiterbar ist und sich auf offizielle Standards stützt. Nach Möglichkeit sollte sie beim Kooperationspartner bereits im Einsatz sein bzw. deutlich verbreitet sein.

In den folgenden Unterkapiteln wird belegt, warum diese Anforderungen von dem Framework Eclipse erfüllt werden.

9.3.2.1 Konzept von Eclipse

Eclipse wurde in erster Linie als Java-Entwicklungsumgebung populär. Hinter dem Konzept steckt aber grundsätzlich mehr. Die Erfinder bezeichnen ihr Produkt als eine universelle Plattform zur Herstellung von Werkzeugen[225].

Bis zur Version 2.1 ist Eclipse als erweiterbare IDE (integrierte Entwicklungsumgebung) konzipiert gewesen. Eine integrierte Entwicklungsumgebung ist ein Programm zur Entwicklung

[224] Vgl. Batik SVG Toolkit 2008 (Online-Quelle)
[225] Vgl. Steppan 2003, S. 51

von Software. Die Abkürzung IDE steht für Integrated Development Environment. Ein solche Umgebung verfügt in der Regel über Standardkomponenten wie Texteditor, Compiler (Übersetzer), Debugger (Fehlersuchwerkzeug) und teilweise auch über Modellierungstools oder Versionsverwaltungssysteme[226].

Mit dem Begriff "erweiterbare" IDE möchte Eclipse darauf hinweisen, dass die Standardkomponenten um zusätzliche Komponenten ergänzt werden können. Diese Komponenten zur Erweiterung der Funktionalität werden als Eclipse-Plug-ins bezeichnet.

Bei Eclipse wurde das Plug-in-Konzept ins Extreme getrieben[227]. Im Gegensatz zu herkömmlichen IDEs ist Eclipse keine monolithische (nach außen geschlossene) Anwendung. Man entwarf stattdessen eine Architektur, bei der die gesamte Umgebung nur auf einem kleinen Kern basiert, den man durch Plug-ins erweitern kann. Deshalb lässt sich Eclipse besser als andere IDEs (JBuilder, .Net Studio etc.) an individuelle Verwendungszwecke durch Hinzufügen oder Entfernen von Modulen anpassen. Das bietet eine Flexibilität, die es vor Eclipse bei IDEs eigentlich nicht gab. Eclipse bietet für diese Arbeit demnach eine sehr gute Basis, um darauf den Editor für ein Diagnosesystem aufzusetzen.

Dies stellt keine Zweckentfremdung der IDE dar, sondern einen gewollten Trend der Hersteller, wie die verschiedensten Einsatzgebiete von Eclipse zeigen.

9.3.2.2 Einsatz von Eclipse

Es gibt bereits zahlreiche Beispiele, in denen Eclipse nicht nur zur Entwicklung von Javaprogrammen eingesetzt wird. Die Entwicklung der Eclipse-Plug-ins hat sich in den letzten Jahren enorm gesteigert[228]. Hierunter fallen z. B. Plug-ins zur Nutzung als Zeichenprogramm, strategisches Modellierungstool, Datenbank-Werkzeug, Nachrichten/Informationssystem und sogar Spiele wurden speziell für Eclipse entwickelt. Diese Entwicklung liegt im vollsten Interesse der Eclipse Foundation, denn diese sieht ihr Gemeinschaftsprodukt als "eine Plattform für alles Mögliche und nichts im Besonderen"[229].

Seit das Projekt 2004 offiziell von IBM ausgegliedert ist, nimmt auch die Mitgliederzahl der Eclipse Foundation stetig zu. Derzeit sind es über 150 teils namhafte Firmenmitglieder. Unter den strategischen Partnern befinden sich z. B. Firmen wie BEA, Borland, Hewlett Packard, Nokia, Intel und SAP[230].

Die Popularität zeigt sich auch in den Downloadzahlen. Die Eclipse-Version 3.0.2 wurde nach ihrem Erscheinen binnen 60 Tagen über eine Million Mal heruntergeladen. Seit Beginn verzeichnet die Eclipse Foundation insgesamt ca. 50 Millionen Downloads (Stand: 06/2007).

9.3.2.3 Gründe für die Auswahl von Eclipse

Aufgrund des wachsenden Bekanntheitsgrads und des erweiterbaren Komponentenkonzepts können mit Plug-ins von Drittanbietern Synergieeffekte erzeugt werden. Im Bereich der Verarbeitung des SVG-Formats wurde dieser Vorteil genutzt. Bei der Entwicklung des Editors steht der Einbau der Diagnosefähigkeit im Zentrum des Interesses. Die bloße Verarbeitung des

[226] Vgl. Irlbeck 2001, S. 421
[227] Vgl. Steppan 2003, S. 51
[228] Vgl. Widder 2003, S. 82
[229] Daum 2006, S. 2
[230] Vgl. Daum 2006, S. 1

SVG-Formats soll nach Möglichkeit nicht neu implementiert werden (siehe Kapitel 9.3.1.5). Mit dem Eclipse-Konzept bietet sich nun der Vorteil, die Bearbeitung des SVG-Formats durch ein bereits bestehendes Plug-in erledigen zu lassen. In Kapitel 9.3.1.5 wurde auf ein Batik Toolkit gestütztes Plug-in namens "SVGWF Editor" von Openvue verwiesen.

Der Vorteil des Synergieeffekts durch Plug-ins wirkt sich natürlich auch in die andere Richtung aus. Nach dem Abschluss der Arbeit können andere Anbieter auf das erstellte Plug-in dieses Diagnosesystems zugreifen und auf dessen Funktionalität aufsetzen. Dadurch wird die Wahrscheinlichkeit gesteigert, dass das Konzept über die Grenzen der Arbeit hinaus weiter genutzt wird.

Nicht zuletzt aus diesen Gründen zählt die Entwicklungsumgebung auch beim Kooperationspartner zu den häufig genutzten Umgebungen. Sie befindet sich auf der Liste von Programmen, die in IT-Projekten eingesetzt werden dürfen und vorzugsweise verwendet werden sollen. Dies erhöht für das neue Diagnosesystem die Chance auf Akzeptanz bei den Nutzern.

9.3.3 Bestimmung der Viewer-Plattform

Die Eclipse-Umgebung hätte aufgrund ihrer Flexibilität auch eine gewisse Eignung für die Viewer-Software. Allerdings beansprucht die Eclipse-Umgebung zu viele Ressourcen und ist auch nicht für mobile Endgeräte konzipiert. Dieser Anspruch stellt sich zwar für den Einsatz in der Verwaltung nicht, wohl aber für die Diagnosetätigkeit in der Nacharbeit oder Analyse.

Für den Viewer gilt es folglich, eine Plattform zu finden, die sowohl den mobilen Einsatz unterstützt als auch auf Arbeitsplatzrechnern lauffähig ist. Des Weiteren besteht für diese Plattform natürlich auch der Anspruch das SVG-Format verarbeiten zu können.

Die Wahl fiel hier auf die .Net-Umgebung von Microsoft.

9.3.3.1 Ziele der .Net-Plattform

.Net ist die aktuelle Softwareentwicklungsumgebung der Firma Microsoft. Sie wurde zum ersten Mal im Sommer 2000 vom Firmengründer Bill Gates vorgestellt. Mit .Net wollte Microsoft zu der sich immer stärker verbreitenden Java-Technologie des Rivalen Sun in Konkurrenz treten. Deshalb wurden die Vorteile von Java, wie überwiegende Plattformunabhängigkeit und konsequente Objektorientierung, als Kernziele bei der Entwicklung verfolgt. Darüber hinaus wurden weitere Ziele gesteckt, die Microsoft einen entscheidenden Vorsprung verschaffen sollten[231]. Eines dieser Ziele war z. B. die Unabhängigkeit von einer Programmiersprache. Während die Java-Umgebung zur Entwicklung nur die Programmiersprache Java anbietet, sind es bei .Net unter anderem die Programmiersprachen C#, VB und Visual C++. Darüber hinaus soll mit .Net auf dem mobilen Kleingerätemarkt Fuß gefasst werden.

9.3.3.2 Konzept des .Net Frameworks

Um Programme, die mit der .Net-Technologie erstellt wurden, starten zu können, wird eine sogenannte Laufzeitumgebung benötigt.

Eclipse benötigt hierfür die Laufzeitumgebung Java Virtual Machine (JVM). .Net-Anwendungen hingegen brauchen das .Net Framework. Dies ist grundsätzlich eine Sammlung

[231] Vgl. Patt 2002, S. 5-7

von Klassen und deren Methoden, die eine Grundfunktionalität bereitstellen, damit die Anwendungen ausgeführt werden können.

Die Laufzeitumgebung trägt bei Microsoft den Namen Common Language Runtime (CLR). Die CLR ist eine Art Software-basierter Prozessor, der die eigentliche Laufzeitleistung vollbringt und das Programm abarbeitet. Die Laufzeitumgebung .Net Framework muss folglich auf jedem Gerät installiert sein, auf dem die .Net-Anwendung gestartet werden soll[232].

Da die Laufzeitumgebungen an Arbeitsplatz-PCs angepasst sind, können sie in dieser Form nicht auf mobilen Geräten installiert werden. Die derzeit auf dem Markt zur Verfügung stehenden PDAs würden den Anforderungen an Speicherplatz, Prozessorgeschwindigkeit und Arbeitsspeicher nicht gerecht werden. Deshalb hat Microsoft eine Laufzeitumgebung speziell für Kleingeräte entwickelt. Um Speicherplatz zu sparen, wurde ein funktional kompakteres Angebot geschaffen, indem Klassen und Methoden reduziert wurden. Daher der Name .Net Compact Framework.

Die Version 1.0 des CF (Compact Frameworks) wurde 2003 veröffentlicht und macht ca. 12% der vollen .Net Framework-Spezifikation aus. Aufgrund der limitierten Ressourcen von Kleingerät-Hardware wurde versucht, einen Kompromiss zwischen dem Speicherbedarf und der Notwendigkeit von Funktionen für ein solches Gerät zu finden. Besonders im Bereich der Einbindung von Komponenten, Datenbank-Servern und Web Technik wurde Funktionalität eingespart[233].

9.3.3.3 Gründe für die Auswahl von .Net

Eine Philosophie, die Microsoft mit der .Net-Plattform verfolgt ist, dass Nutzer "jederzeit, überall und auf jedem Gerät von der […] Funktionsvielfalt profitieren können"[234]. Dieser Gedanke deckt sich auch mit dem Verbesserungsansatz dieser Arbeit, das Diagnosesystem soll auf mobilen und Terminalgeräten verfügbar gemacht werden. Deshalb bietet die .Net-Technologie einige Vorteile, die den Einsatz für diese Arbeit begünstigen.

Der entscheidende Vorteil ist, dass geschriebener Code zwischen dem .Net Framework und Compact Framework nahezu ausgetauscht werden kann. Das bedeutet im engeren Sinne, dass der Code einer Desktop-Anwendung für die mobile Anwendung wieder verwendbar ist. Hierdurch wird die Effizienz im Rahmen der Anwendungsentwicklung erheblich gesteigert, sowie die Look & Feel-Gleichheit zwischen den Endgeräten gefördert.

Dieser Aspekt erweist sicht im Hinblick auf die Neuentwicklung des Diagnosesystems als grundlegender Vorteil, denn die Architektur fordert die Bereitstellung von zwei Viewern: Einen für den mobilen (Mobile Viewer) und einen für den Desktop-Bereich (Desktop Viewer). Die Viewer sollen auf beiden Endgeräten gleiche Funktionalität bieten und sich auch auf gleiche Weise bedienen lassen. Je höher der Anteil an gemeinsam genutztem Code ist, desto größer ist auch die Wahrscheinlichkeit, dass die Anwendung das gleiche Verhalten zeigt. Es reduziert außerdem den Wartungs- und Korrekturaufwand und die Fehlerhäufigkeit. Ein entdeckter Fehler kann parallel in beiden Anwendungen behoben werden.

Des Weiteren bietet das .Net Compact Framework eine zuverlässige und sichere Umgebung für die Ausführung der Applikation. Die Laufzeitumgebung stellt sicher, dass ein Fehler einer

[232] Vgl. Vasters 2001, S. 10-12
[233] Vgl. Wigley 2003, S. 14
[234] MSDN Deutschland 2004 (Online-Quelle)

Anwendung nicht zum Absturz des Geräts führen kann. Gleichzeitig garantiert das Sicherheitsmodell, dass bösartigem Code der Zugriff auf sichere Systemressourcen nicht gestattet wird.

Im Anwendungsbereich der Diagnose von Kraftfahrzeugen muss eine solche Sicherheit gewährleistet sein, da bei der Diagnose auch kritische Bereiche der Fahrsicherheit untersucht werden müssen. Der Schutz gegen Absturz des Geräts ist also für die Qualität des Diagnoseprozesses und die Sicherheit des Diagnostikers erforderlich.

Außerdem verspricht das .Net Compact Framework eine hohe Leistungsperformance, da es speziell für Geräte mit begrenzten Ressourcen konzipiert wurde. Das Framework selbst arbeitet sehr speichereffizient und gibt Systemressourcen von Anwendungen zurück, sobald sie nicht länger benötigt werden. Dies wird unter anderem durch den integrierten JIT-Compiler (Just-in-Time) erreicht. Anwendungen, die im .NET Compact Framework ausgeführt werden, werden als systemeigener Code ausgeführt, der vom JIT zur Laufzeit erstellt wird. Die JIT-Technologie sorgt somit für eine höhere Leistung bei der Codeausführung.

Auch versucht Microsoft mit der .Net-Technologie Standardisierung durch die Unterstützung vereinheitlichter Schnittstellen zu fördern. So wird z. B. die Verarbeitung des offenen W3C-Standards XML nativ durch .Net unterstützt. Wie in Kapitel 9.3.1 erläutert, basiert auch das in der Arbeit verwendete Grafikformat SVG auf XML.

9.3.3.4 Herausforderungen der SVG Integration in das Compact Framework

Trotz der XML-Unterstützung von .Net muss für die grafische Darstellung des SVG-Formats ein entsprechender Treiber eingebunden werden. In Kapitel 9.3.1 wurde der Treiber (Plug-in) mit dem das SVG-Format im Viewer verarbeitet werden soll, bereits vorgestellt. Das ausgewählte eSVG-Plug-in bietet die Möglichkeit, SVG-Bilder grafisch darzustellen. Treiber, wie auch das eSVG-Plug-in, die eine grafische Ausgabe anbieten, sind für Microsoft Betriebssysteme meist als ActiveX-Komponenten realisiert.

ActiveX ist eine entsprechende Erweiterung des COM-Konzepts von Microsoft. COM ist eine Software-Schnittstelle, über die Software-Komponenten Daten miteinander austauschen. COM-Komponenten werden normalerweise in Dynamic-Link-Libraries (DLL)-Dateien transportiert. Die Technologie COM wurde proprietär entwickelt, um die Verteilung von Programmfunktionalität zu erleichtern. ActiveX-Komponenten sind meist kleinere Anwendungen, die dazu dienen, in Programmen einen abgegrenzten funktionalen Bereich zu übernehmen, wie z. B. das Darstellen eines Grafikformats[235].

In dem .Net Framework können ActiveX-Elemente problemlos eingebunden werden. Deren Integration und Ausgabe wird vollständig unterstützt. Im Compact Framework von .Net trifft man hier allerdings auf eine Einschränkung, die nicht nur im Rahmen dieser Arbeit eine Herausforderung darstellt und zu zusätzlichem Programmieraufwand führt. Dies ist die fehlende Unterstützung von ActiveX-Komponenten. Die Funktionalität wurde zugunsten der Ressourcenbeanspruchung bei dem Compact Framework eingespart (siehe Kapitel 9.3.3.2).

Der Zugriff auf COM- oder ActiveX-Komponenten muss deshalb durch eine Hülle gekapselt werden. Die Hülle wird programmiertechnisch als Wrapper bezeichnet. .Net enthält für die Entwicklung solcher ActiveX-Wrapper keinen Codegenerator. Die Wrapper für die ActiveX-

[235] Vgl. Yao, S. 5

Komponente müssen folglich selbst entwickelt werden. Fallstricke und zeitaufwendig sind hierbei die recht unverständliche Syntax bei der Entwicklung des Wrappers sowie die Abbildung der ActiveX-Daten auf .NET-Datentypen[236].

Wie dieser Herausforderung bei der Entwicklung des Viewers begegnet wird, ist Inhalt des Kapitels 10. In diesem Kapitel wird die Umsetzung der ausgewählten Basistechnologien (Eclipse-Plattform, .Net-Plattform und SVG-Format) zu einem lauffähigen Diagnosesystem beschrieben.

[236] Vgl. Yao, S. 187

10 Umsetzung des Entwurfs

Die Entwicklung von Software ist mehr als eine reine Programmieraufgabe. Die Anforderungen des Benutzers sowie alle fachlichen und technischen Rahmenbedingungen müssen erkannt und berücksichtigt werden. Die Zusammenführung all dieser Informationen mündet in die Softwarearchitektur[237]. Die Softwarearchitektur legt die wesentlichen Strukturen eines Softwaresystems fest und ist die Grundlage für die Umsetzung[238].

In Kapitel 9 wurden das Zusammenspiel der zentralen Softwarekomponenten und die ihnen zugrunde liegenden Technologien erläutert. Das Kapitel befasste sich mit der Makroarchitektur.

Um diese in die Implementierung zu übertragen, ist es notwendig, die wesentlichen Mechanismen der einzelnen Softwarekomponenten zu erfassen. Dies erfolgt mit der Mikro- bzw. technischen Architektur.

Das mit dieser Arbeit entwickelte Diagnosesystem gliedert sich in die zwei Softwarekomponenten Editor und Viewer auf. Für beide Komponenten wird jeweils die technische Architektur erläutert. Die technische Architektur einer Anwendersoftware wird üblicherweise anhand von logischen Softwareschichten beschrieben. Dies ist Gegenstand von Kapitel 10.1. Ergänzt wird die Erläuterung durch die Beschreibung der physischen Verteilung der Komponenten[239]. Die Ausführungen hierzu erfolgen in Kapitel 10.2.

Abschließend werden in Kapitel 10.3 und 10.4 die Schwerpunkte der Implementierung aufgezeigt. In Kapitel 10.3 erfolgt dies für die Softwarekomponente Editor und in Kapitel 10.4 für den Viewer. Für das Verständnis dieser beiden Kapitel sind tiefere Kenntnisse in den Programmiersprachen Java, C#, JavaScript und XML sowie Kenntnisse der Eclipse- und .Net-Architektur empfohlen.

Zielsetzung des Kapitels 10 insgesamt ist es, anhand der Erläuterung der Schwerpunkte zu belegen, dass das Konzept softwaretechnisch umsetzbar ist. Das Ergebnis der Umsetzung in Form einer Prototypentwicklung wird anschließend in Kapitel 11 im Rahmen der Projektierung vorgestellt.

---

[237] Vgl. Forbrig 2004, S. 48
[238] Vgl. Dunkel 2003, S. 1
[239] Vgl. Dunkel 2003, S. 15-16

10.1 Logische Softwareschichten

Aus der Abbildung der Makroarchitektur in Kapitel 9.2 gehen zwei elementare Softwarekomponenten hervor: der Editor und der Viewer. Softwarekomponenten, die direkt vom Anwender bedient werden können, werden auch als Anwendersoftware bezeichnet. In Abbildung 9.1 ist zu erkennen, dass der Diagnostiker den Viewer und der Verwaltungsmitarbeiter den Editor bedient. Für die Software-basierte Umsetzung müssen Editor und Viewer technisch tiefgehend strukturiert werden. Die Basistechnologien hierfür wurden bereits in Kapitel 9.3 erläutert.

In der Softwarearchitektur bieten sich im Allgemeinen drei Schichten an, um die grundsätzlichen Aufgaben und Funktionen von Anwendersoftware wiederzugeben[240]:

- Präsentationsschicht
- Anwendungsschicht
- Persistenzschicht

Die Präsentationsschicht stellt die Benutzeroberfläche dar. Sie dient dazu, Informationen für den Benutzer abzubilden und auf dessen Eingaben zu reagieren (Interaktion). In dem vorliegenden System fällt unter diesen Bereich die Oberfläche des Viewers und des Editors.

Um die Informationen, die auf der Oberfläche angezeigt werden sollen, zu beschaffen und diese aufzubereiten, ist unterhalb der Präsentationsschicht die Anwendungsschicht angesiedelt. Sie versorgt die Oberflächenschicht mit Daten für die Anzeige und verarbeitet die Aktionen des Benutzers. Auf dieser Ebene befindet sicht, die fachliche Logik des Programms angesiedelt. In der Anwendungsschicht wird demnach die komplette fachliche Funktionalität der Applikation realisiert. Im Fall des Diagnosesystems ist dies hauptsächlich die Verarbeitung des Grafikformats und der Steuergerätedaten.

Die Persistenzschicht stellt Dienste bereit, um die dauerhaft gespeicherten (persistenten) Daten zu verwalten. Damit sind vorrangig Daten gemeint, die in einem Datenbanksystem gespeichert sind. Das Diagnosesystem ist allerdings eine Stand-alone-Anwendung. Das Softwaresystem ist dementsprechend autonom auf einem Rechner ohne Verbindung zu einem Server, wie z. B. einem Datenbanksystem, lauffähig. Deshalb beschränkt sich die Beschreibung der Persistenzschicht des Editors und des Viewers auf die Verarbeitung von Dateien (Speichern/Lesen). Darunter fallen vorrangig das Diagnoseskript, die Sprachdateien und die Bibliotheken mit den Diagnoseformen.

In den folgenden Kapiteln 10.1.1 und 10.1.2 werden die einzelnen Softwareschichten für den Editor und für den Viewer erläutert.

[240] Vgl. Dunkel 2003, S. 17-19

10.1.1 Editor

10.1.1.1 Übersicht

Seit der ersten Version verfolgt Eclipse das Konzept einer reinen Plug-in Architektur. Das bedeutet, dass nicht nur Funktionen über Plug-ins erweitert werden können, sondern praktisch jeder funktionale Bereich in Eclipse selbst ein Plug-in darstellt. Dieser streng komponentenbasierte Ansatz bietet größtmögliche Flexibilität für die Erstellung eigener Anwendungen[241].

Bei der Erstellung eines eigenen Plug-ins kann auf die Funktionen bestehender Plug-ins zugegriffen werden. Für ein Plug-in, das die IDE erweitern soll, sind hier insbesonders drei Bereiche relevant: die Laufzeitumgebung, die Ressourcenverwaltung des Eclipse Workspace (Projekte, Verzeichnisse und Dateien) und die Arbeitsfläche des Benutzers (Workbench). Die Abbildung der technischen Architektur zeigt, wie diese drei Bereiche die Eclipse-Plattform dominieren und wie sie sich über alle drei Softwareschichten verteilen:

[241] Vgl. Daum 2006, S. 364

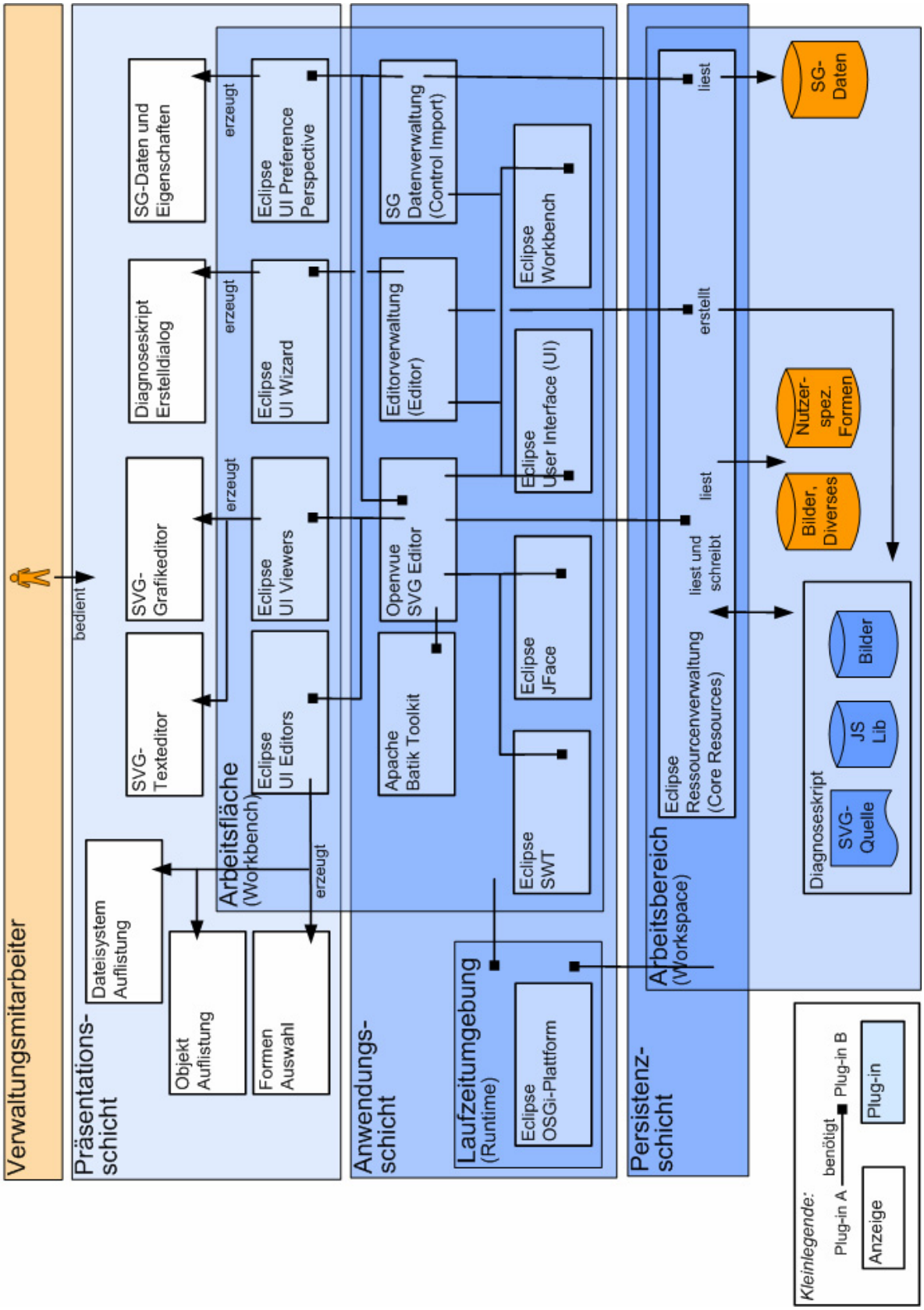


Abbildung 10.1 Technische Architektur des Editors

Den Kern der Plug-in-Architektur bildet die Laufzeitumgebung. Das Plug-in OSGi-Plattform stellt den Ablaufkern dar und ist das einzige Plug-in, das in jeder Eclipse-Anwendung vorhanden sein muss. Es läuft vollständig autark und benötigt keine anderen Plug-ins.

Die Plug-ins der Bereiche Workbench und Workspace müssen sich, um lauffähig zu sein, in dieses Plug-in einhängen. Das Anknüpfen erfolgt über Erweiterungspunkte (Extension Points). Der Anschluss an einen Extension Point wird in der Abbildung durch ein Quadrat am Pfeilende simuliert. Plug-ins können beliebig viele Erweiterungspunkte zur Verfügung stellen und so ihre Funktionalität für andere Plug-ins der Plattform zur Verfügung stellen. Durch dieses Konzept ergibt sich der in Kapitel 9.3 beschriebene Synergieeffekt, der unter anderem zu dem Entschluss für Eclipse geführt hat.

10.1.1.2 Präsentationsschicht

Für die Realisierung einer Benutzeroberfläche stellt Eclipse standardmäßig mehrere Plug-ins zur Verfügung. Diese sind in der Regel von dem User Interface (Benutzerschnittstellen)-Plug-in abgeleitet. Die in der Architektur dargestellten Plug-ins stellen nur eine kleine Auswahl der Möglichkeiten dar. Es steht eine Ansammlung von verschiedenen Editoren, Anzeigeflächen oder Formularen mit Eclipse zur Verfügung[242].

Die vom User Interface (UI) abgeleiteten Plug-ins stellen ihren Inhalt in Fenstern dar. Die Fenster können vom Benutzer beliebig eingeblendet und in ihrer Größe und Position verschoben werden.

Ein Plug-in, das von dem Plug-in UI ableitet, ist das Plug-in Wizard. Mit diesem Plug-in können Formularabfolgen gestaltet werden. Für den Editor des Diagnosesystems wurde dieses Plug-in genutzt, um ein Dialogfenster zum Anlegen eines neuen Projekts zu erstellen. Mit dem Dialog kann ein Projekt mit den zugehörigen Verzeichnissen, Bibliotheken und einem Diagnoseskript angelegt werden. Dieses Formular stellt somit den Einstieg in die Arbeit mit dem Editor dar.

Nach dem Anlegen und Öffnen des Diagnoseskripts stehen zum Bearbeiten des Diagnoseskripts zwei Editoren zur Verfügung. Beide Editoren sind in einem Fenster integriert und sind von dem Eclipse-Plug-in Editor abgeleitet. Dieses beruht im Kern auch auf dem Plug-in UI. Per Reiter (Tabs) kann zwischen den Editoransichten gewechselt werden. Eine Ansicht dient zum grafischen Bearbeiten des SVG-Bildes und die andere zum manuellen Bearbeiten des Quellcodes. Eine Änderung in einer Editoransicht führt dazu, dass beim Wechsel zur anderen Editoransicht die Änderung sofort übernommen und angezeigt wird.

Neben dem Editorfenster stellt die Editorumgebung dem Benutzer noch weitere Auflistungen und Auswahlfenster zur Verfügung. Diese sind über das Plug-in Viewer realisiert. Die Eclipse IDE bietet z. B. ein Dateiexplorer-Fenster an. Damit können die Diagnoseskripte, Bilder oder andere Dateien per Doppelklick geöffnet werden.

In ein geöffnetes Skript können über das grafische Editorfenster Diagnoseformen (Objekte) hinzugefügt werden. Die Objekte können über ein Formenauswahl-Fenster in das Diagnoseskript per Drag & Drop geschoben werden. In Kapitel 11.2.1.3 liefert hierzu einen Screenshot. In dem Auswahlfenster werden sowohl die Standardformen der Diagnose und durch das Pro-

[242] Vgl. Daum 2006, S. 366

jekt ergänzte nutzerspezifische Formen angezeigt. In Kapitel 10.4 wird auf die Hintergründe der nutzerspezifischen Formen noch weiter eingegangen.

Die in ein Skript eingefügten Objekte werden in einem weiteren Fenster aufgelistet. Durch Klick auf eines der Objekte zeigt das Eigenschaftsfenster deren Daten an, wie z. B. Größe, Position oder formenspezifische Werte.

Die formenspezifischen Schlüsselwerte für den Diagnoseeditor sind die Steuergerätezugriffsdaten. Die Zuweisung der Schlüsselwerte, wie z. B. des Steuergerätebefehls, zu einer Diagnoseform erfolgt per Auswahlbox über das Eigenschaftsfenster. Welche Auswirkungen diese Zuweisung hat und wie mit der Zuweisung die Informationen im Diagnoseskript eingetragen werden wird in Kapitel 10.1.1.3 mit der Anwendungsschicht erläutert.

Um die Steuergerätedaten für die Eigenschaftszuweisung zur Verfügung zu stellen, müssen sie allerdings zuvor über die Editoreinstellungen importiert werden. Für die Darstellung von Einstellungsseiten bietet die Workbench von Eclipse ein Plug-in Preference Perspective. Mittels dieses Plug-ins kann in dem Menü Einstellungen des Editors eine Seite zum Laden der Steuergerätedaten eingebunden werden.

Für den physischen Zugriff auf das Diagnoseskript oder die Verarbeitung von Steuergerätedaten arbeiten die User Interface-Plug-ins mit Plug-ins aus den darunter liegenden Softwareschichten zusammen. Eine dieser Schichten ist die Anwendungsschicht.

10.1.1.3 Anwendungsschicht

Auf die zentrale Stellung des Laufzeit-Plug-ins wurde bereits verwiesen. Das Plug-in OSGi liefert die Basis zur Verwaltung der Dienste bzw. Komponenten in Eclipse. Das Plug-in ist eine Entwicklung der Open Services Gateway Initiative[243]. Ergänzt durch das Plug-in Runtime bildet es den Kern für jede Eclipse-Entwicklungsumgebung und somit den Ausgangspunkt für die Anwendungsschicht. Jedes der vier elementaren Workbench-Plug-ins benötigt den direkten Zugriff auf das Plug-in Runtime. Diese vier Plug-ins sind das Standard Widget Toolkit (SWT), JFace, User Interface und Workbench Plug-in.

Die Notwendigkeit des User Interface-Plug-ins wurde in der Präsentationsschicht bereits deutlich. Es stellt das Grundgerüst für alle weiteren UI-Plug-ins dar, mit denen Benutzeroberflächen realisiert werden.

Das Plug-in Workbench übernimmt die Verwaltung der Fensterarchitektur auf der Benutzeroberfläche. Mit dem Plug-in wird der Editorumgebung unter anderem die Möglichkeit gegeben, innerhalb der Arbeitsfläche verschiedene Fenster anzuzeigen und diese in weitere Reiter aufzuteilen[244].

Auf Basis des SWT-Plug-ins können die Oberflächen mit grafischen Elementen versorgt werden. Durch die enge Anbindung an das Betriebssystem wird durch das SWT das Look & Feel der Anwendungen dem des Betriebssystems sehr ähnlich. Trotz der Betriebssystemabhängigkeit von SWT ist Eclipse auf gängigen Betriebssystemen wie Microsoft, Linux oder MAC OS lauffähig[245].

[243] Vgl. Daum 2006, S. 365
[244] Vgl. Daum 2006, S. 411
[245] Vgl. Daum 2006, S. 165

JFace setzt auf SWT auf und ist eine Erweiterung zur Darstellung von komplexeren Benutzeroberflächen. So bildet JFace z. B. die Basis für Dialogformulare[246]. Um beispielsweise den Erstelldialog für das Anlegen eines Diagnoseskripts zu ermöglichen, benötigt das Eclipse-Plug-in Wizard das JFace-Plug-in.

Die Verzahnung mit den vier elementaren Workbench-Plug-ins und deren Zugriffe untereinander wurden aufgrund des Komplexitätsgrads in der Grafik ausgespart. Ziel der Abbildung ist es, die Eingliederung der diagnoserelevanten Plug-ins in die technische Architektur von Eclipse zu zeigen. Diese Plug-ins sind der SVG-Editor von Openvue und die eigenentwickelten Plug-ins zur Editorverwaltung und Steuergerätedatenverwaltung.

Das Plug-in Editorverwaltung erzeugt die Dialogformulare, mit denen der Benutzer die Diagnoseskripte anlegen kann. Durch das Öffnen eines angelegten Diagnoseskripts wird wiederum das SVG-Editor-Plug-in aktiviert.

Das SVG-Editor-Plug-in stellt die komplette Verarbeitungslogik für das SVG-Format dar. Es reagiert auf Veränderungen auf der Zeichenoberfläche (Editorfenster), wie z. B. das Einfügen einer Diagnoseform oder dessen grafische Bearbeitung. Es wandelt diese Bearbeitungsaktionen in SVG-Quellcode um und ändert im gleichen Zuge den Quellcode des Skripts. Für die Generierung des SVG-Quellcodes und dessen Anzeige bedient sich das SVG-Editor-Plug-in des Batik Toolkit der Apache Foundation. Apache hat diese fast vollständige Implementierung des SVG-Standards entwickelt, um Programmen wie Eclipse- oder Web-Anwendungen die Möglichkeit zu geben SVG-Grafiken zu verarbeiten.

Des Weiteren verwaltet das SVG-Editor-Plug-in die Anzeige der Diagnoseformen und deren Eigenschaften sobald sie einer Diagnosemaske hinzugefügt wurden. Dafür ist die Interaktion mit dem Eclipse-Viewer-Plug-in notwendig.

Die entscheidende Erweiterung der Editorfunktionalität, um im Rahmen der Diagnose eingesetzt werden zu können, ist die Zusammenführung des SVG-Plug-in mit dem Plug-in zur Steuergerätedatenverwaltung (Control-Import-Plug-in). Dadurch erhält der Editor die Funktionalität Steuergerätebefehle anzuzeigen und diese einer Diagnoseform zuzuweisen. Erst durch das Einfügen einer Diagnoseform in ein Diagnoseskript und die Verankerung eines Steuergerätebefehls in einer solchen Form erlangt das Skript die Fähigkeit zur Steuergerätediagnose. Wie dieser Kernbereich softwaretechnisch gelöst wurde, wird in Kapitel 10.3 beschrieben.

Um die Steuergerätebefehle anzuzeigen, lädt das Plug-in Control Import die Dateien, die in dem Menü Einstellungen dafür angegeben wurden (siehe Kapitel 10.1.1.2 Präsentationsschicht,). Die Dateien sind im XML-Format gespeichert und enthalten die Steuergerätedaten. Sie können variabel an das Basissystem des Kfz-Herstellers angepasst werden und stellen somit eine offene Schnittstelle dar. Der Aufbau der XML-Datei wird in Kapitel 10.3.2 im Rahmen der Implementierung näher erläutert.

Der Zugriff auf Dateien, wie z. B. die XML-Steuergerätedatei oder das Diagnoseskript, wird in der Eclipse IDE über den Workspace-Bereich gelöst. In der technischen Architektur findet sich der Workspace in der Persistenzschicht wieder.

[246] Vgl. Daum 2006, S. 243

10.1.1.4 Persistenzschicht

Die Persistenzschicht sorgt dafür, dass Daten dauerhaft gespeichert und auch wieder geladen werden können. Diese Aufgabe fällt bei der Arbeit innerhalb einer Eclipse IDE unter den Bereich des Workspace bzw. des Eclipse-Plug-ins Ressource.

Innerhalb eines Eclipse-Projekts werden Verzeichnisse und Dateien als Ressourcen bezeichnet. Der Aufbau eines Projekts kann über ein Navigatons-Fenster betrachtet und bearbeitet werden (Dateisystem in Abbildung 10.1). Die Funktionalität stellt das Ressource-Plug-in zur Verfügung und zudem noch einige höhere Dienste, die es im Dateisystem des Betriebssystems normalerweise nicht gibt. Dazu gehören ein Mechanismus zum Verwalten von Anmerkungen oder eine Ereignisverarbeitung bei Ressourcenänderungen[247].

Die Ereignisverarbeitung macht es dem SVG-Editor-Plug-in möglich festzustellen wenn ein SVG-Bild zur Laufzeit außerhalb der Eclipse-Umgebung geändert wurde. Es fragt daraufhin ein Neuladen der Datei ab.

Da bei dem Diagnosesystem kein Datenbankzugriff vonnöten ist, beschränkt sich der Zugriff in der Persistenzschicht auf gespeicherte Dateien. Diese Dateien werden erstellt, gelesen und geändert.

Im Falle der Steuergerätedaten erfolgt ausschließlich ein Lesen der XML-Struktur. Hierfür lädt das Control-Import-Plug-in die in den Einstellungen ausgewählte XML-Datei.

Das erste Anlegen eines Diagnoseskripts wird über das Plug-in Editorverwaltung angestoßen. Gelesen und bearbeitet werden die Skripte mit den zugehörigen Dateien, wie Bildern oder JavaScript-Bibliotheken, durch das SVG-Editor-Plug-in. Nach dem Verarbeiten der Skripte bedient sich das SVG-Editor-Plug-in erneut dem Eclipse-Plug-in Ressource um das Skript zu speichern.

Fertiggestellte Diagnoseskripte können auf dem Viewer ausgeführt werden. Dazu werden die Dateien entweder direkt aus dem Workspace auf das Diagnosegerät aufgespielt oder in ein Serververzeichnis hochgeladen. Den Hochlademechanismus stellt das Editorverwaltung-Plug-in bereit. Dabei überträgt das Plug-in die Dateien aus dem Workspace in ein angegebenes Zielverzeichnis. Der Mechanismus unterstützt das FTP-Protokoll. Da der Upload optional ist und keine zentrale Kernfunktionalität darstellt, wurde er in der technischen Architekturabbildung nicht erfasst.

Die Verarbeitung eines erstellten Diagnoseskripts durch die Viewer-Software wird mit dem folgenden Kapitel 10.1.2 beschrieben.

10.1.2 Viewer

10.1.2.1 Übersicht

Bei der technischen Architektur ist es ratsam, sich nicht im Detail zu verlieren, sondern die Kernaussage des Schemas zu erhalten. Kernthema der Viewer-Funktionalität ist die Darstellung des Grafikformats SVG und der Zugriff auf das Steuergerät. Deshalb stehen diese Themen im Vordergrund der Abbildung.

[247] Vgl. Daum 20006, S. 366

Aufgabenbereiche, die nicht mit dieser Kernthematik in Verbindung stehen und über standardisierte Methoden des .Net Frameworks gelöst werden, werden aus Gründen der Übersichtlichkeit ausgespart. Dies trifft z. B. auf die Verwaltung der Mehrsprachigkeit zu. Die hier verwendete Technik des Ressource Managers ist auf der .Net-Plattform des Viewers (siehe Kapitel 9.3.3) eine verbreitete Möglichkeit zur Internationalisierung. Sie wird deshalb in der Abbildung nicht erfasst.

Die Abbildung gliedert sich neben der Aufteilung in die logischen Softwareschichten noch in die Bereiche mobil und Desktop auf. Die Abbildung wird beginnend mit der Diagnoseoberfläche (Präsentationsschicht) erläutert.

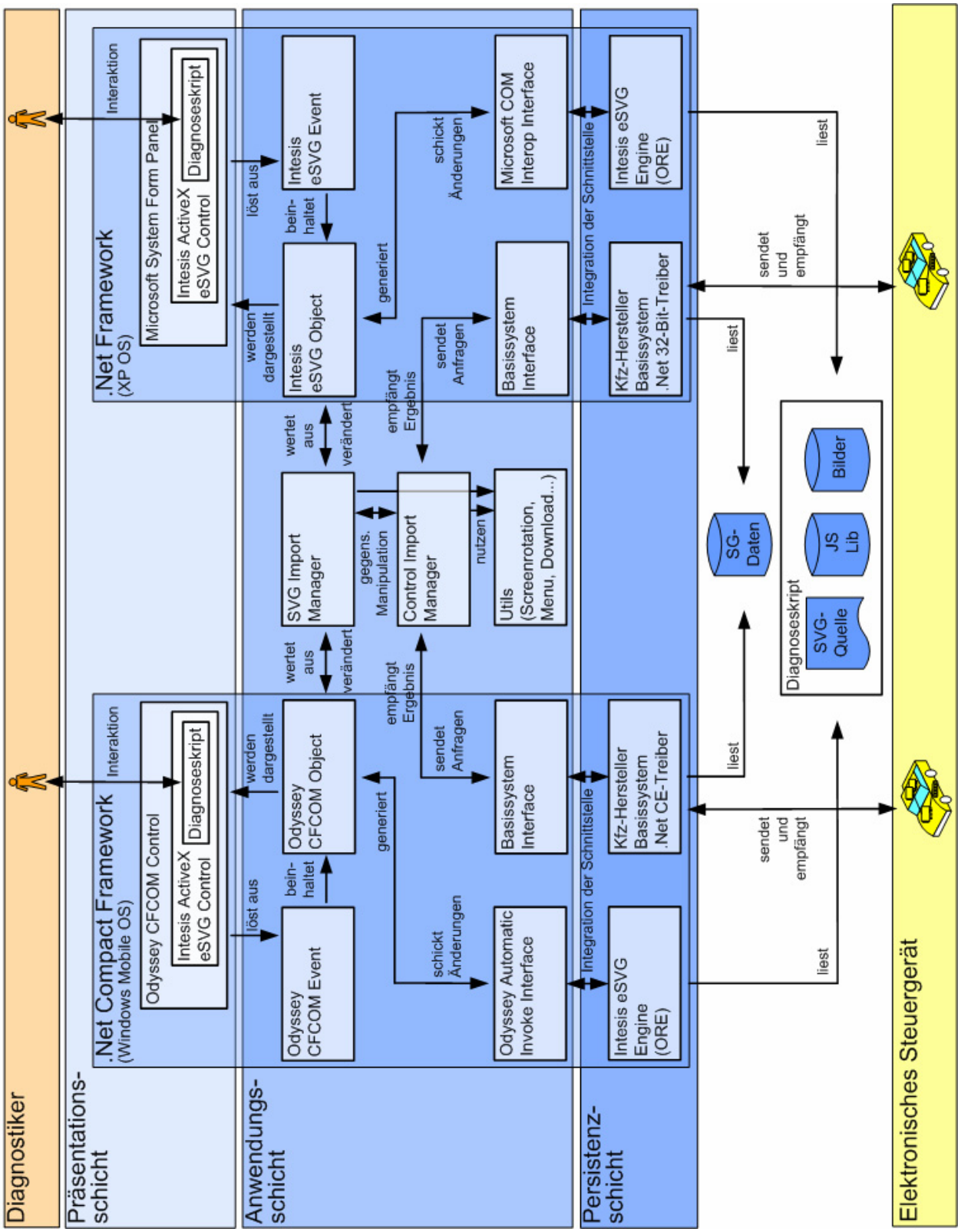


Abbildung 10.2 Technische Architektur des Viewers

10.1.2.2 Präsentationsschicht

Wie in Kapitel 9.3.3 festgelegt wurde, wird der Viewer auf Basis der .Net-Plattform entwickelt.

.Net bietet für die Entwicklung der Anwendung zwei Frameworks an. Die Laufzeitumgebung des Rahmenwerks dient zum Ausführen der Anwendung. Die Laufzeitumgebung ist unter anderem aus Gründen der Performanceoptimierung und technischen Realisierbarkeit endgerätspezifisch. Deshalb wird ein .Net Framework für Desktop-Anwendungen und ein .Net Compact Framework für mobile Anwendung zur Verfügung gestellt. Das .Net Framework ist unter anderem auf XP- und 2000er-Betriebsystemen lauffähig. Das Compact Framework benötigt Betriebssystem CE oder das Windows Mobile.

Aufgrund der getrennten Frameworks muss programmiertechnisch jeweils eine separate Oberfläche erstellt werden. Deshalb ist die Präsentationsschicht in der Abbildung aufgeteilt in die Oberfläche Desktop XP und in die Windows Mobile. Für den Benutzer bedeutet dies zwei Versionen der Viewer-Software: eine Desktop-Viewer-Software und eine Mobile-Viewer-Software. Intern können allerdings zentrale Softwareteile auf beiden Varianten genutzt werden. Dies wird im Verlauf der Beschreibung der Architektur noch verdeutlicht.

Die Viewer-Software erhält ihre Diagnosefunktionalität durch das Laden der Diagnoseskripte. Diese bestehen aus der SVG-Quelle, den zugehörigen JavaScript-Bibliotheken (mit den Diagnoseformen) und etwaigen Bildern. Um das Diagnoseskript zu laden muss ein SVG-Plug-in in die Viewer-Oberfläche integriert werden. Das ausgewählte Plug-in trägt den Namen eSVG Control von der Firma Intesis (siehe Kapitel 9.3.1.5). Dieses wird für Microsoft Betriebssysteme in Form eines ActiveX-Controls bereitgestellt. Die zugrunde liegende Technik der Controls wurde bei den Basistechnologien in Kapitel 9.3.3.5 erläutert.

Im .Net Framework gestaltet sich das Einbinden des ActiveX-Controls relativ problemlos. Es kann direkt in die Oberfläche des Viewers integriert werden. Die Integration kann z. B. in ein .Net-Panel-Objekt erfolgen. Ein Panel dient der Darstellung z. B. von grafischem Inhalt. Das ActiveX-Element stellt seinen Inhalt (das geladene SVG-Bild) in dem Panel dar. Die unter der Präsentationsschicht liegende Anwendungsschicht kann im .Net Framework direkt auf deren Objekte zugreifen.

Bei der Erläuterung der Basistechnologien wurde allerdings schon darauf hingewiesen, dass die Einbindung im Compact Framework ungleich komplexer ist. Das .Net Compact Framework (CF) ist im Grunde eine reduzierte Version des .Net Frameworks. Deshalb stößt man bei anspruchsvolleren Anwendungen an die Grenzen der Kompaktversion. Die fehlende Unterstützung der ActiveX-Komponenten ist eine solche Grenze. Dies wird auch in Fachzeitschriften krtisiert.

Um dieses Defizit auszugleichen, hat Microsoft folgende Lösung veröffentlicht: Da man aus dem Compact Framework heraus die Komponente nicht direkt ansprechen kann, muss eine Art Brückenbibliothek, eine native Wrapper-DLL geschrieben werden. Der Wrapper ist an dieser Stelle als Schnittstelle zwischen ActiveX und CF zu verstehen. Das Erstellen des Wrappers ist sehr zeitaufwendig, da alle Objekte und Funktionen der ActiveX-Komponente abgebildet werden müssen. Die Wrapper-DLL muss in einer Programmiersprache geschrieben werden, die das CF verarbeiten kann. Diese Programmiersprache unterscheidet sich aber zwangsläufig von der Programmiersprache der ActiveX-Komponente. Dieser Unterschied ist die Ursache für viele Komplikationen bei der Erstellung und später bei der Verwendung des Wrappers. Das Problem der Konvertierung von Datentypen zwischen zwei Programmiersprachen ist dabei nur ein

Beispiel. Aus der Wrapper-DLL muss nach dem Erstellen für dessen Verwendung eine Wrapper-Klasse abgeleitet werden. Die Wrapper-Klasse muss wieder alle Methoden der Wrapper-DLL abbilden. Die Abbildung erfolgt programmiertechnisch über den Befehl Plattform Invoke (PInvoke). Eine CF-Anwendung kann die Wrapper-Klasse aufrufen und so auf die Funktionalität der ursprünglichen ActiveX-Komponente zugreifen. Diese sehr abstrakte Beschreibung des Vorgangs gibt nur einen kurzen Einblick in die tatsächliche Komplexität und Probleme bei der Verwendung von Wrappern im Compact Framework.

In der derzeit offiziellen Version .Net Compact Framework 2.0 SP1 (Stand: 01/2008) wurde die Unterstützung der DLL-Komponenten weiter verbessert. Es können z. B. COM-Komponenten automatisiert eingebunden werden. Aber für die grafische COM-Komponente, ActiveX, ist auch in der noch im Beta-Stadium befindlichen Version 3.5 keine Lösung in Aussicht[248]. Aufgrund des mühevollen und teilweise nicht Erfolg versprechenden Wegs des Erstellens eines Wrappers soll mit dieser Arbeit eine Alternative vorgestellt werden. Diese Lösung wurde auch von Microsoft als mögliche Alternative propagiert[249].

Wie in der technischen Architekturabbildung zu erkennen ist, wird das eSVG-Plug-in in ein CFCOM-Control der Firma Odyssey eingebettet. Die Einbindung in der Präsentationsschicht ist damit ähnlich unkompliziert wie im .Net Framework. Die Vorteile des CFCOM-Controls zeigen sich vorrangig bei der Verarbeitung der ActiveX-Funktionalität und werden deshalb in Kapitel 10.1.2.3 der Anwendungsschicht weiter vertieft.

10.1.2.3 Anwendungsschicht

Die Anwendungsschicht repräsentiert die komplette fachliche Verarbeitungslogik des Viewers. In der Erklärung der HMI-Schnittstelle (Kapitel 2.2) wurde sie deshalb als Logikschicht bezeichnet.

Zu deren Umfang gehört z. B. die Verarbeitung von Benutzeraktionen der Oberfläche. Eine solche Aktion wäre z. B. ein Klick auf einen Button auf der Oberfläche eines geladenen Diagnoseskripts. Der Button in Form eines grafischen SVG-Elements löst ein Ereignis aus. Das Ereignis wird auf der Präsentationsschicht (Oberfläche) ausgelöst und muss in der Anwendungsschicht verarbeitet werden. Aus dem Ereignis lässt sich das zugehörige Objekt (SVG-Element) ableiten, das das Ereignis ausgelöst hat. In dem Beispiel eben der Button der gedrückt wurde.

Im Fall von .Net Framework kann direkt auf die Objekte und Ereignisse des eSVG-Plug-ins zugegriffen werden. Durch den Einsatz der CFCOM-Technik gelingt dies ähnlich einfach auch auf der Compact Framework-Version des Viewers.

Ohne diese Technik müsste der im vorherigen Kapitel beschriebene komplexe Weg der Wrapper-DLL und -Klasse gewählt werden. Durch die Verarbeitung über CFCOM steht aber auch hier ohne komplexere Wrapper-DLLs der Zugriff auf das Ereignis und dessen Objekte zur Verfügung.

Wie sich der ActiveX-Einbau für das .Net Framework und das Compact Framework programmiertechnisch gestaltet, wird in Kapitel 10.4.2 bei der Implementierung näher erläutert. Für das Verständnis der technischen Architektur soll die Erläuterung genügen, dass trotz unterschiedli-

[248] Vgl. PInvoke 2008 (Online-Quelle)
[249] Vgl. Sjöström 2003 (Online-Quelle)

cher Frameworks bzw. Betriebssysteme eine ähnliche Weise gefunden wurde, in der Anwendungsschicht das SVG-Format zu verarbeiten.

Auch bei dem Zugriff auf das Steuergerät trifft man auf eine betriebssystemabhängige Lösung. Vom Kooperationspartner wird jeweils ein Treiber für Windows Mobile (CE) und einer für XP (32-Bit)-Systeme zur Verfügung gestellt. Um den Grad an gemeinsam nutzbarer Software entsprechend hoch zu halten, muss auch hier eine Möglichkeit gefunden werden, die Treiber möglichst flexibel einzubinden. Hierfür wurde das Konzept einer programmiertechnischen Schnittstelle (Interface) genutzt. Die Abbildung zeigt eine Schnittstelle, die den Zugriff auf den Treiber des Basissystems überbrückt. Durch die Überbrückung mittels einer Schnittstellenklasse bleibt das System anpassungsfähig für weitere Basissysteme, ohne dabei im Kern verändert werden zu müssen. Die Hintergründe und Vorteile der Schnittstellenprogrammierung werden in Kapitel 10.4.1 noch näher beschrieben.

Die wahre Stärke der .Net-Technologie und weshalb sie für den Viewer als Plattform ausgewählt wurde, zeigt sich allerdings erst bei der Verbindung des Basissystemzugriffs mit der Verarbeitung der SVG-Grafik. Das Zusammenführen der grafischen Darstellung mit der Steuergerätetechnologie macht den Kernbereich der Viewer-Software aus.

Wie in Abbildung 10.2 gezeigt wird, konnte dieser Bereich der Verarbeitungslogik plattformunabhängig programmiert werden. Er ist in der Abbildung zentral in der Anwendungsschicht, ohne notwendige Zugehörigkeit zu einem Framework, dargestellt. Dies schließt den SVG und Control Import Manager sowie diverse Hilfsfunktionen ein. Hintergründe der Manager werden in Kapitel 10.4.1 erläutert. Die aus der Plattformunabhängigkeit entstehenden Vorteile, wie die Verwendbarkeit in beiden Viewer-Versionen und die Reduzierung der Fehleranfälligkeit, wurden in Kapitel 9.3.3 bereits angeführt. Sie sind entscheidende Vorteile der .Net-Technologie.

Dieser umfangreichste Teil der Programmierung stellt die zentrale Programmintelligenz dar. Es werden die eingehenden SVG-Ereignisse verarbeitet, die daraus resultierenden notwendigen Befehle an das Steuergerät weitergeleitet und Antworten grafisch an den Benutzer zurückgegeben. Das Leistungspotenzial der Dynamisierung und Interaktion steckt hier. Deshalb ist die plattformunabhängige Programmierung an dieser Stelle besonders wichtig für die Qualitätssicherheit. In Kapitel 10.4.3 wird das Konzept und die Implementierung der zentralen Verarbeitung noch weiter vertieft.

Für den tatsächlichen Kontakt zur Hardware (Steuergerät) sind plattformabhängige Treiber notwendig. Sie werden in der Anwendungsschicht über Schnittstellen angesprochen (siehe: Basissystemschnittstelle). Aus architektonischer Sicht finden sich die Treiber in der Persistenzschicht wieder.

10.1.2.4 Persistenzschicht

Die Persistenzschicht stellt beim Viewer die endgerätspezifische Schnittstelle zu dem Steuergerät und den physischen Dateien dar.

Diese Schnittstelle ist zum einen der Treiber für das Basissystem. Dieser wird im vorliegenden Fall vom Kooperationspartner bzw. einem beauftragten Softwarezulieferer bereitgestellt. Über das in dieser Arbeit erstellte Schnittstellenkonzept für das Basissystem (siehe Kapitel 10.1.2.3 und 10.4.1) können auch (proprietäre) Basissysteme anderer Kfz-Hersteller eingebunden werden. Für die Kommunikation mit dem Fahrzeug greift der Treiber des Kooperationspartners

noch zusätzlich auf eine Steuergeräteinformationsdatei zu. Nähere Erläuterungen hierzu finden sich in Kapitel 3.2.

Neben den Steuergerätedaten zählt natürlich auch das Diagnoseskript zu den persistenten (dt. dauerhaften) Daten, die vom Viewer verarbeitet werden müssen. Wählt der Benutzer auf der Diagnoseoberfläche ein Diagnoseskript, sprich ein SVG-Quelle, zum Laden aus, so wird auf der Oberfläche (Präsentationsschicht) an sich nur der Pfad ausgewählt. Über die Anwendungsschicht wird der Aufruf an die Persistenzschicht weitergeleitet. Die eSVG Engine bestimmt daraufhin mit dem entsprechenden Pfad die Datei und liest deren Inhalt ein. Daraufhin erfolgen die Verarbeitung und das Anzeigen des Bildes. Für das Anzeigen des Bildes lädt die Engine noch die zugehörigen JavaScript-Bibliotheken und eingebundene Bilder. Nach diesem Vorgang kann das Bild in der Anwendungsschicht verarbeitet und in der Präsentationsschicht dargestellt werden.

Neben den für den Programmablauf relevanten Dateien ist natürlich auch die Viewer-Software selbst auf dem Dateisystem des Endgeräts gespeichert bzw. installiert. Diese Speicherung der Software wird in der IT-Architektur als physische Verteilung bezeichnet. Sie ist Inhalt des nächsten Kapitels.

10.2 Physische Verteilung

Die physische Verteilung definiert, wie die Softwarekomponenten auf den Rechnern verteilt sind und über welche Netzwerke sie miteinander kommunizieren[250].

Die Klärung der Verteilung ist besonders bei Client/Server-Systemen relevant, da damit auch funktionelle Zuordnungen festgelegt werden. So liegt z. B. bei einem Internetbrowser als schlanker Client (Thin Client) die Verarbeitungslogik größtenteils auf dem Server.

In dem vorliegenden System zur Diagnose handelt es sich bei der Anwendersoftware, Editor und Viewer, jeweils um Stand-alone-Anwendungen. Sie sind also ohne den Kontakt zu einem Server lauffähig. Diese Anforderung wurde in Kapitel 7.3.2.2 als Voraussetzung definiert. Deshalb gestaltet sich die Erläuterung der physischen Verteilung relativ einfach. Sie wird in den Kapitel 10.2.1 und 10.2.2 kurz dargestellt.

Abschließend wird die Netzwerkkommunikation behandelt. Die Kommunikation des Editors für den Up- und Download der Diagnoseskripte erfolgt über ein übliches FTP- bzw. SFTP-Protokoll. Sie ist aber für die Funktionalität des Diagnosesystems (Editors) nicht zwingend notwendig und wird deshalb nicht näher erläutert. Für den Viewer dagegen ist die Verbindung zum Steuergerät elementar. Wie aber bereits in Kapitel 2.2 abgegrenzt wurde, ist die Verarbeitung der Kommunikation Aufgabe des Basissystems und nicht der HMI-Software (Viewer). Es werden deshalb in Kapitel 10.2.3 nur einige Beispiele dafür gegeben, wie die Diagnosesoftware mit dem Fahrzeug kommunizieren kann.

[250] Vgl. Dunkel 2003, S. 20

10.2.1 Editor

Die Diagnoseskripte werden vorrangig in der Verwaltung erstellt. Dort finden sich beim Kooperationspartner typischerweise als Arbeitsplatzrechner Desktop-PCs oder Notebooks. Für den Editor besteht keine Anforderung auf mobilen Kleingeräten lauffähig zu sein.

Da der Editor als Plug-in für Eclipse entwickelt wird, ist er auf jedem Rechner, auf dem diese Plattform lauffähig ist, auch nutzbar. Der Editor kann integriert in eine vollständige Eclipse-Plattform oder als einzelne Plug-in-Sammlung verteilt werden, je nachdem, ob eine Neuinstallation der Eclipse-Umgebung erforderlich ist oder der Editor in eine bestehende Umgebung integriert werden soll.

Der Editor wurde optimiert entwickelt für die Eclipse-Version 3.1, ist aber auch in die aktuelle Version 3.3 (Stand: 02/2008) integrierbar. Eclipse ist auf einer großen Bandbreite von Betriebssystemen ausführbar. Dazu zählen Windows, Linux, Solaris und Mac OS X. An die Hardware wird nur die Empfehlung von mindestens 500 MB Arbeitsspeicher ausgesprochen. Abschließend wird die Java Virtual Machine (VM) gefordert, die die Plattform als Laufzeitumgebung benötigt. Diese sollte mit einer JDK-Version von mindestens 1.4.2 (Patch 13) installiert sein[251].

10.2.2 Viewer

Für den Viewer präsentiert sich die physische Verteilung aufgrund des mobilen Einsatzes etwas differenzierter.

Da auch der Viewer in eine Plattform integriert ist, benötigt er, wie der Editor eine entsprechende Laufzeitumgebung. Bei dem Viewer ist dies das .Net Framework für den Desktop Viewer und das .Net Compact Framework für den Mobile Viewer. Das Framework muss jeweils auf dem Endgerät installiert sein. Der Viewer ist optimiert entwickelt für die Version 1.1, ist aber auch auf der derzeit aktuellen Compact Framework Version 2.0 SP1 und der .Net Framework 3.0 einsetzbar (Stand: 02/2008).

Neben dem Framework müssen noch die Treiber zur Verarbeitung des SVG-Formats und zum Zugriff auf das Basissystem installiert sein. Beide stehen in einer Windows XP- und Windows Mobile-Version zur Verfügung.

Durch den Einsatz der .Net-Technologie wurde das beabsichtigte große Spektrum an möglichen Endgeräten erreicht. Auf nahezu jedem Gerät, auf dem das Betriebssystem Windows XP, NT, CE oder Mobile installiert ist, ist die Viewer-Software theoretisch lauffähig. Dadurch ergibt sich eine Spannbreite an Handys, Smartphones, Blackberrys, PDA, Palms, Tablet PCs, Notebooks, Desktop-PCs, Terminal-PCs etc. als Einsatzmöglichkeiten.

Da besonders der mobile Markt ständig in Bewegung ist und ein beständiges Redesign der Endgeräte zu beobachten ist[252], erscheint es nicht sinnvoll, mit dem Konzept bzw. der Architektur des Diagnosesystems eine Empfehlung für ein Endgerät auszusprechen. Es soll vorrangig die Flexibilität herausgestellt werden. Welches Gerät sich als optimal erweist, ist durch die Projektierung der Software zu bestimmen. Innerhalb des Projekts kann der genaue Einsatzort

[251] Vgl. Daum 2006, S. 11
[252] Vgl. Schiller 2003, S. 23

untersucht werden. Deshalb erfolgt eine Abwägung der Endgeräte in Kapitel 11.3 im Rahmen der Projektierung beim Kooperationspartner.

10.2.3 Netzwerke

Aufgrund der Trennung zwischen HMI-Software und Basissystem (siehe Kapitel 2.2) braucht muss der Viewer nicht direkt mit dem Fahrzeug kommunizieren. Diese Aufgabe übernimmt das Basissystem. Der Viewer kann somit unabhängig von der Netzwerktopologie gestaltet werden. Da die Mobilität des Endgeräts aber nur in dem Grad möglich ist, wie es auch der Zugriff auf das Fahrzeugnetzwerk zulässt, sollen an dieser Stelle kurz die Möglichkeiten hierfür erörtert werden.

Kfz-Netzwerke unterscheiden sich wie elektronische Netzwerke grundsätzlich nach der Art ihrer Kommunikationsverfahren. Es wird nach dem leitungsgebundenen und leitungslosen Verfahren unterschieden. Bei den gebundenen Verfahren wird zur Übertragung ein physisches Medium, z. B. ein Kabel, benötigt. Bei den anderen erfolgt die Übertragung per Funkwellen[253].

Im Fahrzeug sind die Steuergeräte entweder direkt per Kabelverbindung oder über ein Bus-Netzwerksystem miteinander verbunden (siehe Kapitel 2.1.4). Da es im Diagnosebereich zu Beginn wenig Standards gab, entwickelte fast jeder Kfz-Hersteller sein eigenes Kommunikationssystem. Deshalb haben sich entsprechend vielfältige Diagnoseanschlüsse und Protokolle entwickelt. Nur wenige davon sind heute standardisiert. Ein Beispiel ist das OBD-I- oder OBD-II-Verbindungskabel bzw. Stecker[254].

Da eine physische Leitung die Mobilität eines tragbaren Diagnosegeräts einschränkt, werden zunehmend Funkschnittstellen für die Diagnosekommunikation verwendet. Hierbei werden Technologien aus der PC-Technik in die Diagnose übertragen. Es werden Funkadapter basierend auf Firewire und Bluetooth für die Diagnose eingesetzt[255].

Ein Beispiel ist der EDICBlue-Stecker der Softing AG. Dies ist ein Steckeraufsatz, der an der Diagnoseschnittstelle des Fahrzeugs angebracht wird und über Bluetooth eine Verbindung zum Diagnosegerät aufbaut. Er unterstützt z B. das beim Kooperationspartner im Einsatz befindliche Diagnosebasissystem. Durch die inzwischen zahlreichen Angebote an Funkdiagnoseschnittstellen ist die Mobilität des Viewers folglich nicht durch den Zugriff auf das Fahrzeugnetzwerk beeinträchtigt.

Mit der Bestimmung der physischen Verteilung der Software ist die Beschreibung der Softwarearchitektur vollständig. Die nun folgenden Kapitel 10.3 und 10.4 beschäftigen sich damit, wie die Architektur programmiertechnisch umgesetzt werden kann. Es werden dafür die Schwerpunkte der Viewer- und Editor-Implementierung betrachtet.

[253] Vgl. Stein 2004, S. 53
[254] Vgl. Schäffer 2007, S. 12
[255] Vgl. Schäffer 2007, S. 10

10.3 Schwerpunkte der Editor-Implementierung

Mittels der technischen Architektur wurde gezeigt, dass die Funktionalität des Editors in Form von Eclipse-Plug-ins realisiert wird. Bekannt wurde Eclipse zunächst als Entwicklungsumgebung für Java-Anwendungen (siehe Kapitel 9.3.2.1). Als eines der ersten seiner Art ist es selbst allerdings auch komplett, inklusive der Laufzeitumgebung, in Java verfasst. Dies bedeutet, dass auch die Plug-ins in der Sprache Java zu entwickeln sind.

Die erste Version von Java wurde 1995 von Sun Microsystems veröffentlicht. Java avancierte seit dieser Zeit zu einer der populärsten Programmiersprachen im Bereich der Anwendungs- und Webentwicklung. Den Durchbruch verursachte z. B. die native Unterstützung in Webbrowsern. Heute ist Eclipse eine etablierte IDE für die Java-Entwicklung[256].

Mit zum Erfolg hat auch die leichte Erlernbarkeit der objektorientierten Sprache beigetragen. Bei den folgenden Erläuterungen werden Java-Codeausschnitte der Plug-ins präsentiert. Ergänzt werden die Erläuterungen durch XML-Codeausschnitte. XML dient in dieser Arbeit unter anderem zur hierarchisch strukturierten Darstellung der Steuergerätedaten.

Die Auszüge aus dem Quellcode beschäftigen sicht mit der Kernthematik der technischen Architektur des Editors. Es wird in Kapitel 10.3.2 gezeigt, wie die Einbindung der Steuergerätedaten realisiert wurde. Kapitel 10.3.3 und folgende beschäftigen sich mit der Verarbeitung des SVG-Formats und dessen Zusammenführung mit den Steuergerätedaten. Hierbei wird besonders auf die Gestaltung der Diagnoseformen eingegangen. Die Formen sind die Schnittstelle, an der das Grafikformat und die Diagnosetechnik aufeinandertreffen. Die technische Verarbeitung dieser Verknüpfung ist die zentrale Leistung des Editors. Die Beschreibung mündet deshalb in die softwaretechnischen Lösung dieser Aufgabe in Kapitel 10.3.5.

Als Einstieg in die Erläuterung der Plug-in-Entwicklung soll zunächst gezeigt werden, wie Plug-ins definiert und in die Eclipse IDE integriert werden (Kapitel 10.3.1).

10.3.1 Definition der Plug-ins

Die Definition eines Plug-ins erfolgt in den Manifest-Dateien. Diese Dateien sind der zentrale Knotenpunkt bei der Plug-in Entwicklung. Sie steuern die Einbettung in die Eclipse-Umgebung. Dabei ist die OSGi-Manifest-Datei für den Klassenpfad und die Ablaufspezifikation zuständig. Die Manifest-Datei plugin.xml definiert dagegen die Zusammenarbeit mit anderen Plug-ins und die Verwaltung der Erweiterungspunkte (Extension Points)[257].

Bei der technischen Architektur wurde bereits darauf hingewiesen, dass die Extension Points das Plug-in-Konzept der Eclipse-Architektur ermöglichen. Plug-ins koppeln über die Extension Points mit anderen Plug-ins und können so deren Funktionalität nutzen.

Das Plug-in Editorverwaltung koppelt z. B. mit dem Wizard-Plug-in (siehe Kapitel 10.1.1.3) um Dialogformulare für das Anlegen von Diagnoseskripte zu generieren.

[256] Vgl. Shavor 2003, S. 4-9
[257] Vgl. Shavor 2003, S. 212

```
<requires>
       <import plugin="org.eclipse.core.resources"/>
       <import plugin="org.eclipse.ui"/>
</requires>
...
<extension point="org.eclipse.ui.newWizards">
       <category name="Editor Management"
              id="DiagnosticEditor.plugin.EditorManagement">
       </category>
...
       <wizard
           name="Default Project"
           icon="icons/Default.gif"
           category="DiagnosticEditor.plugin.Editor"
           class="DiagnosticEditor.plugin.Editor.wizards.
                     EditorDefaultProjectCreationWizard"
           project="true"
           id="DiagnosticEditor.plugin.Editor.wizards.defaultwizard">
           <description>
              Creates a Default Project.
              </description>
       </wizard>
...
</extension>
```

Abbildung 10.3 Quellcode: Plugin.xml Plug-in Editorverwaltung

Die Abbildung zeigt die Kopplung des Plug-ins Editorverwaltung (DiagnosticEditor.plugin.EditorManagement) mit dem Plug-in Wizard. Der Zugriff erfolgt über den org.eclipse.ui.newWizards-Extension Point. Durch die Verlinkung kann die Editorverwaltung dem Benutzer Dialoge (Wizards) zur Verfügung stellen. Ein Beispiel ist der "Default Project"-Wizard zum Anlegen eines Diagnoseprojekts. Durch die Verlinkung über den Extension Point erhalten die Wizards der Editorverwaltung die gleiche Struktur und das gleiche Aussehen, wie die bereits in Eclipse vorhandenen Wizards. Somit wird das einheitliche Bedienkonzept erhalten.

Wählt der Benutzer den Dialog "Default Project" aus, dann startet Eclipse den Wizard, der unter der Klasse mit dem Wert "DiagnosticEditor.plugin.Editor.wizards.EditorDefaultProjectCreationWizard" definiert ist. In der Java-Klasse EditorDefaultProjectCreationWizard kann durch die Verlinkung über den Extension Point auf die Funktionalität des Wizard Plug-ins zugegriffen werden. Die folgende Abbildung zeigt z. B. das Erstellen einer Dialogseite über das Plug-in bzw. dessen Bibliothek org.eclipse.ui.dialogs.

```
public void addPages() {
       ...
       projectPage = new org.eclipse.ui.dialogs.WizardNewProjectCreationPage("New
Project");
       projectPage.setTitle("Diagnostic Editor Project");
       projectPage.setDescription("Create a project according to selected template");
       ...
       addPage(projectPage);
}
```

Abbildung 10.4 Quellcode: Klasse EditorDefaultProjectCreationWizard

Auf diese Weise koppeln auch die anderen Plug-ins, wie der SVG-Editor und das Control-Import-Plug-in mit anderen Plug-ins. Das Openvue SVG-Editor-Plug-in nutzt z. B. den Erweiterungspunkt für den Editorbereich.

```
<extension point="org.eclipse.ui.editors">
       <editor
           name="SVG Widget Framework Editor"
           icon="icons/SVGLogo16.gif"
           extensions="svg"
           contributorClass="net.openvue.svgwf.editors.
              SVGWFEditorContributor"
           class="net.openvue.svgwf.editors.SVGWFEditor"
           id="net.openvue.svgwf.editors.SVGWFEditor">
       </editor>
</extension>
```

Abbildung 10.5 Quellcode: Plugin.xml Plug-in SVG-Editor

Neben den vorgeschriebenen Parametern beim Aufruf eines Erweiterungspunkts können auch individuelle Übergabewerte definiert werden. Es wird hier z. B. der Parameter extensions="svg" definiert. Dies führt dazu, dass sobald der Benutzer in der Eclipse-Umgebung eine Datei mit der Endung svg öffnet, das Openvue SVG-Editor-Plug-in zum Darstellen des Inhalts benutzt wird. Dadurch wird die Eclipse IDE um die Fähigkeit erweitert, SVG-Format zu verarbeiten.

Das Control-Import-Plug-in nutzt den Extension Point um im Menü Einstellungen (Preferences) von Eclipse eine zusätzliches Fenster anzuzeigen. Dadurch werden die Einstellungen von Eclipse erweitert. Sie erhalten über das Plug-in Control Import die Fähigkeit, Steuergerätedateien zu laden.

```
<extension point="org.eclipse.ui.preferencePages">
       <page
           name="Diagnostic Control Import"
           class="DiagnosticEditor.plugin.ControlImport.
              preferences.ControlPreferencePage"
           id="DiagnosticEditor.plugin.ControlImport.
              preferences.ControlPreferencePage">
       </page>
</extension>
```

Abbildung 10.6 Quellcode: Plugin.xml Plug-in Control Import

Gestützt durch Extension Points stellen die Plug-ins die Basis für die Funktionalität und Flexibilität von Eclipse dar. Welche Schritte innerhalb der für diese Arbeit relevanten Plug-ins ablaufen, wird in Kapitel 10.3.2 erläutert. Begonnen wird mit der Integration der Steuergerätedaten.

10.3.2 Einbinden der Steuergerätedaten

In den Kapiteln 2 und 3 wurde darauf verwiesen, dass bei vielen Kfz-Herstellern proprietäre Basissysteme zum Einsatz kommen. Deshalb ist ein Kriterium des Anforderungskatalogs die Daten des Basissystems über eine flexible Schnittstelle in den Editor integrieren zu können. Der Editor realisiert dies über eine offene XML-Schnittstelle. Die Struktur ist angelehnt an das standardisierte Basissystem des MCD-Systems der ASAM e.V. (siehe Kapitel 2.1.4). Sie wird

hier in vereinfachter Form dargestellt. Die Abbildung zeigt unter anderem die vier elementaren Hierarchietypen (siehe gelbe Markierung), die im Folgenden noch erläutert werden.

```
<!-- properties -->
<!ENTITY % common.atts
       "id ID #REQUIRED name CDATA #IMPLIED">

<!-- hierarchy  -->
<!ELEMENT database  (job+)>
<!ELEMENT job       (result+, argument*, comment?)>
<!ELEMENT result    (type, comment?) >
<!ELEMENT argument  (type, comment?) >

<!-- value tags  -->
<!ELEMENT comment   (#PCDATA) >
<!ELEMENT type      (#PCDATA) >
<!ELEMENT id        (#PCDATA) >
<!ELEMENT name      (#PCDATA) >

<!-- attributes -->
<!ATTLIST database  %common.atts; >
<!ATTLIST job       %common.atts; >
<!ATTLIST result    %common.atts; >
<!ATTLIST argument  %common.atts; >
```

Abbildung 10.7 Quellcode: Basissystem Schnittstelle im DTD-Format

XML stützt sich für die Definition der Datenstruktur auf das DTD-Format. Mit dem DTD-Format kann die Gültigkeit von Daten überprüft werden. Nur wenn die Daten dieses Format erfüllen, werden sie in den Editor importiert. Da die Struktur aber nicht hart codiert im Java-Quellcode des Plug-ins verankert ist, sondern ausgelagert in der DTD definiert ist, können an proprietären Basissysteme Änderungen oder Erweiterungen vorgenommen werden. Dies ist vor allem im Bereich der Datentypen von Vorteil. Die Importschnittstelle bleibt so flexibel.

Elementar für die Struktur sind allerdings die vier Elemente Steuergerät (database), Befehl (job), Ergebnistyp (result) und Aufrufparameter (argument). Aus ihnen bezieht das Plug-in die elementaren Informationen, um Steuergerätebefehle aufzurufen. Beim Laden der Daten per XML überprüft das Plug-in, ob die in der DTD vorgegebene Struktur erfüllt ist. Erfolgreich geladene Daten erscheinen in der Eigenschaftsliste der Diagnoseformen (siehe Kapitel 10.1.1.2). Aus dieser Liste kann der Benutzer ein Steuergerät und die zugehörigen Steuergerätedaten auswählen. Mit der Auswahl wird der Steuergerätebefehl einer Diagnoseform zugewiesen.

Um die Steuergerätedaten zu laden und in der Eigenschaftsliste der Diagnoseformen anzuzeigen, enthält das Plug-in Control Import die Java-Klasse ControlProvider. Der ControlProvider lädt zunächst die entsprechenden Steuergerätedaten aus der XML-Datei. Zur Verarbeitung von XML bedient sich der Provider des Xerces-Moduls. Ein freiverfügbares Werkzeug, um mit Java XML zu lesen und zu schreiben. Der folgende Ausschnitt zeigt das Einlesen der Daten:

```
private void load() {
       DOMParser parser = null;
       try {
              parser = new DOMParser();
              InputSource is =
                     new InputSource(new FileInputStream(filename));
              parser.parse(is);
       } catch (SAXException e) {
```

```
                e.printStackTrace();
                return;
        } catch (IOException e) {
                e.printStackTrace();
                return;
        }
        doc = parser.getDocument();
        db = doc.getDocumentElement();

        // set db name
        setControlDB(
        XMLUtils.getAttribute(db,XML_ELEMENT_ID));
}
```

Abbildung 10.8 Quellcode: Control Provider (Laden der Steuergerätedaten)

Nach dem Laden können über die Klasse ControlProvider alle Steuergerätedaten zu einem Steuergerät abgefragt werden. Dafür enthält die Klasse verschiedene Get-/Set- (dt. Erhalten-/Setzen-)Methoden. Nach der Auswahl eines Steuergeräts per Set-Methode, können z. B. mit der Get-Methode getJobs() alle Steuergerätebefehle des Steuergeräts abgefragt werden.

```
public String[] getJobs(){
        StringBuffer sb = new StringBuffer();
        if ( doc!=null && db != null ) {
                Node[] job = XMLUtils.getChildren(db, UNIT_PREFIX_JOB);
                for ( int i=0; i < job.length; i++ ) {
                        sb.append(XMLUtils.getAttribute(
                                        job[i],XML_ELEMENT_ID)+UNIT_LIST_SEPERATOR);
                }
        }
...
        if (sb.length()> 0)
                return sb.toString().split(UNIT_LIST_SEPERATOR);
        else
                return null;
}
```

Abbildung 10.9 Quellcode: Control Provider (Ausgabe der Steuergerätebefehle)

Wählt der Benutzer für eine Diagnoseform einen Steuergerätebefehl aus, wird dieser über die Set-Methode setCurrJobId(String id) dem ControlProvider mitgeteilt werden. Nach der Bestimmung des Jobs liefert der ControlProvider mit der Methode getArguments() alle notwendigen Parameter sowie die Ergebnistypen mit getResults() für die Anzeige zurück.

Auf diese Weise versorgt das ControlImport-Plug-in die Eigenschaftsanzeige der Diagnoseformen mit Steuergerätedaten. Weist der Benutzer diese Daten per Auswahl einer Diagnoseform zu, muss der Aufruf des Befehls für die jeweilige Form im Diagnoseskript eingetragen werden. Um diesen Ablauf näher beschreiben zu können, wird zunächst auf die Struktur der Diagnoseformen eingegangen.

10.3.3 Gestaltung der Diagnoseformen mit SVG

Bei der Beschreibung der Basistechnologien in Kapitel 9.3 wurde auf die JavaScript-Unterstützung bei SVG verwiesen. Sie dient als Basis für Dynamiken und Interaktion. Das Konzept der Diagnoseformen stützt sich auf diese Fähigkeiten von JavaScript.

Obgleich JavaScript nicht alle Merkmale einer objektorientierten Programmiersprache besitzt, z. B. keine Vererbung, sind weitreichende Annäherungen erkennbar. JavaScript wird deshalb nicht als objektorientierte, sondern als objektbasierte Sprache bezeichnet. Objekte sind zentrale Elemente in JavaScript. JavaScript entfaltet seine Stärken erst durch die Verwendung von Objekten[258].

Die Ansätze der Objektorientierung in JavaScript wurden genutzt, um die Diagnoseformen zu erzeugen. Es wurde hierfür eine flache Hierarchie von Objekten erzeugt. Obwohl JavaScript, wie erwähnt, keine Vererbung unterstützt, kann auf andere Art eine Hierarchie erzeugt werden. Die Objekthierarchie ergibt sich dadurch, dass ein Objekt innerhalb anderer Objekte verwendet werden kann und so in der Hierarchie unter ihnen liegt[259].

Das in der Hierarchie an unterster Stelle stehende Objekt ist die Komponente (SVGWF_Component). Dieses Objekt stellt die elementare Ereignisbehandlung bereit, die standardmäßig in jedem Objekt gebraucht wird. Dies ist z. B. auf Maus- oder Tastaturbedienung zu reagieren. Des Weiteren werden die für ein Objekt typischen Get-/Set-Methoden bereitgestellt. Im Grunde erzeugt das Initialisieren dieser Komponente ein rudimentäres SVG-Objekt mit verschiedenen Größen und Positionsattributen. Es bildet die Basis einer Diagnoseform.

```
function SVGWF_Component(parent, svarid, x, y, w, h)
{
	this.svgObject = createSvg(x, y, w, h);
	...
}
...
function createSvg(x, y, w, h)
{
	s = createElement("svg");
	s.setAttribute("x", x ? x : 0);
	s.setAttribute("y", y ? y : 0);
	s.setAttribute("width", w ? w : 0);
	s.setAttribute("height", h ? h : 0);
	return s;
}
...
function createElement(elemName)
{
	return document.createElementNS(svgNS, elemName);
}
```

Abbildung 10.10 Quellcode: SVGWF_Component (Objektinitialisierung)

Durch in der Hierarchie übergeordnete Objekte wird dieses SVG-Objekt verfeinert. So erweitert z. B. die Komponente SVGWF_Button die Komponente SVGWF_Component, indem sie das SVG-Objekt mit einem abgerundeten Rechteck umrandet und das Rechteck mit Text und Farbe füllt. Dadurch entsteht das Bild eines Knopfes.

```
function SVGWF_Button(parent, svarid, text, x, y, w, h)
{
	this.base = SVGWF_Component;
	this.base(parent, svarid, x, y, w, h);
	this.background = createRect(1, 1, w, h, h / 2, h / 2);
...
```

[258] Vgl. Hirsemann 2003, S. 108
[259] Vgl. Hirsemann 2003, S. 121

```
function createRect(x, y, w, h, rx, ry)
{
	r = createElement("rect");
	r.setAttribute("x", x ? x : 0);
	r.setAttribute("y", y ? y : 0);
	r.setAttribute("rx", rx ? rx : 0);
	r.setAttribute("ry", ry ? ry : 0);
	r.setAttribute("width", w ? w : 0);
	r.setAttribute("height", h ? h : 0);
	return r;
}
```

Abbildung 10.11 Quellcode: SVGWF_Button (Objektinitialisierung)

Neben den optischen Erweiterungen werden der rudimentären Komponente noch zusätzliche Ereignisbehandlungen und Get-/Set-Methodem hinzugefügt. Ein Button kann z. B. zusätzlich auf ein Ereignis reagieren, dass von einem Steuergerät ausgelöst wird, oder einen Befehl an das Steuergerät schicken. Diese Kopplung der Komponenten mit den Steuergerätedaten wird in Kapitel 10.4 noch konkreter erläutert.

Durch die hierarchische Erhöhung und somit die weitere Spezifizierung der Objekteigenschaften und Fähigkeiten können beliebig komplexe Objekte zur Diagnose (Diagnoseformen) geschaffen werden. Dies zeigt z. B. die Textausgabe. Aufbauend auf der SVGWF_Component wird ein Container erzeugt, der Daten enthalten kann. Der Container wird umschlossen von einem SVGWF_ScrollablePanel-Objekt. Damit erhält das Objekt die Eigenschaft, zu große Inhalte mit einer Laufleiste innerhalb des Containers verschiebbar zu machen. Über dem SVGWF_ScrollablePanel liegt die SVGWF_TextEdIT-Komponente. Diese erweitert das ScrollablePanel um ein Textfeld. Es entsteht damit ein Eingabe- und Anzeigemöglichkeit von blätterbaren (engl. to scroll – blättern) Textinhalten. Um der Textbearbeitung noch zusätzlichen Komfort zu geben, kann die Hierarchie beliebig erhöht werden.

Anhand dieses Konzepts stellt der Editor flexible Diagnoseformen bereit, die somit innerhalb eines Anwenderprojekts beliebig erweitert werden können. Als Basis bietet der Editor an Diagnoseformen z. B. Textfelder, Knöpfe, LED oder Gradienten an. Damit der Benutzer die Formen auf der Zeichenoberfläche grafisch bearbeiten und positionieren kann, müssen sie im Editor entsprechend registriert und verarbeitet werden. Diese Zusammenhänge werden im nächsten Kapitel 10.3.4.

10.3.4 Verarbeitung der Diagnoseformen im Editor

In Kapitel 10.3.3 wurde verdeutlicht, dass die auf JavaScript basierenden Diagnoseformen beliebig erweiterbar sind. Dieser Anspruch bezüglich der Flexibilität ist auch im Anforderungskatalog verankert. Um die Diagnoseformen für die grafische Verarbeitung im Editor zur Auswahl bereitzustellen, müssen sie im SVG-Editor-Plug-in eingetragen werden. Entsprechend der Forderung des Anforderungskatalogs darf folglich auch die Registrierung im Editor keine Java-Quellcode-Veränderungen des Plug-ins notwendig machen, sondern sie muss flexibel parametrierbar sein. Deshalb erfolgt die Parametrierung über eine XML-Datei. Dort können nach Kategorien getrennt die einzelnen Formen definiert werden.

```
<?xml version="1.0" encoding="UTF-8"?>
<svgwidgetregister>
```

```
<family name="Standard Diagnostic Forms">
	<widget classname="SVGWF_Led" type="led" icon="images/Led.bmp">
		<property name="x" displayname="Left" category="Dimension"
			type="number" />
		<property name="y" displayname="Top" category="Dimension"
			type="number" />
...
		<property name="lgreen" displayname="Value to set light green"
			category="Values" type="text" />
		<property name="lred" displayname="Value to set light red"
			category="Values" type="text" />
...
	<widget>
</family>
<family name="Cooperation Partner Diagnostic Forms">
	<widget classname="SVGWF_Parkdistance_Gradient" type="PD"
		icon="images/PD-gradient.bmp">
		<property name="maxdistance" displayname="Max possible distance"
			category="Values" type="number" />
		<property name="mindistance" displayname="Minimum required distance"
			category="Values" type="number" />
...
	<widget>
</family>
</svgwidgetregister>
```

Abbildung 10.12 Quellcode: Diagnoseform Registrierung

Die Abbildung zeigt, wie in die Standard- und nutzerspezifische Kategorie die Diagnoseformen eingebunden werden. Für die Anzeige im Auswahlfenster können auch kleine Bilder (Icons) definiert werden. Die Kategorie der Standardformen enthält z. B. eine LED. Für den Kooperationspartner wird speziell ein Gradient zur Anzeige der Einparkentfernung eingebunden. Kapitel 11.3.3.3 zeigt eine Abbildung des Gradienten.

Auf der Editoroberfläche werden die parametrierten Diagnoseformen in einem Auswahlfenster angezeigt. Im Auswahlfenster werden der Name und das Icon angezeigt. Kapitel 11.2.1 liefert hierfür Bildschirmausschnitte. Das Füllen des Auswahlfensters übernimmt die Klasse SVGWFRegister.

```
SVGWFRegister reg = SVGWFRegister.getRegister();
final Node[] families = reg.getFamiliesArray();
URL imageFileURL = null;

for (int i = 0; i < families.length; i++) {
	final Node[] widgets = reg.getWidgetsArray(families[i]);
	final String catName = SVGWFRegister.getAttribute((Node)families[i], "name");

	for (int j = 0; j < widgets.length; j++) {
		final String dispName = SVGWFRegister.getAttribute(widgets[j], "type");
		final String fileName = SVGWFRegister.getAttribute(widgets[j], "icon");
		final String cmdID = SVGWFRegister.getAttribute(widgets[j], "class-
name");
		...
		final Image image = ImageDescriptor.createFromURL(imageFileURL)
			.createImage();
		items[j] = new IOutlookSelectorItemModel() {
			public Image getImage() {
				return image;
			}
			public String getText() {
				return dispName;
```

```
                    }
                    ...
             };
             ...
      }
};
```

Abbildung 10.13 Quellcode: SVGWFSelectorViewPart (Setzen der Auswahlformen)

Über die Klasse SVGWFRegister werden die definierten Kategorien (getFamiliesArray()) und deren Formen (getWidgetsArray(families[i])) abgefragt. Für die Anzeige der Formen nutzt die Klasse einen Extension Point zu den User Interface-Plug-in (siehe Kapitel 10.3.1). Das Ergebnis der Auflistung kann direkt dem Extension Point (Klasse IOutlookSelectorItemModel) übergeben werden. Daraufhin erscheinen die Formen in einem zusätzlichen Auswahlfenster.

Aus dem Auswahlfenster kann der Benutzer die Objekte per Drag & Drop in das grafische Editorfenster gezogen werden. Innerhalb des grafischen Editorfensters übernimmt das Batik Toolkit die Verarbeitung des SVG-Formats und des JavaScripts.

```
public void loadSVGDocument(String url) {
      String oldURI = null;
      if (svgDocument != null) {
            oldURI = svgDocument.getURL();
      }
      final ParsedURL newURI = new ParsedURL(oldURI, url);

      stopThenRun(new Runnable() {
            public void run() {
                  String url = newURI.toString();
                  fragmentIdentifier = newURI.getRef();

                  loader = new DocumentLoader(userAgent);
                  nextDocumentLoader = new SVGDocumentLoader(url, loader);

                  Iterator it = svgDocumentLoaderListeners.iterator();
                  while (it.hasNext()) {
                        nextDocumentLoader.addSVGDocumentLoaderListener(
                              (SVGDocumentLoaderListener) it.next());
                  }
                  startDocumentLoader();
            }
      });
}
```

Abbildung 10.14 Quellcode: Laden eines SVG-Bildes mit dem Batik Toolkit

Die Klassen DocumentLoader und SVGDocumentLoaderListener sind Bestandteil des org.apache.batik.swing.svg-Pakets. Mit ihrer Hilfe wird das Dokument geladen und der Ereignisbehandlung hinzugefügt. Da die Diagnoseformen mit SVG und JavaScript gestaltet sind, kann das Batik Toolkit sie direkt verarbeiten und darstellen. Der Benutzer braucht sie hierfür nur auf die Zeichenfläche zuschieben.

Um allerdings die Steuergerätedaten einer Diagnoseform zu verarbeiten, muss die Funktionalität des SVG-Editor-Plug-ins erweitert werden. Die Implementierung dieses Schritts wird in Kapitel 10.3.5 beschrieben.

10.3.5 Zuweisung der Steuergerätedaten auf eine Diagnoseform

Die Kernleistung des Editors, um in der Diagnosetechnik eingesetzt werden zu können, ist die Verknüpfung der Steuergerätedaten mit einer Diagnoseform.

Die Registrierung der Diagnoseformen,um sie im Editor verfügbar zu machen, wurde in Kapitel 10.3.4 erläutert. Über die Registrierung können der Diagnoseform verschiedene Eigenschaften, wie z. B. Kategoriezugehörigkeit und Icon, zugewiesen werden. Für die Diagnosetechnik wurden die Registrierungseigenschaften um weitere Einträge ergänzt. Eine Standard-Diagnoseform wie der Button erhält z. B. die Eigenschaften Steuergerät, Steuergerätebefehl, Ergebnistyp und Aufrufparameter.

```
<widget classname="SVGWF_Button" type="button" icon="images/NaviButton.bmp">
...
	<property name="job__ctrldb" displayname="Job-List"
		category="ControlImport" type="combo" />
	<property name="result__ctrldb" displayname="Result-List"
		category="ControlImport" type="combo" />
	<property name="argument__ctrldb" displayname="Argument"
		category="ControlImport" type="array" />
	<property name="db__ctrldb" displayname="Control-DB"
		category="ControlImport" type="combo" />
</widget>
```

Abbildung 10.15 Quellcode: Diagnoseform Registrierung (Button)

Wird die Diagnoseform aus dem Auswahlfenster in das graphische Editorfenster verschoben, dann erscheinen im Eigenschaftsfenster des Editors die Felder aus der Registrierung. Jeder mit Property (dt. Eigenschaft) bezeichnete Eintrag wird angezeigt.

Soll eine Eigenschaft angezeigt werden, die die Endung __ctrldb besitzt, werden aufgrund des vorangestellten Schlüsselworts, wie db (Steuegerät) oder job (Steuergerätebefehl), beim Control-Import-Plug-in die entsprechenden Werte dafür abgefragt. Für das Property job__ctrldb wird folglich eine Eigenschaft für die Diagnoseform ausgegeben, die alle möglichen Steuergerätebefehle für ein Steuergerät auflistet. Wie die Steuergerätedaten vom Control-Import-Plug-in zur Verfügung gestellt werden, ist in Kapitel 10.3.2 erläutert worden.

Folgende Quellcodeausschnitt zeigt, wie das SVG-Editor-Plug-in mit dem Control-Import-Plug-in zusammenarbeitet, um für eine Diagnoseeigenschaft die Werte zu erfragen. Da die Liste der Eigenschaften je Diagnoseform bzw. je Basissystem variabel sein kann, muss die Abfrage entsprechend abstrakt gehalten sein.

```
(SVG Editor Plug-in)
private String[] getValueArray(ControlProvider cp, Object id, String def) {
	String[] array = null;
	if ( def!=null && def.length()>0 ) {
		array = def.split(";");
	} else if ( cp!=null && id.toString().endsWith(ControlProvider.UNIT_APPENDIX)
) {
		String prefix = ControlProvider.getPrefix(id.toString());
		array = cp.getUnitList(prefix);
	}
	...
	return array;
}
```

```
(Control-Import-Plug-in)
public String[] getUnitList(String unit){
       String[] list = null;
       if ( isValidUnit(unit) ){
              if ( unit.equalsIgnoreCase(UNIT_PREFIX_JOB) )
                     list = getJobs();
              else if ( unit.equalsIgnoreCase(UNIT_PREFIX_ARG) )
                     list = getArguments();
              else if ( unit.equalsIgnoreCase(UNIT_PREFIX_RES) )
                     list = getResults();
              ...
       }
       return list;
}
```

Abbildung 10.16 Quellcode: Laden der Eigenschaft Steuergerätebefehl

Entsprechend der Endung der Eigenschaft erkennt das SVG-Editor-Plug-in, dass eine Steuergeräteeigenschaft abzubilden ist (ControlProvider.UNIT_APPENDIX="__ctrldb"). Mit dem vorangestellten Schlüsselwort (prefix) der Eigenschaft, wie job oder result, erfragt der SVG-Editor die Werte beim Control-Import-Plug-in. Diese Abfrage erfolgt über die Klasse ControlProvider (Kapitel 10.3.2) mit der allgemeinen Methode getUnitList. Der ControlProvider erkennt aufgrund des übergebenen Prefixes welcher Typ (unit) an Steuergerätedaten ausgegeben werden soll und liefert die Werte zurück. Für den Prefix job werden intern über die Methode getJobs() die Steuergerätebefehle abgefragt und das SVG-Editor-Plug-in übergeben. Der Benutzer erhält so eine Liste der möglichen Steuergerätebefehle zur Auswahl. Wählt er Steuergerätedaten für die Diagnoseform aus, werden diese im Diagnoseskript für die Diagnoseform verankert.

```
var button = new SVGWF_Button();
button.set('text','Rechts Blinker');
button.set('height','22');
button.set('width','125');
button.set('x','180');
button.set('y','7');
button.set('db__ctrldb','LSZ_2');
button.set('job__ctrldb','STEUERN_IO');
button.set('argument__ctrldb','ORT_1=BLK_RV__');
```

Abbildung 10.17 Quellcode: Diagnoseskript Neu-System (Button für rechten Blinker)

Der Ausschnitt des Diagnoseskripts zeigt, wie für die Diagnoseform Button der Steuergerätebefehl (STEUERN_IO) zum Aktivieren des rechten Blinkers eingetragen wurde. Wird dieses Skript im Editor erneut geladen, dann werden diese Werte in der Eigenschaftsliste der Diagnoseform vorbelegt. Erst mit dem Klick auf die Auswahlliste (zum Ändern der Eigenschaft) erfragt das SVG-Editor-Plug-in die Steuergerätedaten beim Control-Import-Plug-in auf die oben beschriebene Weise nach.

Mit diesem Konzept kann der Benutzer den grafischen Elementen der Zeichenoberfläche per Auswahllisten Steuergerätefunktionalität zuweisen und nachträglich ändern. Die Vorteile, wie der Wegfall zusätzlicher Applikationen, werden in der Evaluation in Kapitel 11 noch deutlich herausgestellt.

Abschließend soll mit Kapitel 10 noch erläutert werden, wie das fertige Diagnoseskript aus Sicht der Programmierung im Viewer verarbeitet wird.

10.4 Schwerpunkte der Viewer-Implementierung

Mit diesem Kapitel wird die Implementierung der zentralen Funktionsbereiche des Viewers beschrieben. Bei der Erläuterung der logischen Softwareschichten wurde verdeutlicht, dass ein Viewer für das .Net Framework und einer für das Compact Framework entwickelt wurde.

Bei der Entwicklung zweier Anwendungen, die beide gleiche Funktionalität bieten sollen, ist die Wiederverwendbarkeit von größter Wichtigkeit. Wie in Kapitel 10.1.2 erörtert wurde, wird dies bereits durch die Auswahl der .Net-Technologie an sich gefördert. Weiter begünstigt wird dieser Faktor durch die Auswahl von C# als Programmiersprache.

Die Sprache C# ist eine objektorientierte und typensichere Programmiersprache, die speziell für die .Net-Laufzeitumgebung entwickelt wurde[260]. Sie vereint Vorteile wie typensichere Konvertierung, strenge Objektorientierung, sprachenübergreifende Verwendung von Code (Bsp. C-Code) sowie ein moderne Syntax. Sie vereint Konzepte aus C, C++, Modula und Java[261].

Was C# aber vor allem im Bereich Wiederverwendbarkeit für diese Arbeit interessant macht, ist die Komponentenorientierung bzw. die Unterstützung der Entwicklung und Verwendung von Komponenten. C# wird auch häufig als erste komponentenorientierte Sprache der C-Familie bezeichnet[262]. Als Komponenten werden z. B. das eSVG-Plug-in und der Basissystemtreiber in den Viewer integriert. Eine direkte Unterstützung dieser Integration durch die Programmiersprache ist folglich von großem Vorteil.

Die programmiertechnische Integration dieser beiden Komponenten mittel der C#-Syntax wird in den folgenden Kapiteln beschrieben.

10.4.1 Einbindung von proprietären Basissystemen

Der Treiber für das Basissystem wird, wie das eSVG-Plug-in, als DLL für das Microsoft Betriebssystem zur Verfügung gestellt. Jeweils eine DLL für das 32-Bit-XP-Betriebssystem und eine für die CE- bzw. Windows Mobile-Version.

Diese DLL ist als klassisches COM-Objekt implementiert. Es liegt demnach die gleiche Technologie, wie bei der eSVG-ActiveX-Komponente zugrunde, nur ohne die grafische Ausgabe. Über die DLL werden Steuerbefehle abgeschickt und deren Ergebnisse abgefragt. Eine grafische Oberfläche ist somit für die DLL nicht notwendig. Dieser Unterschied macht es allerdings um einiges einfacher, das COM-Objekt in das .Net Framework und Compact Framework zu integrieren.

Auch bei der Integration eines nicht grafischen COM-Objekts muss ein Wrapper geschrieben werden (siehe Kapitel 10.1.2.2). Dessen Gerüst kann allerdings durch ein Dienstprogramm von Microsoft automatisch generiert werden. Der Wrapper muss in der Regel noch ergänzt und nachkorrigiert werden. Dies liegt daran, dass die COM-Objekte in einer .Net-fremden Sprache verfasst sind. Der Wrapper muss jedoch in einer .Net-Sprache, wie z. B. C#, verfasst werden.

---

[260] Vgl. Wille 2000, S. 24
[261] Vgl. Vasters 2002, S. 27
[262] Vgl. Vasters 2002, S. 42

Ziel des Wrappers ist es, die Funktionen und Methoden der Basissystem-DLL abzubilden. Dies sind vorrangig Methoden, um Steuergerätebefehle ans Fahrzeug zu schicken und deren Ergebnisse abzufragen. Folgender Codeausschnitt zeigt, wie die Wrapper-Klasse die Funktionen der 32-Bit-Basissystem-DLL kapselt.

```
using System.Runtime.InteropServices;
...
      [DllImportAttribute("api32.dll", EntryPoint="__apiJob", CharSet=CharSet.Ansi)]
      private static extern void __apiJob (uint h, sbyte[] ecu,
            sbyte[] job, sbyte[] para, sbyte[] result);
      [DllImportAttribute("api32.dll", EntryPoint="__apiResultSets",
            CharSet=CharSet.Ansi)]
      private static extern bool __apiResultSets(uint h, out ushort buffer);
...
```

Abbildung 10.18 Quellcode: Basissystem-Wrapper 32-Bit-OS

Die Kapselung der CE-Betriebssystem-DLL des Basissystems gestaltet sich ähnlich. Die Methode erhält allerdings den Zusatz CE.

```
using System.Runtime.InteropServices;
...
      [DllImportAttribute("apiCE.dll", EntryPoint="__apiJob", CharSet=CharSet.Ansi)]
      private static extern void __apiCeJob(uint h, sbyte[] ecu, sbyte[] job,
            sbyte[] para, sbyte[] result);

      [DllImportAttribute("apiCE.dll", EntryPoint="__apiResultSets",
            CharSet=CharSet.Ansi)]
      private static extern bool __apiCeResultSets(uint h, out ushort buffer);
...
```

Abbildung 10.19 Quellcode: Basissystem-Wrapper CE-OS

Der hier genutzte Dienst zur Interoperabilität zwischen .Net-fremdem Code und .Net-Wrapper nennt sich Platform Invoaction Services. Er wird im .Net-Namensraum (Bibliothek) System.Runtime.InteropServices bereitgestellt. Der Dienst, auch kurz PInvoke bezeichnet, ermöglicht es, mit .Net-Code auf Funktionen und Strukturen von COM-Objekten, die in Nicht-.Net-Code geschrieben sind, zuzugreifen. Die im Beispiel genutzte Methode von PInvoke ist DLLImport oder DLLImportAttribute[263].

Durch die Importdeklaration entstehen innerhalb des Wrappers jeweils die Methoden __apiJob und __apiCeJob zum Aufruf des Steuergerätebefehls. Da diese Methoden mit den Aufrufparametern sbyte und uint einen untypischen Datentyp für .Net oder andere höhere Programmiersprachen verlangen, wird noch eine weitere Methode dem Aufruf vorgelagert. Innerhalb dieser Methode wird die entsprechende Typkonvertierung durchgeführt (ByteToSByteArray).

```
public static void apiJob(string ecu, string job, string para, string result) {
...
      sbyte[] ecuSB=Utils.ByteToSByteArray(Utils.StrToByteArray(ecu));
      sbyte[] jobSB=Utils.ByteToSByteArray(Utils.StrToByteArray(job));
      sbyte[] paraSB=Utils.ByteToSByteArray(Utils.StrToByteArray(para));
      sbyte[] resultSB=Utils.ByteToSByteArray(Utils.StrToByteArray(result));
      __apiJob(h, ecuSB, jobSB, paraSB, resultSB);
}
```

[263] Vgl. Archer 2001, S. 305-310

Abbildung 10.20 Quellcode: Basissystem-Wrapper 32-Bit-OS

Analog für die CE-Version:

```
public static void apiJob(string ecu, string job, string para, string result) {
...
       sbyte[] ecuSB=Utils.ByteToSByteArray(Utils.StrToByteArray(ecu));
       sbyte[] jobSB=Utils.ByteToSByteArray(Utils.StrToByteArray(job));
       sbyte[] paraSB=Utils.ByteToSByteArray(Utils.StrToByteArray(para));
       sbyte[] resultSB=Utils.ByteToSByteArray(Utils.StrToByteArray(result));
       __apiCeJob(h, ecuSB, jobSB, paraSB, resultSB);
}
```

Abbildung 10.21 Quellcode: Basissystem-Wrapper CE-OS

Durch das Kapseln über eine zusätzliche Methode kann auf den Basissystemtreiber mit .Net-typischen Attributen (wie z. B. String) zugegriffen werden. Außerdem ergibt sich der Vorteil, dass der Methodename apiJob auf dem .Net Framework und dem Compact Framework gleich ist.

Die Koordination (engl. managing) des Steuergerätezugriffs erfolgt über den Control Import Manager (siehe Anwendungsschicht, Kapitel 10.1.2.1). Der Aufruf der Wrapper-Methoden, wie z. B. apiJob, ist aber im Control Import Manager nicht hart codiert. Das Diagnosesystem soll flexibel auch Basissysteme anderer Hersteller einbinden können. Deshalb wird eine weitere Aufrufschnittstelle zwischengeschaltet. Diese wurde mit der Abbildung der technischen Architektur (Kapitel 10.1.2) als Basissystem Interface eingeführt.

Für jeden herstellerspezifischen Basissystemtreiber muss somit lediglich eine Ableitung dieses Interface programmiert werden. Diese Ableitung des Interface enthält die spezifischen Methoden des Basissystems (wie z. B. apiJob). Der Control Import Manager ruft hingegen lediglich die Methode des Interface auf. Die Methodennamen des Interface heißen bei allen Ableitungen des Interface gleich. Dadurch kann unabhängig davon, wie viel herstellerspezifische Basissysteme integriert werden, der Code des Control Import Manager gleich bleiben.

Folgende Abbildung zeigt einen Ausschnitt der Ableitung des Basissystem-Interface für das Basissystem des Kooperationspartners:

```
public class KoopBasissystem : BasissystemI {

public int BasissystemI.mfn_Job(string pchEcu,string  pchJob,string pchAttr,string
pchRes, int nTimeOut)
{
       ...
       API.apiJob(pchEcu, pchJob,pchAttr,pchRes);
       ...
}
```

Abbildung 10.22 Quellcode: Basissystem Interface Ableitung

Die Abbildung zeigt den Aufruf der Methode apiJob des Basissystem-Wrappers API und die Ableitung der Klasse vom Basissystem-Interface (BasissystemI). Im Control Import Manager findet sich lediglich der Aufruf der Methode BasissystemI.mfn_Job. Mit dem Aufruf der Methode mfn_Job wird der Vorgang eingeleitet, den Befehl an das Steuergerät zu schicken.

Damit wird der eine Teilbereich der elementaren Viewer-Funktionalität für den Einsatz in der Diagnosetechnik bereitgestellt. Wie die Funktionalität des Control Import Manager mit der SVG-Technologie verknüpft wird, wird in Kapitel 10.4.3 erläutert. Hierfür soll zunächst die Integration des SVG-Formats in den Viewer näher betrachtet werden.

10.4.2 Einbindung des SVG Plug-ins

Neben dem Zugriff auf das Basissystem ist die Verarbeitung des SVG-Formats gefordert. Dafür muss das eSVG-Plug-in in die Viewer-Anwendung eingebunden werden. Wie mit der technischen Architektur angeführt, stellt das eSVG-Plug-in eine COM-DLL mit grafischer Ausgabe dar. Diese sind typischer Weise als ActiveX-Controls realisiert.

Die Einbindung eines ActiveX-Controls in das .Net Framework ist mit überschaubarem Aufwand möglich. Es muss wie bei den klassischen COM-Objekten ein Wrapper geschrieben werden, der dann als grafisches Objekt in die Oberfläche eingebunden und direkt angesprochen werden kann. Die Implementierung eines solchen Wrappers wurde in Kapitel 10.4.1 bereits erläutert. Oft werden diese Wrapper für das .Net Framework bereits mit der DLL zusammen ausgeliefert. Auch der Anbieter Intesis stellt für den Einsatz im .Net Framework einen Wrapper zur Verfügung.

Folgender Codeausschnitt erläutert die Integration des ActiveX-Controls in den Desktop Viewer unter Zuhilfenahme des Wrappers:

```
private System.Windows.Forms.Panel panel1;
private System.IntPtr esvgControl1=IntPtr.Zero;
...
eSVGInterface.svg_initialize(false);
RECT rect;
GetWindowRect(panel1.Handle, out rect);
Rectangle rec=new Rectangle(rect.left+6, rect.top+6, rect.right-rect.left-12,
rect.bottom-rect.top-12);
Rectangle clientRect = RectangleToClient(rec);
esvgControl1 = eSVGInterface.svg_createControl(this.Handle, clientRect.Left, clien-
tRect.Top, clientRect.Right, clientRect.Bottom);
```

Abbildung 10.23 Quellcode: Basissystem eSVG Integration .Net Framework

Die Klasse eSVGInterface gehört zum Namensraum Intesis.eSVG und enthält eine statische Methode zum Initialisieren des ActiveX-Controls. In dem Ausschnitt wird gezeigt, wie das eSVG-Wrapper-Interface auf ein Panel abgebildet wird. Ein Panel ist ein .Net-Element zur grafischen Ausgabe. Nach dieser Zuweisung kann auf das eSVG-Control zugegriffen und in der Anwendung weiterverarbeitet werden.

Im Compact Framework stellt sich die Integration etwas problematischer dar. Das .Net Compact Framework stellt eine funktionell reduzierte Version des .Net Frameworks dar. Die Gründe dafür wurden in Kapitel 9.3.3.3 erläutert.

Ein Defizit ist die fehlende unidirektionale Kommunikation mit COM- bzw. ActiveX-Objekten. Wie in Kapitel 10.4.1 vorgestellt, erfolgt die Integration des COM-Objekts im Wrapper über die Methode PInvoke. Über diese Methode können an eine Funktion des COM-Objekts Werte übergeben werden. Dadurch kann die Funktion Daten an den Wrapper aber auch Daten von dem Wrapper an die .NET-Anwendung zurückgeben. Allerdings unterstützt PInvoke im .NET Compact Framework, im Gegensatz zum vollständigen .Net Framework, keine

Rückrufe. Im .NET Framework werden Rückrufe über Delegates (objektorientierte Funktionszeiger) an die COM-Funktion übergeben. Die COM-Funktion ruft dann den Wrapper über die Adresse des Delegaten auf, sobald ein Ergebnis des Funktionsaufrufs vorliegt[264]. So kann das COM-Objekt Rückrufe an die .Net-Anwendung schicken. Diese Rückruffunktionen sowie die damit verbundene Ereignisverarbeitung sind allerdings elementar für Interoperabilität mit ActiveX-Controls. Über den Rückruf steuert das ActiveX-Objekt z. B. Veränderungen auf der grafischen Oberfläche. Da dies im Compact Framework nicht unterstützt wird, können ActiveX-Controls nicht über die herkömmliche Art mit PInvoke integriert werden. Die fehlende Rückruf- und Ereignisbehandlung muss auf komplexe aufwendige Weise kompensiert werden. Hierfür gibt es, laut der Microsoft .Net Compact Framework Core Reference, weder standardisierte Verfahren noch Garantien für den tatsächlichen Erfolg[265].

Da die Darstellung des SVG-Formats im Viewer und somit die Einbindung des ActiveX-Controls elementar für die Funktionalität sind, wurde eine alternative Integrationsmöglichkeit gewählt. Mit dem Einsatz der CFCOM-Technologie von Odyssey ist es möglich, ActiveX-DLLs voll funktionsfähig in das Compact Framework einzubinden und zu verarbeiten.

Odyssey stellt hierzu ein CFCOM-Control bereit. Dieses wird wie jedes andere Objekt in .Net auch initialisiert und bekommt danach eine Verlinkung auf das jeweilige ActiveX-Objekt zugewiesen. Nach der Zuweisung kann das ActiveX-Control über das Objekt CFCOM-Control direkt angesprochen und verarbeitet werden. Die weitere Verarbeitung ist dann der im .Net Framework unter der Verwendung eines .Net Framework-Wrappers ziemlich ähnlich. Deshalb spricht Odyssey bei seiner Technik von einem automatisierten Wrapping für das Compact Framework[266].

Die Verwendung reduziert den Programmieraufwand gewaltig und erhöht auch die Zuverlässigkeit des Einbaus. Für die Initialisierung des eSVG-ActiveX-Control mit CFCOM-Control sind, wie beschreiben, im Quellcode lediglich folgende Schritte notwendig.

```
private Odyssey.CFCOM.ComControl esvgControl1;

esvgControl1 = new Odyssey.CFCOM.ComControl()
esvgControl1.ProgID = "ACTIVEXESVG.ActiveXeSVGCtrl.1";
esvgControl1.Size = new System.Drawing.Size(316, 165);
esvgControl1.ComEvent +=
       new Odyssey.CFCOM.ComEventHandler(esvgControl1_ComEvent);
```

Abbildung 10.24 Quellcode: Basissystem eSVG Integration .Net Compact Framework

Der Ausschnitt zeigt neben dem Verweis auf das eSVG-Control auch die Handhabung der ActiveX-Ereignisse, die auch vollständig unterstützt werden.

Die Voraussetzung, dass CFCOM für ein ActiveX-Control eingesetzt werden kann, ist die Unterstützung der IUnknown- oder IDispatch-Schnittstelle. Dies sind Schnittstelle, mit denen die Funktionalität und der Lebenszyklus von COM-Objekten verwaltet wird[267]. Über diese Schnittstellen wird auch in der Core Reference von Microsoft vorgeschlagen, zu versuchen, ActiveX

[264] Vgl. Wigley 2003, S. 715
[265] Vgl. Wigley 2003, S. 732
[266] Vgl. CFCOM Manual 2003 (HTML Dokument)
[267] Vgl. Templeman 2003, S. 106

einzubinden. Mit CFCOM wird dieser Weg automatisiert. Das Resultat ist ein Wrapper mit Rückruffunktion für das ActiveX-Control, ein sogenannter Callable Wrapper (CCW)[268].

Das Ergebnis dieses Kapitels 10.4.2 ist, dass im .Net Framework und im Compact Framework ein Weg gefunden wurde, das eSVG-ActiveX-Control einzubinden und auf ähnliche Weise weiterzuverarbeiten. In Kapitel 10.4.1 wurde die Bereitstellung der Steuergerätefunktionalität beschrieben. Beide funktionalen Bereiche gilt es nun für die Diagnosetechnik zusammenzuführen. Diese Verknüpfung macht die zentrale Verarbeitungslogik im Viewer aus. Sie ist Thema des Kapitels 10.4.3.

10.4.3 Zentrale Verarbeitungslogik

Bei der Beschreibung der Anwendungsschicht des Viewers in Kapitel 10.1.2.3 wurde erläutert, dass die Verbindung des SVG-Grafikformats mit dem Basissystem den funktionalen Kernbereich des Viewers ausmacht.

Es wurden zwei Klassen implementiert, die sich dieser Anforderung annehmen. Dies sind der Control Import Manager für die Verwaltung des Steuergerätezugriffs und der SVG Import Manager für die Verwaltung der SVG-Verarbeitung. Aufgrund der Verwendung von Interfaces konnten die beiden Verwaltungsklassen plattformunabhängig programmiert werden. Der Quellcode konnte also wieder verwendet werden und ist in nahezu homogener Form im Desktop Viewer und im Mobile Viewer wiederzufinden. Dass dieser Vorteil entscheidend war, für den Einsatz der .Net-Technologie, wurde bereits vertieft.

Während der Laufzeit befinden sich beide Klassen initialisiert im Speicher der Anwendung (Session) und greifen von dort gegenseitig aufeinander zu. Durch diese Interoperation werden die notwendigen Anforderungen für die Diagnosetechnik abgedeckt.

Wird z. B. auf der SVG-Oberfläche ein Ereignis ausgelöst (Klick auf den Button für linken Blinker), dann bewirkt das Ereignis, dass der SVG Import Manager den Control Import Manager anstößt, den jeweiligen Befehl an das Fahrzeug weiterzugeben. Den Befehl bekommt er vom SVG Import Manager, der ihn aus dem Diagnoseskript ausliest. Ein Beispiel für ein Diagnoseskript wurde mit der Beschreibung der Editor-Implementierung angeführt (siehe Kapitel 10.3.5).

Gleiche Interoperabilität ist natürlich auch in umgekehrter Richtung möglich. Ist z. B. eine Diagnoseoberfläche geladen, in der die Drehzahl des Motors angezeigt wird, so greift der Control Import Manager auf den SVG Import Manager zu. Der Control Import Manager fragt zyklisch die Steuergerätezahl ab. Sobald sich diese geändert hat, ruft er bei dem SVG Import Manager die Methode auf, den Wert entsprechend zu ändern. Der SVG Import Manager ändert daraufhin die Oberfläche, im Beispiel Motordrehzahl den Zahlenwert der Drehzahl und den Neigungswinkel der Gradanzeige.

Anhand eines solchen Anwendungsfalls soll abschließend erklärt werden, wie sich die Manager-Interoperation im Quellcode widerspiegelt. Ausgegangen wird von einer SVG-Aktion, die einen Steuergerätebefehl auslösen soll. Der Anwendungsfall basiert auf dem Beispiel, dass der Diagnostiker den Button für das linke Fernlicht gedrückt hat.

[268] Vgl. Templeman 2003, S. 105

Das Klicken löst ein SVG-Event aus. Der SVG Import Manager ist so strukturiert, dass er die verschiedenen Ereignistypen verarbeiten kann. Aus dem Event kann der SVG Import Manager das zugehörige SVG-Objekt ableiten. Die Events sind für die Versionen .Net Framework und Compact Framework anhand einer gemeinsamen Schnittstelle strukturiert. Dadurch muss der Import Manager nicht unterscheiden, ob das Event in dem CFCOM-Control von Odyssey (Compact Framework) oder von dem Intesis-eSVG-Wrapper (.Net Framework) ausgelöst wurde. In dem abgeleiteten SVG-Objekt, das auf den gedrückten Knopf verweist, sind die Informationen hinterlegt, welcher Job mit welchen Parametern ausgeführt werden soll. Diese Informationen werden an den Control Import Manager weitergegeben.

Folgende Abbildung veranschaulicht den Ablauf im Quellcode:

```
private void esvgControl1_Callback1(
    SVGImportManager.SVGElement svgElement  ) {
...
string objId = svgElement.getAttribute(ControlImportManager.ATTR_CB_TARGET);
string svgId = svgElement.getAttribute(ControlImportManager.ATTR_ID);
string[] dbs = esvgControl1_GetDBs(svgElement);
...
bool sentJob=false;
if ( dbs!= null && dbs.Length>0 ) {
    for(int i=0; i<dbs.Length; i++ ) {
       if (dbs[i]==null || dbs[i].Length<=0)
              continue;
       string job=svgElement.getAttribute(ControlImportManager.UNIT_PREFIX_JOB+...);
       string res=svgElement.getAttribute(ControlImportManager.UNIT_PREFIX_RES+...);
       string arg=svgElement.getAttribute(ControlImportManager.UNIT_PREFIX_ARG+...);
       Hashtable argTable = ControlImportManager.parseArray(arg);
       string argValues = null;
       if (argTable.Count>0) {
              ArrayList al = new ArrayList(argTable.Values);
              argValues = (string)al[al.Count-1];
       }
       if ( ControlImportManager.mfn_Job(dbs[i],job,...)
              sentJob=true;
    }
    if (sentJob) {
       ResultSet rs = new ResultSet();
       if ( ControlImportManager.mfn_GetResultSet(true,rs,...)
                  ==(int)BasissystemI.EErrors.eOk )
              MessageBox.Show(BasissystemI.EErrors.eOk...);
       else
              SVGImportManager.SwitchToSuccess(objId, svgId);
    } else
       MessageBox.Show(ControlImportManager.mfn_GetLastErrorText() ,
              "Driver Error",...);
}
```

Abbildung 10.25 Quellcode: Behandlung eines SVG-Ereignisses im SVG Import Manager

Mit dem ausgelösten Event Callback1 wird das zugehörige SVG-Objekt svgElement übergeben. Aus dem Objekt svgElement kann der SVG Import Manager über getAttribute die Bezeichnungen für Job, Attribute und Ergebniswerte abfragen.

Mit den ausgewerteten Attributen kann über den Control Import Manager der Job ausgeführt werden. Hierzu wird die in Kapitel 10.4.1 vorgestellte Schnittstellenmethode mfn_Job ausgeführt. Dadurch wird der Aufruf intern an das initialisierte Basissystem weitergeleitet. Mit dem Aufruf ControlImportManager.mfn_GetResultSet können die Ergebnisse des Aufrufs ausge-

wertet werden. Konnte der Job erfolgreich ausgeführt werden, dann wird dies dem Benutzer angezeigt. Es wird die Methode SVGImportManager.SwitchToSuccess aufgerufen. Im vorliegenden Beispiel erhält der Knopf eine entsprechende Färbung.

Bei dem Anwendungsfall ging die Initiative von einem Ereignis innerhalb des SVG Import Manager aus. Dieser hat daraufhin den Control Import Manager aktiviert. Wie erwähnt verläuft die Zusammenarbeit auch in die andere Richtung. Mögliche Beispiele wie das Ändern von Steuergerätewerten oder Zuständen wurden bereits genannt. Die Zusammenarbeit dieser beiden Klassen stellt die zentrale Logik in der Viewer-Software dar und liefert die Basis für die Diagnosefunktionalität der Anwendung.

Anhand der Erläuterung der technischen Architektur und der Implementierung wurde gezeigt, dass das Konzept dieser Arbeit softwaretechnisch realisierbar ist. Seine tatsächliche Umsetzung in eine einsatzfähige Software soll in Form eines Prototyps für eine Pilotprojektierung erfolgen. Auf diese Inhalte wird im nächsten Kapitel eingegangen.

11 Projektierung und Leistungsbewertung auf Basis des Prototyps

Mit der Architekturbeschreibung und dem Beleg der programmiertechnischen Umsetzbarkeit des neuen Konzepts aus Kapitel 9 und 10 wurde an den Kooperationspartner herangetreten. Es wurde die Durchführung einer testweisen Pilotprojektierung beschlossen. Die einzelnen Schritte der Durchführung werden in Kapitel 11.1 erläutert.

Im Rahmen des Pilotprojekts wurde ein Prototyp auf Basis des neuen Konzepts entwickelt. In Kapitel 11.2 wird der Editor- und Viewer-Prototyp vorgestellt. Unterstützt durch Screenshots wird deren Funktionalität erläutert.

Um die Leistung des Prototyps zu präsentieren und diese zu beurteilen, wird die Software beim Kooperationspartner integriert und eingesetzt. Das Kapitel 11.3 behandelt die dafür notwendigen Schritte.

Die Projektierung dient vordergründig dem Beleg, dass die mit der Arbeit gesetzten Ziele erreicht wurden. Hierfür werden zum einen die Fähigkeiten des Prototyps mit dem Anforderungskatalog verglichen (Kapitel 11.4.2). Ferner werden Messungen bei verschiedenen Anwendungsfällen durchgeführt. Es werden zu diesem Zweck exemplarisch Tätigkeiten mit dem neuen Diagnosesystem durchgeführt und ausgewertet (Kapitel 11.4.3).

Der Vergleich mit den Anforderungen und die Auswertung der Projektdaten sollen den theoretischen und praktischen Nachweis für den Erfolg der Arbeit erbringen (Kapitel 11.4.4).

11.1 Vorgehen bei der Projektierung

Mit einer Pilotprojektierung soll ein neues System unter realitätsnahen Bedingungen in seiner zukünftigen Einsatzumgebung getestet werden. Damit wird ermittelt, ob das System den Anforderungen des Anwendungsbereichs entspricht. Gerade bei komplexen Systemen, die an

mehreren Standorten oder in Geschäftsbereichen eingesetzt werden sollen, ist eine Pilotprojektierung fast unverzichtbar[269].

Außerdem kann durch einen Pilotbetrieb die Akzeptanz des neuen Systems auf ein hohes positives Niveau gebracht werden. Die Arbeit mit der neuen Anwendung ist vor Ort präsentierbar und Kritiken bzw. Verbesserungsvorschläge der Nutzer können direkt aufgenommen werden.

Da es sich im vorliegenden Fall um eine Neuentwicklung handelt, die ihren Ersteinsatz im Rahmen des Testprojekts hat, ist ungewiss, in welchen Maße Nachbesserungen notwendig sind und Funktionalität noch ergänzt werden muss. Auch kann ein Pilotprojekt bei unbefriedigenden Bewertungsergebnissen zum Abbruch der Softwareintegration führen. In Pilotprojektierungen, in denen der endgültige Funktionsumfang und der tatsächliche Einsatz noch nicht feststehen, bietet sich die Entwicklung eines Prototyps an. Dadurch können früh Missverständnisse und/oder Anforderungstendenzen ausgemacht werden und noch in die Software einfließen. Außerdem werden bei einem Abbruch die Kosten der Herstellung einer verkaufsreifen Vollversion gespart[270].

Eine Methode, die das Entwickeln von Prototypen ermöglicht und darüber hinaus weitere Vorteile zur Qualitätssicherung bietet, ist die inkrementelle (oder evolutionäre) Softwareentwicklung.

Bei der inkrementellen Softwareentwicklung erfolgt die Leistungserbringung über ein Spiralmodell. Wie in Form einer Spirale führt der Zyklus des Modells wiederkehrend durch die einzelnen Phasen. Die Phasen sind Planung, Entwicklung, Integration und Qualitätssicherung. Jeder Durchlauf der Phasen repräsentiert einen Prototyp. In jeder Phase wird der Prototyp geplant, entwickelt, beim Nutzer eingesetzt und bewertet. Auf diese Weise werden nach und nach (inkrementell) Prototypen entwickelt, die sich den Vorstellungen des Nutzers anpassen und die geforderten Zielsetzungen erfüllen[271].

Die finale Zielsetzung des Kooperationspartners ist, mit dem Pilotprojekt ein Roll-out-fähiges System zu erhalten. Der Roll-out beschreibt die Bereitstellung der neuen Anwendung in den verschiedenen Einsatzbereichen[272]. Das System ist demnach reif, in den Produktivbetrieb übernommen zu werden, und verlässt aus Sicht der Versionierung den Status eines operationellen Prototyps in eine Vollversion[273].

Dieser Reifegrad ist allerdings nicht mehr Zielsetzung der Arbeit. Für die Bereitstellung einer produktiv verwendbaren Software in der Fertigung eines Automobilkonzerns ist professionelle Softwareentwicklung notwendig. Hierbei geht es neben politischen Erwägungen für die Auswahl des Softwarezulieferers unter anderem auch um die Bereitstellung von dauerhaften Serviceleistungen[274].

Deshalb liegt mit dem Thema der Arbeit "Entwicklung und Einsatz einer innovativen HMI-Software" die Zielsetzung nicht in einem produktiven Vertrieb, sondern in der Schaffung eines Systems, das durch neue Ansätze (Innovationen) das Potenzial für Verbesserung bietet. Dieses

[269] Vgl. Heilmann 2003, S. 184
[270] Vgl. Forbrig 2004, S. 37
[271] Vgl. Thaller 2000, S. 44
[272] Vgl. Heilmann 2003, S. 185
[273] Vgl. Thaller 2000, S. 45
[274] Vgl. Heilmann 2000, S. 188

Potenzial soll mit der ersten Prototypphase der Pilotprojektierung belegt werden. Ein Ausblick auf die Entwicklung in den weiteren Phasen wird mit Kapitel 12.2 gegeben.

Als Zielsetzung für die erste Prototypphase wurde zum einen ein Beleg für die Erfüllung des Anforderungskatalogs festgelegt. Es kommt hierbei die Methode der analytischen Evaluation zum Einsatz[275]. Des Weiteren sollte der praktische Einsatz den Grad der Verbesserung zeigen. Das geforderte Niveau wurde für diese frühe Prototypphase noch nicht deklariert, aber anhand der Prozesszeitmessung innerhalb von Anwendungsfällen soll belegt werden, dass das neue Diagnosesystem Einsparungspotenzial aufweist. Für die Softwarebewertung anhand von Anwendungsfällen bietet sich die experimentelle Evaluation an. Sie basiert auf kontrollierten Experimenten, die z. B. Anwendungsfälle sein können. Ein positives Resümee der analytischen und experimentellen Evaluation würde die zweite Entwicklungsphase des Pilotprojekts einleiten. Die Bewertung und das Ergebnis werden in Kapitel 11.4 erläutert.

Als Voraussetzung für die erste Pilotphase muss die Implementierung aus Kapitel 10 in einen Prototyp überführt werden. Das softwaretechnische Produkt der Implementierung wird als MobiViSt bezeichnet. Es sollen damit die drei Ansatzpunkte der Arbeit, nämlich Mobilität, Grafik und Funktionalität widergespiegelt werden. Bei dem Akronym stehen die ersten vier Buchstaben "Mobi" für Mobilität, "Vi" für Visualisierung im Zusammenhang mit den grafischen Herausforderungen und als primäre Funktionalität steht "St" für das Steuern der elektronischen Steuergeräte. Die Softwarekomponenten von MobiViSt werden in Kapitel 11.2 vorgestellt.

Um MobiViSt einer entsprechenden Bewertung zu unterziehen, sind die für das Pilotprojekt üblichen realitätsnahen Bedingungen zu schaffen. Die Software wird also in abgegrenzten Bereichen integriert. Innerhalb der Bereiche werden explizite Anwendungsfälle ausgewählt, die mit dem bisherigen System und dem neuen System für einen späteren Datenvergleich bearbeitet werden. Als Bereiche wurden die Verwaltung und die Nacharbeit bestimmt. In diesen Bereichen findet das Diagnosesystem die häufigste Anwendung. Die Kombination beider Bereiche deckt außerdem ein breites Anwendungsspektrum des Systems ab. Im Rahmen der notwendigen Eingrenzung können die Verwaltung und Nacharbeit deshalb am ehesten repräsentativ für die übrigen Geschäftsprozesse ausgewählt werden. Die Verwaltung dient der Bewertung des Editors und die Nacharbeit der Bewertung des Viewers. Für die Integration ist es z. B. notwendig, Hardware für die Installation der Software bereitzustellen und unter anderem Diagnoseskripte für den Einsatz in der Nacharbeit anzufertigen. Die Integrationsleistung wird in Kapitel 11.3 näher erläutert.

11.2 Funktionsweise des Prototyps

Im folgenden Kapitel wird die Funktionsweise des Prototyps beschrieben. Die vorgestellte Version V0.1 repräsentiert den Stand, der mit der ersten Phase des inkrementellen Softwareentwicklungszyklus erstellt wurde. Prototypphasen werden üblicherweise mit einer 0er Version gekennzeichnet.

Das Kapitel erhebt nicht den Anspruch einer umfassenden Bedienungsanleitung, es soll vielmehr die Arbeitsweise mit dem Prototyp verdeutlichen. Bei der Beschreibung der Integrationsarbeit und der abschließenden Evaluation des Prototyps wird auf die Arbeitsweise Bezug ge-

[275] Die Techniken und Methoden der Evaluation werden in Kapitel 11.4.1 ausführlich erläutert

nommen. Das Ziel der Präsentation liegt deshalb darin, eine anschauliche Basis für die Erläuterungen der Folgekapitel 11.3 und 11.4 zu liefern.

Entsprechend dem Konzept gliedert sich das Diagnosesystem in zwei zentrale Softwarekomponenten auf. Zunächst wird die Entwicklungsumgebung erläutert. Sie wird im Projekt als Editor bezeichnet. Anschließend wird die Funktionsweise des Viewers erläutert. Der Viewer repräsentiert im Kontext der Diagnosetechnik die HMI-Software.

11.2.1 Bedienkonzept des Editors

Der Editor ist als Eclipse-Plug-in realisiert. Er integriert sich somit in die Eclipse-Entwicklungsumgebung. Über dessen Oberfläche erfolgt demnach die Bedienung.

11.2.1.1 Übersicht der Benutzeroberfläche

Nach der Installation des Plug-ins in eine Standard-Eclipse-Umgebung wird diese um zusätzliche Fenster ergänzt bzw. bestehende Fenster in ihrer Funktionalität erweitert.

Die folgende Abbildung fasst die Fenster zusammen, die für Nutzung im Diagnosebereich relevant sind:

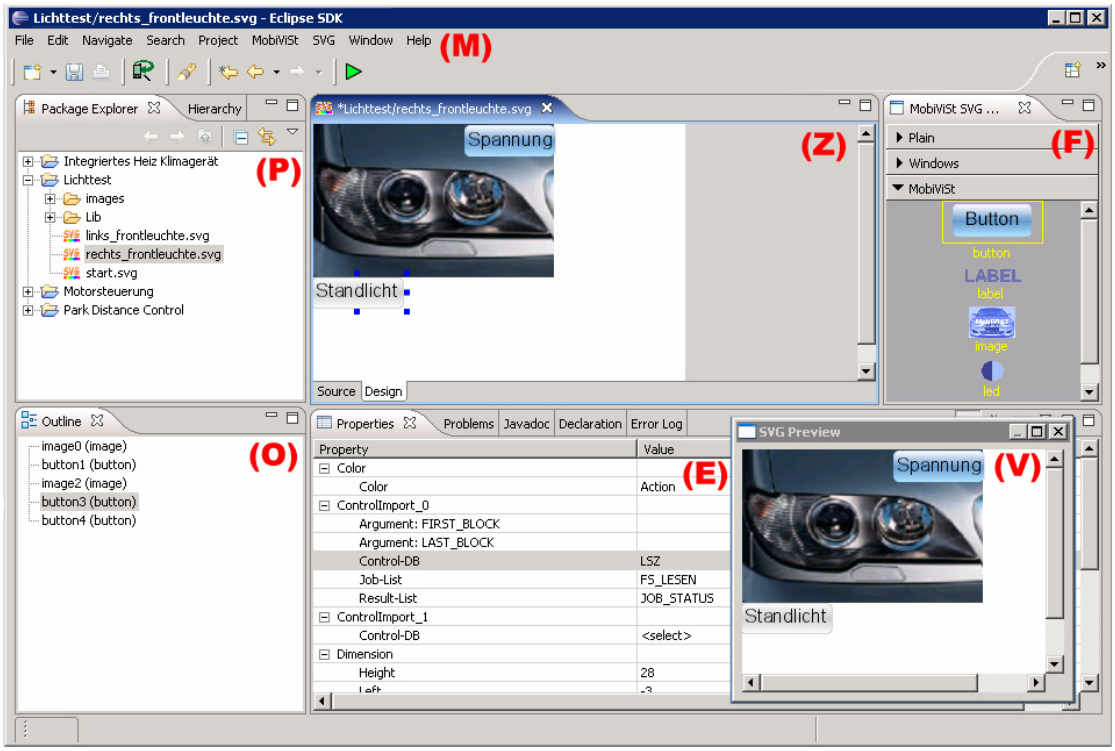


Abbildung 11.1 Benutzeroberfläche des Editors

Die Oberfläche zeigt folgende Abschnitte:

- Zentrale Zeichenoberfläche (Abschnitt Z in Abbildung)
- Auswahlliste mit Diagnoseformen und Zeichenwerkzeugen (Abschnitt F)
- Vorschaufenster zur funktionalen Überprüfung der Diagnosemaske (Abschnitt V)

- Eigenschaftsfenster mit Zuweisungsmöglichkeit der Steuergerätebefehle (Abschnitt E)
- Objektnavigator zur Auswahl der Elemente der Zeichenoberfläche (Abschnitt O)
- Projektnavigator mit den Verzeichnisinhalten (Abschnitt P)
- Menüleiste mit einer Werkzeugleiste zur Direktauswahl (Abschnitt M)

Bis auf die Menüleiste sind alle Abschnitte als Fenster realisiert. In der Eclipse-Umgebung werden diese Fenster als Views (dt. Ansichten/Fenster) bezeichnet. Die Fenster können beliebig angeordnet, in ihrer Größe verändert und ausgeblendet werden. Der Benutzer ist auf diese Weise flexibel, die äußere Form der Benutzeroberfläche selbst zu arrangieren. Das Vorschaufenster (Abschnitt V) oder diverse Dialogformulare, die noch vorgestellt werden, sind nicht als Views in der Oberfläche verankert, sondern werden als externe Fenster außerhalb der Editorumgebung gestartet.

Die Fenster und deren Funktionen werden im Folgenden kurz erläutert. Die Reihenfolge der Beschreibung erfolgt entsprechend ihrer Nutzung bei der Erstellung eines Diagnoseskripts. Damit soll der rote Faden des Bedienkonzepts verdeutlicht werden.

11.2.1.2 Verwaltung eines Projekts

Wird der Editor erstmalig gestartet, muss ein Speicherort für die künftigen Projekte bestimmt werden. Dieser Speicherort ist in der Verwaltung vorzugsweise ein Netzwerkpfad. Auf diese Weise wird sichergestellt, dass alle Skriptprogrammierer auf die gleichen aktuellen Skripte zugreifen und diese bearbeiten.

Nach dem Bestimmen des Speicherorts muss ein erstes Projekt angelegt werden. Ein Projekt umfasst im Eclipse-Umfeld Dateien und Verzeichnisse. Welche inhaltliche Bedeutung ein Editorprojekt hat, kann durch jeden Kfz-Hersteller selbst definiert werden. Im Pilotbetrieb, soll ein Projekt eine Fahrzeugtechnologie, wie z. B. die Lichttechnik, beinhalten. Innerhalb des Projekts finden sich die Diagnoseskripte, Bilder und Bibliotheken wieder. Deren Notwendigkeit wurde in Kapitel 10.1 erläutert. Um ein Diagnoseprojekt bzw. ein Skript anzulegen, erweitert das Prototyp-Plug-in die Eclipse-Umgebung um spezifische Dialoge für den Diagnosebereich. Nach dem Anlegen eines Diagnoseprojekts erscheint im Projektnavigator (Abschnitt P, Abb. 11.1) das Projekt mit dem jeweiligen Technologienamen und einem Startskript. Die Abbildung zeigt den Dialog zum Anlegen eines neuen Projekts und den Navigator.

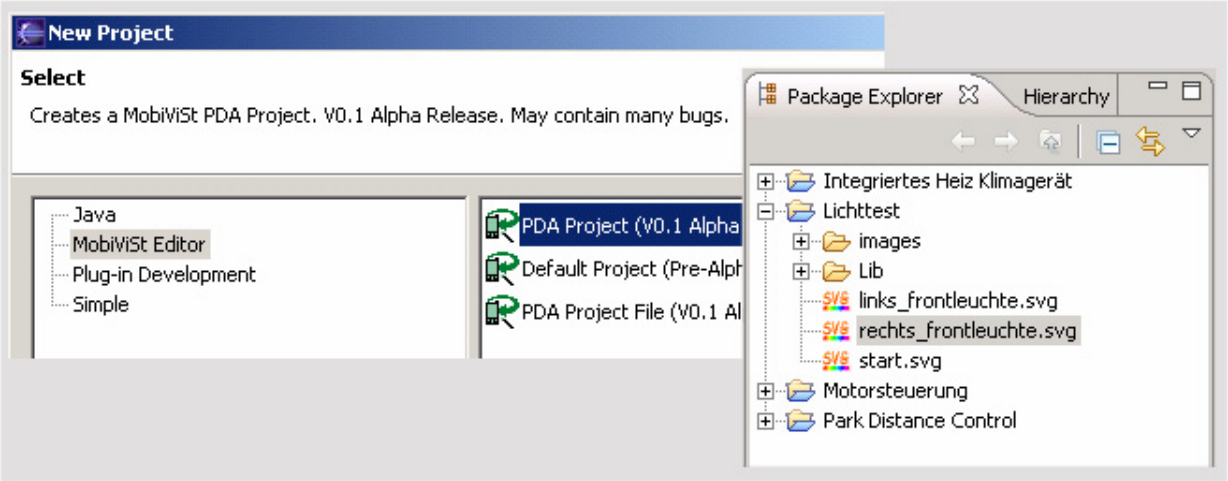


Abbildung 11.2 Verwaltungsoberflächen des Editors

Neben dem Startskript können noch beliebig weitere Diagnoseskripte einem Projekt hinzugefügt werden. In ähnlicher Form kann ein Skript auch mehrere Masken enthalten. Für die Erstprojektierung enthält aus Gründen der vereinfachten Wartung jedes Skript nur eine Maske. Nach dem Öffnen des Skripts per Doppelklick wird die Maske auf der Zeichenoberfläche (Abschnitt Z, Abb. 11.1) dargestellt und kann bearbeitet werden.

11.2.1.3 Gestaltung einer Maske

Die Zeichenoberfläche (Abschnitt Z, Abb. 11.1) unterteilt sich in die Reiter Design und Source (dt. Quelle). Im Design-Modus wird die Maskenoberfläche in der Form grafisch dargestellt, wie sie auch im Viewer erscheint. Auf den Source-Modus wird in Kapitel 11.2.1.5 noch weiter eingegangen.

Aus dem Auswahlfenster für Diagnoseformen (Abschnitt F, Abb. 11.1) können Formen zur Gestaltung der Maske ausgewählt werden. Die Auswahl besteht aus Standard-Zeichenwerkzeugen und Diagnoseformen. Die Zeichenwerkzeuge sind übliche Gestaltungsformen wie Linie, Rechteck, Kreis, Textausgabe etc. Die Diagnoseformen enthalten vorgefertigte Elemente, in denen unter anderem Steuergerätefunktionalität hinterlegt werden kann. Hier bietet der Editor als Standard Button, Label, Image, LED und Gradient an. Diese Grundbedürfnisse können um beliebig weitere Elemente ergänzt werden. Das zugrunde liegende Konzept wurde bei der Implementierung verdeutlicht (siehe Kapitel 10.3).

Die Diagnoseformen können wie bei handelsüblichen Editoren aus dem Auswahlfenster in die Zeichenoberfläche gezogen oder per Mausklick punktiert an einer Stelle erzeugt werden. Nach dem Einfügen können sie in Form, Farbe und Funktionalität verändert werden. Die Abbildung zeigt das Einfügen eines Buttons (dt. Knopf), der mit Steuergerätefunktionalität hinterlegt wird.

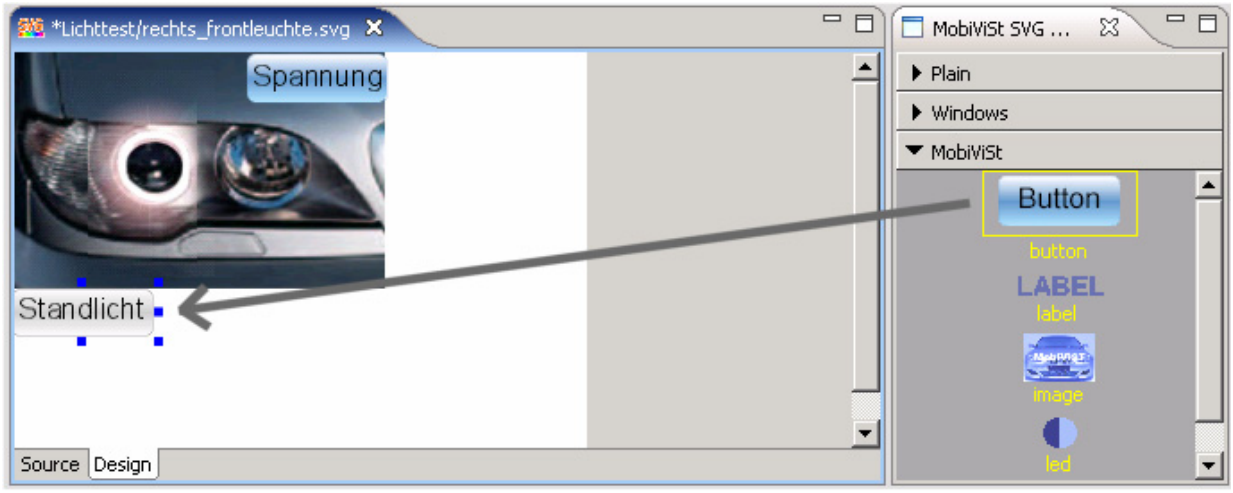


Abbildung 11.3 Gestaltungsoberfläche des Editors

Des Weiteren kann z. B. ein Knopf auch Navigationsfunktionalität erhalten. Die Abbildung zeigt hierfür den Knopf mit der Beschriftung Spannung, der zu einer weiteren Maske (in diesem Fall Diagnoseskript) verzweigt.

Da durch diese Gestaltungsmöglichkeiten, die eigentliche Diagnoseoberfläche durch den Nutzer völlig frei gestaltbar ist, entstehen keine Einschränkungen für die Einhaltung von Richtlinien. So können z. B. die für den Einsatz in der Automatisierungstechnik vorgegebenen VDI-Richtlinien oder ISO-Normen bei der Gestaltung von Oberflächen vollkommen umgesetzt werden. Der Grad der Realisierung von Verordnungen liegt hier in dem Engagement der Projektierung und wird nicht durch die Oberfläche begrenzt.

Wie einer eingefügten Diagnoseform Steuergerätefunktionalität zugewiesen werden kann, wird in Kapitel 11.2.1.4 erläutert.

11.2.1.4 Zuweisung der Steuergerätedaten

Die auf der Zeichenoberfläche dargestellten Objekte können über den Objektnavigator (Abschnitt O, Abb. 11.1) bzw. per Klick auf das Objekt ausgewählt werden.

Nach der Auswahl eines Oberflächenelements werden dessen Eigenschaften im Eigenschaftsfenster angezeigt (Abschnitt E, Abb. 11.1). Um die Arbeit mit dem Editor für die (mit Eclipse vertrauten) Nutzer so intuitiv wie möglich zu gestalten, ist die Zuweisung der Steuergerätefunktionalität in das Eclipse-Eigenschaftsfenster integriert.

Bevor die Steuergerätefunktionen im Eigenschaftsfenster angezeigt werden können, müssen die Steuergerätedaten geladen werden. Die Dateien mit den Steuergerätedaten werden über die Eclipse-Einstellungen importiert. In den Einstellungen wird ein Dialog zur Verfügung gestellt, mit dem die Dateien geladen werden. Folgende Abbildungen zeigen einen Bildschirmausschnitt der Dialogform:

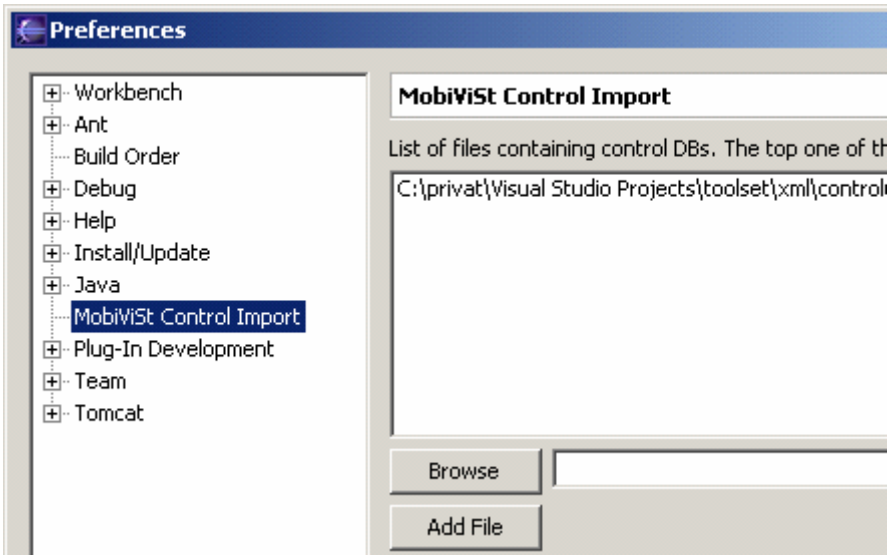


Abbildung 11.4 Einstellungen des Editors

Nach dem Hochladen können im Eigenschaftsfenster die notwendigen Parameter einer Diagnoseform zugewiesen werden. Die Abbildung zeigt das Bespiel eines Buttons, mit dem das Standlicht aktiviert werden soll.

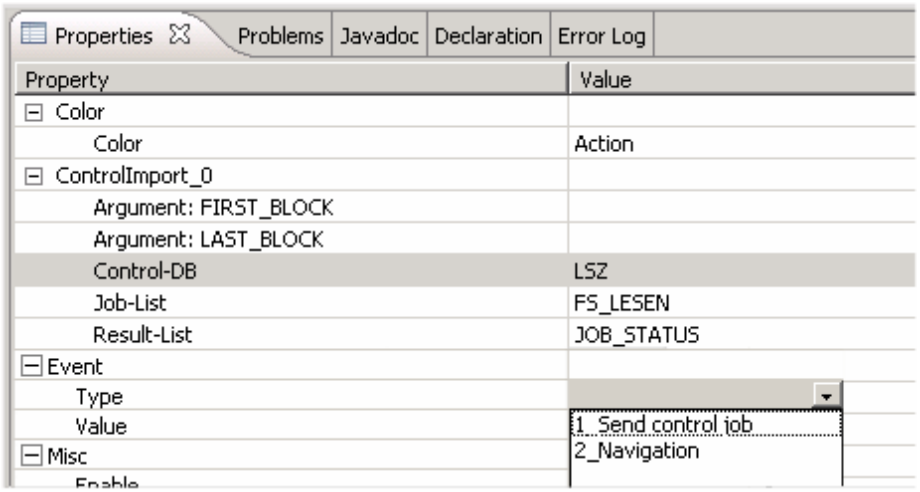


Abbildung 11.5 Eigenschaftsfenster des Editors

Der Benutzer wählt zunächst das zugehörige Steuergerät (Control-DB, siehe Abbildung) aus. Im vorliegenden Fall ist dies das Lichtschaltzentrum (LSZ). Danach stehen ihm alle möglichen Steuergerätebefehle (Abb.: Job-List: FS_LESEN) zur Verfügung. Entsprechend dem Job werden abschließend der Ergebnistyp (Abb.: Result-List: JOB_STATUS) und die Job-spezifischen Aufrufparameter (Abb.: Argument: FIRST_BLOCK und LAST_BLOCK) definiert. Die Auswahl erfolgt per Auswahllisten und ermöglicht so ein schnelles Zuweisen mit geringer Fehlerhäufigkeit.

Jede Diagnoseform kann unterschiedliche Ereignisse erzeugen. So kann z. B. der Button als Ereignistyp (Event-Type) einen Steuergerätebefehl oder ein Navigationsereignis auslösen. Beim Gestalten der Maske wird der Ereignistyp zugewiesen, der später zur Laufzeit ausgelöst

wird. Beim Standlicht-Button ist es das Auslösen des Steuergerätebefehls (Bsp. FS_LESEN). Über das Vorschaufenster kann die Ereignisverarbeitung getestet werden.

11.2.1.5 Test und manuelle Nachbearbeitung

Das Vorschaufenster (Abschnitt V, Abb. 11.1) wird über die Menüleiste (Abschnitt M, Abb. 11.1) mit der grünen Rechteck-Taste geöffnet. Die Vorschau dient der Überprüfung der Ereignisverarbeitung. Die Oberflächengestaltung kann bereits während der Bearbeitung auf der Zeichenoberfläche kontrolliert werden (WYSIWYG-Konzept, siehe Kapitel 7.3.4).

In weiteren Prototypphasen ist geplant, mit der Vorschaufunktion direkt den Viewer aufzurufen. In der ersten Realisierungsphase wird hier ein Editor-interner Viewer geöffnet, der aber auch die Möglichkeit bietet, die korrekte Ereignisverarbeitung zu überprüfen.

In den frühen Prototypphasen ist es durchaus möglich, dass beim Test Fehler auffallen, die im Design-Modus der Zeichenoberfläche nicht behoben werden können oder sogar durch diesen Modus verursacht werden. Dafür bietet der Editor die Möglichkeit, die Zeichenoberfläche in den Source-Modus umzuschalten. In diesem Modus kann der Benutzer den Quelltext direkt bearbeiten. Aufgrund der Verwendung von offiziellen Standards, wie SVG und JavaScript, muss für die Bearbeitung keine proprietäre Sprache erlernt werden.

Der Quellcode-Modus ist in späteren Phasen der Projektierung auch dafür vorgesehen, dass die Benutzer funktionale Erweiterungen direkt eingeben können. Dadurch ist die Maskengestaltung nicht auf die vorgegebene Funktionalität eingeschränkt.

Nach dem Ändern des Quellcodes und dem Wechsel in den Designmodus wird die Änderung sofort interpretiert und dargestellt. Auf diese Weise bleibt auch durch den Source-Modus das WYSIWYG-Konzept erhalten.

11.2.2 Bedienkonzept des Viewers

Der Viewer dient der Verarbeitung und Darstellung der Diagnoseskripte. An den Viewer wird der Anspruch gestellt, eine flexible Diagnoseoberfläche zu liefern. Die eigentliche Diagnosefunktionalität erhält der Viewer über die geladenen Diagnoseskripte. Die im Viewer fest verankerte Funktionalität wurde daher auf ein Minimum reduziert. Dies hat zum Vorteil, dass das grundsätzliche Bedienkonzept für den Nutzer leicht zugänglich ist und sich in wenigen Schritten erklären lässt.

11.2.2.1 Übersicht der Benutzeroberfläche

Der Viewer bietet die Möglichkeit das Diagnoseskript unverändert auf mobilen Endgeräten und Desktop- bzw. Tower-PC-Anlagen anzuzeigen. Hierfür werden der Mobile Viewer für das Windows-Mobile-, Windows-CE- und Pocket-PC-Betriebssystem sowie der Desktop Viewer für Windows NT, 2003 und XP bereitgestellt.

Die folgende Abbildung zeigt, dass beide Viewer in Bezug auf ihre Oberfläche den gleichen Aufbau aufweisen. Die Funktionsgleichheit gilt konzeptionell für den gesamten Viewer. Deshalb sind alle folgenden Erläuterungen für den Mobile Viewer wie auch für den Desktop Viewer gültig. Es wird deshalb auf eine Unterscheidung in der Beschreibung fortan verzichtet.

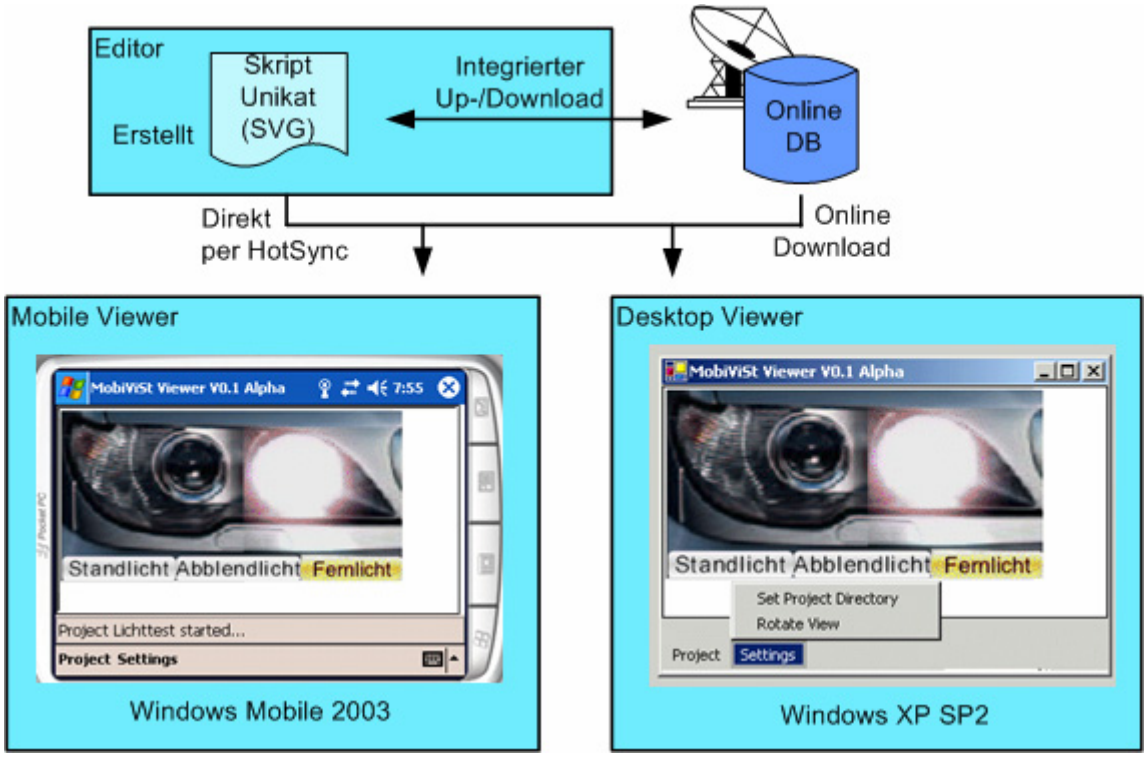


Abbildung 11.6 Benutzeroberflächen des Viewers

Der fest verankerte Bereich der Oberfläche besteht lediglich aus einer Menüleiste. Mit der Menüleiste kann über die Funktion "Project" das Diagnoseprojekt geladen werden (siehe Kapitel 11.2.2.2). Die weitere Funktionsgruppe "Settings" enthält diverse Bedienelemente, die für die Handhabung nützlich sind (siehe Kapitel 11.2.2.3).

Den Hauptbereich der Viewer-Anzeige macht die Darstellung der Diagnosemaske aus. Dieser Bereich gestaltet sich flexibel über den jeweiligen Inhalt des Diagnoseskripts und liefert die eigentliche Diagnosefunktionalität. Die grundlegende Tätigkeit, bevor die Funktionalität der Diagnosemaske genutzt werden kann, ist folglich das Laden des Diagnoseskripts.

11.2.2.2 Laden eines Diagnoseskripts und Navigation

Wie die Abbildung 11.6 der Benutzeroberfläche zeigt, werden die Diagnoseskripte, Bibliotheken und zugehörigen Bilder entweder per Synchronisation (nur für mobile Betriebsysteme), direkt oder von einem Server auf das Endgerät geladen.

Bei der Beschreibung des Editors wurde erläutert, dass gemäß der Eclipse-Technik die Dateien in sogenannten Projekten gespeichert werden. Beim Transfer der Dateien wird die Projektstruktur beibehalten. Anhand der Projektnamen können die Diagnosemasken geladen werden. Dazu muss der Benutzer das Vater-Verzeichnis bestimmen, in das die Projekte kopiert wurden. Die Verzeichnisauswahl erreicht der Benutzer über die Menüleiste und die Funktionsgruppe "Settings".

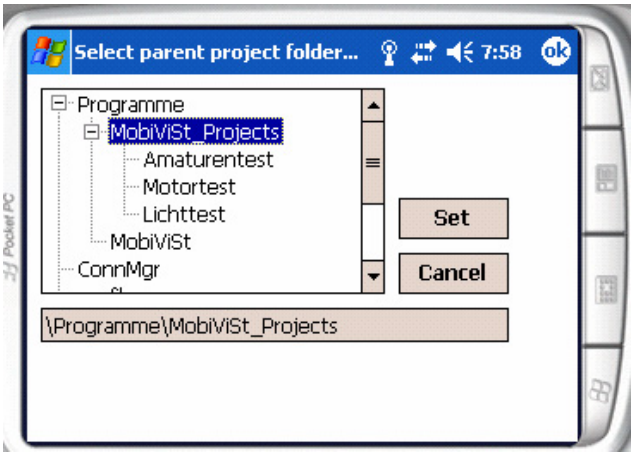


Abbildung 11.7 Verzeichnisauswahl des Viewers (mobil)

Nach dem Bestätigen des ausgewählten Verzeichnisses erscheinen unter dem Menüpunkt "Project" die installierten Projekte. Folgende Abbildung zeigt eine Projektliste, die nach dem Klick auf das Feld "Project" eingeblendet wird.

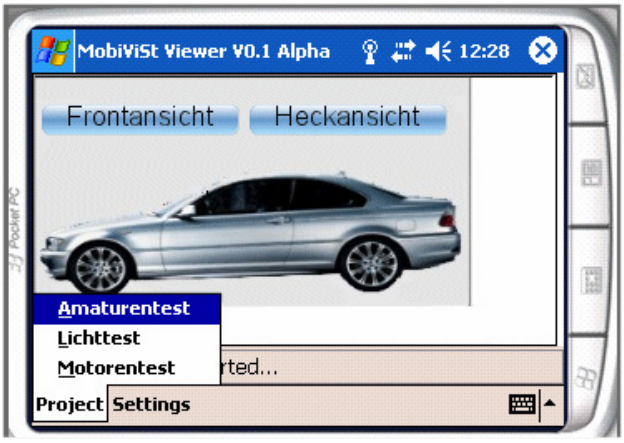


Abbildung 11.8 Projektauswahl des Viewers (mobil)

Nach einem weiteren Klick auf das jeweilige Projekt wird dessen Startmaske geladen. Die Startmaske ist eine Diagnosemaske mit dem Dateinamen start.svg. Diese wird durch den Editor beim Anlegen eines neuen Projekts automatisch miterstellt. Von der Startmaske aus navigiert der Diagnostiker zu den einzelnen Diagnosemasken und beginnt mit der Diagnosearbeit.

11.2.2.3 Hilfsfunktionen zur Bedienung

Die Kernfunktionalität ist das Darstellen der Diagnoseskripte. Hierbei sind gerade im mobilen Bereich einige Funktionen hilfreich, um den Benutzerkomfort zu steigern. Da die Hilfsfunktionen für alle Masken bzw. Projekte von Vorteil sind, sind sie nativ in der Grundfunktionalität des Viewers verankert. Die Funktionen werden über den Menüpunkt "Settings" aufgerufen.

Eine Funktion ist das Rotieren der Ansicht. Erst ab dem Betriebssystem Windows Mobile 2003 SP2 wird diese Funktion durch das Betriebssystem angeboten. Auf älteren Endgeräten muss diese Aufgabe die Applikation selbst bewältigen. Für eine vernünftige Darstellung von Diagnosemasken ist allerdings das Querformat notwendig. Die Defaultausgabe von Windows Mobi-

le ist allerdings im Hochformat. Daher wurde der mobile Viewer um die Funktion "Bildschirm rotieren" erweitert.

Eine weitere elementare Hilfsfunktion ist das Zoomen des Bildschirmausschnittes. Ist eine Maske für einen Diagnoseterminal-Monitor konzipiert, wird sie zwar durch die Skalierung auf die Größe des Handheld-Monitors verkleinert, ist aber danach für den Benutzer schwer lesbar. Deshalb wird eine Funktion angeboten, mit der Bildschirmausschnitte in ihrer Darstellungsgröße verändert werden können. Die Maske kann schrittweise vergrößert bzw. verkleinert werden. Dabei kann der zu skalierende Bereich per Markierung oder punktiert per Klick auf eine Stelle ausgewählt werden. Nach dem Zoomen kann die Maske verschoben oder wieder automatisch an die Monitorgröße angepasst werden. Der Zoomvorgang verhält sich auch auf Handheldgeräten sehr performant. Der Benutzer wird folglich bei der Nutzung dieser Hilfsfunktion nicht durch Wartezeiten behindert.

Allerdings hängt die Geschwindigkeit der Verarbeitung von gehaltvollen SVG-Grafiken, besonders im mobilen Bereich, grundsätzlich von der Leistungsstärke des Endgeräts abhängt. Deshalb gilt es bei der Integration des Prototyps, im Rahmen der Pilotprojektierung, auch das richtige Endgerät auszuwählen. Diese Thematik ist Gegenstand des nächsten Kapitels.

11.3 Integration und Einsatz des Prototyps beim Kooperationspartner

Für die Einführung neuer Software im Zuge einer geplanten Prozessverbesserung sind einige organisatorische Aufgaben zu bewältigen. Becker führt in diesem Zusammenhang aus, dass es keinen "Königsweg" bzw. generell kein optimales Vorgehen gibt, sondern der organisatorische Umfang vielmehr situativ entschieden werden muss[276].

Die Einführungsstrategie der Software wurde bereits zu Beginn der Projektierung festgelegt. In Kapitel 11.1 wurde erläutert, dass das Produkt noch nicht den nötigen Reifegrad hat bzw. die Eignung als Diagnosesystem erst überprüft werden muss. Deshalb ist als Integrationsstrategie eine unternehmensweite zeitgleiche Einführung in allen Einsatzbereichen wie mit Big Bang[277] nicht geeignet. Es bietet sich sehr viel mehr das abgegrenzte schrittweise Verfahren des pilotierten Roll-outs an[278]. Eine ausführliche Begründung für die Pilotierung mit Bezug auf die Softwareentwicklungsstrategie findet sich in Kapitel 11.1.

Als Pilotbereiche wurden die Nacharbeit und die Verwaltung gewählt. Für den dortigen Einsatz sind lizenzrechtliche, technische und prozessbasierte Integrationsleistungen zu erbringen.

Die lizenzrechtliche Klärung sieht eine Aufschlüsselung vor, welche Lizenzen mit den einzelnen Softwarekomponenten zum Tragen kommen. Besonders für Großkonzerne ist die Offenlegung von Bedeutung, da teilweise eigene Vereinbarungen mit den Lizenzpartnern oder Nutzungsausschlüsse festgelegt sind. Kapitel 11.3.1 erörtert die zu berücksichtigenden Softwarelizenzen des Prototyps.

Als nächster Schritt der Integrationsleistung ist die technische Realisierung durchzuführen. Diese beinhaltet zunächst die Bereitstellung der notwendigen Hardware (Kapitel 11.3.2).

Der nächste technische Schritt der Anbindung muss dann bereits innerhalb der jeweiligen Geschäftsprozesse erfolgen (Kapitel 11.3.3). Der Schritt sieht die Zusammenführung der Hardware und Softwarekomponenten vor (Kapitel 11.3.3.1). Hierbei ist sowohl die Installation der Software als auch die physische Kopplung der Hardware durchzuführen. In der Nacharbeit wäre dies z. B. die Installation der HMI-Software auf dem Endgerät und dessen Anschluss an das Fahrzeug.

Nach der technischen Organisation kann der Prototyp innerhalb der Prozesse eingesetzt werden. Dabei ist eine intensive Betreuung in der ersten Prototypphase unabdingbar. Die Begründung dafür wird einleitend in Kapitel 11.3.3 nochmals verdeutlicht.

Des Weiteren darf bei dem Einsatz die Zielsetzung der Pilotphase nicht außer Acht gelassen werden. Neben dem Marketingeffekt für das neue Diagnosesystem unter den Mitarbeitern soll eine abschließende Evaluation über das Weiterführen der Prototypentwicklung entscheiden. Der Einsatz soll für die Evaluation entsprechende Daten bereitstellen. Aufgrund der beschränkten Ressourcen für das Testprojekt werden die Anwendungsfälle gezielt ausgewählt und abge-

[276] Vgl. Becker 2000, S. 233
[277] Vgl. Becker 2000, S. 236
[278] Vgl. Becker 2000. S. 235

arbeitet. Das Tagesgeschäft darf nicht beeinträchtigt werden. Die Kapitel 11.3.3.2 und 11.3.3.3 stellen exemplarisch Anwendungsfälle aus der Verwaltung und Nacharbeit vor.

11.3.1 Softwarelizenz und Vertrieb

Für die Integration der Software in die IT-Infrastruktur des Kooperationspartners müssen die zugrunde liegenden Technologien und deren Lizenzbestimmung offengelegt werden. Der IT-Betrieb führt bei jeder neu einzuführenden Software eine Verifizierung durch, ob ein Ausschluss für die verwendeten Technologien bzw. deren Lizenzpartner besteht. Falls nicht, werden eventuelle firmeneigene Lizenzvereinbarungen mit dem Urheber der Technologie geprüft.

Da der Viewer und der Editor auf verschiedenen Plattformen aufsetzen, sind für den Prototyp grundsätzlich zwei Lizenzverträge relevant. Diese ergeben sich aus der Nutzung der Eclipse-Plattform durch den Editor und der .Net-Plattform durch den Viewer.

Für den Editor basierend auf der Eclipse-Version 3.1 gilt die Eclipse Public License (EPL). Als Modifizierung der Common Public License (CPL) beinhaltet sie eine Open-Source-Lizenz. Sie gewährt ein kostenloses Recht zur Nutzung und Auslieferung von Eclipse-Plug-ins. Die EPL und CPL sehen weiter vor, dass eigene Applikationen (Plug-ins) kommerziell vertrieben werden dürfen. Das kommerzielle Vertriebsrecht gilt weiter auch für Derivate. Es dürfen unter EPL und CPL registrierte Plug-ins verändert werden und nach der Änderung unter eigenen Lizenzbedingungen kommerziell vertrieben werden[279].

Die Eclipse-Plug-ins sind mit Java programmiert. Seit dem zweiten Quartal 2007 steht die komplette Java-Technologie unter der Lizenzbestimmung der GNU General Public License (GPL). Diese ist der CPL sehr ähnlich und garantiert, dass damit erstellte Produkte kommerziell vertrieben werden dürfen.

Die beiden im Rahmen der Arbeit erstellten Plug-ins:

- de.gefasoft.mobivist.plugin.Editor (Verwaltung der Editoreinstellungen)
- de.gefasoft.mobivist.plugin.ControlImport (Verwaltung des Steuergerätezugriffs)

dürfen somit verkauft werden. Des Weiteren kann der Editor als ganzheitliches Produkt verkauft werden. Es dürfen die zusätzlich notwendigen Plug-ins zu einem lauffähigen Eclipse-Programm zusammengefasst werden und als Editor des Diagnosesystems kommerziell vertrieben werden. Hierbei muss darauf hingewiesen werden, dass die unterstützenden Plug-ins unter die EPL-Lizenzvereinbarungen fallen. Beim Kooperationspartner sind die Java und Eclipse-Technologie sowie deren Lizenzbestimmungen zugelassen. Der Editor kann somit an den Kooperationspartner vertrieben werden.

Der kommerzielle Vertrieb von eigenständigen Eclipse-Plug-ins ist in vielen Branchen üblich. Auch in der Automobilindustrie hat er Einzug erhalten. Viele Softwarezulieferer vertreiben diverse Anwendungen als Eclipse-Plug-ins. So z. B. der IT-Automobilzulieferer GEFASOFT AG, der ein Konfigurationstool für sein Kontrollsystem als Eclipse-Plug-in verkauft.

Auch die Plattform des Viewers von Microsoft ist auf die Entwicklung und den kommerziellen Vertrieb von damit entwickelter Software ausgelegt.

[279] Vgl. EPL V1.0 2005 (Online-Quelle)

Microsoft stützt sich für den Schutz seines .Net Frameworks auf die EULA-Lizenz. EULAs (End User License Agreement) sind spezielle Lizenzvereinbarungen, welche die Benutzung von Software regeln. Diese sollen den Benutzer verstärkt an die Wahrung des Urheberrechts binden. Microsoft verpflichtet diese mit der Nutzung z. B. von .Net Studio (Entwicklungsumgebung) dazu, die Microsoft. NET Framework Redistributable EULA zu akzeptieren.

Die EULA besagt, dass auf einer lizenzierten Kopie eines Microsoft Betriebssystem das .Net Framework installiert und frei genutzt werden darf. Es darf weiterhin mit einer darauf basierenden Anwendung mitinstalliert bzw. verteilt werden[280]. Für die Verteilung legt Microsoft noch weitere Regeln fest. Die aus rechtlicher Sicht einfachste Variante ist der Download von der offiziellen Microsoft Internetseite. Auf die weiteren Verteilmöglichkeiten und Vorschriften braucht im Rahmen der ersten Prototypphase nicht weiter eingegangen zu werden.

Für das Projekt ist an dieser Stelle wichtig, dass das Framework aus lizenzrechtlicher Sicht auf registrierten Kopien des Microsoft Betriebssystem eingesetzt werden darf. Für den Viewer ergibt sich somit die Möglichkeit, das basierende Framework kostenlos mit auszuliefern und ihn selbst kommerziell zu vermarkten.

Hierbei ist allerdings zu berücksichtigen, dass für den Viewer noch das kostenpflichtige CFCOM-Control der Firma Odyssey und das eSVG-Plug-in von Intesis (siehe Kapitel 10.1.2.2) notwendig sind. Für die beiden Plug-ins müssen bei einem produktiven Betrieb des Viewers noch entsprechend Lizenzen erworben werden.

Der kommerzielle Vertrieb des Viewers und des Editors wird besonders bei einer Weiterführung des Projekts in den nächsten Prototypphasen relevant. Da diese allerdings nicht mehr Zielsetzung der Arbeit sind, soll an dieser Stelle der Verweis auf die lizenzrechtlichen Rahmenbedingungen genügen. Ein professioneller Einsatz wird abschließend im Ausblick diskutiert (siehe Kapitel 12.2).

Nach der Klärung der Lizenzverhältnisse kann die technische Integration mit der Beschaffung der notwendigen Hardware beginnen.

11.3.2 Auswahl der passenden Endgeräte

Die Einführung der Editor-Software erfordert beim Kooperationspartner keine zusätzlichen Investitionen im Bereich Hardware. In Kapitel 10.2 bei der physischen Verteilung des Diagnosesystems wurde erläutert, dass die Eclipse-Umgebung auf gängigen Arbeitsplatz-Betriebssystemen wie die von Microsoft, Sun und Apple lauffähig ist. Beim Kooperationspartner arbeiten die Verwaltungsmitarbeiter, so auch die Diagnoseskript-Programmierer, derzeit mit dem XP-Betriebssystem. Auf diesem ist der Editor lauffähig. Für die Pilotprojektierung müssen in der Verwaltung folglich keine Endgeräte neuangeschafft werden.

Sehr viel mehr Aufmerksamkeit muss hingegen der Hardwarebeschaffung für den Einsatz in der Nacharbeit gewidmet werden. Da der Markt an mobilen Endgeräten ein breiteres Spektrum bietet, soll an dieser Stelle kurz differenziert werden, welche Möglichkeiten zur Verfügung stehen und ob sie sinnvoll erscheinen. Für die Integrationsarbeit ist es in der ersten Pilotphase nicht nötig, den kompletten Markt an mobilen Endgeräten zu evaluieren. Die Arbeit will an dieser Stelle vielmehr sinnvolle Lösungen abgrenzen, die aufgrund einsatzspezifischer Kriterien in Frage kommen.

[280] Vgl. Microsoft .NET Framework EULA (Online-Quelle)

Eines der entscheidenden Kriterien für dieses Projekt ist die Display- bzw. Gesamtgröße des Endgeräts. Klassifiziert nach ihrer Displaygröße werden die Endgeräte auf ihre Einsatzfähigkeit untersucht.

Die kleinsten Vertreter, Pager und Wearables, sind aufgrund ihrer Displaygröße nicht sinnvoll einsetzbar. Pager bieten meist nur einen 1-2 Zeilen Lauftext. Wearables, die als Ziel haben, am Körper unsichtbar getragen zu werden (z. B. integriert in Kleidung), sind so konzipiert, dass sie nahezu ohne Display auskommen.

Die nächste Kategorie mobiler Endgeräte umfasst die Gruppe der sogenannten Handsets. Diese schließt die einfachsten Modelle von Handys bis hin zu den Smartphones ein. Die Eingabe erfolgt über die Tastatur. Die Geräte sind bezüglich der Kommunikation sprachzentriert. Die Handsets haben wie die Pager den Schritt zum industriellen Einsatz bereits vollzogen. Rufsysteme in der Fertigungsindustrie sind seit einigen Jahren schon mit SMS Gateways ausgestattet und alarmieren den zuständigen Leiter per Textmitteilung. In neueren Meldesystemen kann dieser auch per spezifizierten Antworttext Steuerbefehle für die Maschine zurückschicken.

Eine neuere Entwicklung der Handsets ist über das Verschicken von Textmitteilungen hinaus für die Industrie interessant geworden: Smartphones, die als Hybridgeräte eine Mischung aus Mobiltelefon und PDA sind, bieten einerseits die Funktionalitäten eines Mobiltelefons, andererseits können sie auch wie ein PDA als kleiner Rechner Anwendungen ausführen. Obwohl Smartphones mit einem etwas größeren Display als normale Handys ausgestattet sind, bieten sie allerdings noch nicht ausreichend Komfort zur Visualisierung. Auch sind sie aufgrund der noch zu geringen Rechnerleistung für einen Einsatz als Diagnoseendgerät nicht geeignet.

Die nächst leistungsstärkere Gruppe sind Handhelds. Die Zugehörigkeit zu den Handhelds ist nicht eindeutig definiert[281]. Der Begriff umfasst in dieser Klassifizierung die Personal-Digital-Assistant-Geräte (PDA). Ein PDA ist ein kleiner tragbarer Computer, der meist mit einem schnell startenden Betriebssystem ausgestattet ist und inzwischen eine große Bandbreite von Anwendungen abdeckt. Bei dem Gerät bestehen ca. 80 % einer Seite nur aus dem Display, das um nur wenige physische Knöpfe ergänzt ist. Bei neueren Geräten ist der Anteil des Displays sogar noch größer und bietet bereits hochauflösende VGA-Grafik (640x480 Pixel) an. Auch die Prozessorleistung der neuesten Geräte liegt mit über einem Gigahertz nur noch eine Generation hinter dem Niveau aktueller Arbeitsplatzrechner. Diese Leistung komprimiert auf einem Gerät – das sich angenehm in der Hand halten lässt (engl. handheld) – bildet eine gute Hardwarebasis für ein mobiles Diagnosesystem. Die handflächengroßen Monitore bieten ausreichend Platz, um eine Diagnosemasken darzustellen. Zugleich ist auch die Gesamtgröße des Geräts noch nicht störend bzw. einschränkend für den Diagnostiker.

Auch das Betriebssystem von PDAs entspricht den Anforderungen des Viewers. Die sogenannte Pocket-PC-Variante von PDAs basiert auf dem Betriebssystem Windows Mobile. Auf diesem ist die Viewer-Plattform, das .Net Compact Framework, lauffähig und größtenteils bereits nativ unterstützt.

Aufgrund der Faktoren Rechnerleistung, Displaygröße und Lauffähigkeit der .Net Plattform wird ein aktuelles Pocket-PC-Gerät als Diagnoseendgerät gewählt.

Die Kategorien der nächst größeren Endgeräte, wie Tablet PCs, Mini-Notebooks und Notebooks wurden aufgrund ihrer Größe von den Diagnostikern in der Nacharbeit als zu störend

[281] Vgl. Schiller 2003, S. 23

und platzraubend empfunden. Diese Tendenz hat sich bereits bei der Analyse der Nacharbeitsprozesse abgezeichnet (siehe Kapitel 6.5).

In anderen Geschäftsprozessen, wie der Steuergeräteanalyse und Steuergeräteentwicklung, erscheint der Einsatz des Diagnosesystems auf Notebooks bzw. Tablet PCs den Mitarbeitern als günstiger. Deshalb sei an dieser Stelle erwähnt, dass bezüglich der Plattform der Einsatz des Viewers auch auf diesen Endgeräten möglich ist.

Mit der Bereitstellung der Hardware ist die Basis geschaffen, die neue Software in den Geschäftsprozessen einzuführen und anzuwenden.

11.3.3 Einführung und Anwendung des Prototyps in den Geschäftsprozessen Verwaltung und Nacharbeit

Bei der Einführung neuer Software wird häufig das Hauptaugenmerk auf die technische Migration gelegt. Der Erwerb der Softwarelizenzen, die Anschaffung neuer Hardware und die Verteilung der neuen Software sind greifbare Kostenpunkte und liegen im Fokus der Einführungsarbeit. In Kapitel 11.3.3.1 wird der letzte noch ausstehende Schritt der technischen Anbindung, die Softwareinstallation, beschrieben.

Besonders wenn ein Pflichtenheft der Einführung vorweggeht, wird erwartet, dass mit der technischen Einführung der neuen Software automatisch Verbesserungen erzielt werden. Diese zu eingeschränkte Erwartungshaltung beschreibt Heilmann in einer Fallstudie zur Geschäftsprozessverbesserung[282]. Heilmann beschreibt in diesem Zusammenhang das Versäumnis einer intensiven Einführungsbetreuung nach der technischen Anbindung. Laut Heilmann kommt es bei der Einführung neuer Systeme im besten Fall zu der vollen Ausschöpfung des Verbesserungspotenzials. Im negativen Sinn kann es aber auch zur Beeinträchtigung etablierter Prozessabläufe kommen. In jedem Fall ist mit Veränderungen einzelner Arbeitsroutinen zu rechnen. Dies liegt für die Prozessanalysten und Führungskräfte in der Natur der Sache, bedeutet aber für die Nutzer der Software eine Änderung seines Arbeitsalltags[283].

Ab Beginn der Prozessanalyse wurde in dieser Arbeit immer wieder darauf hingewiesen, dass die Akzeptanz der künftigen Nutzer elementar für die erfolgreiche Prozessmodifikation ist. Deshalb wurden sie in der konzeptionellen Phase bei der Festlegung von Verbesserungsansätzen miteingebunden. Der Aspekt der Anwenderzufriedenheit prägt einige Punkte des Anforderungskatalogs. Die Punkte werden in der folgenden Evaluation explizit auf Erfüllung hin geprüft. Diese Rücksichtnahme, sowie der gemäßigte Umstieg durch den pilotierten Roll-out, mit dem sich der Nutzer langsam an die Software gewöhnen kann, kommen der Akzeptanz des neuen Diagnosesystems zugute.

Allerdings ist die Wirkung dieser positiven Rahmenbedingungen gefährdet, wenn die erste Pilotphase nicht durch ausgeprägte Beratungsleistung begleitet wird. Heilmann macht dies in der Fallstudie deutlich. Eine versäumte persönliche Migrationsleistung kann zu einer lang anhaltenden Ablehnung führen[284]. Deshalb wurden die Erstnutzer des Prototyps bei der Einarbeitung entsprechend in die Bedienung des neuen Systems eingeführt.

[282] Vgl. Heilmann 2003, S. 298-301
[283] Vgl. Heilmann 2003, S. 302
[284] Vgl. Heilmann 2003, S. 303

Erst nachdem sich die Anwender mit dem System vertraut gemacht hatten, wurden Anwendungsfälle aus dem Tagesgeschäft nachgestellt und selbständig bearbeitet. Die Bearbeitung erfolgte mit dem Ist- und dem Neu-System (Prototyp).

Die Ablaufdaten der Anwendungsfälle und ersten Eindrücke der Anwender dienen als Basis für die Leistungsbewertung. Sie werden in Kapitel 11.4 explizit evaluiert. Die Kapitel 11.3.3.1 bis 11.3.3.3 sollen hingegen mehr einen Einblick in die Veränderungen geben, die durch das neue System entstehen. Die Veränderungen werden anhand von Beispielen veranschaulicht. In Kapitel 11.3.3.2 werden hierfür Anwendungsfälle aus der Verwaltung und in Kapitel 11.3.3.3 Anwendungsfälle aus der Nacharbeit angeführt.

11.3.3.1 Technische Anbindung in der Verwaltung und Nacharbeit

In Kapitel 11.3.2 wurden die unterschiedlichen Hardwareanforderungen und deren Erfüllung innerhalb der Geschäftsprozesse erörtert. Jetzt soll das Neu-System auf der bereitgestellten Hardware installiert werden und lauffähig gemacht werden.

In der Verwaltung erfordern die Systemanforderungen des Editors keine Erweiterung der bestehenden Hardware. Das System, basierend auf der Eclipse-Umgebung, kann auf den bestehenden Arbeitsplatzrechnern installiert werden.

Um die Anwendungsfälle der Verwaltung mit dem Editor abarbeiten zu können, ist lediglich das Bereitstellen der Steuergerätedaten erforderlich. Um flexibel für diverse proprietäre Basissysteme zu bleiben, fordert der Editor die Daten in einer erweiterbaren XML-Struktur. Die Struktur wurde in Kapitel 10.3.2 definiert. Zunächst wurden nur die Steuergerätedaten portiert, die für die Anwendungsfälle der ersten Pilotphase notwendig waren. Diese waren aus dem Bereich der Klimatechnik, Lichtschaltelektronik, Motorsteuerung und Einparkhilfe. Da die Mitarbeiter durch ihre tägliche Arbeit mit dem proprietären Basissystem bestens vertraut sind, konnte der Export in XML über ein Kommandozeilen-Tool mit geringen Aufwand automatisiert werden. Der Kooperationsparnter konnte die für den Pilotbetrieb notwendig Editorumgebung innerhalb eines Manntages bereitstellen. Die Bearbeitung von Anwendungsfällen nach einer betreuten Anlernphase wird in den Kapiteln 11.3.3.2 und 11.3.3.3 erläutert.

Für die Installation des Viewers in der Nacharbeit mussten zusätzliche mobile Endgeräte angeschafft werden. Art und Typ wurden in Kapitel 11.3.2 näher bestimmt. Über einen OBD-Funkadapter stellt das Endgerät mit dem Steuergerät bzw. dem Fahrzeug eine Verbindung her (siehe Kapitel 5.6.1). Der OBD-Funkstecker wird ans Fahrzeug angeschlossen und überträgt per WLAN die Daten an die spezifische Netzwerkadresse des PCs. Die ausgewählten mobilen Endgeräte sind deshalb mit einer integrierten WLAN-Karte ausgerüstet. Der Anschluss wäre aber auch über ein OBD-Kabel möglich, das mit einem 9-poligen seriellen Anschluss angeschlossen werden kann. Auch für den seriellen Anschluss würde für das mobile Endgerät ein entsprechender Adapter bereitstehen. Das Kabel würde allerdings die Mobilität einschränken, weshalb die WLAN-Kommunikation bevorzugt wurde.

Installiert wurde die Viewer-Software neben den mobilen Endgeräten auch auf dem Diagnoseterminal. Wie im Rahmen der Lizenzvereinbarungen bereits erwähnt wurde ist hierfür das .Net Framework (kostenfrei) auf den registrierten Windows Betriebssystemen zu installieren.

Für den Betrieb der Viewer-Software wird zur Laufzeit noch der Treiber für das Basissystem benötigt. Auf dem Diagnoseterminal ist er aufgrund der Verwendung durch das Ist-System be-

reits vorhanden. Für den mobilen Einsatz stellen die Hersteller des proprietären Treibers eine zusätzliche Version für Windows Mobile bereit.

Im Anschluss an die Installation erfolgt die Konfiguration des Netzwerkanschlusses. Danach kann das Endgerät über den Funkadapter die Kommunikation mit dem Basissystem bzw. Steuergerät aufnehmen. Der Viewer ist somit für die Nacharbeit einsatzbereit. Die Einführung erfolgt in einem abgetrennten Bereich auf zwei Diagnoseterminals sowie auf zwei mobilen Endgeräten. Für die Abarbeitung von experimentellen Nacharbeitsfällen (siehe Kapitel 11.3.3.3) müssen aber zunächst die entsprechenden Diagnoseskripte in der Verwaltung erstellt werden.

11.3.3.2 Einsatz des Prototyps in der Verwaltung

Die Prozessmodellierung hat ergeben, dass die Hauptaufgabe der Verwaltung in Bezug auf das Diagnosesystem darin besteht, Diagnoseskripte zu erstellen und nachzubearbeiten.

Für beide Tätigkeiten wurden in der Analyse Schwachstellen aufgedeckt. Deshalb sollen für beide Tätigkeitsbereiche Anwendungsfälle durchgeführt werden. Die Anwendungsfälle werden anhand von Ablaufprotokollen dokumentiert und sind in der Quelle MobiViSt-Prozessdatenerfassung 2008 erfasst. Sie werden in der Evaluation anschließend ausgewertet, um zu entscheiden, ob mit dem neuen System Verbesserungen erreicht werden.

Die Abarbeitung erfolgt durch Verwaltungsmitarbeiter des Kooperationspartners. Da neben der Erbprobung des Neu-Systems noch die Bearbeitung des Tagesgeschäfts erfolgen muss, kann der Kooperationspartner folglich nur ein beschränktes Zeitfenster zur Verfügung stellen. Die Anwendungsfälle müssen deshalb sorgfältig ausgewählt werden und sich auf das Wesentliche konzentrieren.

Für den Anwendungsfall Neuerstellungt eines Diagnoseskripts wurde ein im Ist-System bereits bestehendes Skript ausgewählt. Da die Neuerstellung eines kompletten Skripts den Zeitrahmen der Pilotprojektierung überschreiten würde, wurde für diesen Anwendungsfall nur eine Maske ausgewählt. Die Maske sollte mit dem Ist-System und dem Neu-System neu erstellt werden. Das grafische Verbessern der Maske ist nicht Ziel dieses Anwendungsfalls. Die Maske sollte lediglich nachkonstruiert werden, um die Arbeitsweisen zu vergleichen. Die folgende Abbildung zeigt beide erstellte Masken nach dem Laden im Viewer:

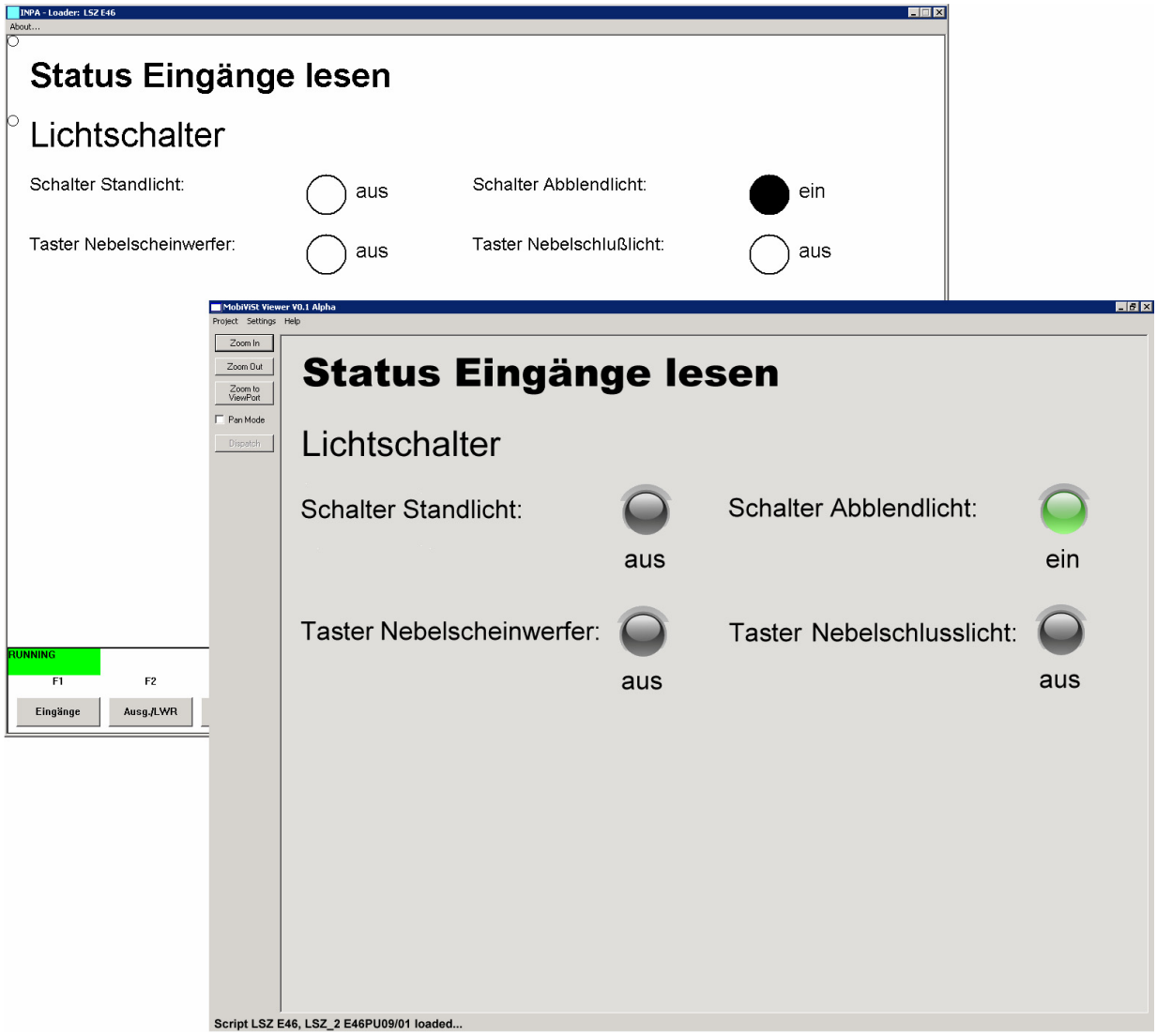


Abbildung 11.9 Diagnosemaske Lichtschalter: Ist-System (links), Neu-System (rechts)

Während der Erstellung der Maske wurde das folgende Ablaufprotokoll für beide Systeme erfasst. Ausführlich ist das Ablaufprotokoll in der Quelle MobiViSt-Prozessdatenerfassung 2008 erfasst. Dort werden die einzelnen Schritte und die Metadaten des Anwendungsfalls beschrieben. Die folgende Tabelle zeigt nur die Überschriften der einzelnen Schritte:

Nr.	Tätigkeit	Dauer-Ist <Sek>	Dauer-Neu <Sek>
1	Entwicklungsumgebung öffnen	45	55
2	Skript öffnen	10	20
3	Grafisches Objekt auswählen	30	3
4	Größe der Grafik bestimmen	10	5
5	Grafik positionieren	15	5
6	Grafik beschriften	30	10
7	Wechseln in EDIABAS Toolset32	5	-

8	Steuergerätedaten laden	20	15
9	Job auswählen	30	30
10	Job markieren & kopieren	20	-
11	Wechseln in Texteditor	3	-
12	Job einfügen	10	-
13	Wechseln in EDIABAS Toolset32	5	-
14	Resultwert auswählen	30	30
15	Result markieren & kopieren	15	-
16	Wechseln in Texteditor	3	-
17	Result einfügen	10	-
18	Schritte 3-18 wiederholen für die drei weiteren LEDs	620	260
19	Skript speichern	3	3
20	Skript kompilieren	40	-
21	Zur Überprüfung der Anzeige in INPA-Loader öffnen	10	-
22	Funktionale Überprüfung	-	-
	Gesamtzeit Sek. (MM:SS)	**964 (16:04)**	**436 (7:16)**

Tabelle 11.1 Systemvergleich bei Anwendungsfall: Maske neu erstellen

Eine detaillierte Beurteilung erfolgt bei der Evaluation der Software. Was allerdings den Programmierern der Diagnoseskripte bereits während der Arbeit positiv auffiel, ist die Tatsache, dass nur eine Anwendung zu starten ist und der Wechsel bzw. das Kopieren zwischen den Applikationen entfällt. Auch die einfache grafische Nachbearbeitung der Objektposition, sowie das Wegfallen des Kompiliervorgangs wurden von den Nutzern positiv bewertet. In der Tabelle lassen sich die entfallenen Schritte an den "-"-Zeichen erkennen. Außerdem ist am Tabellenende die erhebliche Reduzierung der Gesamtzeit auffällig – von 16:04 Minuten auf 7:16 Minuten.

Um der Evaluation noch mehr Daten für den Vergleich der Bearbeitungszeiten zu liefern, wurden weitere Anwendungsfälle dokumentiert. Ein anderer Anwendungsfall sieht z. B. das Ergänzen einer bestehenden Maske um zusätzliche Funktionalität vor. Hierfür wurde ein Szenario geschaffen, in dem eine bereits nachkonstruierte Maske um eine Statusanzeige erweitert werden soll. Das vollständige Ablaufprotokoll ist in der Quelle MobiViSt-Prozessdatenerfassung 2008 hinterlegt und wird auch bei der Evaluation untersucht.

Ferner wurden Anwendungsfälle durchgeführt, um die grafischen Potenziale des Neu-Systems zu überprüfen. Für diese eher subjektive Beurteilung wurden bestehende Masken des Ist-Systems, die besonders bei der Analyse als nicht benutzerfreundlich eingestuft wurden, anhand des Neu-Systems nachkonstruiert. Der Fokus lag hierbei nicht auf der zeitlichen Ersparnis, sondern der grafischen bzw. funktionalen Verbesserung.

Bei den Masken des derzeitigen Systems sind aufgrund der gestalterischen Möglichkeiten die Masken zur Wertanzeige von den Masken zur Steuerung getrennt. Der Diagnostiker muss folglich zwischen den Masken wechseln, falls er einen Wert ändern und im Anschluss daran Werte

auf ihre Veränderung hin beobachten will. Auch ist aufgrund der grafischen Auflösung des Ist-Systems die Anzahl der anzeigbaren Werte pro Maske sehr gering. Dies erhöht die Notwendigkeit des Maskenwechsels bei der Diagnose noch weiter.

Die folgende Abbildung zeigt, wie eine Anzeige- und eine Steuermaske im Bereich der Heiz- und Klimatechnik in einer Maske des Neu-Systems zusammengefasst wurde. Durch die grafischen Möglichkeiten wurden an vielen Stellen Textbezeichnungen gespart. Der Inhalt wird dadurch für den Diagnostiker schneller erfassbar.

Darüber hinaus kann in der Maske des Neu-Systems noch zusätzliche Steuerfunktionalität untergebracht werden. Die Sollwerte der Innenraumtemperatur sind im Ist-System nur als Anzeigewerte dargestellt. Sie müssen im Ist-System über eine dritte (nicht abgebildete) Maske gesteuert werden. Die Maske des Neu-Systems ermöglicht die Ausgabe der Sollwerte und macht sie gleichzeitig über einen Regler steuerbar.

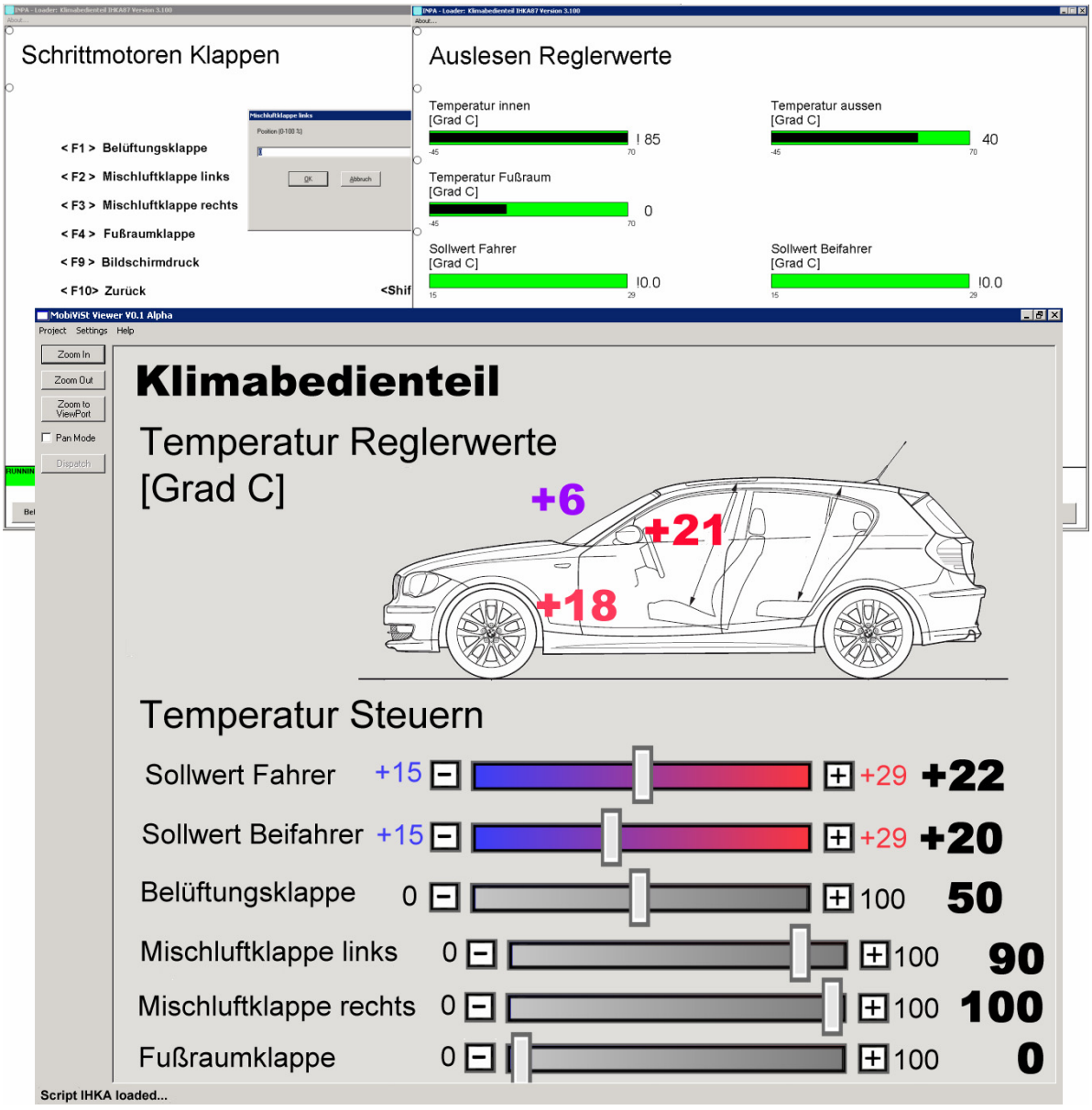


Abbildung 11.10 Diagnosemaske Bedienteil aus Heiz- und Klimabereich (Ist-/Neu-System)

Der Nutzen solcher grafischer Veränderungen ist letztlich stark subjektiv zu beurteilen. Obwohl sich die zeitliche Verbesserung durch die Reduktion der Maskenwechsel innerhalb einer Pilotprojektierung kaum nachweisen lässt, ist der Vorteil einer angenehmeren und benutzerfreundlicheren Diagnoseoberfläche offensichtlich.

Mit der Bereitstellung der neuen Diagnosemasken durch die Verwaltung kann die Abarbeitung der Anwendungsfälle in der Nacharbeit durchgeführt werden. Dort finden sich weitere Verbesserungen, die durch den Einsatz des neuen Diagnosesystems entstehen.

11.3.3.3 Einsatz des Prototyps in der Nacharbeit

Aufgrund von Qualitätssicherungsmaßnahmen ist in der Nacharbeit die Einführung innerhalb eines abgegrenzten Pilotbetriebs noch wesentlich wichtiger. Neue Software, die sich noch in der Pilotphase befindet und noch nicht den nötigen Status an Zuverlässigkeit aufweist wird zunächst an konstruierten Anwendungsfällen getestet. Erst wenn ein entsprechender Reifegrad bzw. ein Roll-out fähiges Produkt entwickelt ist, wird es für die Bearbeitung realer Fälle aus dem Tagesgeschäft zugelassen.

Die Anwendungsfälle dieser ersten Prototypphase sind demnach nachgebildete Fälle an Testfahrzeugen. Die Fälle entsprechen real eingetretenen Nacharbeitsfällen. Auch sie werden sowohl mit dem Ist-System, als auch mit dem Neu-System bearbeitet. Die Fehlerursache wird manuell herbeigeführt und ist dem Diagnostiker unbekannt. Sie bekommen realitätsgetreu nur die Beschreibung der Fehlermeldung aus dem finalen Prüfsystem ausgehändigt.

In der Analyse und Soll-Modellierung wurde verdeutlicht, dass die Verbesserungen in der Nacharbeit vornehmlich durch ein mobiles und grafisch hochwertigeres System zu erreichen sind. Im Hinblick auf die Überprüfung dieser Annahme in der Evaluation, wurden entsprechende Anwendungsfälle ausgewählt.

In Kapitel 11.3.3.2 wurde bereits gezeigt, dass mit dem neuen System die Aussagekraft und Funktionalität der Masken gesteigert werden kann. Dadurch soll dem Diagnostiker die Möglichkeit geboten werden, zielstrebiger die Fehlerursache zu finden.

Als Anwendungsfall in der Nacharbeit soll zur Veranschaulichung ein bekanntes Beispiel aus der Prozessanalyse aufgegriffen werden. In Kapitel 6.5.2.4 wurde ein defekter Sensor zur Abnahme der Innenraumtemperatur als Fehlerursache diagnostiziert. Um die Fehlerursache zu ermitteln, musste der Diagnostiker bei dem Ist-System verschiedene Masken überprüfen, bis er auf die Maske für den Innenraumsensor trifft. Anhand der grafischen Darstellung, die den eingestellten Sollwert der Temperatur und den gemessenen Wert des Sensors auf einer Maske zusammenfasst (siehe Abbildung 11.10), erkennt der Diagnostiker sofort die Fehlerursache. Das Ablaufprotokoll des Anwendungsfalls ist in der Quelle MobiViSt-Prozessdatenerfassung 2008 hinterlegt.

Ein weiteres Verbesserungspotenzial soll der mobile Einsatz bieten. Das Diagnoseskript kann unverändert sowohl auf dem mobilen Endgerät als auch auf dem Diagnoseterminal eingesetzt werden. Folgende Abbildung zeigt eine Maske des Ist-Systems, die mit dem Neu-System nachkonstruiert wurde. Sie ist somit auf dem Handheld und dem Diagnoseterminal in gleicher Form verfügbar.

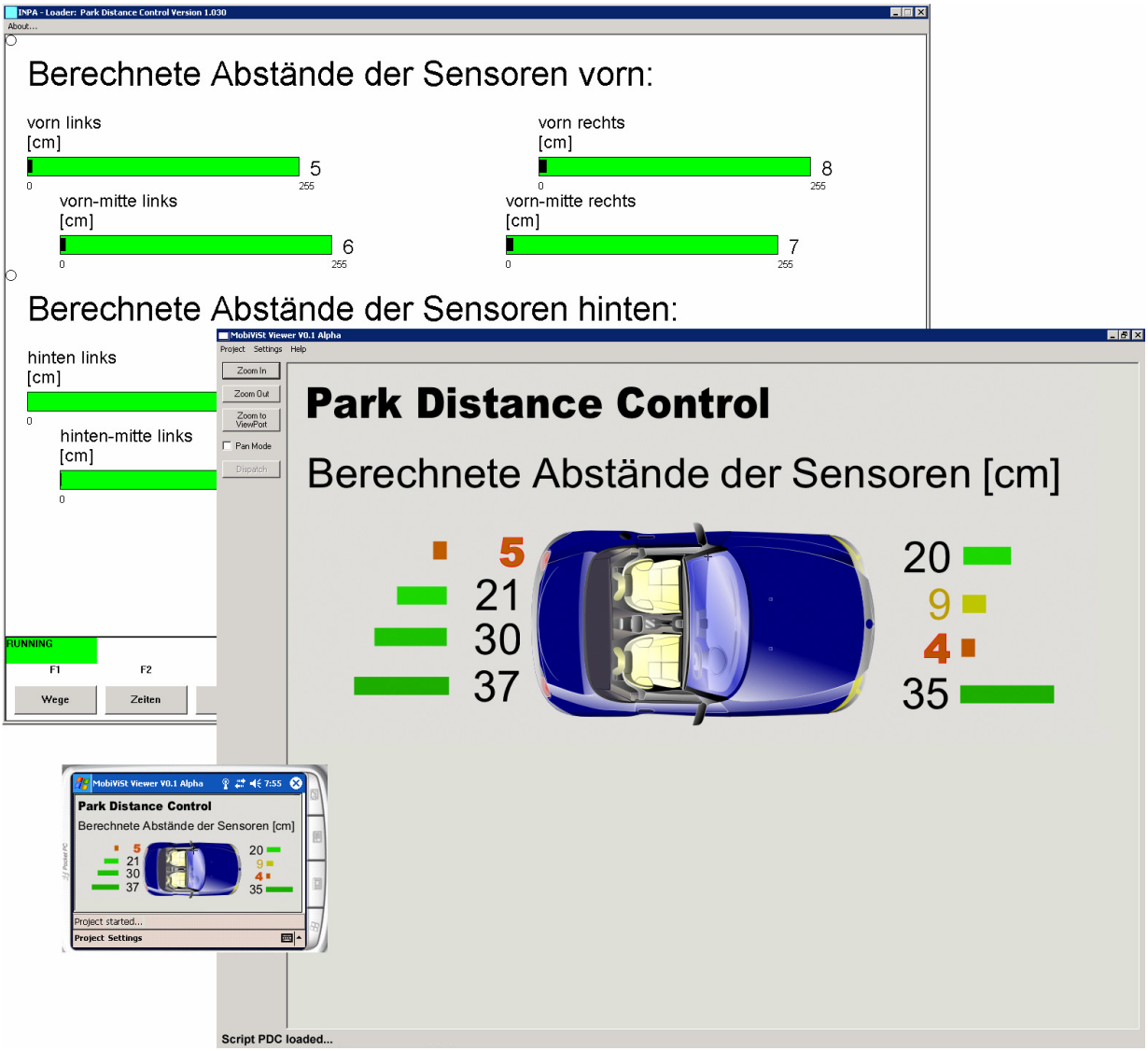


Abbildung 11.11 Diagnosemaske auf mobilem Endgerät und Diagnoseterminal

Durch den mobilen Einsatz ergibt sich der Vorteil, dass die Diagnostiker bei dem Steuergerät vor Ort am Fahrzeug bleiben können und nicht für die Ausführung von Steuergerätebefehlen am Diagnoseterminal die Arbeitsstelle verlassen müssen.

In Kapitel 6.6 wurden hierfür bereits anschauliche Beispiele gegeben. Die Anwendungsfälle wurden rekonstruiert und mit beiden Systemen erneut abgearbeitet. Folgende Tabelle gehört zu einem Anwendungsfall, bei dem der Blinkerschalter defekt ist. Die Tabelle stellt einen Auszug des Ablaufprotokolls dar. Der vollständige Anwendungsfall ist in der Quelle MobiViSt-Prozessdatenerfassung 2008 erfasst.

Nr.	Tätigkeit	Dauer-Ist <Sek>	Dauer-Neu <Sek>
1	Fahrzeuginnenraum betreten	25	25
2	Aktivieren des Blinkerschalters -> Schalter eingerastet kein Blinksignal	15	15

3	Fahrzeuginnenraum verlassen	15	-
4	Über Diagnosesystem des Lichtschalt-Steuergeräts Licht für Blinker rechts aktiviert -> Licht geht an => Licht i.O., Blinker Elektronik n.i.O.	45	50
5	Fahrzeuginnenraum betreten	20	-
6	Armaturenverkleidung im Fahrzeuginnern um Schalterhebel herum entfernt.	260	260
7	Stecker am Steuergerät überprüft Steckerverbindung am Steuergerät zu Schalterkonsole ausgetauscht und Schalter aktiviert -> Kein Blinksignal	30	30
8	Fahrzeuginnenraum verlassen	15	-
9	Maske am Diagnosesystem gewechselt, um Spannung am Anschluss zum Blinkerschalter zu messen -> Anzeige zeigt keine Spannung Vermutung: Blinkerschalter defekt	45	50
10	Fahrzeuginnenraum betreten	20	-
11	Spannung manuell auf Blinkerschalterverbindungsstecker am Steuergerät gelegt (spezielles Messwerkzeug) -> Kein Blinksignal	60	60
12	Fahrzeuginnenraum verlassen	15	-
13	Über Diagnosesystem Spannung am Anschluss zum Blinkerschalter gemessen -> Anzeige zeigt keine Spannung => Kontakt am Steuergerät defekt	25	25
14	Fahrzeuginnenraum betreten	20	-
15	Steckerkontakt ausgetauscht und Kabelverbindung angeschlossen	110	110
16	Schalter aktiviert -> Blinksignal	10	10
17	Fahrzeuginnenraum verlassen	15	-
18	Über Diagnosesystem Spannung am Anschluss zum Blinkerschalter gemessen -> Anzeige zeigt Spannung	25	25
19	Fahrzeuginnenraum betreten	20	-
20	Armaturenverkleidung um Schalterhebel herum montiert.	200	200
21	Fahrzeuginnenraum verlassen und Diagnosetätigkeit und Reparatur beendet	20	20
	Gesamtzeit Sek. (MM:SS)	**1010 (16:50)**	**880 (14:40)**

Tabelle 11.2 Systemvergleich bei Anwendungsfall: Blinkerschalter defekt

Der Diagnostiker ist aufgrund der Position des Steuergeräts gezwungen, liegend im Fußraum des Fahrzeugs zu arbeiten. Von dort kann er weder die Terminalanzeige sehen, noch das Ter-

minal bedienen. Dadurch ist er bei dem Ist-System gezwungen, den Arbeitsbereich mehrmals zu verlassen, um Befehle auszuführen. Die Tabelle zeigt deutlich, dass unter der Nutzung eines mobilen Endgeräts dieser Weg entfällt. Er kann mit dem mobilen Viewer, die zur Diagnose notwendigen Werte vor Ort ablesen und auch Befehle an das Steuergerät schicken. Dadurch verkürzt sich auch hier die Gesamtdauer der Bearbeitung.

Bei weiteren Anwendungsfällen zeichnete sich eine ähnliche zeitliche Verbesserung ab. Darüber hinaus steigert das Entfallen des ständigen Verlassens der Arbeitsfläche die Anwenderzufriedenheit. Der Diagnostiker kann konzentrierter und zielstrebiger ohne Unterbrechungen an dem Nacharbeitsfall arbeiten.

Bereits während der Bearbeitung der Anwendungsfälle sind den Mitarbeitern in der Verwaltung und Nacharbeit die Verbesserungen durch das Neu-System aufgefallen. Ob die empfundenen Verbesserungen tatsächlich begründet sind und bis zu welchem Grad sie bestehen wird im nächsten Kapitel 11.4 beurteilt. Anhand analytischer und experimenteller Evaluationsverfahren werden die Daten aus dem Pilotbetrieb in der abschließenden Leistungsbewertung des Prototyps ausgewertet.

11.4 Leistungsbewertung des Prototyps

Die Leistungsbewertung eines Prototypen ist ein wichtiger Schritt zur Informationssammlung innerhalb des iterativen Softwareentwicklungsprozesses. Mit diversen Bewertungsmethoden sollen Fehler und Schwachstellen ausfindig gemacht und die Erfüllung von Zielsetzungen überprüft werden. Grundsätzlich soll sichergestellt werden, dass die Software den Bedürfnissen des Benutzers und des Prozessumfeldes nachkommt[285].

Für die Bewertung von Software (Evaluation), finden sich zahlreiche Definitionen. Die deutsche Gesellschaft für Evaluation deklariert sie allgemein, als die systematische Untersuchung des Nutzens oder Werts eines Gegenstandes, wobei derartige Gegenstände z. B. Systeme, Systemkomponenten oder Programme sein können[286].

Für die Untersuchung von Softwareobjekten steht heute eine große Bandbreite von Methoden und Werkzeugen zur Auswahl. Sie reichen von einfachen Checklisten über Inspektionsverfahren bis hin zu aufwendigen Usability-Tests[287]. In Kapitel 11.4.1 werden nun die Methoden beschrieben, die aus dem Angebot an Techniken für die Arbeit ausgewählt wurden.

11.4.1 Auswahl der Technik zur Leistungsbewertung

Für die Evaluation von Software können verschiedenste Techniken angewendet werden. Die Frage nach der besten Technik bleibt allgemeingültig unbeantwortet. Vielmehr ergibt sich die Eignung der Evaluationstechnik aus dem Bewertungsgegenstand und der Zielsetzung der Eva-

[285] Vgl. Nielsen 1993, S. 93-95
[286] Vgl. DeGEval 2002, S. 1
[287] Vgl. Hegner 2003, S. 6

luationsaussage jedes Mal aufs Neue[288]. Deshalb strebt die folgende Auflistung der Techniken nicht nach Vollständigkeit, sondern nach der Eingrenzung der gewählten Technik.

Die Zielsetzungen einer Evaluation hängen grundsätzlich vom Zeitpunkt der Durchführung ab. Diesbezüglich werden in der Literatur in erster Linie zwei Gruppen, die formativen und die summativen Evaluationen, diskutiert. Während die formative Evaluation innerhalb des gesamten Entwicklungsprozesses möglich ist, hat die summative Evaluation das Ziel, das System abschließend zu bewerten[289]. Die Gruppe der formativen Evaluationen ist somit ein wichtiger Bestandteil iterativer Softwareentwicklung. Sie bewertet Prototypen im Laufe ihres Entwicklungsprozesses und soll deshalb in dieser ersten Prototypphase Anwendung finden.

Nach Baumgartner unterteilt sich die Evaluation in einen vierstufigen Prozess[290], der sich auch in der Gliederung dieser Arbeit widerspiegelt. Der Ablauf beginnt mit der Formulierung von Wertekriterien und den zugehörigen Leistungsstandards. Diese beiden ersten Schritte finden sich in den Kapiteln 6 und 7 der Arbeit wieder. Die Wertekriterien sind die Bereiche, in denen Schwachstellen entdeckt wurden. Für diese Wertekriterien wurden in Kapitel 7 die Anforderungen definiert. Die von Baumgartner als Leistungsstandards definierten Anforderungen sind im Anforderungskatalog (Kapitel 7.3) dargelegt. Diesen Leistungsstandard gilt es mit Bewertungsgegenstand (der Software) zu erbringen. So wurde z. B. für das Wertekriterium Grafik, die Anforderung einer realitätsgetreuen Bildqualität aufgestellt. Deshalb enthält der Anforderungskatalog den Anspruch, dass die Software mindestens den Leistungsstandard einer True-Color-Farbtiefe erfüllen muss (siehe Kapitel 7.3.2.3). Das Erreichen dieser Forderungen wird mit dem dritten Schritt nach Baumgartner gemessen. Der Bewertungsgegenstand wird dann in Kapitel 11.4.2 und 11.4.3 daraufhin geprüft, ob die vorgegebenen Leistungsstandards erfüllt werden. Der letzte Schritt des Evaluationsprozesses verknüpft die verschiedenen Ergebnisse zu einem einheitlichen Werturteil. Das Urteil wird in Kapitel 11.4.4 dokumentiert.

Nach der Auswahl der Gruppe der formativen Evaluation und der Bestimmung des Ablaufs gilt es für die Evaluation die passende Technik zu finden. Wie bereits bei der Auswahl der Gruppe erwähnt, hängen die Zielsetzungen vom Zeitpunkt der Evaluation ab. So lässt sich charakteristisch über den Zeitpunkt und die Zielsetzung eine passende Technik für die Evaluation finden.

Zunächst soll die Zielsetzung der Evaluation bestimmt werden. Die Zielsetzung definiert den Umfang der Evaluation mit und gibt eine Richtung für die auszuwählende Technik vor. Eine Zielsetzung der Evaluation kann z. B. das Auffinden von Schwachstellen der Software sein. Hierfür ist allerdings ein umfangreicher Usability-Test nahezu Voraussetzung, um z. B. Mängel in der Bedienbarkeit zu erfassen. Daneben gibt es weitere Zielsetzungen, wie z. B. den bewertenden oder vergleichenden Evaluationszweck. Bei beiden soll aus einer Reihe von Softwarelösungen das am Besten passende Produkt ausgewählt bzw. die Eigenschaften der einzelnen Produkte abschließend bewertet werden. Die genannten Zielsetzungen sind mehr für Prototypen späterer Entwicklungsphasen bzw. für einsatzfähige Software geeignet. Ein vollständiger Usability-Test erscheint auch erst bei solchen Softwareständen sinnvoll.

Hegner ergänzt die Liste der Zielsetzungen um den schlichten Stopp/Go-Evaluationszweck[291]. Mit der Evaluation soll die Entscheidung getroffen werden, ob eine Maßnahme weitergeführt wird oder nicht. Im vorliegenden Fall basiert diese Maßnahme auf der Einleitung eines weite-

[288] Vgl. Nielsen 1993, S. 17
[289] Vgl. Wottawa 1998, S. 5-7
[290] Vgl. Baumgartner 1999, S. 3-10
[291] Vgl. Hegner 2003, S. 9

ren Iterationsschritts in der Prototypentwicklung. Die Stopp/Go-Zielsetzung erscheint für die vorliegende Prototypphase als passend. Das Urteil einer weiteren Prototypphase hängt entscheidend davon ab, ob der Anforderungskatalog erfüllt wird und Prozesszeitverbesserungen zu erkennen sind. Eine umfassende Überprüfung der Softwareergonomie und Benutzerfreundlichkeit sind also weniger die Ziele dieser Evaluation. Mit der ersten Prototypphase soll die grundsätzliche Einsetzbarkeit und Verbesserungsfähigkeit der Prozesse nachgewiesen werden.

Entsprechend der Bestimmung der Zielsetzungen und des Zeitpunkts werden die passenden Evaluationstechniken und zugehörigen Methoden ausgewählt.

Die Notwendigkeit eines begrenzten Pilotbetriebs bei einem so komplexen und bereichsübergreifendem System wie dem vorliegenden, wurde bereits in Kapitel 11.1 erläutert. Mit der Evaluation verhält es sich ähnlich. Eine ganzheitliche Beurteilung in allen Einsatzgebieten und allen Anwendungsfällen ist bei der Steuergerätediagnose in einem Automobilgroßkonzern nicht möglich und auch in der ersten Prototypphase nicht sinnvoll. Für die Bewertung bietet sich bei solchen Rahmenbedingungen deshalb die experimentelle Evaluation an. Diese Technik dient zur Überprüfung theoretischer Annahmen und zum Vergleich verschiedener Systeme[292]. Es wird dabei das bisher eingesetzte System mit dem Prototyp verglichen. Für den Test wird ein spezifischer Bereich definiert, in dem beide Systeme die gleichen Aufgaben bekommen. Man spricht deshalb bei dieser Technik auch von kontrollierten Experimenten. In dieser Arbeit wurden die Experimente anhand ausgewählter Anwendungsfälle dargestellt. In Kapitel 11.3.4 wurde hierfür die Vorarbeit, wie z. B. die Integration der Software, geleistet. Das neue und das bestehende Diagnosesystem durchliefen Anwendungsfälle in den Integrationsbereichen und wurden anhand ihrer Prozessdaten bewertet. Die Experimente lieferten somit die Basis für den Kausalschluss, ob die Aussage der Prozessverbesserung vertretbar ist. Die Auswertung der kontrollierten Experimente wird in Kapitel 11.4.3 beschrieben.

Neben der Beurteilung der Prozessverbesserung ist das andere wichtige Kriterium für die Stopp/Go-Entscheidung die Erfüllung des Anforderungskatalogs. Im Vergleich zu der objektiven Erfassung quantitativer Prozessdaten durch Zeitmessungen und Stückzahlberechnungen im Rahmen der experimentellen Evaluation werden für die Bewertung des Anforderungskatalogs auch subjektive Methoden notwendig. Subjektive Evaluationsmethoden knüpfen unmittelbar an die Beurteilung durch den Benutzer an. Bei subjektiven Evaluationsmethoden werden eher "weiche" (qualitative) Daten gewonnen, ob die Benutzung des Systems z. B. angenehm, klar oder nachvollziehbar ist. Bei objektiven Methoden versucht man, subjektive Einflüsse weitgehend auszuschalten[293].

Die Bewertung des Anforderungskatalogs erfordert eine Mischung aus beidem. Es werden zum einen objektive Systemeigenschaften und zum anderen subjektive Funktionalitäten bewertet. Eine Funktionalität, die eine subjektive Bewertung erfordert, ist die Forderung nach einer umfangreichen grafischen Gestaltungsmöglichkeit. Eine Systemeigenschaft, die sich mehr objektiv überprüfen lässt, ist z. B. die Unterstützung von proprietären Basissystemen. Für eine derartige Mischung bietet die Evaluationsbandbreite die Technik der analytischen Evaluation. Diese stützt sich auf subjektive und objektive Bewertungsmethoden[294].

Bei der analytischen (leitfadenorientierten) Evaluation wird das System von einem Experten geprüft. Durch die persönliche Beteiligung eines Prüfers ist die Evaluation zunächst subjektiv.

[292] Vgl. Hegner 2003, S. 19
[293] Vgl. Nielsen 1993, S. 33-37
[294] Vgl. Hegner 2003, S. 15

Dieser bemüht sich allerdings durch vorgegebene Leitfäden, die Begründung so objektiv wie möglich zu gestalten. Ein Diagnostiker, der z. B. überprüft, ob der Viewer mobil lauffähig ist, fällt sein Urteil nach Erfolg oder Misserfolg der Startfähigkeit auf dem mobilen Endgerät. Der Leitfaden schreibt z. B. die Anzeige der Startmaske auf dem Bildschirm des Endgeräts als Kriterium vor. Allerdings kann der Benutzer nicht erkennen, ob interne Systemfehler aufgetreten sind oder ob das System stabil läuft. Die Objektivität beschränkt sich also auf den Grad der vorgegebenen Richtlinien. Für eine Erstbegutachtung bietet sich das analytische Verfahren aufgrund der Vorgabemöglichkeit von Prüfleitfäden deshalb sehr gut an. Die Leitfäden können im vorliegenden Fall entsprechend den Anforderungen des Katalogs definiert werden. In Kapitel 11.4.2 werden die Ergebnisse der analytischen Evaluation vorgestellt.

Zusammenfassend lässt sich die durchgeführte Bewertung wie folgt charakterisieren. Sie stellt eine formative Evaluation dar, die zum Ziel hat, mittels experimenteller und analytischer Techniken, über objektive Prozessdaten und subjektive Benutzerbeschreibungen zu einem kausalen Stopp/Go-Urteil zu kommen.

11.4.2 Erfüllung der Anforderungsdefinition

Die Struktur dieses Kapitels orientiert sich an der Anforderungsdefinition aus Kapitel 7.3. Sie gliedert sich in die Bereiche Schnittstellen-Anforderungen, technische Anforderungen und funktionale Anforderungen. Innerhalb der Bereiche wird mittels Checkboxen (dt. Kontrollkästchen) symbolisiert, welche Anforderungen damit konkret an das Diagnosesystem gestellt wurden.

Der Beleg für die Erfüllung einer Anforderung wird größtenteils anhand objektiver Daten – der Systemeigenschaften der Diagnosesoftware – erbracht. Die Daten ergeben sich aus der Softwarearchitektur (Kapitel 9 und 10), der Integrationsleistung beim Kooperationspartner (Kapitel 11.3) und der Fähigkeiten des Prototyps (Kapitel 11.2). Gestützt werden die Erläuterungen durch Expertenbegutachtungen. Die Fachkräfte aus der Nacharbeit und Verwaltung streben bei den Gutachten nach größtmöglicher Objektivität.

11.4.2.1 Schnittstellen-Anforderungen

- **Unterstützung von proprietären Basissystemen**

  Die Unterstützung des Basissystems des Kooperationspartners ist elementare Voraussetzung, um das Diagnosesystem überhaupt einsetzen zu können. Die Erfüllung dieser Anforderung belegt sich direkt aus der erfolgreichen Integration im Rahmen des Pilotbetriebs. Die Diagnoseskripte für die ausgewählten Steuergeräte hätten nicht erstellt werden können und der Viewer hätte nicht mit dem Fahrzeug kommunizieren können ohne das Zusammenspiel mit dem proprietären Basissystem.

  Der Anspruch dieses Punkts liegt allerdings nicht nur in der Integration des Basissystemtreibers, sondern auch darin, aus der Unterstützung keine Einschränkung werden zu lassen. Das Diagnosesystem sollte nicht restriktiv auf den Einsatz mit dem aktuellen Basissystem des Kooperationspartners beschränkt sein. Bei einem Wechsel des Basissystems beim Kooperationspartner oder der Erschließung neuer Nutzer soll das Konzept eine Erweiterung in diese Richtung unterstützen. Deshalb sind Viewer und Editor anpassungsfähig gestaltet, was die Verarbeitung der Basissystemdaten betrifft.

Der Viewer erlangt diese Flexibilität durch das Verwenden programmiertechnischer Interfaces (dt. Schnittstellen). Diese werden durch die Programmiersprachen der .Net-Technologie unterstützt und stehen somit auf der Viewer-Plattform zur Verfügung. Die technischen Hintergründe und die genaue Form der technischen Umsetzung sind in Kapitel 10.4.1 ausführlich erläutert. Für den Viewer ergibt sich daraus der generelle Vorteil, dass bei der Integration eines neuen Basissystems die Kernfunktionalität nicht verändert werden muss. Es ist lediglich eine zusätzliche Implementierung der Schnittstelle für den neuen Basissystemtreiber durchzuführen. Damit erfüllt der Viewer den Anspruch, flexibel proprietäre Basissysteme zu unterstützen.

Der gleiche Anspruch wird für den Editor erhoben. Hier mündet die Erfüllung der Forderung in einer anpassungsfähigen Importmöglichkeit der Zugriffsinformationen. Im folgenden Kontrollpunkt wird dieser Anspruch überprüft.

- **Import der Steuergeräteinformationen in die Entwicklungsumgebung**

  Für den Import der Steuergeräteinformationen in den Editor gilt es die gleiche Flexibilität bereitzustellen, wie bei dem Zugriff auf das Basissystem durch den Viewer. Beide Softwarekomponenten dürfen nicht starr auf das Basissystem des Ersteinsatzes eingeschränkt sein.

  Dafür muss zunächst sichergestellt werden, dass das Importmuster der Steuergeräteinformationen anpassbar ist. Erfüllt wird dieser Anspruch durch die Bereitstellung einer individuell parametrierbaren XML-Schnittstelle. Die XML-Datei gibt das Datenmodell vor, in dem die Zugriffsinformationen strukturiert sind. Anhand dieser Datenhierarchie stellt der Editor die Zugriffsinformationen dar und kann sie den Diagnoseformen zuweisen. Die technischen Hintergründe der XML-Schnittstelle sind in Kapitel 10.3.2 beschrieben.

  Elementar für die Verarbeitung sind folgende Hierarchieelemente: der Job (Steuergerätebefehl) mit den zugehörigen Aufrufparametern und den Result-Sets (Ergebniswerten). Der Job ist einem Steuergerät zugeordnet. Für die Bewertung des Basissystemimports wurde versucht, die Struktur eines anderen Basissystems abzubilden.

  Ein Format zur Beschreibung von Steuergerätedaten, das zunehmend an Bedeutung gewinnt, ist der ODX-Standard des ASAM MCD-Systems. Er stellt einen offenen Referenzstand für die Diagnose von Steuergeräten dar und wurde von dem Arbeitskreis für Standardisierung Automatisierungs- und Messsysteme (kurz ASAM) konzipiert. In dem Konzept finden sich für den Steuergerätezugriff ähnliche zentrale Elemente wie im Neu-System wieder. Die Spezifikation beschreibt einen Job, der mit Ein- und Ausgabeparameter definiert werden muss und auf dem Diagnosesystem ausgeführt wird[295]. Intern ruft der Job einen Diagnosedienst des Steuergeräts auf. Diese Form der Jobs wird bei ODX als SINGLE-ECU-JOB bezeichnet. Die nähere Betrachtung des Formats und die Befragung von Fachkräften zeigt, dass ODX als offener Standard natürlich weitaus mehr konzeptionelle Möglichkeiten bietet. Beispiele sind die Unterstützung von multiplen Steuergerätejobs[296] oder der direkte Aufruf eines Steuergeräte-internen Dienstes[297]. Durch das parametrierbare XML-Schnittstellenkonzept und die nutzerspezifische

[295] Vgl. ASAM ODX 2006, S. 89-92
[296] Vgl. ASAM ODX 2006, S. 177
[297] Vgl. ASAM ODX 2006, S. 277

Gestaltung der Diagnoseformen (siehe Kapitel 10.3.3) hat sich allerdings grundsätzlich gezeigt, dass der Editor für den Import dieses Basissystems erweiterbar wäre.

Dieser Erfüllungsgrad soll zur positiven Bewertung der Anforderung in der ersten Prototypphase genügen.

11.4.2.2 Technische Anforderungen

- **Betriebssystemübergreifende Lauffähigkeit der HMI-Software (Mobilität)**

  Der Viewer ist auf Basis der .Net-Technologie entwickelt und stützt sich im mobilen Bereich auf das Compact Framework. Der Mobile Viewer ist folglich auf allen mobilen Geräten lauffähig, auf denen das Compact Framework installiert betriebsfähig ist. Bei der Integration wurden für das Pilotprojekt Handhelds als geeignete mobile Endgeräte bestimmt. Diese werden unter anderem mit Microsoft Betriebssystemen vertrieben. Die mobilen Betriebsysteme von Microsoft, wie Windows Mobile, CE oder Pocket PC, unterstützen das Compact Framework nativ oder können dahingehend nachgerüstet werden.

  Der Desktop Viewer für Tower-PCs und Arbeitsplatzrechner benötigt das .Net Framework und ist somit auf Microsoft Betriebssystemen wie NT, 2003 und XP lauffähig. Diese sind auf den Diagnoseterminal und den Verwaltungsrechnern beim Kooperationspartner bereits im Einsatz.

  Damit ist im Hinblick auf die Pilotprojektierung die betriebssystemübergreifende Lauffähigkeit des Viewers erfüllt. Neben dem Einsatz beim Kooperationspartner ist aufgrund der großen Verbreitung der Microsoft Betriebssysteme auf Arbeitsplatzrechner und aufgrund der zunehmenden Etablierung im mobilen Sektor (siehe Kapitel 9.3.3.4) auch eine gewisse allgemeine betriebssystemübergreifende Lauffähigkeit als erfüllt zu bewerten.

- **Stand-alone-Lauffähigkeit der HMI-Software**

  Für den ortsungebundenen Einsatz, z. B. in Testlabors der Steuergeräteentwicklung, bei der Analyse, in der Nacharbeitswerkstatt oder beim Service vor Ort, ist eine eigenständige Lauffähigkeit ohne Netzwerkverbindung gefordert.

  Die HMI-Software läuft in der Desktop- und mobilen Version ohne den Zugriff auf einen Server. Sie benötigt lediglich eine Verbindung zum Fahrzeug, um mit dem Steuergerät kommunizieren zu können. Alle dafür notwendigen Ressourcen wie Treiber, Steuergeräteinformationen oder Diagnoseskripte können lokal von dem Endgerät geladen werden. Damit sind die geforderten Eigenschaften des Stand-alone-Betriebs im Sinne dieser Arbeit erfüllt.

- **Ausgabe in hoher Bildqualität**

  Diese Anforderung entstand beim Kooperationspartner aus der nicht mehr zeitgemäßen Beschränkung des Ist-Systems auf wenige Grundfarben. Die Diagnosemasken sind generell in Schwarz und Weiß gestaltet, mit der Möglichkeit Rot und Grün als Signalfarbe zu verwenden.

Mit dem neuen System sollen Grafiken und Bilder realitätsgetreu dargestellt werden. Es wird also eine hohe Farbtiefe und entsprechende Auflösung gefordert.

Mit der Unterstützung von True Color wird eine realitätsnahe Abbildung verbunden. Kombiniert mit der auflösungsunabhängigen Vektorgrafik wird das SVG-Format und somit der Viewer diesem Anspruch gerecht (siehe Kapitel 7.3.2.3). Mit der im Projekt ausgewählten mobilen Hardware kann dieses Potenzial nicht ganz ausgeschöpft werden. Gängige Pocket PCs (Handhelds mit Windows Mobile) liefern aber bereits eine Farbtiefe von 16 Bit und eine Auflösung 640x480 Pixeln (siehe Kapitel 11.3.2). Damit kann eine VGA-Grafik mit High Color erzeugt werden. Um Farbverläufe und technische Abbildungen gut erkennbar darzustellen, wurde die Qualität von den Benutzern als völlig zufriedenstellend gewertet.

Für das Diagnosesystem ist die Forderung nach hoher Bildqualität damit erfüllt. Die Gesamtleistung des Systems kann durch den Einsatz künftiger leistungsfähigerer Hardware noch gesteigert werden.

- **Unterstützung von Mehrsprachigkeit**

  Dieser Punkt fordert neben der Darstellung der Inhalte in mehreren Sprachen auch, dass diese keine zusätzliche Speicherung der Diagnoseskripte zur Folge hat (siehe Kapitel 11.4.2.3 "Redundanzfreies Erstellen der Masken"). Das gleiche Diagnoseskript soll folglich in mehreren Sprachen ausgegeben werden können. Dadurch wird eine verwaltungsintensive redundante Aufbewahrung der gleichen Dateien in mehreren Sparchversionen vermieden.

  Bei dem neuen Diagnosesystem sind die Übersetzungen in gesonderte Dateien ausgelagert. Für jede Sprache gibt es eine zusätzliche Datei, die auf dem Endgerät hinterlegt wird. Das Diagnoseskript enthält Schlüssel für die Textausgabe. Diese werden über die Sprachdateien aufgelöst und liefern so den Ausgabetext zurück. Auf diese Weise kann zur Laufzeit in eine beliebige Sprache umgeschaltet werden. Das Diagnoseskript kann für zusätzliche Sprachen in seiner ursprünglichen Form erhalten bleiben.

  Da .Net den Unicode-Sprachsatz (siehe Kapitel 7.3.2.4) unterstützt, können alle gängigen gesprochenen Sprachen, auch z. B. aus dem asiatischen Raum, mit dem Viewer dargestellt werden.

  Die Möglichkeiten der Internationalisierung sind damit über die Ansprüche des Kooperationspartners hinaus abgedeckt.

- **Verwendung von Standards**

  Die technischen Richtlinien des Kooperationspartners schränken bei diesem Punkt das am Markt befindliche Potenzial zur Standardisierung ein. Das IT-Zentrum des Kooperationspartners gibt Softwarestandards vor, die bei der eingesetzten Software verwendet werden sollen oder zum Teil auch müssen.

  Die Plattform-Technologien des Viewers und des Editors (.Net und Eclipse), sind beide als Empfehlungen beim Kooperationspartner verzeichnet. Gleiches trifft auf die unterstützenden Sprachen C#, Java und JavaScript zu. Des Weiteren sind die Standardformate der W3C Group SVG und XML vermerkt.

Bis auf die Odyssey- und Intesis-Plug-ins, die ausschließlich zur Darstellung der SVG-Grafik auf dem Viewer dienen, werden nur kostenfreie öffentlich zugängliche Standardtechnologien verwendet. Aus Projektsicht ist der Standardisierungsgrad somit völlig ausreichend. Für eine vollständige Standardisierung, die unabhängig von kostenpflichtigen Softwarekomponenten ist, wird in Kapitel 12.2 ein Ausblick für eine mögliche Realisierung gegeben.

11.4.2.3 Funktionale Anforderungen an die HMI-Software

- **Interaktion mit Benutzer**

  Die Interaktion stellt wie die Unterstützung des Basissystems eine elementare Grundvorrausetzung für den Einsatz im Diagnosebereich dar (siehe 7.3.3.1).

  In Kapitel 9.3.1.3 wurde die Interaktionsfähigkeit von SVG beschrieben. Die Fähigkeit erlangt das Grafikformat durch die Unterstützung von JavaScript. Dieser konzeptionelle Beleg, kombiniert mit der erfolgreichen Integration im Pilotbereich, soll an dieser Stelle genügen. Denn ohne Interaktionsfähigkeit könnte das Diagnosegerät weder auf Steuerungsaufträge des Benutzers reagieren noch Werte des Steuergeräts anzeigen. Es wäre nicht möglich gewesen, in der Nacharbeit die Anwendungsfälle zu bearbeiten.

- **Diagnoseoberfläche unterstützt Maskensystem**

  Als Ausgangspunkt der Arbeit wurde die steigende Anzahl der Steuergeräte im Fahrzeug bestimmt. Damit das Diagnosesystem die momentane und künftige Variation der Steuergerätetypen beherrschen kann, ist eine flexible Oberflächengestaltung notwendig. Innerhalb der HMI-Software muss es demnach möglich sein, frei gestaltbare Diagnosemasken zu laden.

  Bei dem neuen Diagnosesystem wird dies durch die Verarbeitung der Diagnoseskripte erreicht. Der Viewer lädt das Skript und baut aus dem darin enthaltenen SVG-Code die Diagnosemaske auf. Beispiele unterschiedlicher Diagnosemasken finden sich in Kapitel 11.3.4.

  Von der geladenen Maske aus kann auf weitere Masken verwiesen werden. Die Navigation kann durch den Benutzer oder auch automatisiert erfolgen. Das System bietet die Möglichkeit, in ein Skript mehrere Masken zu integrieren oder pro Maske ein Skript anzufertigen. Die Verlinkung auf weitere Masken kann also auch skriptübergreifend erfolgen.

  Die vorgeführten Beispiele aus Kapitel 11.3.4 und der Einsatz in der Nacharbeit liefern den exemplarischen Beleg für die Unterstützung des Maskensystems.

- **Freie Skalierbarkeit der Oberfläche**

  In dem Anforderungskatalog wurde der Anspruch erläutert, dass erst durch eine qualitätsverlustfreie Skalierung der Diagnosemasken die Vorteile der betriebssystemübergreifenden Nutzung vollständig ausgeschöpft werden können (siehe Kapitel 7.3.3.3). Die Masken können für Desktop-Monitore konzipiert werden und auf handflächengroßen PDA-Monitoren auf eine lesbare Größe skaliert werden. Umgekehrt kann eine für den Handheld erstellte Maske zur Anzeige auf einem Desktop-Monitor auf Vollbild

vergrößert werden. In beiden Fällen bleibt die Diagnosemaske in ursprünglicher Bildqualität erhalten.

Die Skalierfähigkeit erlangt das verwendete Grafikformat SVG durch seine Eigenschaft als Vektorgrafik. Vektorgrafiken erfüllen den Anspruch der qualitätsverlustfreien Skalierung (siehe Kapitel 9.3.1.1). Da das Diagnosesystem auf SVG basiert, erfüllen die Masken den geforderten Anspruch dieses Punkts. Kapitel 11.3.4 liefert hierfür ein Beispiele in Form eines Screenshots.

11.4.2.4 Funktionale Anforderungen an die Entwicklungsumgebung

- **Kompilierung nicht erforderlich und WYSIWYG-Unterstützung**

  Mit der Umsetzung des WYSIWYG-Konzepts wird eine deutliche Einsparung in der Bearbeitungszeit von Diagnoseskripten erwartet. Aufwendige Kompilierungs- und Prüfvorgänge einer Diagnosemaske sollen damit verkürzt, wenn nicht sogar überflüssig werden (siehe Kapitel 7.2.2). Die Verbesserung der Arbeitsabläufe wird in Kapitel 11.4.3 untersucht.

  Im Rahmen der analytischen Evaluation wird die grundsätzliche Umsetzung dieser Anforderung beurteilt. In einem Editor-Viewer-Konzept wie dem vorliegenden, fällt der Beweis in den ersten Prototypphasen allerdings sehr schwer.

  Durch die Abarbeitung der Anwendungsfälle in der Verwaltung konnte belegt werden, dass beim grafischen Gestalten eines Diagnoseskripts der Quellcode direkt miterzeugt wird. Dieser Quellcode muss für die grafische Anzeige in der Zeichenoberfläche des Editors nicht kompiliert werden. Er wird direkt interpretiert und angezeigt. Innerhalb des Editors gilt also, dass das, was man sieht, auch das ist, was erzeugt wird (What You See, Is What You Get – WYSIWYG).

  Allerdings verlässt das Skript für den Diagnoseeinsatz den Rahmen des Editors. Der Quellcode des Skripts wird auch vom Viewer nur interpretiert und dargestellt, allerdings auf einer neuen Plattform. Der Editor basiert auf Eclipse und interpretiert das SVG mit dem Batik Toolkit. Der Viewer hingegen basiert auf .Net und interpretiert mit dem Intesis SVG-Plug-in. Beide unterstützen die SVG-Hauptspezifikation. Allerdings gibt es bei beiden Interpretern Einschränkungen von Spezifikationsbereichen, die noch nicht vollständig implementiert sind. Auch birgt ein öffentlicher Standard, besonders im Grafikbereich, immer Spielraum für unterschiedliche Auffassungen der Spezifikationen.

  Diese Divergenzen lassen sich allerdings innerhalb der ersten Prototypphase schwer erkennen, sondern zeigen sich in der Regel erst über einen längeren Zeitraum des Einsatzes.

  Deshalb wird an dieser Stelle die WYSIWYG-Unterstützung nur mit Vorbehalt bestätigt. Innerhalb des Pilotbetriebs wurden keine Unterschiede in der Verarbeitung festgestellt. Die Masken hatten im Viewer das gleiche Aussehen wie im Editor. Auch trat kein Fall auf, bei dem eine durch den Editor erstellte Diagnosefunktionalität im Viewer nicht ausgeführt werden konnte.

- **Redundanzfreies Erstellen der Masken ("Unikat")**

  Durch Mehrsprachigkeit oder mobile Lauffähigkeit machen heutige Systeme oft ein zusätzliches Erstellen der bestehenden Maske erforderlich. Sie wird erneut für die jeweilige Sprache oder für das Endgerät erstellt, ist aber an sich in Funktion und Form der ursprünglichen Maske gleich. Durch die zusätzlichen Dateien vermehrt sich der Verwaltungsaufwand bei der Datenhaltung und bei Änderungen enorm (siehe Kapitel 7.3.4.2).

  Das Konzept des Diagnosesystems zeigt, dass ein (unikates) Diagnoseskript für den mobilen und Desktop-Betrieb erstellt wird. In Kapitel 9.1 wird diese Fähigkeit durch die Eigenschaft der qualitätsverlustfreien Skalierung begründet. Die Mehrsprachigkeit kann über eine zusätzliche Sprachdatei ergänzt werden. Diese Sprachdatei kann für beliebig viele Diagnoseskripte genutzt werden. Die Textstellen der Diagnosemaske werden zur Laufzeit übersetzt. Die Notwendigkeit mehrerer Sprachversionen der gleichen Masken entfällt folglich.

  Für die Verwaltung entsteht unabhängig von betriebssystemübergreifenden oder internationalem Einsatz nur der Aufwand, das Diagnoseskript mit den zugehörigen Sprachdateien zu verwalten. Durch die ausgelagerten Sprachdateien ergibt sich für die Verwaltung noch der zusätzliche Vorteil, dass man um eine neue Sprache hinzuzufügen, nur die bestehenden Sprachdateien übersetzen und einpflegen muss. Der funktionale Bereich muss nicht verändert werden.

  Die Verwaltungsmitarbeiter haben somit im Rahmen des Pilotbetriebs bestätigt, dass keine redundante Speicherung der Diagnoseskripte aufgrund von Internationalisierung, Kompilierung oder mobilen Einsatzes notwendig ist.

- **Grafisches Erstellen der Masken**

  Mit der grafischen Erstellmöglichkeit der Diagnosemasken wird vorrangig eine Verkürzung der Prozesszeiten angestrebt. Deshalb wird dieser Punkt noch ausführlich in Kapitel 11.4.3 behandelt. Für die Bewertung des Anforderungskatalogs soll diese Eigenschaft des Editors zunächst nur konzeptionell nachgewiesen werden.

  In Kapitel 11.2.1 wurden die Werkzeuge aufgelistet, mit denen eine Maske erstellt werden kann. Innerhalb eines Fenster der Editoroberfläche werden Diagnoseformen angeboten. Mit den Diagnoseformen können z. B. Werte des Steuergeräts grafisch dargestellt werden oder Befehle an das Steuergerät geschickt werden. Die grafischen Möglichkeiten der Diagnoseformen werden im folgenden Kontrollpunkt näher untersucht. Neben den Diagnoseformen bietet der Editor die üblichen Zeichenwerkzeuge eines grafischen Editors an (Linie, Kreis oder Rechteck). Auch die Handhabung ist der eines handelsüblichen Grafikdesigner ähnlich. Über Datei/Neu können leere Diagnoseskripte angelegt werden und per Drag & Drop können die Diagnoseformen in das leere Skript gezogen werden. Per Klick auf ein Zeichenwerkzeug bzw. eine Diagnoseform wird dieses bzw. diese ausgewählt.

  Die Zuweisung der Steuergerätefunktionalität zu den Diagnoseformen erfolgt über ein Eigenschaftsfenster. Per Auswahllisten werden sequenziell erst das Steuergerät, dann der Befehl und anschließend die zugehörigen Parameter bzw. Ergebniswerte ausgewählt.

Mit den Diagnoseformen, Zeichenwerkzeugen, Eigenschaftslisten und der Zeichenoberfläche kann die gesamte grafische Gestaltung der Diagnoseoberfläche bewältigt werden. Es sind hierfür keine manuelle Eingabe bzw. manuelles Änderungen des Skript-Quellcodes notwendig. Damit ist der Anspruch, eine Diagnosemaske grafisch erstellen zu können erfüllt.

- **Umfassende grafische Gestaltungsfunktionalität**

  Das Angebot einer umfassenden grafischen Gestaltungsweise wird größtenteils durch subjektive Maßstäbe der Nutzer beurteilt. Es lässt sich nicht an konkreten Systemeigenschaften festmachen.

  Positiv auf das Urteil wirken sich z. B. die im vorherigen Kontrollpunkt beschriebenen Zeichenwerkzeuge aus. Linien. Kreise oder Rechtecke bieten einen gestalterischen Komfort wie in handelsüblichen Zeichenprogrammen.

  Der Kooperationspartner hatte mit dem Anspruch einer "umfassenden grafischen Gestaltungsfunktionalität" auch die Verbesserung der Diagnoseformen im Sinn. Das Ist-System bietet nur eine geringe Anzahl an vorgefertigten Objekten zur Diagnose (Textfelder, LEDs, Regler etc.) an. Die Erweiterung um weitere Formen sowie deren grafische Aufbesserung ist umständlich bzw. durch den Kooperationspartner selbst nicht möglich.

  Im neuen System basieren die Diagnoseformen auf dem SVG-Format mit JavaScript-Unterstützung. Sie bieten somit hohen grafischen und funktionalen Komfort. Auch kann der Umfang nutzerspezifisch leicht erweitert werden. Es können zusätzliche Formen mit SVG und JavaScript gestaltet werden und im Editor registriert werden (siehe Kapitel 10.3.3). Des Weiteren ist JavaScript eine verbreitete Skriptsprache, die von vielen Programmierern des Kooperationspartners bereits beherrscht wird. Die Diagnoseformen können somit auch selbstständig ergänzt werden. Auch für den Einsatz bei weiteren Kfz-Herstellern wird mit dem Konzept der erweiterbaren Diagnoseformen eine entsprechende Flexibilität bewahrt.

  Bei dem vorliegenden Pilotprojekt wurde die grafische Gestaltungsfunktionalität durch die Verwaltungsmitarbeiter als deutlich verbessert beurteilt.

- **Unterstützung von Upload/Download**

  Der Editor bietet die Möglichkeit, die Diagnoseskripte direkt über einen angegebenen Pfad auf einen Server zu übertragen und von diesem herunterzuladen.

  Die Up- und Downloadmöglichkeit wird in der implementierten Form als ausreichend beurteilt, ist aber in dieser Projektphase von geringerem Interesse. Der Abgleich mit dem Server wird erst in späteren Phasen verstärkt zum Einsatz kommen, sobald mehrere Diagnoseskripte umgesetzt und im Einsatz sind.

  Viele der in diesem Kapitel beurteilten Ansprüche des Anforderungskatalogs sollen hingegen bereits in der ersten Prototypphase ihre Vorteile zeigen und entsprechende Prozessverbesserungen bewirken. Mit deren Überprüfung beschäftigt sich das nächste Kapitel.

11.4.3 Verbesserung innerhalb der Geschäftsprozesse

Ziel des Anforderungskatalogs ist es, ein verbessertes Diagnosesystem zu definieren. Die Verbesserung soll Systembrüche auflösen, Prozesszeiten verkürzen und die Anwenderzufriedenheit steigern.

Im letzten Kapitel wurde auf analytische Weise der Erfüllungsgrad des Anforderungskatalogs bewertet. In die Bewertung sind größtenteils objektive Systemeigenschaften eingeflossen. Systemeigenschaften müssen nicht über mehrere und unterschiedliche Anwendungsfälle getestet werden. Das System erfüllt entweder eine Eigenschaft oder eben nicht.

Für den Beleg von Prozessverbesserungen in einem Pilotbetrieb ist man hingegen auf die Auswertung mehrerer Anwendungsfälle angewiesen. Eine Alternative zu den Anwendungsfällen wäre eine Simulation der Prozesse. Die Neumodellierung und Simulation der Prozesse wäre aber in einer frühen Prototypphase nicht sinnvoll. Denn die Arbeitsabläufe sind durch mögliche Optimierungen in den weiteren Prototypphasen noch zu starken Schwankungen unterworfen, um sie fundiert neu erfassen zu können. In den Anfängen des Pilotbetriebs bietet sich deshalb die experimentelle Evaluation über Anwendungsfälle an (siehe Kapitel 11.4.1).

Anhand von kontrollierten Experimenten (Anwendungsfällen) wird der Prototyp bewertet. Zur Beurteilung der Prozessverbesserung werden in den Anwendungsfällen größtenteils die Veränderungen der Prozesszeiten untersucht. Des Weiteren werden für die Beurteilung der Anwenderzufriedenheit die Prozessbeteiligten nach der Abarbeitung des Anwendungsfalls befragt, wie sie die Arbeit mit dem neuen System empfanden.

Neben Anwenderzufriedenheit und Prozesszeiten wurde in Kapitel 7 noch eine dritte Schwachstelle ausfindig gemacht, die Systembrüche. Aufgrund ihres eindeutig definierten Auftretens kann die Aufhebung der Systembrüche, neben den Anwendungsfällen, auch anhand der theoretischen Soll-Prozesse aus Kapitel 7 belegt werden.

Die Fähigkeit, die Schwachstellen aufzulösen, erhält der Prototyp durch die Umsetzung der Anforderungsdefinitionen. Die Erfüllung der Definitionen durch den Prototyp wurde bereits bewertet. Im Folgenden soll gezeigt werden, dass diese Anforderung auch ihre Berechtigung hat. Deshalb werden zu jedem Geschäftsprozesse die Anforderungen aufgelistet, die vorrangig zur Auflösung der Schwachstellen beigetragen haben.

11.4.3.1 Geschäftsprozess Verwaltung

Im Verwaltungsprozess erfolgt die Neuerstellung, Bearbeitung und Bereitstellung der Diagnoseskripte (siehe Kapitel 5.8).

Die Schwachstellenanalyse aus Kapitel 6 hat ergeben, dass vorwiegend bei der Maskengestaltung und der Integration der Steuergerätefunktionalität Defizite bestehen. Speziell die folgenden Punkte des Anforderungskatalogs setzen sich mit diesen Defiziten auseinander. In Klammern ist der Anforderungstyp aus Kapitel 7.3 vermerkt:

- Import der Steuergeräteinformationen in die Entwicklungsumgebung (Schnittstellenbasiert)
- Verwendung von Standards (technisch)
- Kompilierung nicht erforderlich und WYSIWYG-Unterstützung (funktional)

- Redundanzfreies Erstellen der Masken – "Unikat" (funktional)
- Grafisches Erstellen der Masken (funktional)
- Umfassende grafische Gestaltungsfunktionalität (funktional)

Die Folgen der Defizite des derzeitigen Diagnosesystems zeigen sich in Systembrüchen, unverhältnismäßig langen Prozesszeiten und in der Benutzerunzufriedenheit. Der Beurteilung soll zeigen, ob der Prototyp die Schwachstellen beseitigt.

- **Auflösung der Systembrüche**

  In Kapitel 6 wurde an zwei Stellen in den Verwaltungsprozessen vermehrt Anwendungssystembrüche erfasst. Der eine Prozessbereich ist das Einfügen von zusätzlichen Steuergerätebefehlen in das Diagnoseskript und die Bearbeitung von bereits eingetragenen Steuergerätebefehlen. Der zweite Prozessbereich beschreibt das Kompilieren und Ausführen des Skripts, um die Bearbeitung zu überprüfen.

  Das Ist-System verlangt von dem Skriptbearbeiter, zwischen mehreren Anwendungen zu wechseln. So muss z. B. zur Übertragung der Steuergeräteinformationen in das Diagnoseskript vom Texteditor in ein Toolset gewechselt werden. Das Toolset zeigt die Steuergerätedaten an. Nach dem Eintragen eines Befehls muss das Skript gespeichert, kompiliert und das Kompilat geladen werden. Dafür ist auch wieder der Wechsel zwischen weiteren systemspezifischen Programmen notwendig. Die Schwachstellen und technischen Hintergründe sind ausführlich in Kapitel 6.4 und 6.5 dokumentiert.

  Weiter wurde analysiert, dass innerhalb der beiden Prozessbereiche teilweise 60 % aller Tätigkeitsübergänge einen Anwendungssystembruch zur Folge haben. Die Brüche, sprich der erzwungene Wechsel zwischen Anwendungen, schaffen eine unruhige und fehleranfällige Arbeitsweise bei der Programmierung der Diagnoseskripte (siehe Kapitel 7.2.1).

  Aus den einzelnen autarken Anwendungssystemen wurde deshalb eine ganzheitliche Entwicklungsumgebung geschaffen.

  Dafür sind zwei der einleitend aufgelisteten Anforderungen elementar wichtig. Zum einen der Import der Steuergeräteinformationen in die Entwicklungsumgebung und zum anderen die funktionelle Unterstützung des WYSIWYG-Konzepts.

  Durch die Möglichkeit, die Steuergeräteinformationen im zentralen Editor des Prototyps abrufen und direkt Diagnoseformen zuweisen zu können, löst sich die Notwendigkeit eines zusätzlichen Toolsets auf. Damit verschwinden an dieser Stelle die Systembrüche.

  In gleichem Maße reduziert WYSIWYG die Systembrüche. Das Bild wird in der Form, in der es entwickelt wird auch dargestellt (interpretiert). Die Kompilierung wird unnötig. Außerdem kann das Ergebnis der Bearbeitung auf der Editor-Zeichenoberfläche jederzeit kontrolliert werden. Damit verschwindet die Notwendigkeit, das Skript mit einem zusätzlichen Programm zu kompilieren und es danach zur Kontrolle mit dem Viewer zu laden. Somit lösen sich auch diese Systembrüche auf.

  Die Anzahl der Systembrüche sinkt in beiden Prozessbereichen auf 0 %. Die Auflösung wird durch die Modellierung der Soll-Prozesse in Kapitel 7.2.1 nachgewiesen. Da der

Prototyp exakt auf Basis der Soll-Ansprüche entwickelt wurde, sind die Soll-Modelle aussagekräftig. Des Weiteren belegt die Beschreibung der Anwendungsfälle in der Verwaltung diese Verbesserung (siehe Kapitel 11.3.4.2).

Die Auflösung der Systembrüche innerhalb der analysierten Prozessbereiche ist folglich für die erste Prototypphase als erfüllt zu bewerten.

- **Verbesserung der Prozesszeit**

  Die Optimierungsmethode TCT (Total Cycle Time) besagt, dass die Auflösung von Systembrüchen auch immer eine Reduktion der Prozesszeit zur Folge hat (siehe Kapitel 7.1). Deshalb ist zu erwarten, dass der Import der Steuergerätedaten in den Editor und die Auflösung der Kompilierungsnotwendigkeit sich auch positiv auf die Prozesszeit auswirken.

  Aber noch ein weiterer Aspekt wurde bei der Prozessoptimierung als wesentliche Schwachstelle definiert das textbasierte Erstellen der grafischen Diagnosemasken. Die grafischen Oberflächen müssen per Texteditor editiert werden. Erst nach dem Kompilieren und Laden des Kompilats kann die Oberfläche betrachtet und kontrolliert werden. Diese macht die Positionskontrolle und potenzielle Nachbesserung sehr zeitaufwendig.

  Deshalb wurden neben dem Import und WYSIWYG-Konzept in den Anforderungskatalog auch noch die Punkte "grafisches Erstellen der Masken" und "umfassende grafische Gestaltungsfunktionalität" als Anspruch aufgenommen.

  Bei der Soll-Modellierung wurde durch den Wegfall der Kompilierung und der grafischen Gestaltung eine Prozesszeitverbesserung von 37 % in Aussicht gestellt (siehe Kapitel 7.2.2.1).

  In der experimentellen Evaluation soll diese Verbesserung nun anhand diverser Anwendungsfälle nachgewiesen werden. Die Anwendungsfälle sind ausgewählte Auftragsszenarien, die in der Verwaltung eingehen. Sie werden auf beiden Diagnosesystemen abgearbeitet. In einer Tabelle werden die Daten des derzeitigen Diagnosesystems denen des neuen gegenübergestellt. Die Anwendungsfälle werden auf dem gleichen Rechner erfasst und von dem gleichen Verwaltungsmitarbeiter durchgeführt.

  Anhand von Ablaufprotokollen werden die Zeiten zu den einzelnen Tätigkeiten erfasst. Exemplarisch wird an dieser Stelle ein Ablaufprotokoll aufgelistet und erläutert. Weitere Ablaufprotokolle finden sich in der Quelle MobiViSt-Prozessdatenerfassung 2008. Alle Anwendungsfälle sind in dem Teilprozess "INPA-Skript erstellen/nachbearbeiten" und dem zugehörigen Prozess "Angeforderte Funktionalität ergänzen/verbessern" angesiedelt.

  Im folgenden Anwendungsfall wird das bereits aus der Analyse bekannte Beispiel einer funktionalen Erweiterung aufgegriffen (Kapitel 6.5.1.1). In Kapitel 6.4.2 wurde gezeigt, dass ein Viertel aller Aufträge zur Bearbeitung von Skripten pro Monat, den Einbau eines zusätzlichen Steuergerätebefehls erfordern. Der Anwendungsfall weist demnach eine hohe Relevanz auf. Es ist somit gerechtfertigt, ihn in die experimentelle Evaluation einfließen zu lassen.

Im vorliegenden Fall soll eine bestehende Diagnosemaske um die Anzeige des Türstatus erweitert werden. Eine LED zeigt über ZU/AUF an, ob eine Tür am Fahrzeug geöffnet ist. Ausführlich ist das Ablaufprotokoll in der Quelle MobiViSt-Prozessdatenerfassung 2008 nachzulesen. Dort werden die einzelnen Schritte und die Metadaten des Anwendungsfalls beschrieben. Die folgende Tabelle zeigt nur die Überschriften der einzelnen Schritte:

Nr.	**Tätigkeit**	**Dauer-Ist <Sek>**	**Dauer-Neu <Sek>**
1	Entwicklungsumgebung öffnen	45	55
2	Skript öffnen	35	20
3	Grafisches Objekt auswählen	30	3
4	Größe der Grafik bestimmen	10	5
5	Zielposition der Grafik freilegen	45	10
6	Grafik positionieren	15	5
7	Grafik beschriften	30	10
8	Wechseln in EDIABAS Toolset32	5	-
9	Steuergerätedaten laden	20	15
10	Job auswählen	30	30
11	Job markieren & kopieren	20	-
12	Wechseln in Texteditor	3	-
13	Job einfügen	10	-
14	Wechseln in EDIABAS Toolset32	5	-
15	Resultwert auswählen	30	30
16	Result markieren & kopieren	15	-
17	Wechseln in Texteditor	3	-
19	Result einfügen	10	-
20	Skript speichern	3	3
21	Skript kompilieren	40	-
22	Zur Überprüfung der Anzeige in INPA-Loader öffnen	10	-
23	Funktionale Überprüfung	-	-
	Gesamtzeit Sek. (MM:SS)	**414 (6:54)**	**186 (3:06)**

Tabelle 11.3 Systemvergleich bei Anwendungsfall: Funktionalität ergänzen

Das Ablaufprotokoll zeigt, dass sich die Durchlaufzeit zur technischen Abarbeitung des Auftrags von 6 Minuten 54 Sekunden (Ist-System) auf eine Dauer von 3 Minuten 06 Sekunden (Neu-System) verbessert hat. Dies würde einer Verkürzung um über 50 % entsprechen. Aufgrund der hohen Frequenz dieses Anwendungsfall-Typs wurde die Arbeit mit dem neuen Editor in diese Richtung optimiert. Da diese Verbesserung aber

eben nur einen Anwendungsfall-Typ repräsentiert und in der ersten Prototypphase keine umfassende Prozesssimulation möglich ist, wurden noch weitere Typen von Anwendungsfällen abgearbeitet. Das abschließende Urteil stellt eine Gesamtaussage über alle Anwendungsfälle dar. Folgende Auswertung führt die Daten der erfassten Anwendungsfälle aus der Verwaltung zusammen:

```
Bereich: Kooperationspartner, Basissystementwicklung

Anwendungsfall: Maske "Digital2" in "SMG3 Sequenzielles M Getriebe" für E60
um Anzeige "Status Tür" erweitern
      Dauer Ist-System (Sek.)    = 414
      Neu Ist-System (Sek.)      = 186
      Verkürzung um (%)          = 55,1

Anwendungsfall: Maske "Lichtschalter" in "Status Eingänge" für LSZ E46 er-
stellen
      Dauer Ist-System (Sek.)    = 964
      Neu Ist-System (Sek.)      = 436
      Verkürzung um (%)          = 54,8

Anwendungsfall: Maske "Heiz-Klima Bedienteil" in "IHKA87" grafisch verbessern
      Dauer Ist-System (Sek.)    = 2350
      Neu Ist-System (Sek.)      = 2105
      Verkürzung um (%)          = 10,4

Anwendungsfall: Maske "Ansteuern Gang einlegen" in "SMG3 Sequenzielles M Ge-
triebe" für E60 grafisch verbessern
      Dauer Ist-System (Sek.)    = 2525
      Neu Ist-System (Sek.)      = 2205
      Verkürzung um (%)          = 12,7

Anwendungsfall: Maske "Digital2" aus "SMG3 Sequenzielles M Getriebe" für E60
ohne Erweiterung erstellen
      Dauer Ist-System (Sek.)    = 1180
      Neu Ist-System (Sek.)      = 740
      Verkürzung um (%)          = 37,3

Anwendungsfall: Maske "Berechnete Abstände der Sensoren" in " PDC_E65" für
E65 grafisch verbessern
      Dauer Ist-System (Sek.)    = 1775
      Neu Ist-System (Sek.)      = 1330
      Verkürzung um (%)          = 25,1

Durchschnittliche Gesamtreduktion auf alle Anwendungsfälle:
      Verkürzung um (%)          = 32,57

(Quelle: MobiViSt-Prozessdatenerfassung 2008)
```

Abbildung 11.12 Auswertung: Durchschnittliche Verbesserungsquoten in der Verwaltung

Der Pilotbetrieb zeigt, dass bei den Masken, die auf einfache grafische Weise gestaltet werden, eine Verkürzung von knapp über 50 % möglich ist. Bei Versuchen, die grafische Gestaltung zu verbessern, reduziert sich der Verbesserungsfaktor – ist aber immer noch deutlich. Gestützt auf die Anwendungsfälle kann mit dem neuen Diagnosesystem eine zeitliche Reduktion der Bearbeitungszeiten um ca. 33 % für den Teilprozess "Diagnoseskript erstellen/nachbearbeiten" bewirkt werden.

- **Erhöhung der Anwenderzufriedenheit**

  Neben der Auflösung der Systembrüche und der Reduktion der Prozesszeiten ist für die Akzeptanz des neuen Systems auch die Anwenderzufriedenheit von Bedeutung. Besonders bei Prozessverbesserungen, die auf der Integration neuer Software beruhen, kommt der Anwenderzufriedenheit besondere Wichtigkeit zu. Das neue System lebt gerade im Pilotbetrieb wie in den Anfängen des Produktivbetriebs von der Annahme und konstruktiven Kritik durch die Nutzer.

  Das Auflösen von Systembrüchen fördert bereits das angenehme Arbeiten mit dem Diagnosesystem. Allerdings wurden bei der Analyse des Ist-Systems noch weitere Schwachstellen von den Nutzern angeführt. Besonders die proprietäre Programmiersprache der Diagnoseskripte und die Verwaltungskomplexität durch die Vielzahl an notwendigen Dateien wurden hier genannt.

  Aus diesem Grund wurden die Verwendung von Standards und das redundanzfreie Erstellen der Diagnosemasken als Punkte in den Anforderungskatalog mitaufgenommen.

  Mit der Erfüllung der ersten Forderung entfällt für den Verwaltungsmitarbeiter die Aufgabe, eine proprietäre Skriptsprache für das neue System erlernen zu müssen. Das Neu-System beruht auf dem SVG-Standard und der JavaScript-Syntax. Besonders bei JavaScript ist aufgrund seiner Verbreitung die Chance sehr hoch, dass der Verwaltungsmitarbeiter in seiner Tätigkeit als Programmierer die Sprache bereits beherrscht. Dies verschafft bereits in der Prototypphase einen großen Vorteil in Bezug auf die Akzeptanz des neuen Systems.

  Die Reduktion der zu verwaltenden Dateien wird erst in den späteren Pilotphasen seinen Vorteil zeigen. Erst wenn die Anzahl der Skripte ein gewisses Niveau erreicht hat, wird für die zuständigen Verwaltungsmitarbeiter der entsprechende Nutzen spürbar. Das derzeitige System fordert für die Verwaltung eines Diagnoseskripts eine Quelle, ein Zwischen-Kompilat und ein Kompilat. Falls noch Fremdsprachen gefordert sind, jeweils ein Kompilat mehr und eine Textquelle mit den Übersetzungen. Das sind für ein Skript mit einer Fremdsprache fünf Dateien. Mit dem neuen System ist keine Kompilierung notwendig und die Übersetzung erfolgt zur Laufzeit. Es gibt folglich nur ein Skript und eine Datei in der jeweiligen Sprache. Die Anzahl reduziert sich also von fünf auf drei (Diagnoseskript + Standardsprachdatei + Fremdsprachendatei). Da die Sprachdateien auch für mehrere Diagnoseskripte genutzt werden können, erschließt sich hier weiteres Einsparungspotenzial. Entscheidend ist, dass das eigentliche auszuführende Skript nicht wie im bisherigen System für jede Sprache zusätzlich (redundant) gespeichert werden muss, sondern es nur ein relevantes Skript (Unikat) gibt. Damit wird die Verwaltungstätigkeit für den zuständigen Mitarbeiter transparenter und gerade bei Änderungen von bestehenden Skripten stark vereinfacht.

  Zusammenfassend stellt das neue System mit der ganzheitlichen Entwicklungsumgebung, der grafischen Gestaltungsmöglichkeit, den offiziellen Standards und dem reduzierten Dateiaufkommen ein benutzerfreundlicheres Diagnosesystem für die Verwaltung dar als das Ist-System des Kooperationspartners.

11.4.3.2 Geschäftsprozess Nacharbeit

In der Nacharbeit werden Fahrzeuge bearbeitet, die bei den finalen Prüfprozessen im Rahmen der Endmontage durchgefallen sind. Das fehlerhafte Fahrzeug soll dort unter Zuhilfenahme entsprechender Diagnosetätigkeiten in einen fehlerfreien Zustand überführt werden. Danach wird es in den ursprünglichen Prozess der Endprüfung zurückgeschleust (siehe Kapitel 5.4 und 5.6). Relevant für die Arbeit sind nur die mechatronischen bzw. das Steuergerät betreffenden Fälle. In Kapitel 5.6.2 werden diese Fälle durch den Teilprozess "Steuergerät diagnostizieren" zusammengefasst.

Bei der mechatronischen Diagnosetätigkeit wurden im Rahmen der Arbeit zwei verbesserungswürdige Bereiche erfasst. Der Prototyp soll Optimierungen im Rahmen der grafischen Gestaltung liefern und den Mobilitätsgrad des Diagnosesystems steigern (siehe Kapitel 6.5 und 6.6).

Um Mobilität in einem Diagnosesystem anzubieten, ist es nicht genug, einen zusätzlichen Viewer in das System mitaufzunehmen, der auf mobilen Betriebssystemen lauffähig ist, sondern die Mobilität muss in das Konzept des Diagnosesystems integriert werden. Daran scheitern einige Angebote auf dem Markt und erfüllen deshalb nicht die Ansprüche, die mit dieser Dissertation gestellt werden (siehe Kapitel 8.3).

Deshalb wurde bei dem neu entwickelten Diagnosesystem, die Mobilität und der grafisch hohe Anspruche von Grund auf konzeptionell verankert. Gewährleistet wird diese Verankerung durch die Aufnahme folgender Punkte in den Anforderungskatalog (Kapitel 7.3):

- Betriebssystemübergreifende Lauffähigkeit der HMI-Software (technisch)
- Ausgabe in hoher Bildqualität (technisch)
- Freie Skalierbarkeit der Oberfläche (funktional)
- Diagnoseoberfläche unterstützt Maskensystem (funktional)
- Umfassende grafische Gestaltungsfunktionalität (funktional)

Die Verbesserungen sollen sich in der Nacharbeit durch die Verkürzung der Diagnosezeiten und eine gesteigerte Anwenderzufriedenheit zeigen. Die Umsetzung dieser Ziele anhand des Prototyps wird im Folgenden evaluiert:

- **Verbesserung der Prozesszeit**

  In der Nacharbeit stehen den Mitarbeitern Diagnoseterminals zur Verfügung, um das derzeitige Diagnosesystem zu nützen. Diese sind aufgrund ihrer Größe sperrig und kaum zu bewegen. Bei der Soll-Modellierung wurde erörtert, dass das Ist-System zwar auf Notebooks lauffähig wäre, diese aber aufgrund ihrer Größe als immer noch zu sperrig und hinderlich angesehen werden (siehe Kapitel 7.2.2). Der Platzmangel bei der Diagnosearbeit ergibt sich, da die Steuergeräte in Folge ihrer steigenden Anzahl im Fahrzeug an Stellen verbaut werden müssen, die für den Diagnostiker nur schwer zugänglich sind.

  Aufgrund der Immobilität des Ist-Systems sind die Diagnostiker gezwungen, die Arbeitsposition (z. B. in der Fahrerkabine oder unter dem Fahrzeug) zu verlassen, um das Diagnosesystem zu bedienen oder um Werte abzulesen.

Deshalb setzt der Anforderungskatalog die betriebssystemübergreifende Lauffähigkeit der HMI-Software als technischen Anspruch voraus. Der Viewer muss sowohl auf Terminals als auch auf mobilen Kleingeräten lauffähig sein. Bei der Integration des Prototyps wurde für den Pilotbetrieb ein handflächengroßer PDA ausgewählt (siehe Kapitel 11.3.2).

In der Analyse wurden knapp 1000 mechatronische Nacharbeitsfälle pro Monat in einem Werk erfasst, bei denen die HMI-Software des Ist-Systems zur Bearbeitung zum Einsatz kommt (siehe Kapitel 6.5.2). 40 % davon wurden entsprechend der Verbauung der Steuergeräte als Fälle ausgemacht, wo sich der Diagnostiker im Innern des Fahrzeugs befinden muss. Für diese Fälle hat die Soll-Modellierung durch den Einsatz eines mobilen Viewers eine Verkürzung der Prozesszeit von ca. 17 % ergeben.

Wie bei der Evaluation in der Verwaltung werden auch hier in gleicher Weise kontrollierte Experimente durchgeführt. Sie sollen als Anwendungsfälle den praktischen Nachweis für die Prozesszeitverkürzung bringen.

Für die Beurteilung stellt das Ablaufprotokoll die Zeiten des Ist-Systems denen des Neu-Systems gegenüber. Die Beispiele stellen konstruierte Abbilder realer Fehlerfälle dar. Die Ablaufprotokolle sind in der Quelle MobiViSt-Prozessdatenerfassung 2008 erfasst.

Repräsentativ soll die Beurteilung anhand eines bekannten Anwendungsfalls erläutert werden. Der Nacharbeitsfall basiert auf einer defekten Rücksitzheizung. Mit dem Steuergerät im Fahrzeuginnenraum vertritt er ca. 40 % aller mechatronischen Nacharbeitsfälle, bei denen das Diagnosesystem zum Einsatz kommt (siehe Analysedaten, Kapitel 6.6 und 7.2). Im Anhang sind die einzelnen Schritte und Rahmenbedingungen des Anwendungsfalls näher erläutert.

Nr.	Tätigkeit	Dauer-Ist <Sek>	Dauer-Neu <Sek>
1	Betreten des Vordersitzbereichs und Aktivieren der Heizung	40	40
2	Betreten des Hintersitzbereichs -> Rechte Sitzheizung wärmt, linke nicht	35	35
3	Hintersitzbereich verlassen	15	-
4	Über Diagnosemasken für Heiz-Klima-Automatik wird die Heizung des linken Rücksitzes manuell aktiviert	45	55
5	Hintersitzbereich betreten	15	-
6	-> Linke Sitzheizung wärmt nicht	5	5
7	Hintersitzbereich verlassen	15	-
8	Maske am Diagnosesystem gewechselt, um Spannung und Druckwerte zu überprüfen -> Werte von Temperatursensor i. O. -> Werte von Aktor n. i. O. (Aktor ist eine Art Ventilator, der Luft verteilt) => Aktor oder Kontakt zu Aktor nicht funktional	100	80
9	Hintersitzbereich betreten	15	-
10	Rücksitzverkleidung wird entfernt und Steuergerät sowie	170	170

	dessen Aktoren im linken Rücksitzbereich freigelegt.		
11	Steckerverbindung zu Aktor wird überprüft (Kabel wird gelöst und erneut eingerastet)	25	25
12	Hintersitzbereich verlassen	25	-
13	Über Diagnosemasken für Heiz-Klima-Automatik wird die Heizung des linken Rücksitzes manuell aktiviert	40	30
14	Hintersitzbereich betreten	25	-
15	-> Linke Sitzheizung wärmt nicht	5	5
16	Kabel wird ausgetauscht	60	60
17	Hintersitzbereich verlassen	25	-
18	Über Diagnosemasken für Heiz-Klima-Automatik wird die Heizung des linken Rücksitzes manuell aktiviert	25	25
19	Hintersitzbereich betreten	25	-
20	-> Linke Sitzheizung wärmt (i. O.)	5	5
21	Verkleidung wieder montiert, Hintersitzbereich verlassen Diagnosetätigkeit und Reparatur beendet	200	200
	Gesamtzeit Sek. (MM:SS)	**915 (15:15)**	**735 (12:15)**

Tabelle 11.4 Systemvergleich bei Anwendungsfall: Rücksitzheizung defekt

Wie bei dem Anwendungsfall, der im Rahmen der Integration vorgestellt wurde (siehe Kapitel 11.3.3), zeigt sich bei Betrachtung der Tabelle das Entfallen der Wege zum Diagnosegerät ("-"-Zeichen). Das Bestreiten des Weges zum Terminal und von diesem wieder zurück sind einzeln nur kurze Zeitspannen. Sie liegen über alle Anwendungsfälle hinweg zwischen 5 und 25 Sekunden. Aber durch die Häufigkeit, mit der sie auftreten, machen sie je nach Komplexität des Anwendungsfalls wiederum einen deutlichen Anteil der Gesamtarbeitszeit aus.

Durch ein mobiles System erübrigt sich der Weg zum Terminal. Das Diagnosegerät kann zu der Arbeitsstelle am Steuergerät vor Ort mitgenommen werden. In dem vorliegenden Fall schafft das mobile System durch den Wegfall der zusätzlichen Wege eine Reduktion von 15 Minuten 15 Sekunden auf 12 Minuten 15 Sekunden. Aufgrund der komplexen Verbauung der Rücksitzheizung in den Polstern und der Vorsicht die bezüglich des Interieurs geboten ist fällt die zeitliche Reduktion von 19,7 % bei solchen Fällen sehr hoch aus.

Auch für den Geschäftsprozess der Nacharbeit soll versucht werden, durch die Abarbeitung mehrerer Anwendungsfälle die Aussagekraft des Gesamturteils zu steigern.

```
Bereich: Kooperationspartner, Kfz-produzierendes Werk (Nacharbeit)

Anwendungsfall: Nacharbeitsfall "Rechter Blinker n. i. O." in Werk Dingolfing,
TD-47 (Nacharbeit)
Dauer Ist-System (Sek.)    = 1010
Neu Ist-System (Sek.)      = 880
Verkürzung um (%)          = 12,9

Anwendungsfall: Nacharbeitsfall "Linke Rücksitzheizung n. i. O."
```

```
Dauer Ist-System (Sek.)    = 915
Neu Ist-System (Sek.)      = 735
Verkürzung um (%)          = 19,7

Anwendungsfall: Nacharbeitsfall "Temperatur in der Fahrerkabine lässt sich
nicht erhöhen"
Dauer Ist-System (Sek.)    = 1470
Neu Ist-System (Sek.)      = 1230
Verkürzung um (%)          = 16,3

Anwendungsfall: Nacharbeitsfall "Defekte Schließanlage Kofferraum"
Dauer Ist-System (Sek.)    = 1825
Neu Ist-System (Sek.)      = 1505
Verkürzung um (%)          = 17,5

Durchschnittliche Gesamtreduktion auf alle Anwendungsfälle:
Verkürzung um (%)          = 16,6

Anwendungsfall*: Nacharbeitsfall "Update Flashspeicher des Steuergeräts und
Löschen des Fehlerprotokolls"
Dauer Ist-System (Sek.)    = 680
Neu Ist-System (Sek.)      = 720
Verkürzung um (%)          = -5,9 (Verlängerung)

Anwendungsfall*: Nacharbeitsfall "Anpassung der Adaptionswerte"
Dauer Ist-System (Sek.)    = 1905
Neu Ist-System (Sek.)      = 1850
Verkürzung um (%)          = 2,9

(Quelle: MobiViSt-Prozessdatenerfassung 2008)
```

Abbildung 11.13 Auswertung: Durchschnittliche Verbesserungsquoten in der Nacharbeit

Die ermittelte Verbesserungsrate bezieht sich auf den Teil der Nacharbeitsfälle, bei denen sich das Steuergerät an einer schwer zugänglichen Stelle befindet, so z. B. im Fahrzeuginnenraum, im Bodenraum, auf der Fahrzeugunterseite, im Kofferraum oder in der Türenverkleidung. Dies sind Stellen, wo der Diagnostiker nicht gleichzeitig das immobile Diagnoseterminal bedienen und am Steuergerät arbeiten kann.

Die mit (*) markierten Fälle zeigen, dass bei leicht zugänglichen Elektronikteilen bzw. einer Nacharbeitstätigkeit, die keinen physischen Zugriff auf Elektronikteile erfordert, der Vorteil durch Mobilität verschwindend gering ist. Es kann aufgrund der schlechteren Hardwareleistung des mobilen Endgeräts sogar zu einer Verlängerung kommen.

Hinweis: Ein ausschließlich am Terminal bearbeitbarer Nacharbeitsfall wie "Update Flashspeicher des Steuergeräts und Löschen des Fehlerprotokolls" würde normalerweise mit dem Desktop Viewer und nicht mit dem Mobile Viewer bearbeitet werden. Er wurde hier rein zu Demonstrationszwecken mit dem mobilen Viewer bearbeitet. Die Notwendigkeit eines Viewers für Desktop-PCs und die sinnvollen Einsatzbereiche des mobilen Viewers sollten deutlich gemacht werden.

Für den Großteil der Fälle (über 60 % aller mechatronischen Nacharbeitsfälle mit Beteiligung des Diagnosesystems) gilt allerdings, dass das Steuergerät schwer zugänglich ist. Bei diesen Fällen wurde im Pilotbetrieb eine durchschnittliche Verkürzung der Bearbeitungsdauer um ca. 17 % ermittelt.

- **Erhöhung der Anwenderzufriedenheit**

  Es wurde gezeigt, dass mit der Erweiterung des mobilen Faktors die Dauer der Diagnosetätigkeit verkürzt werden kann. Für den Diagnostiker ergibt sich durch die Mobilität noch ein weiterer Vorteil, der sich allerdings weniger durch objektive Daten messen lässt. Mit dem Entfallen der Wege zum Diagnoseterminal entfallen für den Diagnostiker auch die permanenten Unterbrechungen der Arbeit am Steuergerät. Der Diagnostiker kann konzentriert am Steuergerät weiterarbeiten, während er über das Handgerät Werte abliest oder Steuergerätebefehle abschickt. Die Arbeit wird dadurch zielstrebiger zu Ende gebracht und für den Diagnostiker angenehmer.

  Neben der fehlenden Mobilität haben auch die Mängel in der grafischen Verarbeitung beim Nutzer Unzufriedenheit bezüglich des Ist-Systems ausgelöst. Es wurde die kaum realitätsnahe Gestaltung der Diagnoseoberfläche bemängelt. Die Kritik umfasst die geringe Farbtiefe, die Zeichenelemente und die optisch sichtbare Aktualisierung. In Kapitel 7.2.3.2 wurden die daraus entstehenden Beeinträchtigungen für die Arbeit mit dem Diagnosesystem ausführlich diskutiert.

  Es entsteht z. B. durch die Aktualisierung der Diagnosemaske im Ist-System ein permanentes Flackern der Ansicht. Da die zugrunde liegende Technik es nicht möglich macht, einzelne Wert zu aktualisieren. Deshalb muss immer die komplette Maske neu aufgebaut werden.

  Des Weiteren können aufgrund mangelnder grafischer Möglichkeiten die Masken nicht entsprechend der Wirklichkeit, wie z. B. den Bedienelementen im Fahrzeug, abgebildet werden. In Kapitel 11.3.3 wird hierfür ein Beispiel aus dem Heiz- und Klimabereich geliefert. Durch diese realitätsfremde Darstellung hat es der Benutzer oft schwer, sich in die Diagnosemaske einzuarbeiten und den für den Nacharbeitsfall relevanten Wert zu finden.

  Um diesen Misstand zu beseitigen, bietet das neue System eine hohe Bildqualität sowie eine umfassende grafische Gestaltungsfunktionalität. Basierend auf SVG können mit dem Prototypen Diagnosebilder in realitätsgetreuer Farbtiefe (True Color) geschaffen werden. Diese können technische Zeichnungen oder funktionale Hintergrundbilder enthalten. Die Integration des Prototyps in den Pilotbetrieb zeigt hier Fotografien von Fahrzeugteilen, die funktional genutzt werden können. Es ist mit dem Prototyp folglich möglich, reale Bilder von Scheinwerfern oder Bedienelementen abzubilden und zu dynamisieren. In den Kapiteln 11.2 und 11.3.4 werden hierfür einige Beispiele angeführt.

  Mit der realitätsgetreuen Abbildung der Diagnosemasken, die auf mobilen und Desktop-Geräten in gleicher Qualität betrachtet werden können, bietet der Prototyp ein Konzept, das aus Sicht der Diagnostiker den Bedienerkomfort im grafischen Bereich deutlich erhöht.

11.4.4 Ergebnis

Die Bewertungsergebnisse beziehen sich auf den Pilotbetrieb in der ersten Prototypphase. Ausgewertet wurden die Geschäftsprozesse Verwaltung und Nacharbeit. Die Ergebnisse wurden durch eine analytische und experimentelle Evaluation gewonnen und lassen sich wie folgt zusammenfassen:

- **Erfüllung aller Punkte des Anforderungskatalogs**
- **Auflösung aller Systembrüche** im Teilprozess "Diagnoseskript erstellen/nachbearbeiten" der Verwaltung
- **Reduktion der durchschnittlichen Bearbeitungsdauer** von repräsentativen Anwendungsfällen in der **Verwaltung** um 33 % (Teilprozesses "Diagnoseskript erstellen/nachbearbeiten")
- **Reduktion der durchschnittlichen Bearbeitungsdauer** von mechatronischen Anwendungsfällen in der **Nacharbeit** um 17 % (Teilprozess "Steuergerät diagnostizieren")
- **Steigerung der Anwenderzufriedenhei**t bezüglich der Arbeit mit dem Diagnosesystem in der Verwaltung und Nacharbeit

Der erste Punkt der Liste stützt sich auf das Ergebnis der analytischen Evaluation. Die analytische Evaluation hat anhand quantitativer Daten (vornehmlich Systemeigenschaften) gezeigt, dass alle mit dem Anforderungskatalog aus Kapitel 7.3 gestellten Forderungen durch das neu entwickelte System erfüllt werden.

Die Evaluationsergebnisse 2 bis 5 wurden mittels experimenteller Evaluation gewonnen. Hierbei wurden neben objektiven Daten, wie Prozesszeiten, auch subjektive Daten, wie Anwenderurteile berücksichtigt.

Mit der Schaffung einer ganzheitlichen Entwicklungsumgebung, die unter anderem den Import von Steuergerätezugriffsdaten ermöglicht, konnten zentrale Systembrüche aufgelöst werden. Die Auflösung steigert den Arbeitsfluss, reduziert die Durchlaufzeiten und erleichtert dem Benutzer das Arbeiten.

Die Entwicklungsumgebung als grafischen Editor zu gestalten, hat darüber hinaus entscheidend dazu beigetragen, die Prozesszeit zu reduzieren. Bei einem Aufwand von ca. 114 Stunden je Mitarbeiter pro Monat zur Neuerstellung und Bearbeitung von Diagnoseskripten (siehe Tabelle 6.2) könnte bei einer 33-%-Reduktion der Mitarbeiter das gleiche Auftragsvolumen mit ca. 4 Manntagen weniger bearbeiten.

Die Verwendung von offiziellen Standards bei den Plattformen des Diagnosesystems und den Skriptformaten erleichtert den Mitarbeitern zudem die Einarbeitung in das System. Es muss keine weitere proprietäre Programmiersprache erlernt werden. Auch ist die Wahrscheinlichkeit hoch, dass das verwendete Format den Programmierern schon bekannt ist. Dies steigert die Akzeptanz und Anwenderzufriedenheit.

In der Nacharbeit ist es vorrangig der gesteigerte Mobilitätsfaktor, der den Prozess der Steuergerätediagnose verkürzt. Die Verkürzung von 17 % begünstigt allerdings nur mechatronische Nacharbeitsfälle mit schwer zugänglichen Steuergeräten. Mechatronische Nacharbeitsfälle, bei denen das Steuergerät leicht erreicht werden kann, profitieren von der Verbesserung nur in geringem Maße.

Ergänzend zu der Mobilität steigert die hochwertige Darstellungsmöglichkeit den Benutzerkomfort bei der Diagnose. Verbesserungen wie qualitätsverlustfreie Skalierung, grafische Gestaltung in Echtfarben und Integration technischer Abbildungen machen die Arbeit mit dem neuen Diagnosesystem für den Diagnostiker angenehmer und fördern die Anwenderzufriedenheit.

Auch wenn die Datenerfassung in der ersten Prototypphase durch den Pilotbetrieb eingeschränkt ist, sind die Verbesserungen gegenüber dem Ist-System deutlich zu erkennen. Die Verbesserungen zeigen, dass die erarbeiteten Punkte des Anforderungskatalogs in die richtige Richtung führen, um die Gesamtqualität der Diagnose elektronischer Steuergeräte zu verbessern.

Die Zielsetzung der Evaluation ist es eine Stopp/Go-Entscheidung zu treffen. Aufgrund signifikanter Verbesserungen durch den ersten Prototyp wird der nächste Schritt der inkrementellen Softwareentwicklung eingeleitet. Mit der zweiten Prototypphase sollen Benutzerkritiken aus der ersten Prototypphase einfließen, um das System noch weiter an die Anforderungen der heutigen Diagnose anzupassen. Auf erste Ideen der Mitarbeiter und künftige technologische Möglichkeiten zur Verbesserung des Diagnosesystems wird im Rahmen des Ausblicks in Kapitel 12.2 eingegangen.

Mit dem erfolgreichen Abschluss der ersten Prototypphase ist auch die Zielsetzung dieser Arbeit erfüllt. Über den Weg der Erfassung, Analyse und Optimierung der Diagnoseprozesse ist es gelungen, mit den Methoden der Wirtschaftsinformatik einen softwaretechnischen Beitrag zur Verbesserung der Diagnosetätigkeit zu leisten. Durch die Umsetzung, den Einsatz und die Bewertung der Software ist auch der gewollte praktische Bezug der Arbeit erfüllt.

12 Zusammenfassung und Ausblick

12.1 Zusammenfassung

Die vorliegende Arbeit behandelt den Gegenstand der Diagnose von elektronischen Steuergeräten in der Automobilindustrie. Mit zunehmender Zahl der Steuergeräte im Fahrzeug steigt deren Aufgabenumfang und Verantwortung für die Funktionstüchtigkeit des Fahrzeugs. Als Folge dieser Entwicklung stehen die Technologien rund um den Steuergerätebereich neuen Herausforderungen gegenüber – so auch die der Fehlerdiagnose. Das Diagnosesystem muss die Komplexität und die Funktionsvielfalt des Steuergeräts beherrschen. Nur dann wird dem Diagnostiker ein adäquates Mittel zur Fehlersuche an die Hand gegeben.

Der erhöhte Anteil der Softwarelogik in einem Steuergerät hat zur Folge, dass das Diagnosesystem eine entsprechende Flexibilität aufweisen muss. Für das Diagnosesystem ist es nicht mehr ausreichend, aus einem rein mechanischen Werkzeug zu bestehen. Es muss variabel Informationen von Steuergeräten abfragen, diese verarbeiten und sie an die Steuergeräte zurückschicken können. Eine solche Lösung ist nur Software-basiert möglich. Durch diesen wachsenden Anspruch verstärkt die Informatik stetig ihre Position in der Diagnose.

Im gleichen Zuge bekräftigt die Diagnose auch ihre ökonomische Relevanz. Imageverluste und hohe Kosten durch Rückholaktionen bedingen die Verbreitung der Steuergerätediagnose in immer mehr Geschäftsprozessen der Automobilkonzerne. Damit wollen die Konzerne die Qualität steigern und den Umfang an Diagnosefällen in den Begriff bekommen. Die Steuergerätediagnose findet sich heute unter anderem im Service, in der Produktion und in der Verwaltung wieder.

Die Diagnose tangiert folglich den Themenbereich der Informatik und der Betriebswirtschaft. Aufgrund dieser Verschmelzung der Softwaretechnik mit betriebswirtschaftlicher Bedeutung, scheint es angemessen, sich der Diagnosethematik mit der Wissenschaft der Wirtschaftsinformatik anzunehmen.

Die Arbeit vereinigt viele Kerngehalte der Wirtschaftsinformatik. Es werden unter anderem die Prozessmodellierung, Prozessanalyse und Prozessoptimierung IT-basierter Prozesse durchge-

führt sowie die Analyse, Architektur, Implementierung und Evaluation von Software. Diese Techniken unterstützen dabei, eine innovative Diagnosesoftware für elektronische Steuergeräte in der Automobilindustrie zu schaffen.

Um bei der Arbeit neben dem wissenschaftlichen Aspekt auch den nötigen praktischen Bezug sicherzustellen, wurde die Arbeit in Kooperation mit einem Automobilkonzern durchgeführt.

Für den Einstieg in die Arbeit wurde in Zusammenarbeit mit dem Kooperationspartner der Ansatzpunkt für ein verbessertes Diagnosesystem bestimmt. In ersten Gesprächen wurden mit Fachkräften neue Herausforderungen erfasst, die der Diagnosemarkt an ein zeitgenössisches Diagnosesystem stellt und die sich auch im Konzern widerspiegeln. Aus den drei Herausforderungsbereichen Funktionalität, Grafik und Mobilität wurde der Ansatzpunkt für die Arbeit bestimmt. Alle drei Bereiche fokussieren den Abschnitt HMI-Software des Diagnosesystems. Dieser wurde somit Kernthema der Arbeit. Die HMI-Software stellt den Oberflächenteil des Diagnosesystems dar und dient folglich dem Diagnostiker zur Interaktion mit dem Steuergerät.

Um fundiert das Verbesserungspotenzial im Bereich der HMI-Software zu erschließen, wurden beim Kooperationspartner alle Geschäftsprozesse erfasst, bei denen die Software zum Einsatz kommt. Die Modellierung umfasst die Nacharbeit, Steuergeräteanalyse, Steuergeräteentwicklung, Verwaltung und den Service. Die Prozesse wurden Top-down von den Kernprozessen des Unternehmens bis hin zu den einzelnen Arbeitsschritten modelliert. Dabei wurden relevante Prozessdaten, wie Durchlaufzeiten und Stückzahlen, erfasst.

Die Analyse konzentriert sich auf die Geschäftsprozesse Nacharbeit und Verwaltung. In diesen Prozessen ist die Nutzung des Diagnosesystems am stärksten frequentiert. Sie konnten deshalb repräsentativ für die übrigen Geschäftsprozesse ausgewählt werden. Die Analyse hat zahlreiche Schwachstellen in Form von Systembrüchen, unverhältnismäßig langen Prozesszeiten (Flaschenhälse in der Verarbeitung) und Benutzerunzufriedenheiten bei dem derzeitigen Diagnosesystem ergeben.

Im nächsten Schritt der Arbeit wurden die erfassten Schwachstellen mit den Methoden der Prozessoptimierung untersucht. Es wurden entsprechende Verbesserungsansätze erarbeitet. Techniken wie Soll-Modellierung, Simulationen und Benutzerbefragungen bilden die Basis für die Optimierungsarbeit. Das Ergebnis der Optimierung war ein Anforderungskatalog. Ein adäquates Diagnosesystem muss folglich den Ansprüchen dieses Anforderungskatalogs gerecht werden.

Nachdem sich nach einer Marktanalyse auf dem Softwaremarkt und in Forschungsprojekten kein entsprechendes Diagnosesystem fand, ist der Entschluss für eine Neuentwicklung gefallen.

Für die Neuentwicklung galt der Anforderungskatalog als Pflichtenheft. Die Anforderungen sollten in ein neues Diagnosekonzept überführt werden, das somit allen Ansprüchen gerecht wird. Der Entwurf enthält das Softwarekonzept, die Architektur, die softwaretechnischen Hintergründe und eine umfassende Beschreibung der Softwareschichten. Der Umfang des Entwurfs begründet sich mit dem Ziel der geplanten vollständigen Implementierung der Software.

Das Konzept beruht auf einer Kombination aus HMI-Software und zugehörigeren Entwicklungsumgebung, wobei mit der HMI-Software Diagnosemasken geladen werden, die mit der Entwicklungsumgebung erstellt werden. Die Diagnosemasken enthalten die eigentliche Diagnosefunktionalität. So bleibt die HMI-Software für beliebige Steuergeräte anwendbar.

Kern der Idee ist zum einen eine ganzheitliche Entwicklungsumgebung und zum anderen eine grafisch hochwertige mobile HMI-Software. Die Entwicklungsumgebung bietet die Möglichkeit, die Diagnosemasken grafisch zu gestalten und die Steuergerätefunktionalität unmittelbar zuzuweisen. Auf Basis des WYSIWYG-Konzepts können Änderungen direkt mitverfolgt werden und erfordern keine Kompilierung der erstellten Diagnoseskripte. Die HMI-Software bietet die Möglichkeit, das Diagnoseskript sowohl auf einem mobilen Endgerät, wie einem Handheld, als auch auf einem Desktop-PC abzuspielen. Durch qualitätsverlustfreie Skalierung der Diagnosemasken können die Masken auf beliebig großen Bildschirmen dargestellt werden. Auch auf kleinsten Monitoren bleiben die Masken anwendbar. Ist eine Maske für eine Gesamtdarstellung auf einem mobilen Monitor zu groß und sind die einzelnen Werte in der Gesamtdarstellung demzufolge unleserlich, können einzelne Bildschirmausschnitte durch Vergrößerung in den Vordergrund geholt werden. Da die Vergrößerung der Bildschirmausschnitte ohne Verlust der Bildqualität durchgeführt wird, bleiben die Masken auch auf kleinsten Monitoren noch lesbar. Die Farbdarstellung in Echtfarbe und die skalierbare Vektorgrafik schaffen ein grafisch hochwertiges Ergebnis.

Nach der Konzepterstellung erfolgt dessen softwaretechnische Umsetzung. Die Beschreibung der Umsetzung behandelt eine Erläuterung der Schwerpunkte der Programmierung. Es werden z. B. die grafische Unterstützung der Vektorgrafik und der Steuergerätezugriff dargelegt. Das Ergebnis der Umsetzung war ein erster Prototyp.

Im Rahmen der abschließenden Projektierung sollte der Prototyp bewertet werden. Der entstandene Prototyp wurde dafür in Form eines abgegrenzten Pilotbetriebs beim Kooperationspartner integriert. Die Integration erfolgte in die repräsentativen Geschäftsprozesse Verwaltung und Nacharbeit. Im Verlauf des Pilotbetriebs konnten Prozessdaten erfasst und Anwenderbefragungen durchgeführt werden. Sie dienen als Basis für die abschließende Evaluation. Die Evaluation gliedert sich in eine analytische und experimentelle Evaluation auf. Mit der analytischen Evaluation sollte der Erfüllungsgrad des Anforderungskatalogs gemessen werden. Die experimentelle Evaluation hatte das Ziel, die Verbesserung der Geschäftsprozessabläufe zu untersuchen. Anhand der Auflösung von Systembrüchen, der Verbesserung von Prozesszeiten und der Steigerung der Anwenderzufriedenheit wurde eine signifikante Verbesserung durch den Prototyp nachgewiesen. Auch die Evaluation des Anforderungskatalogs hat ergeben, dass alle geforderten Ansprüche durch den Prototyp abgedeckt sind.

Basierend auf dem Ergebnis der Evaluation wird auch die Gesamtzielsetzung der Arbeit als erfüllt betrachtet. Es konnte mit der neuen Software ein positiver Beitrag zur Verbesserung der Diagnosetätigkeit geleistet werden. Für einen produktiven Einsatz des Diagnosesystems sind allerdings noch weitere Entwicklungs- und Verwaltungsschritte notwendig. Da dessen Verwirklichung nicht mehr Zielsetzung der Arbeit ist, werden sie in Form eines Ausblicks im abschließenden Kapitel 12.2 vorgestellt.

12.2 Ausblick

"Automobilhersteller, Systemlieferanten und Komponentenzulieferer, alle waren sich einig: Die Elektronik wird im Auto der Zukunft weiter an Bedeutung gewinnen", so schreibt ein Autor der Global Electronics über das 5. Kompetenztreffen Automobilelektronik[298]. Das am 4. Dezember 2007 in München abgehaltene Kompetenztreffen ist nicht das einzige seiner Art, das sich aus aktuellem Anlass immer größerer Teilnehmerzahlen erfreut. So fand vom 10. bis 11. November 2007 im internationalen Kongresszentrum München (ICM) die 3. Electronica Automotive Conference statt. Auch sie beschäftigt sich mit der Zukunft der Fahrzeugelektronik und präsentiert dazu Beiträge internationaler Unternehmen aus der Automobil-, Zuliefer- und Elektronikindustrie. Nicht zuletzt in dieser Reihe ist auch die 12. Jahrestagung der Elektronik-Systeme im Automobil vom 5. bis 8. Februar 2008 zu nennen, bei der leitende Firmenvertreter z. B. von Audi, BMW, Daimler und Bosch die Elektronikentwicklungen im Automobil diskutieren.

Die Quintessenz aller aktuellen Messen ist, dass die Verfolgung der Ziele wie Steigerung der Sicherheit, des Komforts oder der Umweltverträglichkeit einhergeht mit der zunehmenden Anzahl und Funktionalität der Elektronikkomponenten. Wie in Kapitel 1.1 gezeigt wurde, wirkt sich diese Tendenz in gleichem Maße auf die Komplexität der betroffenen Komponenten und der damit in Verbindung stehenden Technologien aus. Eine dieser Technologien ist die Steuergerätediagnose.

Ihrer zukünftigen Entwicklung wurde auf der Jahrestagung "Elektronik-Systeme im Automobil" im Februar ein Fachtag gewidmet. Beiträge wie "Migration auf neue Diagnose-Konzepte" oder "Konzepte für die zukünftige Fahrzeugdiagnose" zeigen, dass man in der Diagnose nach wie vor nach neuen Wegen und Möglichkeiten sucht, um die Komplexität und Vielfalt der Steuergeräte zu bewältigen.

Im Folgenden soll anhand von aktuellen Vorträgen, Forschungsarbeiten und Softwareansätzen verdeutlicht werden, dass das Konzept dieser Arbeit, die vor drei Jahren begonnen wurde, im Spiegel künftiger Entwicklungsvorhaben steht.

Ein Kernaspekt der Arbeit ist, sich der Diagnoseproblematik über den Ansatz der Prozessoptimierung zu nähern. Da sich die Diagnosetätigkeit in diversen Geschäftsprozessbereichen etabliert hat, wurde die Erfassung aller betroffenen Prozesse als wichtiger Meilenstein der Arbeit festgelegt. Auf der Basis der relevanten Prozesse wurde ein ganzheitlicher Anforderungskatalog für eine neue Diagnosesoftware definiert. Die Stellung der Diagnose innerhalb der Geschäftsprozesse und deren Bedeutung als Ansatzpunkt für Verbesserungen werden durch renommierte Vertreter der Automobil-Zulieferbranche deutlich hervorgehoben. So war z. B. Inhalt eines Vortrags im Rahmen der 12. Jahrestagung "Elektronik-Systeme im Automobil" die "Diagnose als Bindeglied von Geschäftsprozessen"[299]. Referent war der Vorstand der Softing AG, Michael Siedentop. Die Softing AG ist Hersteller von Diagnosesoftware und hat z. B. das Diagnose-Basissystem für die BMW AG entwickelt. In seinem Vortrag "Diagnose elektronischer Steuergeräte in der Prozesskette" macht Siedentop deutlich, dass die Diagnose in immer

[298] Global Electronics 2007 (Online-Quelle)
[299] Siedentop 2008, Tagungs-Infomappe

mehr Geschäftsprozessen Einzug hält und auch Prozessschnittstelle zwischen ihnen ist. In dieser Arbeit wurde z. B. die Ausschleusung von Fahrzeugen aus dem Nacharbeitsprozess in den Steuergeräteanalyseprozess beschrieben. In beiden Geschäftsprozessen wird allerdings die Diagnose mit der gleichen HMI-Software durchgeführt. Die Ansprüche der Diagnostiker an die Diagnoseoberfläche sind dadurch unterschiedlich. Durch den Schritt der ganzheitlichen Prozesserfassung konnten die Ansprüche beider Einsatzbereiche berücksichtigt werden. Diese Art der Vorgehensweise postuliert auch Siedentop in seinem Vortrag.

Ein weiterer Bereich, in dem sich die Arbeit konzeptionell als zukunftsfähig erweist, ist der Bereich der mobilen Einsatzfähigkeit und Flexibilität. Hersteller, wie z. B. Bosch, preisen mit der neuesten Version ihres Produkts ESI[tronic] die "Mobilität und Bewegungsfreiheit in der Werkstatt" als zentrale Produkteigenschaft an[300]. Auch ein aktuell veröffentlichtes Softwareprodukt der Siemens VDO wird auf einem mobilen Diagnosegerät, einem Tablet PC, präsentiert. Weiter wird die Fähigkeit hervorgehoben, mit über 15.000 verschiedenen Steuergerätetypen kommunizieren zu können[301]. Beide Präsentationen der Produkte versuchen deutlich die grafische Gestaltung der Oberfläche hervorzuheben.

An den jüngsten Produkten ist deutlich zu erkennen, dass versucht wird, die Aspekte Mobilität, grafische Gestaltungsweise und flexible Funktionalität herauszustellen. Diese Themen waren es auch, die für die vorliegende Arbeit als zu bewältigende Herausforderungen angesetzt wurden. Allerdings gibt es im Rahmen von Forschungsprojekten durchaus Ansätze, die über den Aspekt der Mobilität dieser Arbeit hinausgehen und eine ubiquitäres Diagnosesystem ansteuern. Das EU-Projekt "MYCAREVENT" hat eine drahtlose ortsunabhängige Fahrzeugdiagnose zum Inhalt. Der Online-Service soll Diagnoseprozesse verkürzen und damit zu einer schnelleren Fehlerbehebung beitragen. Das Konzept der Online-Ferndiagnose wird von der Automobilindustrie derzeit allerdings als langfristiger Lösungsansatz angesehen[302].

Da Kosten von Rückholaktionen und Imageverlust gegenwärtige Probleme darstellen, ist die Automobilindustrie bemüht, diesen mit zeitnahen Lösungen entgegenzutreten. Aktuelle Vorträge (siehe oben) zeigen, dass sich der Ansatz und die Herangehensweise der Arbeit zeitgemäß behaupten können. Abschließend wird erörtert, dass das Ergebnis der Arbeit die Basis für eine zeitnahe Lösung bietet und in diesem Sinne eine Weiterführung geplant ist.

Ziel der ersten Softwareentwicklungsphase dieser Arbeit war es, zu zeigen, dass durch den Einsatz des Prototyps die erfassten Prozessabläufe verbessert werden. Als Ergebnis der Leistungsbewertung in Kapitel 11.4 wurde hinreichend gezeigt, dass dieses Ziel erreicht wurde – weshalb der Kooperationspartner auch die zweite Phase der Prototypentwicklung eingeleitet hat.

Mit der zweiten Phase sollen in Gesprächen mit den Erstnutzern des Prototyps unter Berücksichtigung von Markttendenzen weitere Verbesserungspotenziale erschlossen werden.

Ein Verbesserungsvorschlag ist z. B. die Abkehr von der rein Steuergeräte-orientierten hin zu einer funktionsorientierten Diagnose. Die Funktionen verteilen sich immer stärker auf mehrere Steuergeräte bzw. finden sich je nach Fahrzeugtyp in unterschiedlichen Steuergeräten wieder. Der Vorschlag sieht deshalb vor, hinter einem Diagnoseaufruf mehrere Steuergerätebefehle hinterlegen zu können. Diese werden entsprechend dem Fahrzeugtyp aufgerufen. Die Aufrufe

[300] Bosch ESItronic, S. 3
[301] Vgl. Siemens VDO 2007 (Online-Quelle)
[302] Vgl. MYCAREVENT 2008, (Online-Quelle)

fixieren sich damit nicht mehr nur auf einen Steuergerätetyp, sondern mehr auf Funktionsbereiche. Dadurch steigert sich auch die Wiederverwendbarkeit der Diagnosemasken.

Eine weitere Idee ist die Protokollierung der Diagnosetätigkeiten. Auf diese Weise können automatisch Fehlerbehebungsanleitungen oder Empfehlungen erstellt werden. Sie würden bei erneutem Auftreten des gleichen oder eines ähnlichen Fehlers unterstützen und Bearbeitungszeit sparen.

Auch technisch werden neue Möglichkeiten in Betracht gezogen. So bietet seit 2007 Microsoft ein eigenes Vektorgrafikformat an[303]. Dieses würde den Vorteil mit sich bringen, dass keine kommerziellen Plug-ins zur Unterstützung des Grafikformats mehr eingebunden werden müssen. Außerdem ist zu erwarten, dass sich durch die native Unterstützung auch die Performance verbessern würde.

Für eine Weiterführung der Prototypphasen bis hin zu einem produktivfähigen Software-Release ist allerdings die Übernahme des Konzepts durch einen professionellen Softwareentwickler notwendig. Für die Inbetriebnahme von produktionskritischer Software wie der Diagnosesoftware sichern sich Großkonzerne üblicherweise unter anderem durch Servicevereinbarungen ab. Dieser Service kann nur von einem professionellen Dienstleister erbracht werden. Die Übernahme des Konzepts durch einen Zulieferer des Kooperationspartners befindet sich derzeit in der Verhandlungsphase.

Das Engagement der Mitarbeiter des Kooperationspartners, den Prototyp durch weitere Vorschläge zu verbessern, und die durch die Arbeit aufgezeigten Defizite am derzeitigen System machen den Nachholbedarf im Bereich der Diagnosesoftware deutlich. Diese konstruktive Arbeit mit dem Prototyp sowie aktuelle Vorträge im Bereich der Steuergerätediagnose zeigen, dass mit dem Konzept und Ergebnis dieser Dissertation ein Schritt in die richtige Richtung gemacht wurde, die Diagnosetätigkeit zu verbessern.

[303] Vgl. Marquardt 2007, S. 5

A. Glossar und Abkürzungsverzeichnis

Legende:

Mit einem * markierte Begriffe sind Bezeichnungen aus dem Kontext des Kooperationspartners

Abkürzung / Begriff	Beschreibung
.NET	Von Microsoft entwickelte Softwareplattform (auch .Net)
ABS	Antiblockiersystem
ACC	Active/Adaptive Cruise Control
ActiveX	Ein Softwarekomponenten-Modell von Microsoft für aktive Inhalte
Akteur	Agierende Einheit innerhalb eines Anwendungsfalldiagramms. Zum Beispiel ein Benutzer oder auch ein anderes technisches System wie das Basissystem.
Anwendungsfalldiagramm	Diagramm, das die Zusammenhänge beschreibt, die zwischen einer Menge von Anwendungsfällen und den daran beteiligten Akteuren besteht. Bildet den Kontext und eine Gliederung für die Beschreibung, wie mit einem Geschäftsvorfall umgegangen wird.
ARIS	Von A.W. Scheer entwickelte Architektur integrierter Informationssysteme. Eine ganzheitliche Methode zur umfassenden Beschreibung von Geschäftsprozessen.
ASAM	Association for Standardisation of Automation and Measuring Systems (e. V.)
ASCII	American Standard Code for Information Interchange
ASR	Antriebschlupfregelung
Attribut	Parameter, um auf dem Basissystem Jobs aufzurufen.
Automobildiagnostiker	Mensch, der die Fähigkeit hat, mit Hilfe technischer Mittel gezielt und effizient Störungen am Fahrzeug zu diagnostizieren und zu beheben.
Basissystem	Siehe Diagnosebasissystem
BMP	Windows Bitmap
BPR	Business Process Reengineering
CAN	Controller Area Network, oft auch CAN-Bus
CASCADE*	Finales Prüfsystem der Produktion (Control Application Sequences for Coding and Diagnostics Execution).
CCP	CAN Calibration Protocol, Diagnoseprotokolle
CE	Betriebssysteme von Microsoft für PDA und Embedded Systems
CF	Compact Framework von .NET
CFCOM	Compact Framework COM Control der Firma Odyssey
Change Management	Umsetzung von geschäftlichen Anforderungen in die IT-Service-Landschaft
CLR	Common Language Runtime
COM	Component Object Model, eine proprietäre Softwaretechnologie von Microsoft
Copy & Paste	Kopieren und Einfügen. Abgekürzt C & P. Ein zweistufiges Prinzip der Übertragung von Daten zwischen Softwareanwendungen
CPL	Common Public License
Desktop Viewer	Die Viewer-Softwareomponente des Diagnosesystems MobiViSt für Terminal-PCs bzw. Desktop-PCs
Diagnose	Im weiteren Sinne, die Zuordnung von Zuständen zu einer Kategorie. Dabei steht die Ursachenforschung im Vordergrund. In der Automobiltechnik ist es die Eingrenzung der Fehlerursache unter Zuhilfenahme von technischen Geräten (z. B. per angeschlossenem Computer). Dabei werden über Hardware- und Software-Schnittstellen Informationen aus den Steuergeräten ausgelesen.
Diagnosebasissystem	Eine Art Betriebssystem zur Kommunikation mit Steuergeräten in der Diagnosetechnik.
Diagnosemaske	Inhalt der (grafischen) Oberfläche der HMI-Software. Diagnosemasken sind innerhalb eines Diagnoseskripts gespeichert.
Diagnoseoberfläche	Anzeigefläche, die durch das Starten der HMI-Software erzeugt wird. Rahmen, um darin Diagnosemasken anzuzeigen.

Abkürzung / Begriff	**Beschreibung**
Diagnoseskript	Durch die HMI-Software ladbarer Datenträger. Inhalt (Daten) sind Diagnosemasken und deren diagnosespezifische Informationen.
Diagnoseskriptquelle	Quellcode des Diagnoseskripts. (Von Menschen) lesbarer Code in einer (Pseudo-)Programmiersprache.
Diagnosesoftware	Die Kombination aus HMI-Software und Diagnosebasissystem. Eine vollständig funktionsfähige Software zur Diagnose elektronischer Steuergeräte.
Diagnosesystem	Die Verbindung von Diagnosesoftware und Endgerät. Ein Diagnosegerät, wie z. B. ein Notebook, auf dem die Diagnosesoftware installiert ist. *Hinweis: Da sich die Arbeit ausschließlich mit der Verbesserung der Diagnosesoftware auseinandersetzt, werden die Begriffe Diagnosesystem und Diagnosesoftware sinngleich benutzt.*
Diagnosewerkzeug	Siehe Diagnosesystem
Diagnostiker	Siehe Automobildiagnostiker
DLL	Dynamic Link Library, Dynamische Verbindungsbibliothek
DME	Digitales Motormanagement
Drag & Drop	Ziehen und Fallen lassen. Technik unter Windows. Objekte werden mit gedrückt gehaltener Maustaste an die gewünschte Position gezogen und dort wieder fallen gelassen.
DTD	Document Type Definition
DTS	Diagnose-Toolset
E/E*	Elektrik / Elektronik
Eclipse	Software-Plattform und integrierte Entwicklungsumgebung
ECMAScript	Skriptsprache der European Computer Manufacturers Association
ECU	Electronical Control Unit. Siehe Steuergerät.
EDIABAS Toolset32*	Dient der Kommunikation mit dem Steuergerät. Lädt SGBD. Listet Daten, wie Jobs, Result-Sets und Attribute auf.
EDIABAS*	Elektronik Diagnosebasissystem. Ein Diagnosebasissystem zur Kommunikation mit dem Steuergerät. Wird als Kommunikationsbasis seit 1992/93 schrittweise bei allen auf Diagnose basierenden Prüf- und Codiersystemen und sonstigen Applikationen in der Fertigung eingesetzt.
EDIC	Enhanced Diagnostic Interface Computer (EDIC). Hardware Schnittstelle zu einem Steuergerät. Hersteller Softing AG.
EDICBlue	Basiert auf EDIC. Erlaubt die kabellose Adaption an die Diagnose-Buchse per Bluetooth
eEPK	Siehe eEPK
EPK	Ereignisgesteuerte Prozesskette. Auch als erweiterte EPK (eEPK) bekannt. Modellierungstechnik, um mit ARIS Prozesse darzustellen.
EPL	Eclipse Public License
Erweiterungspunkt	Siehe Extension Point
ESP	Elektronisches Stabilitätsprogramm
EU4	Emissionsklasse 4 (Schadstoffausstoß-Richtlinie)
EULA	End User License Agreement
Extension Point	Von der Eclipse-IDE definierte Punkte, an denen Plug-ins eigenen Code (Funktionalität) in der Entwicklungsumgebung zur Verfügung stellen können und so die Funktionalität der IDE erweitern.
FGNR*	Fahrgestellnummer
FIFO	First In First Out
Firewire	Von Apple entwickelte digitale Schnittstelle
Flexray	Serielles, deterministisches und fehlertolerantes Feldbussystem

Abkürzung / Begriff	Beschreibung
FMEA	Fehler-Möglichkeiten und Einfluss-Analyse
FPY	First Pass Yield
FTP	File Transfer Protocol
Get/Set	Dt. Erhalten / Setzen. Methodensammlung, um Eigenschaften eines programmiertechnischen Objekts abzufragen (Get) und diese zu setzen (Set).
GNU	"G-N-U" steht für "GNU-Nutzer-Unterstützung".
GP	Geschäftsprozess
GPL	GNU General Public License
GPM	Geschäftsprozessmanagement
GPO	Geschäftsprozessoptimierung
GUI	Graphical User Interface
HMI	Human Machine Interface
HMI-Software	Oberflächen- und Logikteil der Diagnosesoftware
HTML	Hypertext Markup Language
HTTP	Hypertext Transfer Protocol
ID	Identifikationsbezeichnung
IDE	Integrierte Entwicklungsumgebung. Abkürzung IDE. Engl. Integrated Development Environment oder Integrated Design Environment. Ein Programm zur Entwicklung von Software. Verfügt in der Regel über folgende Komponenten: Texteditor, Compiler bzw. Interpreter, Linker, Debugger und Quelltextformatierungsfunktion.
IDS Scheer	Integrierte Datenverarbeitungssysteme Scheer
IFH	Interfacehandler
IHKA*	Integrierte Heiz- und Klimaanlage
INDIA	Intelligente Diagnose in der industriellen Anwendung
INPA*	Interpreter für Prüfabläufe. Wird in den Werken u. a. in der Nacharbeit zur Elektronik-Fehlersuche eingesetzt.
INPA-Interpreter*	Integriertes System zum interaktiven Entwickeln von INPA-Skripten und zum Kompilieren der erzeugten INPA-Skriptquellen, sodass diese zur Laufzeit in INPA mit minimalem Zeit- und Speicheraufwand geladen und abgearbeitet werden können.
INPA-Loader*	Dient zum Laden und Abarbeiten von vorkompilierten INPA-Skripten (Diagnoseskripten). In der allgemeinen Diagnosetechnik bezeichnet der Begriff die HMI-Software.
INPA-Skript*	Siehe IPO-Datei
INPA-Skriptquelle*	Siehe SRC-Datei
INPA-Skriptvorlage*	INPA-Skriptquelle mit bereits vordefinierter Grundfunktionalität, wie z. B. Standardmenüs, Menüsteuerung, o. Ä.
IPC	Industrie-PC
IPO-Datei*	Aus der SRC-Datei erstelltes Kompilat. Maschinencode, der mit dem INPA-Loader geladen werden kann. In der allgemeinen Diagnosetechnik bezeichnet der Begriff das Diagnoseskript.
IPS-Datei*	Klartextdatei für den INPA-Interpreter. Zwischenkompilat. Alle Bibliotheksverweise aus der SRC-Datei sind aufgelöst. Für INPA-Loader nicht lesbar.
ISO 9141	Bezeichnet Bussystem von K-Line
ISO 9141-CARB	Bezeichnet Diagnoseprotokoll von K-Line
IT	Informationstechnik
JDK	Java Development Kit
JIT	Just in time

Abkürzung / Begriff	Beschreibung
Job	Dt. Steuergerätebefehl. Mit dessen Aufruf können Daten aus dem Steuergerät ausgelesen werden oder Werte auf dem Steuergerät manipuliert werden.
JPEG	Joint Photographic Experts Group
JVM	Java Virtual Machine
KAIZEN	Japanisch, "Veränderung zum Besseren"
K-Line	Bidirektionaler Ein-Draht-Bus für die Datenübertragung
KOVP*	kundenorientierte Vertriebsprozess
KP	Kernprozess
KVP	kontinuierlicher Verbesserungsprozess
KWP 2000	Key-Word-Protocol 2000. Diagnoseprotokoll.
LabVIEW	Laboratory Virtual Instrument Engineering Workbench
LED	Leuchtdiode, auch Lumineszenz-Diode. Grafisch: An-/Aus-Schalter in Kreisform.
L-Line	Mittels L-Line kann die K-Line auch unidirektional betrieben werden. Die L-Line ist gemäß den Normen lediglich zur Initialisierung des Steuergerätes konzipiert.
Look & Feel	Anwender findet eine optisch vertraute Oberfläche vor. Die Oberfläche ist im Stil des übrigen Systems bzw. Betriebsystems.
LSZ*	Lichtschaltzentrum
MAC OS	Macintosh operational system
Maske	Siehe Diagnosemaske
MCD	Measurement,Calibration and Diagnosis System des ASAM e. V.
MDA	Multiple Displacement Amplification, Funkdiagnose-Interface
MDZ	Mittlere Durchlaufzeit
Mobile Viewer	Die Viewer-Softwarekomponente des Diagnosesystems MobiViSt für mobile Endgeräte, wie z. B. Tablet PCs oder Pocket PCs von Microsoft
MobiViSt	Mobile Visualisierung und Steuerung. Das aus der Dissertation entstandene Diagnosesoftware-Paket. Das Paket besteht aus den Komponenten Viewer (HMI-Software) und Editor (IDE).
monolithisch	Aus einem großen Ganzen bestehend. Im Gegensatz zu aus Einzelteilen zusammengesetzt oder modularer Bauweise.
MOST	Media-oriented Systems Transport. Serielles Bussystem zur Übertragung von Audio- und Video-, Sprach- und Datensignalen
MYCAREVENT	Mobility and Collaborative Work in European Vehicle Emergency Networks
Namensraum	Siehe Namespace
Namespace	Bestimmter Namensraum, in dem bestimmte Objekte verfügbar sind. Diese Objekte können z. B. Klassen, Interfaces oder Enumeratoren sein. Das Klassensystem des Microsoft .NET Frameworks ist z. B. in viele Namespaces aufgeteilt. Jeder Namespace verkörpert ein Teilsystem.
NT	Windows NT ist ein Betriebssystem der Firma Microsoft
OBD	On-Board-Diagnose
ODX	Open Diagnostic Data Exchange. Formaler Beschreibungsstandard der Steuergerätediagnose
OS	Operational System
OSGi	Open Services Gateway initiative
PABS*	Prüfablaufbeschreibungssprache von INPA. C-ähnliche Programmiersprache.
PCMCIA	Personal Computer Memory Card International Association
PDA	Personal Digital Assistant
PDC	Park Distance Control

Abkürzung / Begriff	Beschreibung
PDCA-Zyklus	Phasen im Prozess der kontiunuierlichen Verbesserung (Planen, Tun, Checken und Aktion)
PEP*	Produktentstehungsprozess
PInvoke	Plattform Invoke
Plug-in	Erweiterungen, mit denen die Eclipse-IDE an neue Aufgaben angepasst und um neue Funktionen erweitert werden kann.
Referenzierung	Freigabe und Veröffentlichung des Referenzstands von Dateien, die für EDIABAS und INPA benötigt werden. Referenziert werden z. B. INPA-Skripte.
Reiter	Element einer Registerkarte. Darstellung einer Registerkarte auf einer grafischen Benutzeroberfläche (nach der englischen Bezeichnung auch Tab genannt).
Ressort*	Interne Bezeichnung für Unternehmensbereiche
Result	Ergebnis eines Job-Aufrufs. Softwaretechnische Daten z. B. als Text, Zahl oder boolescher Wert.
Result-Set	Sammlung von Results. Ein Job kann mehrere Results bzw. Typen von Results zurückgeben.
RGB	Rot, Grün und Blau
SFTP	Secure File Transfer Protocol
SG	Steuergerät(e)
SGBD INPA Dienste Oberfläche*	Anwendersoftware, die diverse Dienste für INPA- und EDIABAS-Systeme anbietet. Darunter fällt das Kompilieren der INPA-Skripte.
SGBD*	Steuergeräte-Beschreibungsdatei. Beinhaltet welche Jobs, Result-Sets und Attribute ein Steuergerät hat.
SIM-Datei*	Simulation für EDIABAS von Steuergeräten. Damit können auf EDIABAS basierende Anwendungen auch dann getestet werden, wenn die verwendeten Steuergeräte nicht vorhanden sind.
Six Sigma	Qualitätsmanagement-Methodik
SMG*	Sequenzielles Motorgetriebe Steuergerät
SOAP	Simple Object Access Protocol
SP	Service Pack
SPS	Speicherprogrammierbare Steuerung
SRC-Datei*	Quellcode für eine INPA-Prüfoberfläche. Klartextdatei. In der Diagnosetechnik bezeichnet der Begriff die Diagnoseskriptquelle.
stand-alone	Dt. alleine stehend. Bezeichnet ein Elektronikgerät oder eine Software, die eigenständig, ohne weitere Zusatzgeräte, ihre Funktion erfüllen kann.
Steuergerät	Engl. Electronic Control Unit (ECU). Elektronisches Modul, das überwiegend an Orten eingebaut wird, an denen etwas gesteuert oder geregelt werden muss. Wird im Kfz-Bereich in diversen elektronischen Bereichen eingesetzt (z. B. Komfort, Sicherheit oder Motorsteuerung).
Steuergerätebefehl	Siehe Job
SVG	Scalable Vector Graphics
SVGWF	SVG Widget Framework
SVGZ	Scalable Vector Graphics Zip
SWT	Standard Widget Toolkit
Syntax	Wortschatz einer Programmiersprache
TCT	Total Cycle Time
Texteditor	Gängiger ASCII-/Unicode-Editor zum Bearbeiten von Textdateien wie Diagnoseskriptquellen
TP	Teilprozess

Abkürzung / Begriff	Beschreibung
UML	Unified Modelling Language
Use Case	Siehe Anwendungsfalldiagramm
VDA	Verband der Automobilindustrie
VDO	Ehemals: Vereinigte Deutsche Tachometer-Werke GmbH und Offenbacher Tachometer Werke. Heute: Unternehmen des Konzerns Continental. Zulieferer der Automobilindustrie für Automobilelektronik.
VGA	Video Graphics Array
VI	Virtuelles Instrument in LabVIEW
VMI	Verlag Moderne Industrie
WLAN	Wireless Local Area Network
WYSIWYG	What You See Is What You Get
XHTML	Extensible HyperText Markup Language
XML	Extensible Markup Language
XP	Betriebssystem der Firma Microsoft. XP steht dabei für „eXPerience“ (engl. für Erfahrung, Erlebnis).
ZKF	Zentralverband für Karosserie- und Fahrzeugtechnik

B. Literaturverzeichnis

Legende:

Mit einem * markierte Quellen sind durch den Kooperationspartner verfasst bzw. herausgegeben

Ref.	Dokument (Titel)	Autor	Herausgeber	Ort	Datum
Adam 2002	SVG – Das Praxisbuch	Alexander Adam	Franzis` Verlag	85586 Poing	2002
Archer 2001	Inside C#	Tom Archer	Microsoft Press Deutschland	München	2001
ASAM ODX 2006	ASAM MCD-2D (ODX) Version 2.1.0 Data Model Specification	Association for Standardisation of Automation and Measuring Systems (ASAM e. V.)	ASAM e. V.	Höhenkirchen	09/2006
Automobil Produktion, November 2002	Wachstumsfaktor Elektrik/Elektronik http://www.automobil-produktion.de/ap/themen/00634/index.php	Chefredakteur: Gerd Scholz	verlag moderne industrie GmbH	Justus-von-Liebig-Straße 1 86899 Landsberg	11/2002
Automobil Produktion, Juli 2003	Wandel der Wertschöpfungsanteile http://www.automobil-produktion.de/ap/themen/01784/index.php	Chefredakteur: Gerd Scholz	verlag moderne industrie GmbH	Justus-von-Liebig-Straße 1 86899 Landsberg	07/2003
Automobil Produktion, August 2004	Was bringen Prozessinnovationen im Fahrzeugbau? http://www.automobil-produktion.de/ap/themen/03760/index.php	Chefredakteur: Gerd Scholz	verlag moderne industrie GmbH	Justus-von-Liebig-Straße 1 86899 Landsberg	08/2004
Automobil Produktion, Februar 2004	In Zukunft vernetzt http://www.automobil-produktion.de/ap/themen/02795/index.php	Chefredakteur: Gerd Scholz	verlag moderne industrie GmbH	Justus-von-Liebig-Straße 1 86899 Landsberg	02/2004
Automobil Produktion, Juni 2005	Automobil Produktion: „Qualität kann man nicht erprüfen“	Armin Götz	verlag moderne industrie GmbH	Justus-von-Liebig-Straße 1 86899 Landsberg	Ausgabe: 06/2005
Automobil Produktion, November 2005	Welt-Rekord: 60 Millionen Fahrzeuge produziert http://www.automagazine.de/ap/news/11788/index.php	Chefredakteur: Gerd Scholz	verlag moderne industrie GmbH	Justus-von-Liebig-Straße 1 86899 Landsberg	11/2005
Balzer 2007	Diagnose in verteilten automotiven Systemen (in Jürgen Gausemeier (Ed.), 5. Paderborner Workshop Entwurf mechatronischer Systeme, 22.-23., Volume 210 of HNI-Schriftenreihe, pp. 243-254)	Heinrich Balzer, Benno Stein, Oliver Niggemann	Heinz Nixdorf Institut	Paderborn	03/2007
Batik SVG Toolkit 2008	Batik SVG Toolkit http://xmlgraphics.apache.org/batik/	The Apache Software Foundation	The Apache Software Foundation	Forest Hill, MD 21050-2747, USA	Link: 02/2008
Baumgartner 1999	10 Testmethoden in der Evaluation interaktiver Lehr- und Lernmedien.	Baumgartner, P.	Waxmann Verlag	Münster:	1999
Becker 2000	Prozessmanagement	Jörg Becker, Martin Kugeler, Michael Rosemann	Springer Verlag	Berlin	2000
Birnbaum 2000	Getting to Know OBD II	Jerry G. Truglia , Ralph Birnbaum	AST Training		12/2000
INPA-Produktions	Prozessdatenerfassung INPA: - Produktionsmeldung Werk	Florian Schwarz	Kooperationspartner*		11/2006

Ref.	Dokument (Titel)	Autor	Herausgeber	Ort	Datum
meldung 2006*	(Anhang D, eingereichte Dissertation Florian Schwarz, April 2008, Universität Passau)				
INPA-Prozessdaten erfassung 2006*	Prozessdatenerfassung INPA: - Nacharbeit Schwerpunktsthemen-Bericht Werk (Anhang D, eingereichte Dissertation Florian Schwarz, April 2008, Universität Passau)	Florian Schwarz	Kooperationspartner*		11/2006
MobiViSt-Prozessdaten erfassung 2008*	Ablaufprotokoll (Verwaltung/Nacharbeit) (Anhang F, eingereichte Dissertation Florian Schwarz, April 2008, Universität Passau)	Florian Schwarz	Kooperationspartner*		01/2008
INPA-Analyse 2007*	Prozessanalyse des Ablaufsystems INPA - Fachkonzept Teil 1: Prozesserfassung und Softwarearchitektur (Anhang C, eingereichte Dissertation Florian Schwarz, April 2008, Universität Passau)	Florian Schwarz	Kooperationspartner*		2007
INPA-Analyseberichte 2007*	Anlayseberichte INPA und MobiViSt (Ist-/Soll-System) (Anhang E, eingereichte Dissertation Florian Schwarz, April 2008, Universität Passau)	Florian Schwarz	Kooperationspartner*		2007
SGBD 2003*	SGBD – Erstellung, Pflege und Absicherung		Kooperationspartner*		04/2003
Bosch 2002	Autoelektronik	Robert Bosch GmbH	Springer Verlag	Stuttgart	09/2002
Bosch 2007, Esitronic	Die Software-Komplettlösung für die Zukunft Ihrer Werkstatt: ESI[tronic] von Bosch	Produktbereich Diagnostics	Robert Bosch GmbH	73207 Plochingen	2007
CFCOM Manual 2003	CFCOM Manual 1.1 (Build 1632) (HTML Dokument)	Odyssey Software Inc.	Odyssey Software Inc.		2003
Charwat 1992	Lexikon der Mensch-Maschine Kommunikation	Hans Jürgen Charwat	Oldenburg Verlag GmbH	München	1992
Daum 2006	Java-Entwicklung mit Eclipse 3.2	Berthold Daum	dpunk.verlag	Heidelberg	2006
DeGEval 2002	Angefragte Gutachten / Meta-Evaluationen auf Basis der DeGEval-Standards	Vorstand der DeGEval	Deutsche Gesellschaft für Evaluation e.V.	Köln	04/2002
digital-world 2006	Mobiles Linux und Windows sollen Marktanteile erobern http://www.digital-world.de/index.cfm?pid=363&pk=132645	Digital-World	© Digital-World 2007 IDG Magazine Media GmbH	München	04/2006
Dudenhöffer 2003	"Kann Deutschland vom Zulieferer-Wachstum profitieren?"	Prof. Dr. Ferdinand Dudenhöffer	Automotive Engineering Partners		02/2003
Dunkel 2003	Softwarearchitektur für die Praxis	Jürgen Dunkel Andreas Holitschke	Springer Verlag	30459 Hannover	2003
EDIABAS 2004	EDIABAS – API SCHNITTSTELLENBESCHREIBUNG	SOFTING AG	SOFTING AG	München	03/2004

Ref.	Dokument (Titel)	Autor	Herausgeber	Ort	Datum
EDIABAS User 2004	EDIABAS – Benutzerhandbuch	SOFTING AG	SOFTING AG	München	03/2004
EDIABAS-SIM 2004	EDIABAS - STEUERGERÄTE-SIMULATOR	SOFTING AG	SOFTING AG	München	03/2004
Ellis 2004	Dienstleistungsmanagement	Avy Ellis, Michael Kauferstein	Springe Verlag	Berlin	2004
EPL 2004	Eclipse Public License http://www.eclipse.org/legal/epl-v10.html	Eclipse Foundation	This Agreement is governed by the laws of the State of New York		2004
eSVG Plug-in 2007	Intesis eSVG Plug-in http://esvg.ultimodule.com/bin/esvg/templates/splash.asp?NC=7892X	Exor International, Inc.	Exor International, Inc.	Cincinnati, OH 45246, USA	Link: 02/2008
F.A.Z. 2005	Produktionskosten VW stellt alle Modelle auf den Prüfstand, Nr. 209 / Seite 16	Autor: rit.	Frankfurter Allgemeine		09/2005
FO 2002*	Unternehmensorganisation	Organisation und Inhouse Consulting	Kooperationspartner*		11/2002
Forbrig 2004	Lehr- und Übungsbuch Softwareentwicklung	Prof. Immo Kerner Prof. Peter Forbrig	Carl Hanser Verlag	Leipzig	2004
Gevatter 2006	Handbuch der Mess- und Automatisierungstechnik im Automobil	Hans-Jürgen Gevatter, Ulrich Grünhaupt:	Springer-Verlag	Berlin	2006
Global Electronics 2007	Der Elektronikanteil im Automobil steigt weiter http://www.global-electronics.net/link/de/18974391#18974391	Global electronics ist das Elektronik-Netzwerk der Messe München GmbH	Messe München GmbH Messegelände	D-81823 München	Link: 02/2008
Grupp 1999	Das DV-Pflichtenheft zur optimalen Softwarebeschaffung	Bruno Grupp	MITP-Verlag GmbH	Bonn	1999
Hammerschmidt 2004	Artikel Nr. 22104749: "Autoelektroniker suchen Wege aus der Diagnosefalle" http://eetimes.eu/germany/showArticle.jhtml?articleID=22104749&queryText=automobilelektronik	Christoph Hammerschmidt	EETimes Germany		07/2004
Hegner 2003	Methoden zur Evaluation von Software	Marcus Hegner	GESIS IZ (ASI e.V.)	Bonn	05/2003
Heilmann 2003	IT-Projektmanagement	Heidi Heilmann	dpunkt.verlag GmbH	Heidelberg	2003
Henning 2003	Taschenbuch Multimedia	Peter A. Henning	Fachbuchverlag Leipzig	Leipzig	04/2003
Herner 2001	Elektrik	Anton Herner	Vogel Buchverlag, Technische Akademie des Kfz-Gewerbes	Würzburg	2001
Hirsemann 2003	JavaScript Wissen dass sich auszahlt	Thorsten Hirsemann, Dorothea Rochusch	SPC TEIA Lehrbuch Verlag GmbH	Berlin	2003
Holzinger 2000	Basiswissen Multimedia Band 1	Andreas Holzinger	Vogel Buchverlag	Würzburg	2000
INDIA 2001	INDIA – Intelligente Diagnose in	Thomas	Technische	München,	2001

Ref.	Dokument (Titel)	Autor	Herausgeber	Ort	Datum
	der industriellen Anwendung (Abschlußbericht zu einem vom BMBF geförderten öffentlichen Projekt)	Guckenbiehl, Lothar Hotz, Peter Struss	Universität München Institut für Informatik, München; 01 IN 509 H1 Robert Bosch GmbH, Stuttgart; 01 IN 509 A2	Stuttgart	
INPA 2003	Benutzerdokumentation INPA	SOFTING AG, u. a. Georg Luderböck	SOFTING AG, u. a. Georg Luderböck	München	01/2003
Landwehr 1977	Darstellung und Analyse eines Kommunikationsmodells	Ecker, H. Landwehr, J.		Düsseldorf	1977
Langmann 1994	Graphische Benutzerschnittstellen	Prof. Dr. Reinhard Langmann	VDI Verlag GmbH	Düsseldorf	1994
Lingnau 2003	Mechatronik	G. Lingnau	Verlag Moderne Industrie	Landsberg	2003
Ludes 2003	Einführung in die Medienwissenschaft. Entwicklungen und Theorien	Jochen Ludes		Berlin	2003
Marquardt 2007	WPF Crashkurs	Bernd Marquardt	Microsoft Press Deutschland	München	2007
Microsoft .NET Framework EULA	Microsoft .NET Framework EULA http://msdn2.microsoft.com/en-us/library/ms994405.aspx	Microsoft Corporation	Microsoft Corporation	Washington	Link: 02.2008
Mircosoft 2003	Dr. International (Group): Developing International Software	Microsoft Press	Microsoft Press	Washington	2003
MSDN Deutschland 2004	Das .NET Compact Framework – Übersicht http://www.microsoft.com/germany/msdn/library/net/compactframework/DasNETCompactFramework1.mspx?pf=true	MSDN Deutschland	Microsoft Corporation		06/2004
MYCAREVENT 2007	MYCAREVENT http://www.mycarevent.com/index.aspx	Heiko Dirlenbach	Forschungsinstitut für Rationalisierung Pontdriesch	52062 Aachen	Link: 02.2008
Nielsen 1993	Usability Engineering	Jakob Niesen	Morgan Kaufmann	Mountain View, California	1993
Patt 2002	Microsoft .Net – Eine Einführung	David S. Patt	Microsoft Press Deutschland		2002
PInvoke 2008	Platform Invoke Support http://msdn2.microsoft.com/de-de/library/h50dxzwx.aspx		Microsoft Corporation		2008
Porter 1989	Globaler Wettbewerb: Strategien der neuen Internationalisierung	Porter, M.		Wiesbaden	1989
Ricken 2005	Ausarbeitung zum Thema „SVG vs. SWF“ für das Studienfach Multimedia & Webtechnologien (MWT)	Christoph Ricken Jens Schäfers Björn Schröder	Fachhochschule Lippe und Höxter	32657 Lemgo	2005
Rupp 2002	Requirements Engineering und -Management	Chris Rupp	Carl Hanser Verlag	München	2002

Ref.	Dokument (Titel)	Autor	Herausgeber	Ort	Datum
Schäffer 2007	Fahrzeugdiagnose mit OBD	Florian Schäffer	Elektor-Verlag GmbH	Aachen	2007
Schäuffele 2003	Automotive Software Engineering	Jörg Schäuffele, Thomas Zurawka	Vieweg & Sohn Verlag	Wiesbaden	07/2003
Scheer 1998	Wirtschaftsinformatik, Studienausgabe. 2. Auflage	Scheer, A.-W	Springer-Verlag	Berlin	1998
Scheer 2001, ARIS Methode	ARIS Methode	IDS Scheer AG	IDS Scheer AG	Saarbrücken	2001
Schilder 2005	Autotechnik heute	Hans-Joachim Schilder, Eberhard Kittler:	Motorbuch Verlag	Stuttgart	2005
Schiller 2003	Mobilkommunikation	Jochen Schiller	Addison-Wesley Verlag		2003
Schmelzer 2004	Geschäftsprozessmanagement in der Praxis	Hermann J. Schmelzer, Wolfgang Sesselmann	Hanser Verlag	München	2004
Schneider 2004	Taschenbuch der Informatik	Uwe Schneider, Dieter Werner	Carl Hanser Verlag	Leipzig	2004
Schönsleben 2001	Integrales Informationsmanagement: Informationssysteme für Geschäftsprozesse, 2. Auflage	Schönsleben, Paul	Springer Verlag	Berlin	2001
Schweitzer 1989, VDI Bericht 787	VDI-Berichte 787Mechatronik. Aufgaben und Lösungen	G. Schweitzer	VDI-Verlag	Düsseldorf	1989
Seminar01379_EDIABAS	Seminar 01379 – Einführung in die Fahrzeugdiagnose und Diagnosestools	Maresa Praxenthaler, TI-430, (Fa. ESG)	Maresa Praxenthaler, TI-430, (Fa. ESG)	München	12/1999
Shavor 2003	The Java Developer's Guide to Eclipse	Sherry Shavor, Jim D'Anjou	Addison-Wesley	Boston	2003
Siedentop 2008, Tagungs-Infomappe	12. Jahrestagung Elektronik-Systeme im Automobil: „Diagnose elektronischer Steuergeräte in der Prozesskette“	Dr. Michael Siedentop, Vorstand, Softing AG	Tagungsunterlagen Euroforum Deutschland GmbH, 40512 Düsseldorf	Softing AG, München	Link: 02/2008
Siemens VDO 2007	Equip Auto 2007: Siemens VDO präsentiert universelles Fahrzeugdiagnose-System http://www.vdo.de/press/archive/replacementparts/2007/sv-200710-001-d.htm	Joachim Töpfer	VDO Automotive AG	Frankfurt	10/2007
Sjöström 2003	Add COM and ActiveX Support in .NET Compact Framework Using Odyssey Software CFCOM http://msdn2.microsoft.com/en-us/library/aa446494.aspx	Andy Sjöström	businessanyplace.net		09/2003
Softing AG Pressestelle 2003	Steuergeräte im (Zu)Griff http://www.softing.com/home/de/pdf/ae/press/professional-article/2003/0303_Steuergeraete_imZuGriff.pdf	Softing AG	Softing AG	Münchne	03/2003
Staud 2001	Prof. Dr. J. Staud: Geschäftsprozessanlayse	2. Auflage	Springer Verlag Berlin	Heidelberg	2001

Ref.	Dokument (Titel)	Autor	Herausgeber	Ort	Datum
Stein 2004	Rechnernetze und Internet	Erich Stein	Carl Hanser Verlag	München	2004
Steppan 2003	iX Magazin: "Sunblocker"	Bernhard Steppan	Heise Zeitschriften Verlag GmbH & Co. KG	Hannover	12/2003
SVGI 2007	SVG Working Group: SVG Implementierung http://www.svgi.org/	W3C			2007
Templeman 2003	COM Programming with Microsoft .Net	Julian Templeman	Microsoft Press	Washington	2003
Thaller 2000	Design und Implementierung	Georg Erwin Thaller	Verlag Technik	10400 Berlin	2000
Triebe 1996	Anforderungskatalog für Softwareentwicklung Auswahl und Anwendung	J. K. Triebe, M. Wittstock	Bundesanstalt für Arbeitsschutz	Dortmund	1996
Valldorf 2003	Advanced Microsystems for Automotive Applications 2003	Jürgen Valldorf	Springer Verlag	Berlin	2003
Vasters 2001	Net-Crashkurs	Vasters, Oellers	Microsoft Press Deutschland		2001
Vasters 2002	.Net-Crashkurs	Vasters, Oellers, Javidi	Microsoft Press Deutschland	München	2002
VDA 2004	Jahresbericht 2004	VDA Presse- und Öffentlichkeitsarbeit	Verband der Automobilindustrie e.V. (VDA):	Frankfurt am Main	2004
VDA 2005	Jahresbericht 2005	VDA Presse- und Öffentlichkeitsarbeit	Verband der Automobilindustrie e.V. (VDA):	Frankfurt am Main	2005
VDA 2006	Jahresbericht 2006	VDA Presse- und Öffentlichkeitsarbeit	Verband der Automobilindustrie e.V. (VDA):	Frankfurt am Main	2006
Versteegen 2001	Change Management bei Software-Projekten	Gerhard Versteegen	Springer Verlag	Berlin-Heidelberg	2001
Versteegen 2004	Anforderungsmanagement	Gerhard Versteegen	Springer Verlag	Berlin-Heidelberg	2004
Voss 1998	Entwicklung von Graphischen Benutzungsschnittsellen	Dr. Josef Voss, Dr. Dietmar Nentwig	Carl Hanser Verlag	München	1998
W3C-Mobile 2003	W3C Recommendation: Mobile SVG Profiles: SVG Tiny and SVG Basic http://www.w3.org/TR/2003/REC-SVGMobile-20030114	W3C	Presseveröffentlichung des http://www.w3.org/		01/2003
W3C-Spec 2003	W3C Spezifikation: Scalable Vector Graphics (SVG) 1.1 http://www.w3.org/Graphics/SVG/	W3C	Veröffentlichung des http://www.w3.org/		01/2003
W3C-Std 2006	W3C veröffentlicht neue Ausgaben der zentralen XML-Standards http://www.w3c.de/Press/2006/xml-pressrelease.de.html	Janet Daly	Presseveröffentlichung des http://www.w3.org/		08/2006
Wehner 2006	"Neue Software zur Elektronik-Diagnose bei VW", Beitrag 2672155 http://www.kfzbetrieb.de/news/kb_beitrag_2672154.html	Andreas Wehner	Vogel Auto Medien GmbH & Co.KG	Max-Planck-Str. 7/9 97082 Würzburg	02/2006
Wenz 2006	Technik erklärt: Die Flash-	Christian Wenz	Neue	Bayerstraße 26	Ausgabe

Ref.	Dokument (Titel)	Autor	Herausgeber	Ort	Datum
	Konkurrenz SVG		Mediengesellschaft Ulm mbH	80335 München	02/2006
Wenzel 2001	ASAM-MCD Einführung	B. Wenzel	ASAM e.V.	München	04/2001
Widder 2003	Javamagazin: "Everything is a Plugin"	Oliver Widder	Software & Support Verlag GmbH		November 2003
Wigley 2003	.Net Compact Framework Core Reference	Andy Wigley	Microsoft Press	Washington	2003
Wigley 2003	Microsoft .Net Compact Framework	Andy Wigley	Microsoft Press	Washington	2003
Wille 2000	C# first guide	Christoph Wille	Markt+Technik Verlag	München	2000
Wottawa 1998	Lehrbuch Evaluation.	Wottawa, H., Thierau, H.	Huber	Bern	1998
Yao 2004	.Net Compact Programming with C#	Paul Yao, David Durant	Addison Wesley	Boston	2004
Zimmermann 2006	ATZ-MTZ-Fachbuch: "Bussysteme in der Fahrzeugtechnik"	Prof. Dr. Werner Zimmermann:	Vieweg Verlag		04/2006
ZKF 2004	Branchenbericht 2004	Dr. Klaus Weichtmann (Hauptgeschäftsführer) Dipl. – Oec. Anette Gundlach (Referatsleiterin)	Zentralverband Karosserie- und Fahrzeugtechnik e.V. (ZKF)	Bad Vilbel	2004