Ein experimenteller Vergleich zwischen einer plan-getriebenen und einer leichtgewichtigen Entwicklungsmethodik zur Spezifikation von eingebetteten Systemen

Alexander Raschke* und Ramin Tavakoli Kolagari**

Zusammenfassung

Agile Methoden spielen eine immer wichtigere Rolle in den unterschiedlichsten Arten von Projekten. Sie stehen für eine flexible Software-System-Entwicklung, die sich an den Anforderungen des Kunden orientiert und die außerdem eine verlässliche Qualität versprechen. Das Experiment, das wir in dem vorliegenden Technischen Bericht beschreiben, wurde als Praktikum im Hauptstudium in der Universität Ulm im Sommersemester 2003 durchgeführt. Darin wurde versucht herauszufinden, inwieweit die Anwendung von agilen Methoden in frühen Phasen der Softwareentwicklung eingebetteter Systeme sinnvoll und durchführbar ist. Wir wandten Elemente bekannter agiler Techniken wie XP und SCRUM in der Requirements Engineering Phase des Entwicklungszyklus an. Da wir durch die zu erwartende Anzahl an Studierenden, die an dem Experiment teilnahmen, nur mit einer kleinen Stichprobe rechnen konnten, haben wir in erster Linie nicht versucht, die Ergebnisse als statistisch stichhaltig zu erfassen, sondern vor allem, Hinweise auf die Anwendbarkeit der agilen Methoden im Requirements Engineering in der täglichen Arbeit der Studierenden zu erfassen. Dies wurde insoweit unterstützt, dass wir über den vergleichsweise langen Zeitraum eines Semesters mit den Studierenden zusammenarbeiten konnten, was viele Einmaleffekte ausglich, die bei einem kurzzeitigen Experiment unweigerlich auftreten. Die Hinweise, die wir herausfinden konnten, legen es nahe, dass die Anwendung von agilen Elementen im Requirements Engineering nicht nur möglich ist, sondern auch die versprochenen Effekte, wie verbesserte Qualität, Kundennähe und auch Spaß an der Entwicklung, erzielt.

*Universität Ulm, Fakultät für Informatik, Abt. Programmiermethodik und Compilerbau, 89069 Ulm, e-mail: Alexander.Raschke@informatik.uni-ulm.de

**DaimlerChrysler AG, Forschungszentrum Ulm, Abteilung Softwareprozessgestaltung, Postfach 23 60, 89081 Ulm, e-mail: Ramin.Tavakoli_Kolagari@daimlerchrysler.com

Inhaltsverzeichnis

1 Einleitung

Seit Anfang des neuen Jahrhunderts ist „Agilität“ in jeder Munde. Was steckt hinter diesem Begriff, der einen solchen Hype in der Forschung und vor allem der Praxis ausgelöst hat? Im Zuge des „Agil-machens“ der unterschiedlichsten Abläufe in der Softwareentwicklung stellte sich für uns die Frage, inwieweit sich agile Elemente bekannter Techniken auch auf das Requirements Engineering übertragen lassen. Vor allem war es interessant für uns, ob das Ziel, das hinter einem agilen Element steckt, auch dann erreicht wird, wenn dieses Element aus seiner Umgebung herausgelöst und in eine andere übertragen wird. Da agile Techniken sich zumeist aus einem großen Pool an „best-practices“ speisen, die gemäß einschlägiger Literatur genau in der vorgeschriebenen Art und Weise anzuwenden sind, um die anvisierten Ziele mit ihnen erreichen zu können, vermuteten wir, dass sich der Weg zu diesen Zielen für das Requirements Engineering anders ausgestaltet als für die Gesamtsystementwicklung.

Klassisches Requirements Engineering ist charakterisiert durch eine vergleichsweise hohe Investition in die Dokumentation der Anforderungen; schwergewichtige Prozesse werden genutzt, um die Anforderungen zu erfassen und konsistent zu halten. Das liegt vor allem daran, dass Requirements Engineering eine Phase im Software Engineering ist, die vor allem dann Sinn macht, wenn die Projekte eine gewisse Größe erreichen und die Systeme damit nicht mehr ohne weiteres zu überblicken sind oder wenn man in einer Auftraggeber/Auftragnehmer Beziehung steht. Betrachtet man daher beispielsweise XP (eXtreme Programming, [BF00]) als Vertreter agiler Methoden, so stellt man fest, dass das Requirements Engineering der Teil der Softwareentwicklung ist, wo am meisten gespart wird. Die Artefakte, die während des Requirements Engineering generiert werden, unterstützten die Entwicklung der Produkte, besitzen als solche aber keinen inhärenten Wert. Indem sie in solcher Art und Weise argumentieren, versuchen agile Methoden klassische Prozesse zu verschlanken, wobei klassische Herangehensweisen regelmäßig überprüft werden, inwieweit sie direkt für die Entwicklung des Projekts genutzt werden. Setzen wir voraus, dass im Zuge immer stärkerer Bedürfnisse des Marktes verbesserte Qualität in kürzeren Innovationszyklen benötigt wird, so müssen auch wir uns Gedanken machen, wie wir das Requirements Engineering stromlinienförmiger und damit flexibler gestalten können. Praktisch ohne das Requirements Engineering auskommen zu wollen, wie es XP vorschlägt, kann dabei für uns nicht in Frage kommen, da es zunächst zu bezweifeln ist, ob man die hohen Qualitätsanforderungen an die Produkte ohne diese Phase überhaupt umsetzen kann und des weiteren ist es häufig nicht verzichtbar, weil die umgebenden Prozesse dies als Voraussetzung haben: So ist es in der Automobilindustrie der Fall, dass die Entwicklung der Steuergeräte auf Seiten der Automobilhersteller im wesentlichen die Spezifikation umfasst, die dann wiederum die vertragliche Grundlage für die Realisierung durch Zulieferer darstellt.

Um einen ersten Eindruck zu bekommen, wie sich ein solches „agiles Requirements Engineering“ ausgestaltet, haben wir das im folgenden beschriebene Experiment gestaltet. Das Experiment wurde an der Universität Ulm im Sommersemester 2003 in Form eines Praktikums für Informatikstudenten im Hauptstudium durchgeführt. Für dieses Experiment wurden die Studenten in zwei Gruppen eingeteilt, die beide mit dem Tool „Statemate“ ([HN96] und [Sta]) Statecharts zu modellieren hatten. Während die eine Gruppe ihre Statecharts in einem in der Praxis erfolgreichen, klassischen Prozess modellierte, ging die andere Gruppe nach einem leichtgewichtigen Prozess vor, der mit agilen Elementen angereichert war, die für dieses Experiment erstellt wurden und sich an bekannten agilen Methoden orientierten. Statecharts sind präzise Beschreibungsmodelle und praktisch so mächtig wie textuelle An-

forderungsdokumente, die Systemeigenschaften beschreiben. Aus diesen Gründen und weil Studenten üblicherweise mehr Spaß daran haben, Statecharts zu modellieren als Textdokumente zu verfassen, haben wir uns entschieden, Statemate als Spezifikationstool zu nutzen. In der Praxis allerdings wird Statemate eher zur ersten prototypischen Darstellung eines Systems genutzt, vor allem daher, weil es neben dem rein beschreibenden Charakter auch die Möglichkeit bietet, das durch die Statecharts modellierte System auszuführen. Diese Eigenschaft wurde auch in diesem Experiment genutzt, da wir in Anlehnung an XP unsere Modelle intensiv getestet haben, was nur für ausführbare Spezifikationen möglich ist: textuelle Spezifikationen lassen sich zwar inspizieren aber nicht testen, was in diesem Zusammenhang einen großen Unterschied darstellt.

Wir verglichen beide gewählten Prozesse bezüglich ihren Fähigkeiten, die Entwicklung von leicht änderbaren und kundenorientierten Spezifikationen zu unterstützen. Im einzelnen versuchten wir Hinweise herauszufinden, inwieweit die Prozesse folgende Schlüsselaspekte unterstützten: Kundenorientierung, Modellierungszeit, Qualität der Testfälle und der Statemate Modelle, Systemverständnis, Kreativität des Entwicklungsteams, Nachvollziehbarkeit der Architekturentscheidungen und Wiederverwendung, Änderbarkeit und Wartbarkeit der Spezifikationen. Da wir nur mit einer kleinen Versuchsgruppe rechnen und damit keine signifikanten Ergebnisse ableiten konnten, verließen wir uns in erster Linie auf die subjektiven Einschätzungen der Studenten neben messbaren Tatsachen (wie beispielsweise Zeit).

Wir erwarteten vor Beginn des Experiments, dass die Spezifikationen, die im leichtgewichtigen Prozess erstellt wurden, leichter zu ändern und zu warten sein würden, da durch die iterative Herangehensweise die leichte Änderbarkeit in gewisser Weise in die Spezifikationen implementiert worden sein müsste. Weiterhin erwarteten wir, dass die Stärken des plangetriebenen Prozesses in der Erstellung vollständiger und aktueller Testfälle liegen würde und dass das Team ein besseres Systemverständnis aufweisen würde, das sich das Team mit Hilfe der eingesetzten Use Cases ([Jac95] und [JC95]) würde erschließen können. Im wesentlichen hat sich ergeben, dass sich in diesem experimentellen Setting keine signifikanten Unterschiede zwischen den beiden Prozessen ergeben haben. Es haben sich Hinweise gefunden, die unsere anfänglichen Thesen stützten, aber auch solche, mit denen wir nicht rechneten.

In den folgenden Kapiteln beschreiben wir den Ablauf des Experiments und die Hinweise und Erfahrungen, die wir ableiten konnten. In Kapitel 2 geben wir einen kurzen Überblick über die Prozesse und Techniken, die als Ausgangspunkt für die im Experiment angewandten Prozesse dienten. In Kapitel 3 beschreiben wir den Ablauf des Experiments im Detail. In den beiden darauf folgenden Kapiteln beschreiben wir die Prozesse des Experiments und beschreiben in Kapitel 6 die Ergebnisse und diskutieren diese kritisch. Der Anhang umfasst die wesentlichen Dokumente für die Studierenden und kann als Grundlage für ähnliche experimentelle Praktika an anderen Universitäten dienen.

2 Prozesse und Techniken

Für uns ist ein Prozess agil oder leichtgewichtig, wenn er zielorientiert und direkt ist, verlässliche Qualität liefert, an den speziellen Bedürfnissen der beteiligten Stakeholder orientiert ist und die beteiligten Personen in den Mittelpunkt stellt, wobei er auf die Erstellung eines Artifakts fokussiert ist. Im Zusammenhang mit unserem Experiment bedeutet das, der Fokus des Prozesses sollte auf der Anforderungsspezifikation (Statemate Modell) liegen. Da die meisten der in der Literatur genannten agilen Methodiken (beispielsweise [ASRW02]) ihren

Schwerpunkt auf den Quellcode legen, mussten wir zentrale Ideen dieser Ansätze auf Spezifikationsdokumente übertragen. Wir entschieden uns als Basis für *eXtreme Programming* und *Scrum*, zwei wohlbekannte agile Methoden.

Beim plangetriebenen Ansatz entschieden wir uns für einen Ansatz mit abstrakten Ebenen, welcher im ITEA Forschungsprojekt EMPRESS ([Heu04]) entwickelt worden ist und die Spezifikationstechnik in der Praxis verbessert. Neben einem kurzen Überblick über die verwendeten Methoden, gibt dieses Kapitel eine kurze Einführung in die wesentlichen Merkmale der eingesetzten Statecharts und das verwendete Tool Statemate.

2.1 eXtreme Programming (XP)

XP ([ASRW02], [BF00]) ist eine innovative Softwareentwicklungsdisziplin, die existierende Methoden und Techniken in einen gemeinsamen Rahmen einbettet. Kent Beck entwickelte XP ungefähr vor acht Jahren in einem Projekt bei DaimlerChrysler ([ACL02]). Mit seinem Kollegen Ward Cunningham arbeitete er einige Jahre zusammen, um einen neuen Ansatz zu erforschen, der die Softwareentwicklung einfacher und flexibler gestalten sollte. Das besondere an XP ist, dass es allgemein anerkannte Techniken zum Extremen ausreizt. Basierend auf den Werten Einfachheit, Feedback, Kommunikation und Mut sind die im folgenden kurz beschriebenen Techniken die wichtigsten innerhalb von XP:

- Gemeinsames *Codeeigentum* der Entwickler, die den Code jederzeit durch „*Pair Programming*“ erstellen und inspizieren können.
- Code wird regelmäßig neu arrangiert (*Refaktorisierung*), um ihn immer möglichst einfach und überschaubar zu gestalten.
- Eine *Systemmetapher* beschreibt die Grundidee des Systems und sollte als erste adressiert werden für Begründungen zu Entscheidungen.
- *User Stories* sind die grundlegenden Kommunikationselemente zwischen einem *Vor-Ort Kunden* und dem Entwicklungsteam, um Anforderungen zu beschreiben.
- Tests, die auf den User Stories basieren, werden jederzeit durchgeführt und nicht nur von den Entwicklern, sondern auch den Kunden, die damit überprüfen können, ob das Produkt ihre Anforderungen erfüllt. Die Beschreibung aller Testfälle ist am ehesten vergleichbar mit klassischen Spezifikationen und wird vor der Implementierung erstellt.
- Iterationszyklen sind sehr viel kürzer als in anderen Methoden.

Im folgenden werden wir uns auf den Gebrauch der User Stories in unterschiedlichen Phasen von XP konzentrieren. Für weitere Informationen bezüglich XP verweisen wir auf [ACL02], [ASRW02] und [BF00].

Im eXtreme Programming sind User Stories eingeführt, um Anforderungen festzuhalten und zu beschreiben. In der *Explorationsphase* werden die Stories regelmäßig kreiert - im Idealfall von einem Vor-Ort Kunden, der ein Repräsentant des Kunden innerhalb des Entwicklungsteams ist. Der Vor-Ort Kunde sollte während der Entwicklung für die Entwickler jederzeit erreichbar und am besten vor Ort sein, um aufkommende Fragen beantworten und regelmäßig Feedback geben zu können. Dieser Kunde muss natürlich mit der entsprechenden

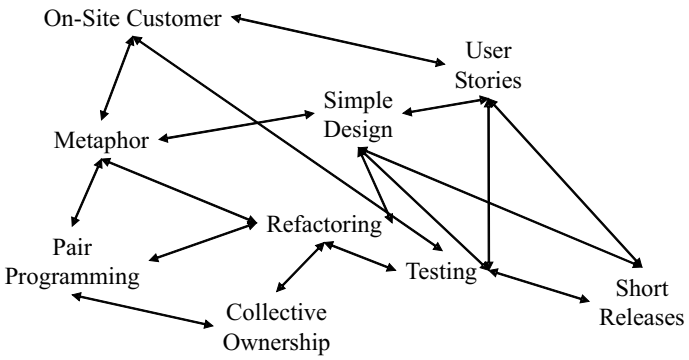

Abbildung 1: Ausgewählte XP-Techniken, ein kohärentes Netzwerk.

Befugnis ausgestattet sein, systemrelevante Entscheidungen zu treffen (beispielsweise Änderungen genehmigen). Dies ist im allgemeinen auch möglich für solche Projekte, die XP vor Augen hat, da die gesamte Systementwicklung nicht länger dauern sollte als drei, im Höchstfall sechs Monate. In der *Planungsphase* werden die Stories vom Kunden ausgewählt, die in dem nächsten Entwicklungsschritt umgesetzt werden sollen. Die Auswahl findet statt auf Grundlage der Priorisierung der Stories durch den Auftraggeber und der Schätzung der Entwickler, wie lange es dauern wird, die Anforderungen der Story umzusetzen. Das bedeutet für die Stories, dass sie konkret genug formuliert sein müssen, um von den Entwicklern abgeschätzt werden zu können, ohne allerdings ihre Stärke zu verlieren, eine Beschreibung eines Szenarios aus Sicht des Kunden zu sein. Diese ausgewählten Stories dienen dann als Input für die *Iterationen zur Release Phase*. Dort bilden sie zunächst die Grundlage für die Erstellung der Testfälle und werden dann zu ihrer Implementierung genutzt. Am Ende der Entwicklung können die Stories noch einmal genutzt werden, um ein Bedienhandbuch für das entwickelte System zu erstellen.

User Stories werden üblicherweise in der Form von *Story Cards* beschrieben. Diese Story Cards stellen im wesentlichen den Rahmen dar, in welchem die Story zu dokumentieren ist; typischerweise enthalten diese einen eindeutigen Identifier, eine Priorisierung und eine Abschätzung der Entwickler, wie lange die Umsetzung der Story dauern wird. Diese Abschätzung wird zumeist auf Grundlage abstrakter Maßeinheiten (eingebürgert haben sich sog. „Gummibärchen“) gegeben, da sich die Dauer schwer in Stunden abschätzen lässt und hinter dem abstrakten Maß für die Entwickler die Möglichkeit besteht, durch Erfahrung hinzuzulernen und sich die Semantik leichter anpassen lässt.

Da Story Cards auch in diesem Experiment eine wesentliche Rolle spielen, können weitere Aspekte zu ihnen in Kapitel 4 und in Kapitel 6 gefunden werden.

2.2 Scrum

Scrum ([ASRW02], [SB02] und [TN86]) wurde zuerst 1986 präsentiert als ein adaptiver, schneller, selbstorganisierender Software-Produktentwicklungsprozess. Dieser empirische Ansatz wendet die Ideen industrieller Prozesskontrolltheorie in der Systementwicklung an. Daraus resultierte ein Ansatz, der die Ideen von Flexibilität und Anpassbarkeit in breiter Anwendung wiedereinführte. Im Gegensatz zu XP werden hier nicht spezifische Softwareentwicklungstechniken definiert, sondern es wird sich darauf konzentriert zu klären, wie die

Teammitglieder vorgehen sollten, um flexibel ein System in einer sich ständig ändernden Umgebung (beispielsweise Stakeholder Anforderungen, Zeit und andere Ressourcen, Teammitglieder, Methoden, Tools und Technologie) zu erstellen; Scrum adressiert also vor allem das Prozess-Rahmenwerk.

Scrum verbessert existierende Produktionspraxis für Systeme, da es unter anderem explizit das Problem der sich ändernden Umgebung adressiert. Weiterhin ist es notwendig, wie es der Fall für die meisten agilen Methoden ist, dass die Teammitglieder erfahrene Ingenieure sind und einem strikten Softwareentwicklungsprozess folgen und dass das zu entwickelnde System eine Innovation ist. Was meistens in der Literatur unerwähnt bleibt, ist, dass viele Systeme, die entwickelt werden, Weiterentwicklungen von existierenden Systemen sind und sich somit der Herausforderung gegenüber sehen, Anforderungen einer sich schon *geänderten* Umgebung zu erfüllen und nicht so sehr ändernden Anforderungen einer sich während der Entwicklung ändernden Umgebung.

Der Scrum Prozess beinhaltet drei Phasen: Pregame, Entwicklung und Postgame (siehe Abbildung 2).

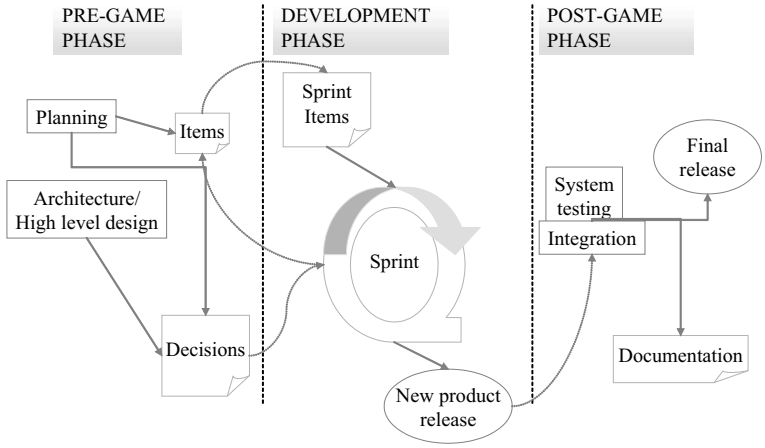

Abbildung 2: Vereinfachter Scrum Prozess.

Im folgenden geben wir einen Überblick über die einzelnen Phasen gemäß [Sch95] und [SB02].

Die *Pregame Phase* beinhaltet zwei Unterphasen: Planung und Architektur.

Planung beinhaltet die Definition des zu entwickelnden Systems. Alle bis zu diesem Zeitpunkt bekannten Anforderungen unterschiedlicher Stakeholder werden festgehalten, priorisiert und deren Umsetzungsaufwand wird geschätzt. Weiterhin beinhaltet die Planung die Definition des Projektteams, der Tools, Techniken, Ressourcen, eine Risikobewertung, Kontrollaufgaben, Trainingsbedürfnisse einzelner Teammitglieder und die Verifikation der Management Zustimmung. Zu jeder Iteration werden all diese Aspekte, wenn nötig, aktualisiert und bilden den Input für die Architektur und Entwicklungsphase.

Die *Architektur*, das abstrakte Design des Systems, wird skizziert auf Grundlage der Dinge, die während der Planung erstellt wurden. Ein Design-Reviewtreffen wird abgehalten, in dem die Vorschläge zur Implementierung geklärt werden und um Designentscheidungen zu treffen, die die Gründe für die spezifischen Lösungen darstellen, die während der Entwicklung implementiert werden.

Die *Entwicklungsphase* umfasst den agilen Teil des Scrum Prozesses. Unvorhersehbare Änderungen in Umgebungs- und technischen Variablen werden regelmäßig überprüft, um gegebenenfalls schnell und flexibel reagieren zu können. Das System wird in sogenannten *Sprints* entwickelt, die iterative Zyklen sind, in denen die Funktionalität des Systems erstellt oder erweitert wird, um neue Releases zu produzieren. Jeder Sprint enthält die traditionellen Phasen der Software Entwicklung: Anforderungen, Analyse, Design, Evolution und Auslieferung. Ein Sprint dauert in der Regel zwischen einer Woche bis ein Monat. Scrum schreibt nicht vor, wie ein Sprint im Detail auszusehen hat: Um flexibel zu sein, muss es nötig sein, unterschiedliche Entwicklungsmethoden während eines Sprints zu nutzen — von klassischen Softwareentwicklungsmethoden wie schwergewichtige Requirements Engineering Praktiken bis zu agilen Methoden wie XP.

Die *Postgame Phase* beinhaltet den Abschluss des Release. Wenn die Dinge, beispielsweise Anforderungen, die in der Planungsphase erstellt wurden, abgearbeitet sind, ist das System fertig zum Release. Weiterhin umfasst die Postgame Phase Aufgaben wie Integration, Testen des Systems und Dokumentation.

Scrum bietet ein Rahmenwerk, das eine agile oder leichtgewichtige Entwicklung von Software ermöglicht. Dabei präsentiert es einen kohärenten Überblick über die Aspekte, die nötig sind, um Flexibilität zu erreichen. Da der Ansatz leicht anzupassen ist, hat er uns wesentlich unterstützt, einen Prozess zur leichtgewichtigen Erstellung von Anforderungsspezifikationen zu erstellen.

2.3 Abstraktionsebenenmodell

Im folgenden Unterkapitel werden wir die plan-getriebene, klassische Requirements Engineering Methode adressieren, die einen wiederverwendungsorientierten Prozess unterstützt. Dabei liegt der Methode die Überlegung zugrunde, dass die meisten zu entwickelnden Systeme Weiterentwicklungen von existierenden Systemen sind und somit ein Großteil an Entwicklungsleistung gespart werden kann, wenn Teile der Spezifikationen der Vorgängersysteme auf strukturierte Art und Weise wiederverwendet werden können. Um diese Art von Wiederverwendung unterstützen zu können, wird nicht nur verlangt, dass die Struktur der Spezifikationsdokumente einen hohen Wiedererkennungswert bezüglich der verschiedenen Teile hat, sondern dass sie auch inhaltlich langlebige Beschreibungen umfasst und als solche kennzeichnet, die als solche wiederverwendet werden können. Ein Modell, das diese Anforderungen umsetzt und einen leichten, änderungsfreundlichen Umgang mit Anforderungen bereithält, ist in der folgenden Abbildung beschrieben; es lässt sich eingänglich in der Pyramidenstruktur veranschaulichen (siehe Abbildung 3).

Die Idee, die hinter diesem Spezifikationsmodell steckt, ist die Anwendung von Abstraktionsebenen. Neben der klassischerweise beschriebenen Ebene der System-Anforderungen werden weitere, abstrakte Ebenen beschrieben: So finden sich Ebenen, in der die Sicht auf das System aus der Perspektive eines Nutzers/Kunden oder aus der des Managements beschrieben wird. Weiterhin ist es denkbar, mehr Ebenen oder Erweiterungen bestimmter Ebenen zu erstellen, um weitere Stakeholder des Systems widerzuspiegeln und adäquat in der Systembeschreibung zu erfassen oder um weitere technische Details einzufügen, falls dies für die Systembeschreibung nötig ist. Die im klassischen Requirements Engineering geforderte strikte Trennung zwischen Problem- und Lösungsraum lässt sich in der Praxis kaum aufrechterhalten: Vorhandenes Wissen sowie klare Vorstellungen werden auch entsprechend beschrieben, auch wenn diese Informationen für die Phase des Requirements Engineering zu lösungsorientiert

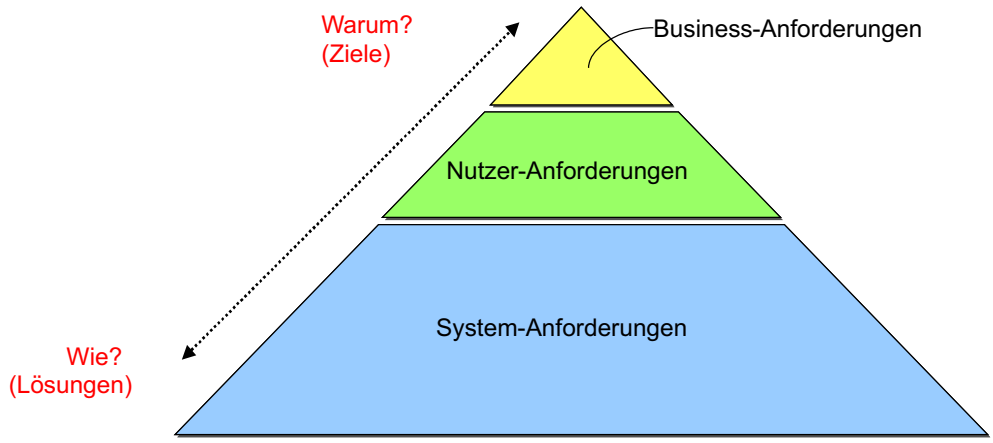

Abbildung 3: Abstraktionsebenenmodell als Pyramide.

sein sollten. Durch das hier beschriebene Abstraktionsebenenmodell wird ein Kompromiss zwischen der klassischen Forderung, zwischen Problem und Lösung klar zu trennen und dem pragmatischen Ansatz, alles was man zu einem bestimmten System weiß, auch entsprechend zu beschreiben, herbeigeführt, da das gesamte Wissen dargestellt wird, allerdings in der Abstufung von „problemorientiert“ zu „lösungsorientiert“.

Mit Hilfe dieses Spezifikationsmodells wird die Grundlage der Wiederverwendung der Anforderungen erstellt. Da sich Elemente aus der System-Anforderungs-Abstraktionsebene typischerweise rapide ändern — da sich die technologische Weiterentwicklung rasant bewegt — sind diese nicht für die Wiederverwendung geeignet. Die dahinterliegenden abstrakten Ideen oder Ziele allerdings sind üblicherweise stabiler und damit geeigneter, um wiederverwendet zu werden.

Nun geben wir eine kurze Einführung in die verschiedenen Ebenen. Für mehr Details zu diesem Thema verweisen wir auf [OTK03] und [Wie99].

Business-Anforderungen sind unterteilt in eine *Vision* und einen *Scope*. Die Ziele/Nutzen des Systems sind der *Vision* untergeordnet. Die folgenden Anforderungskategorien können als Vision erwartet werden:

- Funktionale Ziele/Nutzen aus der Perspektive eines Nutzers: Diese werden beschrieben als kurze Sätze und/oder als Einführungen zu den essentiellen funktionalen Aspekten des Systems, also die „Highlights“ des Systems.

- Nicht-Funktionale Ziele/Nutzen aus der Perspektive eines Nutzers: Diese sind essentielle nicht-funktionale Aspekte des Systems, die für die künftigen Kunden interessant sind.

- Nicht-Funktionale Ziele/Nutzen aus der Perspektive des Managements: Diese sind essentielle nicht-funktionale Aspekte des Systems, die früh zu kommunizieren sind, um elementare Geschäftsanforderungen zu erfüllen und die über die nicht-funktionalen Ziele/Nutzen aus der Perspektive der Nutzer hinausgehen.

Essentielle Restriktionen des Lösungsraums durch das Management werden im *Scope* beschrieben. Die folgende Anforderungskategorie kann als Scope erwartet werden:

- Einschränkungen, die das Management früh kommuniziert, die elementare Aspekte der Systementwicklung oder des künftigen Produktes vorschreiben.
  Diese Einschränkungen können nicht leicht von den nicht-funktionalen Zielen/Nutzen aus der Perspektive des Managements unterschieden werden, aber als Daumenregel kann man sagen, dass die Elemente der Vision einen Lösungsraum aufspannen, während Elemente des Scope Teile aus diesem Lösungsraum herausschneiden.

Wir schlagen vor, *Nutzer-Anforderungen* mit Hilfe von Features und Use Cases zu beschreiben.

- *Features* ([Heu04], [KCH+90] und [KKL+98]) beschreiben den statischen Charakter des Systems. Da wir Menschen zur gedanklichen Vergegenwärtigung eines Systems häufig an dessen Hauptcharakteristiken anstelle an ein wie auch immer geartetes „Abstraktum“ denken, ist diese Art der Beschreibung bekannt und für die verschiedenen Stakeholder leicht zu verstehen — die zumeist nicht eine gemeinsame Fachsprache sprechen.

- *Use Cases* ([Coc01], [Jac95], [JC95] und [JCJv92]) beschreiben das dynamische Verhalten des Systems und komplettieren damit die Systembeschreibung durch Features. Ihr an eine Geschichte erinnernder Aufbau unterstützt ebenso das Verständnis des Systems, da Geschichten alte kulturelle Errungenschaften sind, so dass der Mensch gut mit ihnen umgehen und Unstimmigkeiten in ihnen aufdecken kann.

Die Dokumentation und das Management der *System-Anforderungen* sind essentiell für jeglichen Requirements Engineering Prozess. Prozesse und Techniken auf dieser Abstraktionsebene unterscheiden sich je nach betrachteter Domäne, aber allen gemeinsam ist die Technikzentriertheit der System-Anforderungen. Folglich bilden sie eine vergleichsweise vollständige Anforderungsspezifikation, die sich allerdings regelmäßig zu großen Teilen von System-Anforderungen weiterentwickelter Systeme unterscheiden wird, da sie durch den rasanten Fortschritt der Technik zumeist nur eine sehr kurze Lebensdauer haben.

2.4 Statemate

In diesem Experiment haben wir uns entschieden, das Statechart-Tool Statemate MAGNUM von I-Logix zu verwenden. Diese Software wird in der Automobilindustrie häufig eingesetzt, um Spezifikationen für eingebettete Systeme zu entwickeln. Die zentrale Modellierungstechnik sind hierarchische Zustandsautomaten (Statecharts), die eine Erweiterung der einfachen Zustandsübergangsdiagramme darstellen. Statecharts wurden von David Harel, der ebenso das Tool Statemate als eine integrierte Modellierungsumgebung für Statecharts begründet hat, entwickelt. Hierarchische Zustandsautomaten bieten als Erweiterung Parallelisierung und Hierarchisierung der Zustände. Dadurch lassen sich Zustände (unter Verwendung von Parallelisierung) und Zustandsübergänge (unter Verwendung von Hierarchisierung) einsparen und somit die Modelle schlanker und eleganter darstellen. Darüber hinaus hat D. Harel eine voll ausführbare Semantik für diese Statecharts definiert, so dass sie simuliert werden können und außerdem sogar direkt C-Code generiert werden kann.

Das Tool Statemate bietet zusätzlich zum Ablauf der Statecharts, die Möglichkeit, einfache graphische Oberflächen mit Knöpfen, Scrollbars und anderen graphischen Objekten zu erstellen. Diese Interaktionselemente können mit den Statecharts verbunden werden und ermöglichen so die Erstellung einer prototypischen Benutzeroberfläche. In Verbindung mit

der Simulation bietet dies eine hervorragende Möglichkeit, mögliche Fehler in dem Modell zu finden. Der Tester kann mit den Modellen „spielen“ und die Korrektheit und Vollständigkeit der Spezifikation gegenüber den vorher erstellten Testfällen überprüfen.

In diesem Praktikum lag der Schwerpunkt auf der Modellierung der geforderten Funktionalität in Statecharts, so dass wir damit in kurzer Zeit bereits ausführbare Modelle hatten. Diese Modelle können — wie oben bereits erwähnt — als eine ausführliche Spezifikation aufgefasst werden und darüber hinaus kann diese auch interaktiv getestet werden.

Ein Grund, warum gerade dieses Tool eingesetzt wurde, war die Ausführbarkeit der Spezifikationen: Um den in XP geforderten Test-Ansatz verfolgen zu können, war es notwendig, dass die Spezifikationen ausführbar waren. Dies spiegelt zwar insoweit nicht die Praxis wider, da es sich dort vor allem um textuelle Spezifikationen handelt, dennoch entschieden wir uns dafür, diesen Aspekt mit aufzunehmen. Die Übertragbarkeit der Experimentergebnisse auf die aktuelle Praxis ist in diesem Punkt brüchig.

Weiterhin war für die Wahl von Statemate die Tatsache ausschlaggebend, dass keiner der Studenten damit vorher schon Erfahrungen hatte. Dadurch war es nicht notwendig, die unterschiedlichen Wissensstände im nachhinein zu kompensieren und der Frustration von Studenten mit weniger Erfahrung wurde vorgebeugt. Zusätzlich zu diesem didaktischen Aspekt sollte auch der experimentelle erwähnt werden: Da das Tool und die Statecharts für alle Beteiligten neu waren, war es wahrscheinlicher, dass Qualitätsunterschiede auf den Modellierungsprozess (agil oder plangetrieben) zurückzuführen waren, als auf die Diskrepanz der Vorkenntnisse. Wir erwarteten zwar technische Fehler in den Modellen von allen Gruppen, aber die entscheidende Frage für unser Experiment war: Welche Spezifikation trifft schließlich genauer die Vorstellungen des Kunden?

3 Ablauf des Experiments

Das Experiment wurde mit sieben Studenten durchgeführt, die sich in den letzten Semestern ihres Studiums befanden. Die Dauer des Praktikums umfasste zwölf Wochen. Die ersten vier Wochen wurden genutzt, um Statemate und die zu verwendenden Prozesse und Methoden (wie weiter unten beschrieben) einzuführen. Nachdem die Studenten ein Grundverständnis der Funktionalität von Statecharts erworben hatten, wurden zwei kleinere Übungsprojekte in Statemate durchgeführt: Die „Zweihandpresse“ (siehe Anhang D) und das „Innenlicht eines PKW“ (siehe Anhang E). Um die im Experiment zu verwendenden Prozesse und Methoden adäquat anwenden zu können, wurden die Studenten aufgefordert, sich mit von uns vorbereiteten Materialien zu Features, Use Cases, Story Cards und Testfällen zu beschäftigen.

Nach dieser Einführungsphase wurden die Studenten in drei Gruppen eingeteilt: Zwei Gruppen entwickelten ihre Modelle in dem leichtgewichtigen Prozess, die dritte Gruppe nutzte dafür den plan-getriebenen Prozess. Der erste Input für alle drei Gruppen war eine gekürzte Version der Spezifikation eines Türsteuergeräts [HP02] (siehe Anhang B). Da sie Fehler und Inkonsistenzen enthielt, war sie keine rundum verlässliche Spezifikation, da es die Aufgabe der Studenten war, eine verlässliche und aktuelle Spezifikation zu erstellen, indem sie das Türsteuergerät in Statemate modellierten.

Das Türsteuergerät in einem PKW ist verantwortlich für die Sitzeinstellung (inklusive des Nutzer-Managements) und für die Kontrolle des Türschlosses. Um dies tun zu können, muss es von unterschiedlichen Sensoren und Ansteuerungsknöpfen direkt oder über einen CAN Bus angesteuert werden. Eine Architektur, die im wesentlichen auf Kommunikation und

Reaktionen auf Kommunikation beruht, kann einfach durch Statecharts beschrieben werden und somit kann eine solche Architektur vor allem Gebrauch von den Stärken von Statecharts und damit vom Tool Statemate machen.

Der Ablauf des Experiments ist in Abbildung 4 dargestellt. Das Experiment besteht aus drei miteinander verbundenen Phasen. In der ersten Phase entwickelte jede Gruppe ihr Statemate Modell gemäß ihrem jeweiligen Prozess. Ein Zeitfenster von sechs Wochen war für diese erste Aufgabe geplant. Während dieser ersten Phasen wurden zwei Änderungen durch die Kunden, die in diesem Setting durch uns, den Betreuern, dargestellt wurden, verlangt. Die Darstellung der Änderungswünsche unterschied sich jedoch bezüglich der Darstellung und des Inhalts: Während die leichtgewichtigen Entwicklungsteams eine abstrakte Beschreibung in Form von Features bekommen haben, erhielt das plan-getriebene Team ein textuelles Dokument, das den Änderungswunsch sehr viel ausführlicher beschrieb. Ab diesem Zeitpunkt trafen sich die leichtgewichtigen Teams nicht mehr mit dem plan-getriebenen Team in gegenseitigen inhaltlichen Diskussionen und wurden von zwei unterschiedlichen Betreuern begleitet. Damit wollten wir zum einen eine Vermischung der Ideen vermeiden und zum anderen wollten wir herausfinden, welcher Prozess das Systemverständnis sowie die Kreativität der Entwickler besser unterstützt.

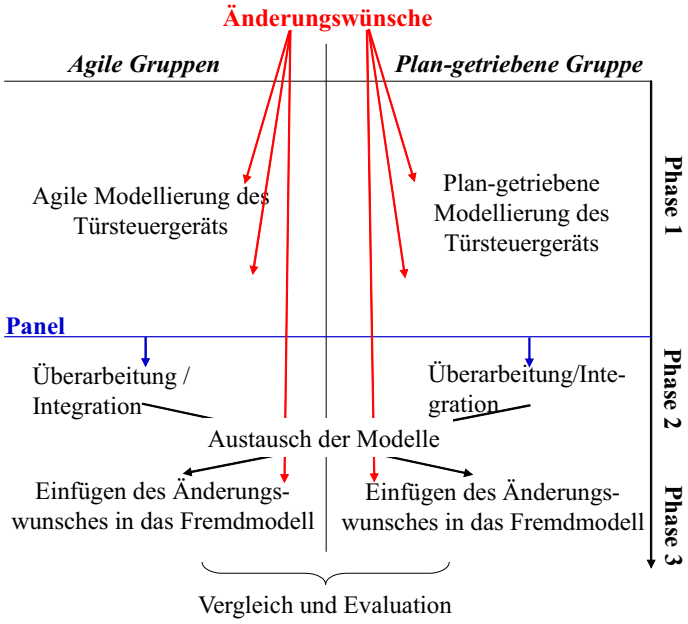

Abbildung 4: Ablauf des Experiments.

Der erste Änderungswunsch war, ein Innenraumlicht zu modellieren, das in Abhängigkeit vom Zustand der Türen (geöffnet, geschlossen) zu leuchten hatte, so wie man es von modernen PKWs kennt (siehe Anhang C.1). Diese Änderung in den Anforderungen war ein „Hinzufügen“, was bedeutet, dass sie umgesetzt werden konnte, ohne existierende Modellteile zu verändern. Im Gegensatz dazu hatte der zweite Änderungswunsch sehr wohl einen Einfluss auf bisher erstellte Teile des Modells. Darin verlangten wir zwei unterschiedliche Geschwindigkeiten für das Sitzeinstellungssystem (siehe Anhang C.2). Mit diesen Modifikationen der Anforderungen wollten wir die Änderbarkeit der Modelle bezüglich der unterschiedlichen

Prozesse untersuchen.

Zu Beginn der zweiten Phase dann wurde ein von den Betreuern erstelltes Panel (Darstellung der das System umgebenden Hardware in Statemate, die von den Statecharts angesteuert werden kann und womit das aktive System simuliert werden kann) den Studenten ausgehändigt. Nachdem sie dieses Panel in ihre Statemate Umgebung integriert hatten, testeten sie ihre Modelle und ihre Spezifikationen gegen die zuvor erstellten Testfälle. Dafür war eine Woche geplant.

Als die Modelle alle Tests erfolgreich durchliefen, wurden die Modelle der leichtgewichtigen Gruppe mit denen der plan-getriebenen Gruppe ausgetauscht. Dann war ein weiterer Änderungswunsch umzusetzen: Das Verhalten der Nutzer-Management Bedienelemente sollte modifiziert werden (siehe Anhang C.3). Dieser letzte Änderungswunsch war einfacher zu modellieren als die vorherigen, da dieser in ein den Studenten fremdes Modell innerhalb einer Woche umzusetzen war. Mit dem Tausch und dem nachfolgenden Änderungswunsch sollte die Wartbarkeit der Modelle überprüft werden.

Wie in der Einführung schon erwähnt, sollten die Studenten dann einen Fragebogen ausfüllen und ihre während des Experiments gemachten Erfahrungen beschreiben.

4 Leichtgewichtiger Entwicklungsprozess

Wie schon oben beschrieben, wurde der leichtgewichtige Prozess des Experiments, so wie er im Folgenden beschrieben wird, für die spezifischen Umstände und Voraussetzungen dieses Experiments entwickelt, wobei wir uns auf die Begrifflichkeit und die Ideen von XP und Scrum gestützt haben. Da die Studenten keine erfahrenen Ingenieure waren und wir ihnen nicht zu viele Prozessschritte (wie beispielsweise „pair programming“, „test-first“, u.v.a.) vorschreiben wollten, um ihnen einen gewissen Grad an Freiheit während der Entwicklung zu belassen, sahen wir uns einer großen Herausforderung gegenüber: Wir hatten einen leichtgewichtigen Prozess zu entwerfen, der aus elementaren agilen Elementen besteht, der Flexibilität und Änderbarkeit unterstützt und der ein System erstellen würde, das nicht nur die Kundenanforderungen erfüllen, sondern dies überdem mit verlässlicher Qualität tun würde.

Die Ziele des leichtgewichtigen Prozesses und die Vorschläge, wie diese durch agile Elemente zu realisieren seien, sind in Tabelle 1 dargestellt.

Abbildung 5 stellt den Rahmen für die agile Entwicklung dar.

In einer Art „Pregame“ Phase erstellten die Kunden (die Veranstalter des Praktikums) Ideen und Skizzen, die in den Spezifikationen verfeinert werden sollten (die Statemate Modelle). Diese Aspekte wurden in intensiver Kommunikation geklärt und diskutiert. Auf Basis der Einschätzungen der Entwickler und der Priorisierung der Kunden wurde entschieden, welche Anforderungen in dem nächsten Release umgesetzt werden sollten. Das Entwicklungsteam erstellte die Spezifikation während der Entwicklungsphase und konnten jederzeit technische Fragen mit den Betreuern klären (die neben der Rolle des Kunden auch die eines technischen Experten inne hatten). Nach einer Woche (ein Release-Cycle) präsentierte das Entwicklungsteam ihre Spezifikation, die von den Kunden evaluiert wurde. Basierend auf dieser Evaluation und weiterer Ideen, gaben sie dann Input für das nächste Release. Obwohl das Entwicklungsteam ermutigt wurde, jederzeit auch inhaltliche Fragen den Kunden zu stellen, was auch intensiv genutzt wurde (wir haben dies durch Vor-Ort-Präsenz an der Universität sowie e-mails/Telefon gewährleisten können), so durften die Kunden von sich aus nur einmal die Woche neuen oder geänderten Input geben. Die Idee, die hinter dieser Strategie steckte,

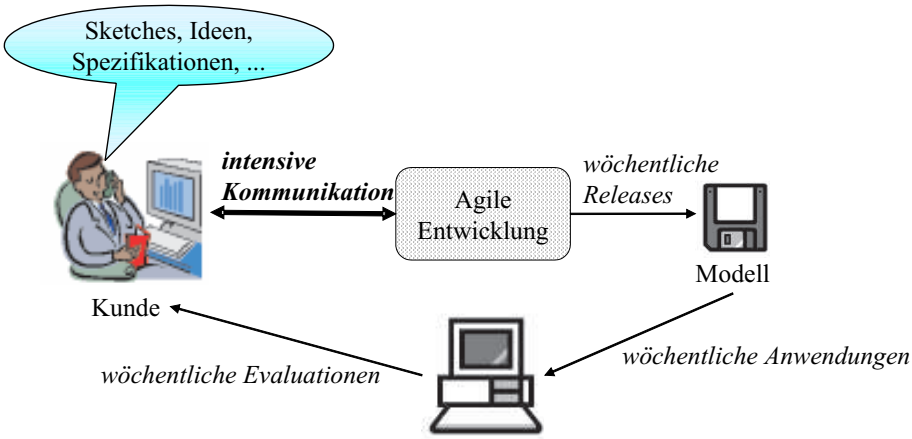

Abbildung 5: Rahmen der agilen Entwicklung.

war es, dem Entwicklungsteam zu ermöglichen, wenigstens für den Zeitraum einer Woche verlässlich planen zu können, ohne durch spezifische oder sich ändernde Anforderungen durch die Kunden unterbrochen zu werden.

Das agile Entwicklungsteam bestand aus drei Rollen (siehe Abbildung 6):

- *Vor-Ort Kunde-Proxy.* Da der „echte“ Kunde (die Betreuer des Experiments) nicht immer persönlich vor Ort sein konnten, wurde eine Rolle in das Entwicklungsteam installiert, die den Kunden repräsentieren sollte. Gab es also inhaltliche Fragen, die kurzfristig von den Kunden nicht beantwortet werden konnten, so hatte der Vor-Ort Kunde-Proxy eine Lösung zu finden. Die Hauptarbeit des Vor-Ort Kunden-Proxy aber war es, Story Cards für die Entwickler zu erstellen. Basierend auf dem textuellen Dokument und der Kommunikation mit den Kunden, erstellte der Kunde-Proxy die Nutzer Stories, um die Entwicklung bezüglich der spezifischen Priorisierung der Kunden und späterer Anforderungen anzuleiten.

- *Entwickler.* Die Entwickler erstellten die Spezifikation auf Basis der Story Cards und dem initialen textuellen Dokument. Sie schätzten die Umsetzungsdauer für jede Story Card. Diese Schätzung und die vorher erwähnte Priorisierung formte die Grundlage für die Auswahl der Story Cards für das nächste Release durch den Kunden-Proxy.

- *Tester.* Der Tester erstellte textuelle Testfälle auf der Basis der Kommunikation mit dem echten Kunden. Diese Testfälle wurden neben der Spezifikation erstellt und mussten regelmäßig während der Entwicklung ausgeführt werden. Im Falle eines unklaren Ergebnisses eines Testfalles, musste der Tester mit dem Kunden zusammen entscheiden, ob der Testfall erfolgreich durchlaufen wurde oder ob die Spezifikation zu ändern war.

Alle Mitglieder des Entwicklungsteams waren verantwortlich darauf zu achten, dass die Spezifikation so gut wie möglich die Kundenanforderungen umsetzt. Durch die hohe interne Kommunikation zwischen den Teammitgliedern wurde ein gemeinsames Bild des Systems regelmäßig aktualisiert.

Ziel	Agiles Element
Schnelle Entwicklung der Spezifikationen	Jegliche Entwicklungsmühe kommt in erster Linie der Entwicklung der Spezifikation zugute — keinem anderen Artefakt.
Hohe Qualität der Testfälle	Testfälle werden während der Entwicklung erstellt, nicht danach.
Tiefes Verständnis des Systems und hohe Kundenzufriedenheit	Intensive Kommunikation zwischen dem Entwicklungsteam und den Kunden unterstützt ein gemeinsames Verständnis und klärt offene Fragen.
Kreativität des Entwicklungsteams	Ein Entwicklungsprozess, der Spaß macht und der den Entwickler für die Arbeit insgesamt verantwortlich macht und auf die Kreativität der Entwickler vertraut, fördert diese. Weiterhin wird die Kreativität durch die offene Lösungsspanne gefördert, die durch die Story Cards aufgespannt wird.
Hohe Änderbarkeit und Wartbarkeit der Spezifikationen	Inkrementelle Entwicklung der Spezifikationen und angepasste Restrukturierung (Refaktorisierung) kreiert änderungsfreundlichen Charakter der Spezifikationen.

Tabelle 1: Ziele des leichtgewichtigen Prozesses

5 Plan-getriebener Entwicklungsprozess

Nachdem wir einen Überblick über den leichtgewichtigen Entwicklungsprozess gegeben haben, arbeiten wir in diesem Kapitel den plan-getriebenen Entwicklungsprozess aus, den wir für das Experiment herangezogen haben. Ähnlich der Tabelle mit den Zielen des leichtgewichtigen Prozesses, beschreiben wir in Tabelle 2 die Ziele und die klassischen Elemente des plan-getriebenen Prozesses.

Abbildung 7 stellt schematisch die Kommunikationsaufgaben dar, die in der Umgebung des plan-getriebenen Prozesses ablaufen. Auch hier wurde wiederum die Rolle des Kunden von den Organisatoren des Praktikums übernommen. Sie gaben zunächst dem Entwicklungsteam eine rohe textuelle Beschreibung ihrer Visionen des Systems als initiales Input-Dokument. Später im Prozess wurde dieses durch Änderungswünsche verfeinert. Das Entwicklungsteam nutzte den Input, um die Visionen in funktionale Features zu kategorisieren. Dann erstellten sie einen exakten Plan, um die für die Features relevanten Modellierungsaufgaben zu ordnen. Um dies durchführen zu können, mussten auch sie den Modellierungsaufwand für die einzelnen Features abschätzen. Der Fortschritt der Entwicklung und sämtliche Design-Entscheidungen, die während der Entwicklung getroffen wurden, mussten mit den Features zusammen in einem Dokument festgehalten werden. Dieses diente sowohl als Planungsdokument als auch als Dokumentation des Verlaufs der Entwicklung. Da es auch die Ideen und Wünsche der Kunden umfasste, musste es regelmäßig nach Änderungswünschen angepasst werden, um zu gewährleisten, dass es den aktuellen Stand des Projekts zu jedem Zeitpunkt darstellte.

Das Entwicklungsteam definierte zwei Rollen, die nicht realen Personen zugeordnet wurden, um den einzelnen Charakteren einen Wechsel ihrer Rollen im Entwicklungsprozess zu

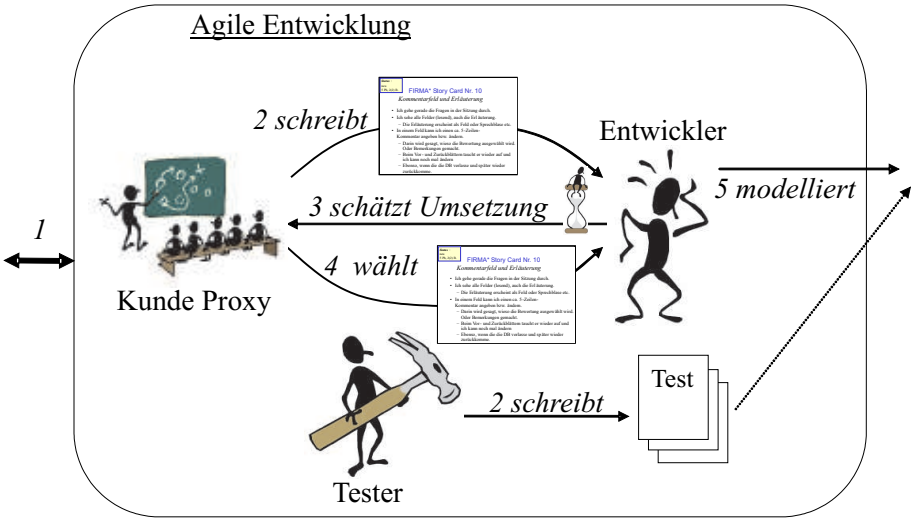

Abbildung 6: Agile Entwicklung (die Nummern an den Pfeilen geben die Reihenfolge der Entwicklungsschritte an).

vereinfachen. Die Rolle des „Entwicklers“ war verantwortlich für die Dokumentation der Features und des Fortschritts des Projekts, für die Erstellung der Use Case Diagramme, der textuellen Use Cases und der Statemate Modelle. Die Rolle des Testers war es, Testfälle aus den Use Cases abzuleiten. Da aber die Use Cases neben der Spezifikation erstellt wurden, konnten die Testfälle direkt nach der Spezifikation erstellt werden: Somit wurde auch in einem gewissen Sinne die Dokumentation der Use Cases vervollständigt (siehe Abbildung 8). Das passt auch in das Bild heutiger Praxis, in der die Erstellung der Testfälle nicht als die vornehmliche Aufgabe der Entwickler gesehen wird und diese somit oft an das Ende der Entwicklung geschoben wird.

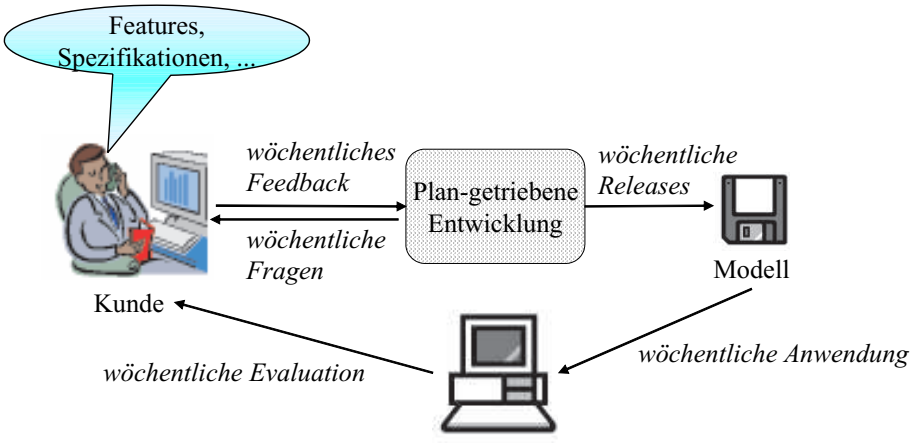

Abbildung 7: Rahmen der plan-getriebenen Entwicklung.

Der Kunde in unserem Experiment evaluierte das aktuelle Release des Systems einmal die Woche und gab Feedback in wöchentlichen Treffen. Nur während dieser Treffen durfte

Ziel	Plan-getriebenes Element
Überwachung der Modellierungszeit	Der kontinuierlich angepasste Arbeitsplan gibt den Nutzern einen Überblick über die verstrichene Zeit und somit die Möglichkeit, die verbleibende Modellierungszeit zu kontrollieren und an Kundenwünsche anzupassen.
Hohe Qualität der Testfälle	Die Testfälle werden nach der Modellierung auf der Basis der Use Cases erstellt. Damit können die Testfälle um das während der Entwicklung gewonnene Wissen angereichert werden.
Leichte Nachvollziehbarkeit der Design-Entscheidungen	Jegliche Änderungen der originalen „high-level“ Anforderungen (Features, Use Cases) werden dokumentiert und im Arbeitsplan beschrieben.
Gute Wartbarkeit und Wiederverwendbarkeit der Spezifikationen	Das Abstraktionsebenenmodell stellt eine gute Sicht auf die Abhängigkeiten zwischen unterschiedlichen Anforderungen dar und unterstützt damit das Auffinden zusammengehörender Spezifikationsteile.

Tabelle 2: Ziele des plan-getriebenen Prozesses

das Entwicklungsteam kundenzentrierte Fragen stellen, also Fragen, die nur der Auftraggeber des Systems beantworten kann. Dieser Umstand spiegelt die Realität insoweit wider, dass der Kunde normalerweise nicht immer verfügbar ist, sondern üblicherweise nur für bestimmte Zeitabschnitte. Technische Fragen allerdings zu Statemate oder Statecharts konnten natürlich dennoch jederzeit an die Betreuer gerichtet werden, um die plan-getriebene Entwicklungsgruppe ebenso wie die leichtgewichtige zu unterstützen.

Ebenso ist es notwendig zu erwähnen, dass die Iterationszyklen (eine Woche) für beide Entwicklungsgruppen gleich waren. Da Iterationen in einer plan-getriebenen Entwicklung üblicherweise länger dauern, mag dieser Umstand überraschen. Bei der Konzeption des Experiments sahen wir uns der Herausforderung gegenüber, einen in der Realität langen Iterationszyklus in den kurzen Rahmen eines Universitätssemesters zu packen. Um den nötigen Einblick in den Fortschritt der Entwicklungsarbeit bekommen zu können, haben wir uns letztlich für kurze Iterationszyklen entschieden. Dies ist insoweit zu rechtfertigen, weil nur ein vergleichsweise kleines System zu entwickeln war; dennoch wird in diesem Punkt die Realität nicht angemessen repräsentiert, weil sich da leichtgewichtige und plan-getriebene Entwicklungsprozesse stark in der Länge der Iterationszyklen unterscheiden.

Die Feature- und Use Case-orientierte Art der Dokumentation passt gut in das Abstraktionsebenenmodell, wie es in Kapitel 2.3 beschrieben ist. Die Visionen des Kunden können als die Business-Anforderungen gesehen werden. Sie sind noch vergleichsweise oberflächlich, decken aber die gesamte Umgebung ab. Diese Anforderungen werden vom Entwicklungsteam durch Use Cases und Features verfeinert, wobei die Use Cases die statische Beschreibung der Features um dynamische Aspekte vervollständigen. Die Use Cases können mit den weiter oben erwähnten Story Cards verglichen werden: sie beschreiben, in einer an Geschichten und Abläufen orientierten Art und Weise die Funktionalität des Systems. Aber anders als Story Cards sollten sich Use Cases an gewisse formale Rahmenbedingungen halten. Das Statemate

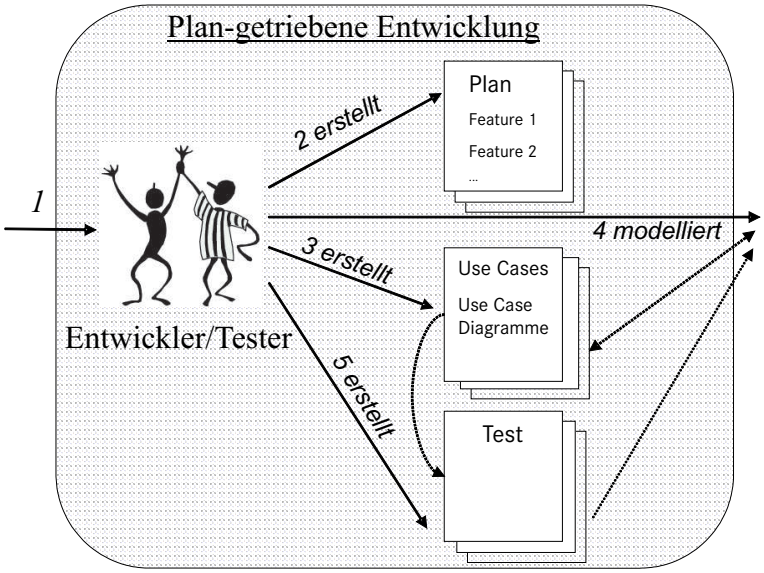

Abbildung 8: Plan-getriebene Entwicklung (die Nummern an den Pfeilen geben die Reihenfolge der Entwicklungsschritte an).

Modell repräsentiert die System-Anforderungen. Diese Modelle beschreiben die technischen Details, die für eine angemessene Implementierung der Anforderungen benötigt werden.

6 Ergebnisse

Die Ergebnisse des Experiments werden in den beiden folgenden Unterkapiteln präsentiert. Wir untersuchen die erhaltenen Ergebnisse bezüglich der Anwendbarkeit der leichtgewichtigen respektive plan-getriebenen Elemente und die tatsächliche Zielerreichung. Abschließen wollen wir mit einer kritischen Diskussion des Experiments.

6.1 Ergebnisse des leichtgewichtigen Prozesses

Die Story Cards waren eine große Unterstützung für die Entwicklungsteams. Zunächst waren sie Schlaglichter der Kommunikation zwischen Teammitgliedern. Sie beschrieben Design-Entscheidungen und stellten somit ein wichtiges Instrument für die Systementwicklung dar, da sie als einziger aktueller Input für die Entwickler anzusehen waren. Weiterhin trugen die Story Cards auf ihre Art zu der Etablierung einer eigenen Teamkultur bei: Story Cards unterschieden sich von Team zu Team und entwickelten sich bei jedem Inkrement weiter, so dass am Ende der Entwicklung jedes Team einen eigenen Stil für ihre Story Cards entwickelt hatte, der sehr stabil war. Weiterhin unterschieden sie sich beispielsweise durch unterschiedliche Arbeitsstile: Wenn die Teammitglieder weitestgehend zusammen arbeiteten, dann umfassten die Story Cards im wesentlichen sehr abstrakte Geschichten; arbeiteten die Teammitglieder eher arbeitsteilig und getrennt, so waren die Story Cards sehr detailliert mit Referenzen auf die technische Lösung.

Es erwies sich als wesentlich, dass die Person in der Rolle des Kunden-Proxy und die in der Rolle des Testers unterschiedliche waren, da in diesem Fall die Spezifikation inhaltlichen

Input über zwei Wege bekam: Zu Beginn der Spezifikation wurde der Input durch den Kunden-Proxy geliefert, am Ende wurde die Spezifikation über den Tester überprüft. Diese Trennung der Rollen sorgte für eine stringentere Überprüfung.

Die Testfälle erfüllten einen erstaunlich hohen Standard an Qualität, Vollständigkeit und Aktualität. Somit waren sie tatsächlich die aktuelle Anforderungsspezifikation für die Modell(spezifikation) — so wie es von XP gefordert wird. Dies könnte daran liegen, dass das leichtgewichtige Entwicklungsteam nur darauf zu achten hatte, dass das Statemate Modell alle Testfälle zu erfüllen hatte. Keine weiteren Dokumente mussten aktuell gehalten werden — nicht einmal die Story Cards, die eben *nicht* eine Spezifikation darstellten. Somit konnte der größte Teil des Aufwandes in die beiden wesentlichen Dokumente (Spezifikation und Testfälle) eingebracht werden, was vermutlich zu dem betrachteten hohen Standard der Qualität geführt hat.

Es wurden viele intrinsische Verbesserungsvorschläge durch das leichtgewichtige Entwicklungsteam generiert. Damit verbunden waren intensive Diskussionen zu Verbesserungen oder systemrelevante Änderungsvorschläge, die mit den Betreuern geführt wurden. Dies legt die Vermutung nahe, dass der leichtgewichtige Prozess Kreativität, Systemverständnis und das Verständnis von Kundenbedürfnissen unterstützt. Dies mag daran liegen, dass die Anforderungen von Kunden nicht als bindende Vertragsgrundlage sondern als vorläufige Vorschläge aufgefasst werden, die durchaus zur Diskussion stehen können. Insgesamt profitiert damit der Kunde von einem innovativen System, das die wesentlichen Anforderungen erfüllt; das Entwicklungsteam auf der anderen Seite fühlt sich respektiert und fühlt sich zufrieden mit den erarbeiteten Ergebnissen, die es maßgeblich mitgestalten konnte.

6.2 Ergebnisse des plan-getriebenen Prozesses

Die Erwartungen, die wir in die Features und Use Cases gesetzt haben, wurden erfüllt. Die Studenten nahmen das Konzept der Use Cases dankbar auf und erschlossen sich mit ihrer Hilfe ein tiefes Systemverständnis. Use Cases beschreiben das Systemverhalten in einer Szenario-ähnlichen Weise, sind aber exakt genug, um daraus detaillierte Funktionalität zu extrahieren. Auf der einen Seite waren die Alternativen und Ausnahmen, die in den Use Cases benannt wurden, hilfreich, um die angemessenen Testfälle zu erstellen. Auf der anderen Seite konnten diese Testfälle inhaltlich eben nur die schon herausgefundenen Use Cases abdecken: Wenn etwas in den Use Cases fehlen würde, so würde es höchstwahrscheinlich auch in den Testfällen der Fall sein.

Weiterhin gibt es einige Schwierigkeiten, die man zu überwinden hat, wenn man mit Use Cases arbeitet: In der Welt der eingebetteten Systeme ist es nicht einfach, den richtigen Aktor zu finden. In diesem Experiment zum Beispiel, stellte der CAN-Bus einen Aktor dar, der allerdings nicht in das klassische Verständnis eines Aktors passt. Damit können wir als Ergebnis festhalten, dass hier gewisser Einarbeitungsbedarf besteht, um Use Cases auch sinnvoll für eingebettete Systeme anwenden zu können.

Das System insgesamt wurde unterteilt in 15 Features, die wiederum in vier Featuregruppen gruppiert wurden. Um die funktionale Struktur der Modelle zu beachten, beinhalteten sie zumeist funktionale Aspekte. Die Interaktion zwischen den Features wurde in 13 Use Cases beschrieben. Da also nicht-funktionale Aspekte kaum eine Rolle spielten, verstanden die Studenten nicht ohne weiteres die Unterschiede zwischen Use Cases und Features.

Weiterhin unterschied sich die Aufwandsschätzung stark von der in Realität benötigten Zeit. Dies lässt sich dadurch erklären, dass die Studierenden kaum Erfahrungen mit der Mo-

dellierung hatten. Zu Beginn waren auch die Schätzungen der leichtgewichtigen Entwicklungsteams realitätsfern, aber sie hatten die Möglichkeit ihre Schätzungen von Woche zu Woche anzupassen. Im Gegensatz dazu, hatte die plan-getriebene Entwicklungsgruppe zu Beginn der Entwicklung den vollständigen Projektplan auszuarbeiten. Ähnlich verhielt es sich mit der Testphase, die nicht durchgeführt wurde, bis die Modellierung beendet war, so dass sie letztlich zu kurz kam. Da sie nicht jede Woche zu testen hatten, unterschätzte die Gruppe den Aufwand, den das Testen mit sich bringt. Dies kann auch häufig in aktuellen Software Projekten beobachtet werden.

Des weiteren gab es Probleme mit der initialen Erstellung des Modells. Da nur wenig Refaktorisierung durchgeführt wurde, konnten sich ändernde Anforderungen nicht adäquat im Modell abgebildet werden. Dies führte zu allerhand hausgemachten Problemen, die erst im Projektverlauf entdeckt werden konnten. Dies hätte allerdings vermieden werden können, wenn die Studierenden mehr Erfahrung mit der Modellierung von Statemate Modellen gehabt hätten.

6.3 Diskussion

Wie wir oben schon erwähnten, hatte jeder Prozess seine Vor- und Nachteile. Die spezifischen Ziele eines Projekts im Blick, muss man regelmäßig entscheiden, welcher Prozess das jeweilige Projekt am besten unterstützt.

Plan-getriebene Ansätze werden momentan weithin genutzt, um Spezifikationen zu entwickeln. Dies liegt zumeist daran, dass Requirements Engineering Methoden — die normalerweise mit Planungsmethoden gekoppelt sind — als erstes benötigt werden, wenn die Größe eines Systems jenseits einer gewissen Zeit- und Ressourcenskala liegt. Kleinere Systeme werden häufig ohne Prozessunterstützung oder mit Hilfe etablierter agiler Prozesse erstellt. Die seit einiger Zeit aufkommende Diskussion zu „agilen Requirements Engineering“ gibt allerdings Hinweise darauf, dass leichtgewichtige Prozesse auch interessant sind für eine klassisch „schwergewichtige“ Phase.

Die Vorteile der leichtgewichtigen Methode, die sich auch auf die Spezifikationsphase übertragen lassen, sind gebunden an eine Kunde – Entwickler Beziehung. In Domänen, wo Innovationen entwickelt werden, die nicht beauftragt werden oder wo die Spezifikationen besondere technische Anforderungen erfüllen müssen, passt eine solche Vorgehensweise nicht. Hier wird mehr Forschung benötigt, um die Anwendbarkeit und Grenzen leichtgewichtiger Prozesse zu klären. Eine Kombination der offensichtlichen Stärken leichtgewichtiger Prozesse und plan-getriebener Elemente könnte den Gap zwischen den Zielen „schneller am Markt“ und „verbesserter Qualität“ verringern.

A Übersicht über die Anhänge

Im Anhang sind einige der von uns den Studierenden zur Verfügung gestellten Dokumente zusammengefasst:

1. Die gekürzte Fassung der Türsteuergerät Spezifikation [HP02], so wie sie im Experiment verwendet wurde

2. Die von uns eingebrachten Änderungswünsche:

   (a) Steuerung eines Innenlichts integrieren (siehe auch Anhang E)

(b) Zwei Geschwindigkeiten bei der Sitzeinstellung

(c) Änderung der Bedienung des Benutzermanagements

3. *Systemspezifikation einer Zweihandpresse*
   Dieses Spezifikation wurde zu Beginn des Semesters zum Erlernen der Modellierung mit Statecharts den Studierenden ausgeteilt.

4. *Systemspezifikation einer Innenlichtsteuerung für einen PKW*
   Auch diese Systembeschreibung wurde zu Beginn des Semesters zum Erlernen der Modellierung mit Statecharts den Studierenden ausgeteilt. Die darin beschriebene Funktionalität wurde teilweise in den Änderungswünschen wieder verwendet.

B Gekürzte Fassung der Beschreibung des Türsteuergerätes

siehe folgende Seiten

Universität Ulm
Fakultät für Informatik

Praktikum im Hauptstudium

Experimentelles Software Engineering

Sommersemester 2003

Dietmar Ernst, Ramin Tavakoli Kolagari, Alexander Raschke

Systembeschreibung
Türsteuergerät

Überarbeitete Version vom 18. Januar 2004

Eine gekürzte Fassung der Fallstudie:

Frank Houdek und Barbara Paech, Das Türsteuergerät – eine Beispielspezifikation, Technischer Bericht, IESE-Report Nr. 002.02/D, Fraunhofer Institut Experimentelles Software Engineering (IESE), 2002.

URL http://www.iese.fhg.de/pdf_files/iese-002_02.pdf.

1 Einleitung

Dieses Lastenheft enthält festgelegte Anforderungen an die unter Abschnitt 2 beschriebene Komponente. Es ist die verbindliche Vorgabe zur Entwicklung dieser Komponente.

Alle Abweichungen von den Anforderungen dieses Lastenhefts bedürfen der Schriftform. Sind für die einwandfreie und uneingeschränkte Funktion erforderliche Randbedingungen in diesem Lastenheft nicht oder abweichend definiert, so ist dies dem Auftraggeber anzuzeigen. Sind dem Entwicklungslieferanten qualitäts- oder zuverlässigkeitserhöhende oder kostensenkende Alternativen bekannt, sind diese dem Auftraggeber anzuzeigen.

In Abschnitt 2 findet sich zuerst eine Übersicht über die einzelnen Funktionalitäten, die das Türsteuergerät (TSG) zur Verfügung stellt. Abschnitt 3 beschreibt das TSG aus Komponentensicht, d.h. Anforderungen an den Einbau des TSG sowie dessen Schnittstellen. Der Abschnitt 4 beschreibt detailliert die Kommunikation des TSG über den CAN-Bus. In Abschnitt 5 werden die einzelnen Funktionalitäten des TSG mit ihren jeweiligen Randbedingungen und speziellen Anforderungen erläutert. Abschnitt **Fehler! Verweisquelle konnte nicht gefunden werden.** enthält zusätzliche Richtlinien für die Durchführung der Fallstudie.

Hinweise zur Notation

Binärzahlen werden durch Unterstreichen dargestellt, z.B. 01101.

Hexadezimalzahlen werden (wie auch in C üblich), mit vorangestelltem 0x dargestellt, z.B. 0x22f.

2 Überblick

In diesem Abschnitt wird ein kurzer Überblick über die Funktionalitäten und Eigenschaften des TSG aus Benutzersicht gegeben. Eine detaillierte Beschreibung der einzelnen Funktionalitäten findet sich in Abschnitt 5. In Zweifelsfällen gelten die Beschreibungen in Abschnitt 5.

2.1 Absatzmarkt

Der Einsatz der beschriebenen Komponente ist für die Baureihe STAL 390 (Coupé, 2 Türen) geplant. Die Markteinführung ist für das 3. Quartal 2003 geplant. Die erwarteten Stückzahlen betragen ca. 20.000 Einheiten pro Jahr.

2.2 Kurzbeschreibung der Komponente

Die in diesem Lastenheft beschriebene Komponente wird als *Türsteuergerät* bezeichnet (kurz: *TSG*). Gegenstand der Modellierung ist das TSG der linken Fahrzeugseite. Dieses TSG kommuniziert mit einem TSG auf der rechten Fahrzeugseite. Das TSG wird in einem Fahrzeug mit zwei Türen verwendet.

Das TSG übernimmt folgende Funktionen im Fahrzeug:

Sitzeinstellung

Verstellen des Lehnenwinkels, der horizontalen Sitzposition, der Höhe des vorderen Sitzbereichs, der Höhe des hinteren Sitzbereichs und der Schalung des Sitzes.

Benutzermanagement

Benutzerspezifisches Abspeichern der Sitzposition.

Türschloss

Auf- und Zuschließen des Fahrzeugs über Schlüssel, Funksender oder CAN.

2.3 Periphere Komponenten

In Abbildung 1 ist das TSG zusammen mit seinen peripheren Komponenten in einer schematischen Zeichnung dargestellt.

Die hellgrau unterlegten Teile der Abbildung beschreiben die Funktionalitäten des TSG. Die weiß unterlegten Elemente bezeichnen diejenigen Fahrzeug-Komponenten, mit denen das TSG direkt interagiert (über Sensoren und Aktuatoren, die direkt mit dem TSG verbunden werden). Neben diesen direkten Ein- und Ausgabeeinheiten gibt es eine Reihe externer Einheiten, mit denen das TSG über den CAN-Bus kommuniziert (z.B. Kombigerät oder Telematiksystem).

Im Folgenden werden die in Abbildung 1 angedeuteten drei Funktionalitäten des TSG in informeller Weise aus Benutzersicht näher beschrieben.

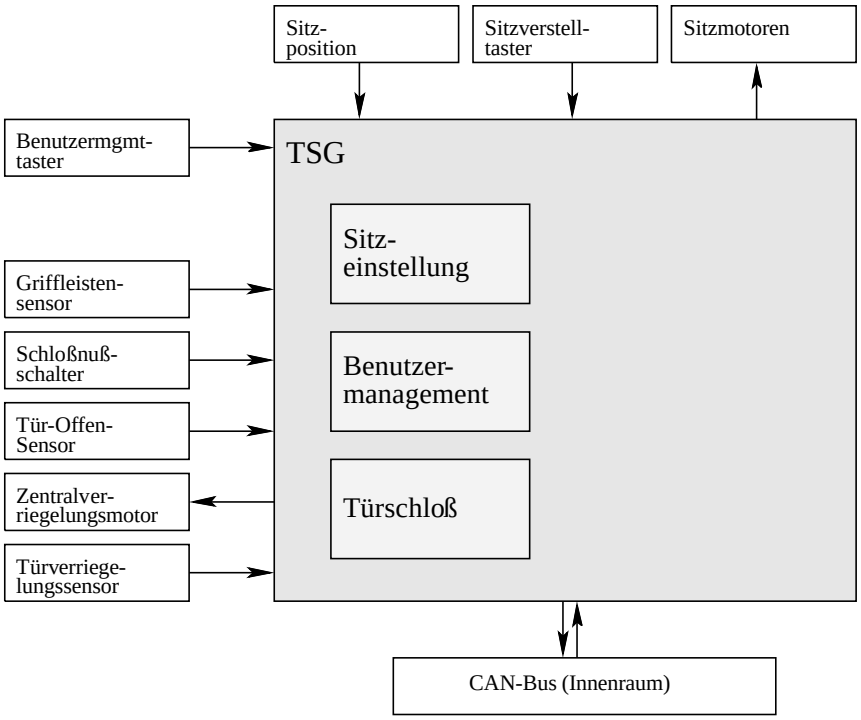

Abbildung 1: Schematische Darstellung des TSG mit seinen peripheren Komponenten

2.4 Sitzeinstellung

Vor dem Antritt einer Fahrt kann der Benutzer den Sitz gemäß seinen Anforderungen einstellen. Er hat dabei die Möglichkeit:

- den Winkel, in dem die Sitzlehne steht,
- die Entfernung des Sitzes vom Lenkrad,
- die Höhe des hinteren Sitzbereichs,
- die Höhe des vorderen Sitzbereichs und
- die Schalung des Sitzes

zu verstellen. Die Einstellung der Sitzpositionen ist nur bei geöffneter Tür möglich. Die näheren Einzelheiten sind in Abschnitt 5.2 beschrieben.

2.5 Benutzermanagement

Ohne Benutzermanagement muss jeder Fahrer Sitzposition vor jedem Fahrtantritt überprüfen und ggf. neu einstellen. Das Benutzermanagement nimmt dem Fahrer diese Aufgaben ab, indem es diese Einstellungen speichert und bei Wiederbenutzung des Fahrzeugs durch denselben Fahrer dessen vorherige Einstellungen wiederherstellt.

Dazu stehen den Benutzern eine Speicherungstaste und vier Zifferntasten zur Verfügung, die vier benutzerdefinierten Einstellungen entsprechen. Die Einstellungen der Plätze eins und zwei werden abgerufen, wenn (a) der Benutzer die entsprechende Zifferntaste drückt oder (b) der Funksender mit der Benutzerkennung eins bzw. zwei verwendet wird. Die Einstellungen der Plätze drei und vier können nur über die entsprechenden Zifferntasten abgerufen werden. Zur Speicherung der ggf. geänderten Einstellungen muss der Benutzer die Speicherungstaste drücken, gedrückt halten, und anschließend die Zifferntaste des entsprechenden Speicherplatzes betätigen. Das Benutzermanagement wird in Abschnitt 5.3 näher erläutert.

2.6 Türschloss

Zum Auf- und Abschließen des Wagens von außen kann der Benutzer wahlweise das Türschloss in der Fahrer- bzw. Beifahrer-Tür verwenden oder aber einen Funksender einsetzen. Die Verriegelung der zwei Türen wird daraufhin gemeinsam geöffnet bzw. geschlossen. Die Öffnung und Verriegelung des Kofferraums wird vom Kofferraum-Steuergerät übernommen.

Das Öffnen der Türen von innen ist immer möglich, unabhängig davon, ob die Zentralverriegelung offen oder geschlossen ist (mechanisches Öffnen). Wird eine Fahrzeugtür von innen geöffnet, werden alle Türen entriegelt.

In Abschnitt 5.4 findet sich die ausführliche Beschreibung der Türschlosssteuerung.

3 Komponentenbeschreibung

3.1 Schnittstellen

Das TSG wird über drei Steckleisten mit seiner Umgebung verbunden.

- Stecker S1: 24-polige Steckerleiste. Über diesen Stecker werden alle Sensoren und Aktoren innerhalb der Tür angeschlossen.
- Stecker S2: 49-polige Steckerleiste. Über diesen Stecker werden alle Sensoren und Aktoren außerhalb der Tür, d.h. im Fahrzeuginnenraum (z.B. Sitztaster), im Fahrzeugrahmen und in der zugehörigen Fondtür angeschlossen.
- Stecker S3: 8-polige Steckerleiste. Über diesen Stecker erfolgt die Stromversorgung sowie die Ankopplung an den Innenraumbus (CAN-Bus).

Nachfolgend werden die einzelnen Schnittstellen detailliert beschrieben.

3.1.1 Stecker S1: Türelemente

Die Kontaktierung ist durchgängig mit versilberten 0.63 mm Kontakten im Macro Tripplelock System auszuführen. Abbildung 2 zeigt das Steckerbild. Tabelle 1 beschreibt die Belegung des Steckers im Überblick.

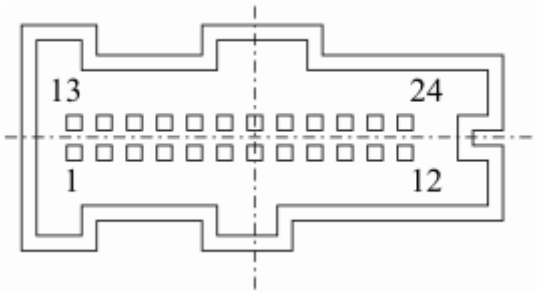

Abbildung 2: Steckerbild S1

Anschluss	Bezeichnung	In / Out	Beschreibung
1	MASSE		Signalmasse
7	MGMT_1	In	Benutzermanagement-Taster *Einstellung 1*
8	MGMT_2	In	Benutzermanagement-Taster *Einstellung 2*
9	MGMT_3	In	Benutzermanagement-Taster *Einstellung 3*
10	MGMT_4	In	Benutzermanagement-Taster *Einstellung 4*
11	MGMT_SET	In	Benutzermanagement-Taster *Speichern*
12	T_OFFEN	In	Fahrer-Tür offen
13	T_GRIFF	In	Fahrer-Griffleiste angehoben
18	T_RIEGEL	In	Fahrer-Tür verriegelt
19	KEY_STATE	In	Status Türschloss
20	ZENTR_MOT1	Out	Schließung Fahrer-Tür
21	ZENTR_MOT2	Out	Schließung Fahrer-Tür
24	V_CC		Betriebsspannung 12 V

Tabelle 1: Steckerbelegung S1

Nachfolgend werden für die einzelnen Signale deren Charakteristiken detailliert beschrieben.

Benutzermanagement-Taster

Anschlüsse: 7 (MGMT_1), 8 (MGMT_2), 9 (MGMT_3), 10 (MGMT_4), 11 (MGMT_SET)

Charakteristik:

Nicht entprellter Taster gegen Masse (siehe Abbildung 3).

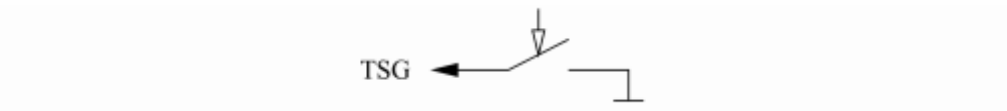

Abbildung 3: Anschlusscharakteristik Benutzermanagement-Taster

Sensor-Eingänge Türgriff, Türverriegelung

Anschlüsse: 12 (T_OFFEN), 13 (T_GRIFF), 18 (T_RIEGEL)

Charakteristik:

Nicht entprellter Mikroschalter gegen Masse (analog Abbildung 3).

Schlossnussschalter

Anschlüsse: 19 (KEY_STATE)

Charakteristik:

Über ein Widerstandsnetzwerk werden die verschiedenen Tasterstellungen codiert (siehe Abbildung 4).

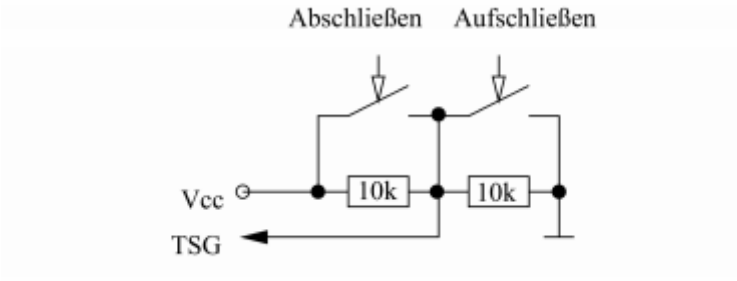

Abbildung 4: Anschlusscharakteristik Schlossnuss

Schließmotor Zentralverriegelung

Anschlüsse: 20 (ZENTR_MOT1), 21 (ZENTR_MOT2)

Charakteristik:

+12 V für 2 sec. ± 0.3 sec. verriegeln die Tür.

-12 V für 2 sec ± 0.3 sec. entriegeln die Tür.

Die Leistungsaufnahme des Schließmotors beträgt 18 W.

3.1.2 Stecker S2: Externe Elemente

Die Kontaktierung ist durchgängig mit versilberten 0.63 mm Kontakten im Macro Tripplelock System auszuführen. Abbildung 5 zeigt das Steckerbild.

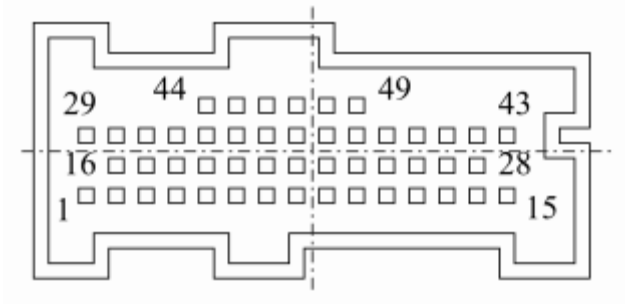

Abbildung 5: Steckerbild S2

Tabelle 2 beschreibt die Belegung des Steckers im Überblick.

Anschluss	Bezeichnung	In/Out	Beschreibung
1	MASSE		Signalmasse
4	SITZ_HOR	In	Sitztaster *Sitz vor/zurück*
5	SITZ_V	In	Sitztaster *Sitzfläche vorne heben/senken*
6	SITZ_H	In	Sitztaster *Sitzfläche hinten heben/senken*
7	SITZ_S	In	Sitztaster *Schalung enger/weiter*
8	SITZ_W	In	Sitztaster *Lehnenwinkel steiler/flacher*
9	SPOS_HOR	In	Sitzposition Ist-Wert *Sitz vor/zurück*
10	SPOS_V	In	Sitzposition Ist-Wert *Sitzfläche vorne heben/senken*
11	SPOS_H	In	Sitzposition Ist-Wert *Sitzfläche hinten heben/senken*
12	SPOS_S	In	Sitzposition Ist-Wert *Schalung enger/weiter*
13	SPOS_W	In	Sitzposition Ist-Wert *Lehnenwinkel steiler/flacher*
32	SMOT_HOR1	Out	Ansteuerung Sitzmotor + *Sitz vor/zurück*
33	SMOT_HOR2	Out	Ansteuerung Sitzmotor - *Sitz vor/zurück*
34	SMOT_V1	Out	Ansteuerung Sitzmotor + *Sitzfläche vorne heben/senken*
35	SMOT_V2	Out	Ansteuerung Sitzmotor - *Sitzfläche vorne heben/senken*
36	SMOT_H1	Out	Ansteuerung Sitzmotor+ *Sitzfläche hinten heben/ senken*
37	SMOT_H2	Out	Ansteuerung Sitzmotor - *Sitzfläche hinten heben/senken*
38	SMOT_S1	Out	Ansteuerung Sitzmotor + *Schalung enger/weiter*

39	SMOT_S2	Out	Ansteuerung Sitzmotor - *Schalung enger/weiter*
40	SMOT_W1	Out	Ansteuerung Sitzmotor + *Lehnenwinkel steiler/ flacher*
41	SMOT_W2	Out	Ansteuerung Sitzmotor - *Lehnenwinkel steiler/ flacher*

Tabelle 2: Steckerbelegung S2

Nachfolgend werden für die einzelnen Signale deren Charakteristiken detailliert beschrieben.

Sitztaster

Anschlüsse: 4 (SITZ_HOR), 5 (SITZ_V), 6 (SITZ_H), 7 (SITZ_S), 8 (SITZ_W)

Charakteristik:

Über ein Widerstandsnetzwerk werden die verschiedenen Tasterstellungen codiert (siehe Abbildung 6).

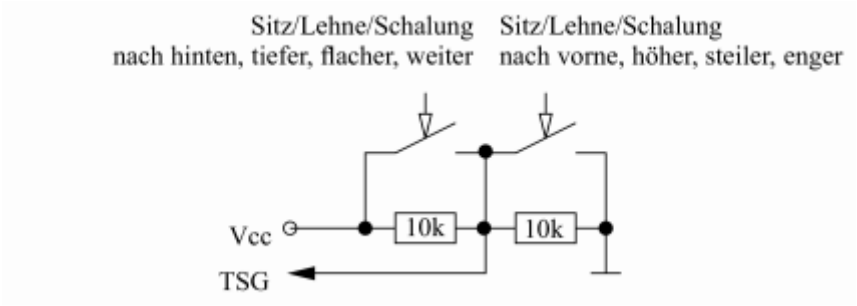

Abbildung 6: Anschlusscharakteristik Sitztaster

Sitzposition

Anschlüsse: 9 (SPOS_HOR), 10 (SPOS_V), 11 (SPOS_H), 12 (SPOS_S), 13 (SPOS_W)

Charakteristik:

Widerstandswert 1 kΩ bis 10 kΩ (Wertebereich wird i.d.R. nicht ganz ausgenutzt), bei Endanschlag Unterbrechung.

Kleinere Werte geben an, dass Sitz/Lehne/Schalung weiter vorne, höher, steiler bzw. enger sind.

Sitzmotor

Anschlüsse: 32 (SMOT_HOR1), 33 (SMOT_HOR2), 34 (SMOT_V1), 35 (SMOT_V2), 36 (SMOT_H1), 37 (SMOT_H2), 38 (SMOT_S1), 39 (SMOT_S2), 40 (SMOT_W1), 41 (SMOT_W2)

Charakteristik: siehe Tabelle 3. Die Leistungsaufnahme je Motor beträgt max. 15 W.

Sitzeigenschaft	Kürzel	Bewegung bei +12 V	Bewegung bei -12 V	Geschw.
Entfernung Sitz Lenkrad	HOR	nach vorne	nach hinten	0.9 cm/sec.
Sitzfläche vorne	V	heben	Senken	0.7 cm/sec.
Sitzfläche hinten	H	heben	Senken	0.7 cm/sec.
Schalung	**S**	heben	Senken	0.5 cm/sec.
Lehnenwinkel	W	steiler	Flacher	3.3 °/sec.

Tabelle 3: Sitzmotoransteuerung

Schließmotor Zentralverriegelung

Anschlüsse: 43 (FZENTR_MOT1), 44 (FZENTR_MOT2)

Charakteristik:

+12 V für 2 sec. ± 0.3 sec. verriegeln die Tür.

-12 V für 2 sec ± 0.3 sec. entriegeln die Tür.

Die Leistungsaufnahme des Schließmotors beträgt 18 W.

3.1.3 Stecker S3: Energie und Kommunikation

Die Kontaktierung ist durchgängig mit versilberten 0.63mm Kontakten im Macro Tripplelock System auszuführen. Abbildung 7 zeigt das Steckerbild. Tabelle 4 beschreibt die Belegung des Steckers im Überblick.

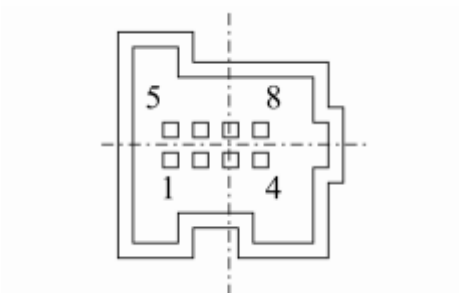

Abbildung 7: Steckerbild S3

Anschluss	Bezeichnung	In / Out	Beschreibung
1	MASSE		Signalmasse
2	V_CC		12 V Versorgungsspannung (Bordnetz)
5	CAN_B_LOW	**In / Out**	CAN-B (Innenraumbus) Low
6	CAN_B_HIGH	In / Out	CAN-B (Innenraumbus) High

Tabelle 4: Steckerbelegung S3

Nachfolgend werden für die einzelnen Signale deren Charakteristiken detailliert beschrieben.

CAN-B Bus

Anschlüsse: 5 (CAN_B_LOW), 6 (CAN_B_HIGH)

Charakteristik:

Anbindung an den Innenraumbus gemäß ISO 11519. Übertragungsrate 83.333 KBit. Anbindung über Differenzsignal.

Die Botschaften, die über den CAN-Bus übertragen werden, sind in Abschnitt 4 definiert.

3.2 Elektrische Spezifikation

Die Betriebsspannung des TSG liegt zwischen 9.0 V und 13.5 V. Die erweiterte Betriebsspannung (Ansteuern der Aktuatoren nicht notwendig) liegt zwischen 7.5 V und 16.0 V.

Im aktiven Zustand darf der Stromverbrauch ohne aktive Aktoren 500 mA nicht überschreiten. Bei aktivierten Aktoren darf der Stromverbrauch entsprechend höher sein (siehe dazu die Beschreibung der einzelnen Ausgänge).

3.3 Gehäuse

Für das Gehäuse gelten die maximalen Abmessungen von (B x H x T) 220 mm x 185 mm x 34 mm. Die in Abbildung 8 eingezeichneten Befestigungspunkte sind zu beachten. Weitere oder geänderte Befestigungspunkte sind nur nach Absprache mit dem Auftraggeber zulässig.

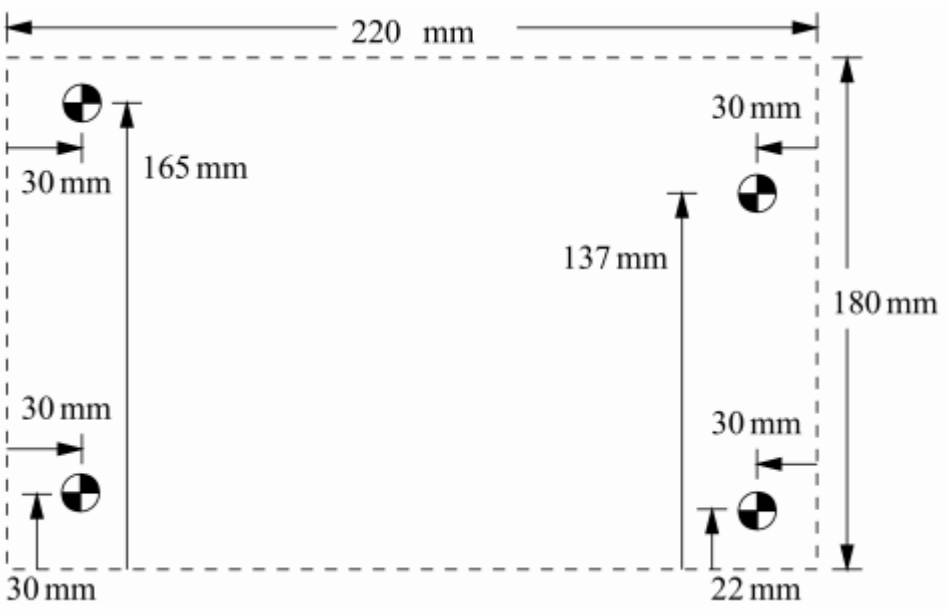

Abbildung 8: Position der Befestigungspunkte

4 CAN-Kommunikation

Nachfolgend sind alle Botschaften zusammengetragen, die das TSG über den CAN-Bus sendet oder empfängt. Die Tabelle selbst ist wie folgt aufgebaut:

- Spalte 1 (TSG_L): Sende/Empfangsstatus für das linke TSG.
  - *e:* Botschaft wird vom linken TSG empfangen
  - *s:* Botschaft wird vom linken TSG gesendet
- Spalte 2 (ID): Identifier der CAN-Nachricht (Hexadezimalwert).
- Spalte 3 (Sig.-Art): Signalart *zyklisch* oder *spontan*.
- Spalte 4 (δt): Im Falle eines zyklischen Signals die Zykluszeit in Millisekunden.
- Spalte 5 (Byte): Die Nummer des nun betrachteten Bytes in der aktuellen Botschaft.
- Spalte 6 (Bit): Die Nummer des nun betrachteten Bits innerhalb des aktuellen Bytes.
- Spalte 7 (Signal-Name): Bezeichnung des Signals.
- Spalte 8 (Signal-Funktion): Bedeutung des betrachteten Signals.
- Spalte 9 (Länge): Länge des betrachteten Signals in Bits.

TSG	ID	Sig.-Art	δt	Byte	Bit	Signal-Name	Signal-Funktion	Länge
e	0x003	zyklisch	100	0	6	ZV_SCHL_A	Zentralverriegelung 00 = Zustand beibehalten 01 = Verriegeln 10 = Entriegeln 11 = ungültig	2
s	0x004	zyklisch	100	0	6	ZV_SCHL_L	*(wie 0x003)*	2
e	0x005	zyklisch	100	0	6	ZV_SCHL_R	*(wie 0x003)*	2
e	0x020	zyklisch	100	0	0	BATT	Batteriespannung 5 V – 18 V Werte 50 - 180, 0.1 V Auflösung	8
e	0x031	zyklisch	100	1	7	MOTOR_LFT	1 = Motor läuft	1

e	0x060	zyklisch	100	1	0	TIME	Systemzeit 0-2^{32} - 1, Auflösung 1 sec., Zeit in Sekunden, die seit dem letzten Anklemmen der Batterie-spannung vergangen ist (0 sec. bis ca. 136 Jahre)	32
e	0x080	zyklisch	100	1	0	FZG_V	Fahrzeuggeschwindigkeit 0 - 1023, Auflösung 0.25 km/h, (0.0 km/h bis 255.75 km/h)	10
e	0x100	zyklisch	100	0	0 1 2 3	KL_15KEY KL_15RADIO KL_15 KL_15START	1 = Schlüssel steckt 1 = Schlüssel auf Radiost. 1 = Schlüssel auf Zündung 1 = Schlüssel auf Anlassen	1 1 1 1
s	0x22d	zyklisch	100	0 0	0 4	BLINK DURATION	Blinken auf den angegebenen Positionen x1, x2, x3, x4 x1: vorne Links (1 = Blinker an) x2: vorne Rechts (1 =Bl. an) x3: hinten Links (1 = Bl. an) x4: hinten Rechts (1 =Bl. an) Blinkdauer in 10 ms 1-255 (10 ms bis 2550 ms)	4 8
e	0x28c	zyklisch	500	0	0 1 2 3 4 5 6 7	M_POS_1 M_POS_2 M_POS_3 M_POS_4 M_SAVE_1 M_SAVE_2 M_SAVE_3 M_SAVE_4	1 = Memorypos. 1 einnehmen 1 = Memorypos. 2 einnehmen 1 = Memorypos. 3 einnehmen 1 = Memorypos. 4 einnehmen 1 = Memorypos. 1 speichern 1 = Memorypos. 2 speichern 1 = Memorypos. 3 speichern 1 = Memorypos. 4 speichern	1 1 1 1 1 1 1 1
s	0x28d	zyklisch	500	0	0 1 2 3 4 5 6 7	M_POS_1 M_POS_2 M_POS_3 M_POS_4 M_SAVE_1 M_SAVE_2 M_SAVE_3 M_SAVE_4	*(wie 0x28c)* *(wie 0x28c)* *(wie 0x28c)* *(wie 0x28c)* *(wie 0x28c)* *(wie 0x28c)* *(wie 0x28c)* *(wie 0x28c)*	1 1 1 1 1 1 1 1
s	0x6fe	spontan		0	0 1 2	B_LOW_SITZ B_LOW_KEY ERROR_KEY	Batterie für Sitzbewegung zu gering Batterie für Schließung zu gering Fehler beim Schließvorgang	1 1 1
e	0x6ff	spontan		0	0 1 2	B_LOW_SITZ B_LOW_KEY ERROR_KEY	Batterie für Sitzbewegung zu gering Batterie für Schließung zu gering Fehler beim Schließvorgang	1 1 1
e	0x710	zyklisch	200	0	0	DOOR_ STATE	Zustand der Türen x1, x2 x1: vorne Links (1 = Tür offen) x2: vorne Rechts (1 = Tür offen)	2
s	0x780	zyklisch	200	0	0	F_T_OFFEN	Zustand der Türen Fahrerseite x1 x1: Fahrertür (1 = Tür offen)	1

Tabelle 5: Verzeichnis aller TSG relevanter Kommunikationssignale

Erläuterung:

- Die Zuordnung einer Botschaft auf zwei TSGs (Sende/Empfangsstatus x) dient der Einsparung von CAN-Identifiern. Andernfalls hätten immer zwei Botschaften vorgehalten werden müssen (wie z.B. bei 0x004 und 0x005), was mehr Realisierungs- und Kommunikationsaufwand zur Folge hätte.
- Botschaft 0x100:
  - Die Signale sind kumulativ, d.h. neben der aktuellen Zündungsstufe sind auch immer alle niedrigeren Zündungsstufen mit aktiviert.
- Botschaft 0x22d:
  - Das eigentliche Blinken wird von entsprechenden Aktuatoren (Blinker, Kombiinstrument) eigenverantwortlich durchgeführt. Sobald eine Botschaft mit BLINK ≠ 0000 empfangen wird, erfolgt ein Blinkzyklus mit 54/100*DURATION Leuchtzeit und 46/100*DURATION Dunkelzeit.
  - Weitere Botschaften, die während Leucht- oder Dunkelzeit eingehen, werden ignoriert.
  - Erst nach Ablauf eines Blinkzyklus werden weitere Blinkbotschaften bearbeitet.
- Botschaft 0x28c:
  - Die Signale M_POS_1 und M_POS_2 werden gesendet, wenn der Benutzer einen Funksender verwendet.
  - Die anderen Botschaften sind für den Einsatz eines Fahrerassistenz-Systems vorgehalten (momentan nicht verwendet).
- Botschaften 0x6fe und 0x6ff:
  - Diese Botschaften werden vom Kombiinstrument ausgewertet; im Fehlerfall kann eine entsprechende Warngrafik im Kombi-Display angezeigt werden.

5 Funktionen

In den nachfolgenden Abschnitten finden sich die detaillierten Anforderungen an die Systemfunktionen des TSG. Für jede Systemfunktion findet sich eine Beschreibung der Systemein- und -ausgänge, die Beschreibung des Verhaltens sowie Angaben zum Initialisierungsverhalten. Die Systemein- und -ausgänge sind wie folgt gekennzeichnet:

- S1.*x:* Eingang von Stecker S1 (siehe Abschnitt 3.1.1)
- S2.*x:* Eingang von Stecker S2 (siehe Abschnitt 3.1.2)
- S3.*x:* Eingang von Stecker S3 (siehe Abschnitt 3.1.3)
- CAN.*x:* CAN-Signal (siehe Abschnitt 4)

5.1 Allgemeines Verhalten

Eingänge

- Zustand Fahrertür:
  S1.T_OFFEN
- Zyklische CAN-Botschaften:
  CAN.KL_15KEY, CAN.KL_15RADIO, CAN.KL_15, CAN.KL_15START, CAN.BATT, CAN.FCODE_T0, CAN.FCODE_T1, CAN.ZV_SCHL_R

Ausgänge

Zyklisch gesendete CAN-Botschaften

- Zustand Zentralverriegelung:
  CAN.ZV_SCHL_L
- Ansteuerung Blinklicht:
  CAN.BLINK, CAN.DURATION
- Benutzermanagement:
  CAN.M_POS_1, CAN.M_POS_2, CAN.M_POS_3, CAN.M_POS_4, CAN.M_SAVE_1, CAN.M_SAVE_2, CAN.M_SAVE_3, CAN.M_SAVE_4
- Zustand der dem TSG zugeordneten Fahrzeugtüren:
  CAN.F_T_OFFEN

Verhalten

Solange das TSG aktiv ist, werden zyklisch die angegebenen CAN-Botschaften gesendet. Die Zykluszeiten finden sich im Abschnitt 4. Die Werte der Signale sind gemäß Tabelle 6 zu ermitteln.

Signal	*Wert bzw. Quelle*
ZV_SCHL_L	*Linkes TSG:* Solange die Tür nicht ver- oder entriegelt wird, wird 00 gesendet (siehe Abschnitt 5.4)
BLINK DURATION	Solange keine Tür ver- oder entriegelt wird wird BLINK=0000 und DURATION=0 gesendet (siehe Abschnitt 5.4)
M_POS_1 M_POS_2 M_POS_3 M_POS_4 M_SAVE_1 M_SAVE_2 M_SAVE_3 M_SAVE_4	normalerweise 0 senden (Ausnahme: siehe Abschnitt 5.3)
F_T_OFFEN	Zustand der Fahrzeugtüren auf Fahrerseite, Werte x1, wobei x1 = 1, wenn Fahrertür offen

Tabelle 6: Werte für zyklisch erzeugte Signale

5.2 Sitzeinstellung

Eingänge

- Zustand der Vordertür:
  S1.T_OFFEN
- Sitzbedienungstasten:
  S2.SITZ_HOR, S2.SITZ_V, S2.SITZ_H, S2.SITZ_S, S2.SITZ_W
- Sitzposition:
  S2.SPOS_HOR, S2.SPOS_V, S2.SPOS_H, S2.SPOS_S, S2.SPOS_W
- Batteriespannung:
  CAN.BATT
- Fahrzeuggeschwindigkeit:
  CAN.FZG_V

Intern gibt es eine Verbindung zum Benutzermanagement.

Ausgänge

- Sitzmotoren:
  S2.SMOT_HOR1, S2.SMOT_HOR2, S2.SMOT_V1, S2.SMOT_V2, S2.SMOT_H1, S2.SMOT_H2, S2.SMOT_S1, S2.SMOT_S2, S2.SMOT_W1, S2.SMOT_W2

- Warnmeldung:
  CAN.B_LOW_SITZ

Verhalten

Generell:

Ein Verstellen der Sitzposition über die Sitztaster ist nur möglich, wenn die dem TSG zugeordnete Vordertür geöffnet ist (S1.T_OFFEN = 1). Das Verstellen der Sitzposition über das Benutzermanagement ist auch bei geschlossener Tür möglich.

Die Sitzposition wird entweder entsprechend der vom Benutzermanagement gesendeten Positionsangaben oder den Sitztasten eingestellt. Dabei gilt das Prinzip, dass immer die zuletzt benutzte Taste (Benutzermanagement oder Sitztaste) die Bewegung des Sitzes bestimmt.

Bewegung des Sitzes:

Zur Bewegung des Sitzes werden die in Tabelle 3 angegebenen Spannungen auf die Sitzmotoren gelegt. Die Bewegung wird solange durchgeführt wie:

- Ist-Wert und Soll-Wert nicht übereinstimmen (bei Anfahren einer Sitzposition über das Benutzermanagement) bzw. die Sitztasten gedrückt werden (bei Verstellen der Sitzposition über die Sitztasten)
- und der Wert der Sitzposition keine Unterbrechung erkennt.

Bewegung über Sitztasten:

Bei der Verwendung der Sitztasten können maximal zwei Bewegungsrichtungen gleichzeitig verwendet werden. Wird während der Sitzverstellung über die Sitztasten eine Taste des Benutzermanagements gedrückt (siehe Abschnitt 5.3), so wird die Sitzverstellung über Tasten abgebrochen und die Sitzverstellung über das Benutzermanagement begonnen.

Ist die Batteriespannung BATT während der Sitzverstellung kleiner als 10 V, so werden die Sitze nicht bewegt bzw. wird die Sitzbewegung abgebrochen. Statt dessen wird die Meldung B_LOW_SITZ = 1 generiert.

Bewegung über Benutzermanagement:

Die Sitzverstellung über das Benutzermanagement ist nur möglich, solange die Fahrzeuggeschwindigkeit (FZG_V) kleiner als 5 km/h ist. Überschreitet die Fahrzeuggeschwindigkeit 5 km/h, so wird die Sitzbewegung sofort abgebrochen.

Es sind zwei Fälle zu unterscheiden:

- *Fall 1:* (Auswahl einer Einstellung über Benutzermanagementtaste)
  - In diesem Fall ist anzunehmen, dass der Fahrer bereits auf dem Fahrersitz sitzt. Um die Bewegung so angenehm wie möglich zu gestalten, sind folgende Regeln bei der Ansteuerung der Sitzposition zu beachten:
    - Zuerst werden die Bewegungen durchgeführt, die eine Entspannung der Sitzposition zur Folge haben, d.h. das Vergrößern der Entfernung Sitz-Lenkrad, das Flacher-Stellen des Lehnenwinkels, das Absenken der Sitzfläche (vorne und hinten) sowie das Öffnen der Schalung.
    - Anschließend werden die entgegengesetzten Bewegungen durchgeführt.
    - Es dürfen zu einer Zeit maximal zwei Richtungen gleichzeitig bewegt werden. Dabei gilt die Reihenfolge Entfernung Sitz-Lenkrad, Lehnenwinkel, Schalung, Sitzfläche vorne, Sitzfläche hinten.
  - Ist die Batteriespannung BATT zu Beginn der Sitzverstellung kleiner als 10 V, so werden die Sitze nicht bewegt. Statt dessen wird die Meldung B_LOW_SITZ = 1 generiert.
- *Fall 2:* (Auswahl einer Einstellung über Funksender)

- In diesem Fall soll die gewünschte Sitzposition so schnell wie möglich eingenommen werden. Dazu werden alle Sitzmotoren gleichzeitig angesteuert.
- Ist die Batteriespannung BATT zu Beginn der Sitzverstellung kleiner als 10 V, so werden die Sitze nicht bewegt. Statt dessen wird die Meldung B_LOW_SITZ = 1 generiert.

Initialisierung

Keine Aktion

5.3 Benutzermanagement

Eingänge

- Position des dem TSG zugeordneten Sitzes:
  S2.SPOS_HOR; S2.SPOS_H; S2.SPOS_V; S2.SPOS_S; S2.SPOS_W
- Tasten des Benutzermanagements:
  S1.MGMT_1; S1.MGMT_2; S1.MGMT_3; S1.MGMT_4; S1.MGMT_SET
- CAN-Botschaften zum Abrufen einer Speicherposition (Fahrer-TSG nur Botschaft 0x28c):
  CAN.M_POS_1; CAN.M_POS_2; CAN.M_POS_3; CAN.M_POS_4
- CAN-Botschaften zum Speichern einer Speicherposition (Fahrer-TSG nur Botschaft 0x28c):
  CAN.M_SAVE_1; CAN.M_SAVE_2; CAN.M_SAVE_3; CAN.M_SAVE_4

Ausgänge

- CAN-Botschaften zum Abrufen einer Speicherposition (nur Fahrer-TSG, Botschaft 0x28d):
  CAN.M_POS_1; CAN.M_POS_2; CAN.M_POS_3; CAN.M_POS_4
- CAN-Botschaften zum Speichern einer Speicherposition (nur Fahrer-TSG, Botschaft 0x28d):
  CAN.M_SAVE_1; CAN.M_SAVE_2; CAN.M_SAVE_3; CAN.M_SAVE_4

Intern gibt es Verbindungen zur Sitzeinstellung.

Verhalten

Generell:

Verwendung der Tasten des Benutzermanagements

Betätigt der Fahrer eine der Tasten MGMT_1, MGMT_2, MGMT_3 oder MGMT_4, so wird die Einstellung, die unter dieser Speicherposition abgelegt ist, abgerufen.

Dazu wird

- die CAN-Botschaft M_POS_1, M_POS_2, M_POS_3 oder M_POS_4 (0x28d) generiert (je nach gedrückter Taste), und
- die Sitzeinstellung angestoßen, wobei als Soll-Werte die Werte genommen werden, die unter der Speicherposition abgelegt sind, die der gedrückten Taste entsprechen.

Wurden unter der jeweiligen Speicherposition noch keine Einstellungen abgespeichert, so entfällt das Ansteuern des Sitzes.

Abspeicherung mittels der Tasten des Benutzermanagements:

Betätigt der Fahrer die Taste MGMT_SET und gleichzeitig eine der Tasten MGMT_1, MGMT_2, MGMT_3 oder MGMT_4, so werden die aktuellen Einstellungen unter der zugehörigen Speicherposition abgelegt.

Dazu wird

- die CAN-Botschaft M_SAVE_1, M_SAVE_2, M_SAVE_3 oder M_SAVE_4 (0x28d) generiert (je nach gedrückter Taste), und

- die Werte der Sitzeinstellung SPOS_HOR, SPOS_H, SPOS_V, SPOS_S und SPOS_W unter der entsprechenden Speicherposition abgelegt.

Abruf einer Einstellung über CAN:

Empfängt ein TSG die Botschaft M_POS_1, M_POS_2, M_POS_3 oder M_POS_4, so wird die Einstellungen für den Sitz entsprechend der gespeicherten Werte angestoßen. Finden sich unter den jeweiligen Speicherposition keine Einstellung, so entfällt das Ansteuern des Sitzes.

Speichern einer Einstellung über CAN:

Empfängt ein TSG die Botschaft M_SAVE_1, M_SAVE_2, M_SAVE_3 oder M_SAVE_4, so speichert das TSG die Werte der Sitzeinstellung in der entsprechenden Speicherposition.

Initialisierung

Keine Aktion

5.4 Türschloss

Eingänge

- Zustand Vordertür und Türschloss:
  S1.KEY_STATE; S1.T_RIEGEL; S1.T_OFFEN
- Schließbefehle über CAN:
  CAN.ZV_SCHL_A; CAN.ZV_SCHL_R
- Zündung:
  CAN.KL_15START
- Motor läuft:
  CAN.MOTOR_LFT
- Batteriespannung:
  CAN.BATT

Ausgänge

- Schließmotoren:
  S1.ZENTR_MOT1; .ZENTR_MOT2
- Schließbefehle über CAN:
  CAN.ZV_SCHL_L
- Ansteuerung Fahrzeugblinker:
  CAN.BLINK; CAN.DURATION
- Ansteuerung Fensterheber:
  CAN.WIN_VL_CL, CAN.WIN_VR_CL
- Fehlermeldungen:
  CAN.ERROR_KEY, CAN.B_LOW_KEY

Verhalten

Generell:

Die Funktion soll dem Prinzip folgen, dass entweder alle Fahrzeugtüren verriegelt oder alle Fahrzeugtüren entriegelt sind. Wird eine einzelne Tür ver- bzw. entriegelt, so müssen alle anderen Türen diesem Vorbild folgen. Im Zweifelsfalle hat Entriegeln Vorrang vor Verriegeln (z.B. wenn ein Fahrgast die Tür von innen öffnet [= entriegeln], während der Fahrer die Türen mittels Funksender verriegeln möchte). Solange wenigstens eine Fahrzeugtür offen ist, kann das Fahrzeug nicht verriegelt werden.

Aufschließen des Fahrzeugs:

Die Fahrzeugtüren werden entriegelt, wenn die Fahrzeugtüren verriegelt sind und entweder:

- mindestens eine Tür (mechanisch) geöffnet wird (d.h. T_OFFEN= 1)
- oder das Signal zum Öffnen der Türen erkannt wird (ZV_SCHL_A= 10 oder ZV_SCHL_R= 10)
- oder der Schlüssel zum Aufschließen der Tür verwendet wird (KEY_STATE= Aufschließen)

Um die Fahrzeugtüren entriegeln zu können, muss die Batteriespannung BATT mindestens 9 V betragen. Unterhalb von 9 V arbeiten die Motoren der Zentralverriegelung nicht mehr zuverlässig. Ein selbständiges Nach-Entriegeln der Türen zu einem späteren Zeitpunkt erfolgt i.d.R. nicht.

Ausnahme: Wird zu dem Zeitpunkt, an dem auf Entriegeln erkannt wird, die Entriegelung nicht durchgeführt, weil die Batteriespannung unter 9 V liegt und wird gleichzeitig ein Anlassversuch unternommen (KL_15START = 1), so wird nach einer Wartezeit von 1 sec. nachdem KL_15START = 0 oder MOTOR_LFT = 1 gilt nochmals selbständig geprüft, ob die Batteriespannung nun ausreichend ist. Falls ja, wird die Tür nach-entriegelt. Kann keine Nach-Entriegelung durchgeführt werden, wird die Nachricht B_LOW_KEY = 1 gesendet.

Beim Entriegeln wird das Signal ZV_SCHL_L = 10 generiert und an die Ausgänge ZENTR_MOT1 und ZENTR_MOT2 wird jeweils die Spannung -12 V für einen Zeitraum von mindestens 2 sec. angelegt. Sobald nach dieser Zeit das Entriegeln einer Tür erkannt wird (T_RIEGEL = 0), wird der entsprechende Motor abgestellt.

Wird innerhalb von 3 sec. nicht das Entriegeln der Tür erkannt, so wird der Entriegelungsvorgang abgebrochen, es wird die CAN-Botschaft ERROR_KEY generiert.

Nach erfolgter Entriegelung (oder wenn das Fahrzeug bereits entriegelt war) werden vom linken TSG, soweit nicht das rechte TSG innerhalb von 1 sec. die Fehler-Botschaft ERROR_KEY sendet, die Signale BLINK = 1111 und DURATION = 100 gesendet. Trat ein Fehler beim Verriegeln auf, so wird nicht geblinkt.

Abschließen des Fahrzeugs:

Die Fahrzeugtüren werden verriegelt, wenn die Türen des Fahrzeugs entriegelt sind, alle Fahrzeugtüren geschlossen sind und entweder:

- das Signal zum Abschließen der Türen erkannt wird (ZV_SCHL_A = 01 oder ZV_SCHL_R = 01)
- oder der Schlüssel zum Abschließen der Tür verwendet wird (KEY_STATE = Abschließen)

Um die Fahrzeugtüren verriegeln zu können, muss die Batteriespannung BATT mindestens 9 V betragen. Unterhalb von 9 V arbeiten die Motoren der Zentralverriegelung nicht mehr zuverlässig. Ein selbständiges Nach-Verriegeln der Türen zu einem späteren Zeitpunkt erfolgt nicht. Kann keine Verriegelung durchgeführt werden, wird die Nachricht B_LOW_KEY = 1 gesendet.

Beim Verriegeln wird das Signal ZV_SCHL_L = 01 generiert und an die Ausgänge ZENTR_MOT1 und ZENTR_MOT2 wird jeweils die Spannung +12 V für einen Zeitraum von mind. 2 sec. angelegt. Sobald nach dieser Zeit das Verriegeln einer Tür erkannt wird (T_RIEGEL = 1), wird der entsprechende Motor abgestellt.

Wird innerhalb von 3 sec. nicht das Verriegeln der Tür erkannt, so wird der Verriegelungsvorgang abgebrochen, es wird die CAN-Botschaft ERROR_KEY generiert.

Nach erfolgter Verriegelung (oder wenn das Fahrzeug ohnehin schon verriegelt war) werden vom linken TSG, soweit nicht das rechte TSG innerhalb von 1 sec. die Fehler-Botschaft ERROR_KEY sendet, die Signale BLINK = 1111 und DURATION = 50 zweimal hintereinander im Abstand von δt = 1000 ms gesendet. Trat ein Fehler beim Verriegeln auf, so wird nicht geblinkt.

Automatisches Schließen der Fenster:

Beim Verriegeln werden die Signale WIN_VL_CL, WIN_VR_CL mit dem Wert 10 (vollständiges Schließen) gesendet, um die Fenster automatisch zu schließen.

Unplausible Sensorwerte:

Wird festgestellt, dass eine Tür offen ist (T_OFFEN = 1), wobei gleichzeitig die Tür verriegelt ist (T_RIEGEL = 1), so gilt:

- die Türen werden entriegelt (s.o.)
- und es wird das Signal zum Türöffnen gesendet (ZV_SCHL_L = 10)

Initialisierung

Bei der Initialisierung des TSG wird geprüft, ob der Zustand der Türen konsistent ist. Wenn nein, dann werden alle Türen geöffnet (s.o.).

C Change Requests

C.1 Change Request I

C.1.1 ... an die klassische Gruppe

Es geht um das Innenlicht des Fahrzeugs. Die tatsächliche Hardware wird von einem (fiktiven) Lichtsteuergerät (LSG) gesteuert, das somit nicht Thema des Change Requests ist, aber es bekommt eben Informationen vom TSG:

- Beim Öffnen der Tür mit Schlüssel geht das Licht an.
- Beim Schließen der Tür geht das Licht nach 10 s aus.
- Bei verschlossener Tür und angehobener Griffleiste geht das Licht für 10 s an.
- Dies kann maximal 3 mal erfolgen, danach muss die Tür zuerst ent- und wieder verriegelt werden.
- Beim Abschließen der Tür geht das Licht sofort aus
- Das TSG regelt bereits das korrekte Verhalten zwischen Beifahrer und Fahrertür, d.h. das LSG soll und darf sich nicht mehr um den Türzustand kümmern, es wertet lediglich die Information des untenstehenden CAN-Bus-Signals aus.

Für die Kommunikation zwischen TSG und LSG wird ein weiteres CAN-Bus-Signal benötigt:

```
0x800 | zyklisch | 100 | 0 | 0 | ILICHT_TUER | 1 = Licht_an | 1
```

C.1.2 ... an die agilen Gruppen

Es geht um das Innenlicht des Fahrzeugs. Die tatsächliche Hardware wird von einem (fiktiven) Lichtsteuergerät (LSG) gesteuert, das somit nicht Thema des Change Requests ist, aber es bekommt eben Informationen vom TSG. Das Verhalten des Innenlichts soll dabei einerseits möglichst kundenfreundlich sein, andererseits aber auch nicht zu viel Batterieenergie benötigen. Für die Kommunikation zwischen TSG und LSG wird ein weiteres CAN-Bus-Signal benötigt:

```
0x800 | zyklisch | 100 | 0 | 0 | ILICHT_TUER | 1 = Licht_an | 1
```

Dieses neue Feature wird dabei direkt nach dem Feature Tür Entriegeln/Verriegeln eingeordnet. Damit sieht die aktualisierte Featureliste folgendermaßen aus:

- Tür Entriegeln/Verriegeln
- Innenlicht
- Entfernung Sitz/Lenkrad
- Lehnenwinkel
- Zentralverriegelung
- Sitzhöhe (vorne/hinten)
- Sitzschale
- Benutzermanagement
- Benutzermanagement über Funkschlüssel

Ergänzung zu der Spezifikation:
Werden mehr als zwei Tasten für die Sitzeinstellung gleichzeitig (!) gedrückt, dann erfolgt *keine* Motoransteuerung.

C.2 Change Request II

Da die Geschwindigkeiten der Sitzeinstellung zu langsam sind, soll in Zukunft, bei längerem Drücken der entspr. Taste (2s) die Geschwindigkeit verdoppelt werden. Dies wird realisiert, indem der entspr. Motor zunächst mit ±6V angesteuert wird. Nach 2s wird er mit ±12V aktiviert. Fällt die Batteriespannung unter 10V, werden die Sitze (wieder) langsamer, also mit 6V bewegt. Fällt sie unter 5V, werden die Sitze nicht mehr bewegt. Es wird die Meldung `B_LOW_SITZ = 1` generiert.

Der Wertebereich der Widerstände wird erhöht auf 1kOhm bis 30kOhm. Sie werden ganzzahlig gemessen und jeder Wert entspricht einem Schritt gemäß untenstehender Tabelle.

Sitzeigenschaft	Kürzel	bei +6V	bei -6V	Geschw.
Entfernung Sitz Lenkrad	HOR	nach vorne	nach hinten	0.9 cm/sec.
Sitzfläche vorne	V	heben	Senken	0.7 cm/sec.
Sitzfläche hinten	H	heben	Senken	0.7 cm/sec.
Schalung	S	heben	Senken	0.5 cm/sec.
Lehnenwinkel	W	steiler	Flacher	3.3 /sec.

Sitzeigenschaft	Kürzel	bei +12V	bei -12V	Geschw.
Entfernung Sitz Lenkrad	HOR	nach vorne	nach hinten	1.8 cm/sec.
Sitzfläche vorne	V	heben	Senken	1.4 cm/sec.
Sitzfläche hinten	H	heben	Senken	1.4 cm/sec.
Schalung	S	heben	Senken	1.0 cm/sec.
Lehnenwinkel	W	steiler	Flacher	6.6 /sec.

C.3 Change Request III

Der „`MGMT_SET`“-Button fällt weg. Stattdessen werden die Sitzpositionen auf den einzelnen Speicherknöpfen durch einen längeren Druck ($\geq$ 2s) gespeichert. Dafür ist es natürlich nötig, die Benutzermanagement-Taster (`MGMT_x`) als „Conditions“ zu betrachten und die entsprechende Aktion erst beim Loslassen des Knopfes zu veranlassen.

D Spezifikation und Systembeschreibung „Zweihandpresse“

D.1 Einleitung

Eine Zweihandpresse ist eine Maschine, mit der Metallteile einem hohen Druck ausgesetzt werden können, um diese zu verformen. Um die Sicherheit des Benutzers zu gewährleisten, muß sichergestellt werden, daß dieser während des Pressvorganges nicht seine Hände in die Presse bekommen kann. Dies wird dadurch bewerkstelligt, daß der Benutzer die Presse nur aktivieren kann, wenn er die beiden Taster, die beidseitig neben der Presse angebracht sind, innerhalb einer kurzen Zeitspanne betätigt und während des Pressvorganges gedrückt hält.

D.2 Beschreibung der Zweihandpresse

Im folgenden sind die Anforderungen an das Steuersystem der Zweihandpresse formuliert.

- Das System befindet sich im Ruhezustand, wenn der Presskolben in oberer Stellung, der Motor der Presse ausgeschaltet und keiner der beiden Taster gedrückt ist.
- Um die Presse zu starten, müssen beide Taster innerhalb von einer Sekunde gedrückt werden. Wenn zwischen den Tastendrücken eine größere Zeitspanne liegt, muß das System sich erst wieder im Ruhezustand befinden, bevor ein neuer Startversuch unternommen werden kann.
- Sobald beide Startknöpfe gedrückt sind, setzt sich der Motor in Bewegung und bewegt den Kolben nach unten.
- Falls ein Knopf losgelassen wird, muß innerhalb von 0,5 Sekunden der Motor anhalten und anschließend wieder zur Ausgangsstellung zurückfahren.
- Nachdem der Presskolben eine bestimmte Position (den sog. *Point of no Return*) erreicht hat, wird der Pressauftrag auch ausgeführt, falls die Knöpfe losgelassen werden. Sobald der Kolben diese Position überschritten hat, ist es nicht mehr möglich, die Hand zwischen Kolben und Werkstück zu legen.
- Sobald der Kolben die untere Position erreicht hat, fährt der Motor den Kolben automatisch in die Ausgangsstellung zurück.
- Erreicht der Kolben nicht innerhalb von 5 Sekunden nachdem der *Point of no Return* überschritten wurde die untere Position (z. B. weil des Werkstück zu groß ist), so fährt der Motor den Kolben ebenfalls wieder in die Ausgansstellung zurück. In diesem Falle wird zudem mit einer Lampe dem Benutzer signalisiert, daß ein Fehler vorliegt. Die Lampe geht wieder aus, wenn das System in den Ruhezustand zurückkehrt.
- Bevor die Presse erneut gestartet werden kann, muß sich die Presse zuerst wieder im Ruhezustand befunden haben.

D.3 Beschreibung der externen Beschaltung

Die Zweihandpresse kommuniziert mit ihrer Umwelt mit mehreren Sensoren und Aktuatoren. Diese sind im folgenden näher beschrieben.

- *Taster*. Das System besitzt zwei Taster, die über die zwei Variablen `LEFT_PRESSED_` und `RIGHT_PRESSED_` abgefragt werden können (True, wenn Taste gedrückt).
- *Motor*. Der Motor wird mit den Befehlen `MOTOR_DOWN_`, `MOTOR_UP_` und `MOTOR_STOP_` angesteuert. Die Befehle bewirken, daß der Motor den Kolben nach unten bewegt, nach oben bewegt bzw. anhält.
- *Positionsmelder*. Der Positionsmelder meldet durch die Events `AT_TOP_` und `AT_BOTTOM_`, daß der Kolben der Presse sich ganz oben, bzw. ganz unten befindet.
- *Sensor*. Der Sensor teilt über den Event `SENSOR_INFO_` mit, wann der Kolben den *Point of no Return* erreicht hat. Die Positition des Sensors ist bei `POS` = 75.
- *Signallampe*. Die Lampe wird über die Bedingung `LAMP` gesteuert. Wird die Bedingung mit `tr!(LAMP)` auf True gesetzt, leuchtet die Lampe. Wird die Bedingung mit `fs!(LAMP)` auf False gesetzt, so geht die Lampe wieder aus.

E Spezifikation und Systembeschreibung „Innenlichtsteuerung eines PKW“

E.1 Einleitung

Moderne Kraftfahrzeuge bieten dem Benutzer eine Vielzahl von Extras, die über die reine Basisfunktionalität des Fahrens hinausgehen. So wird auch die Innenraumbeleuchtung nicht mehr einfach nur eingeschaltet, wenn die Türe aufgemacht wird, sondern z. B. ein Anheben der Griffleiste schaltet das Licht ein, so daß man im Dunkeln das Schlüsselloch leichter findet.

Die hier beschriebene Steuerung basiert auf dem Handbuch eines BMW's der 7er Baureihe und gibt die Funktionsweise der dort eingebauten Innenlichtbeleuchtung identisch wieder. Da das Handbuch in einigen Punkten nicht genau war, wurde die Spezifikation logisch ergänzt.

Abschnitt E.2 beschreibt die Bestandteile der Steuerung und ihre Funktion (Spezifikation). Außerdem findet sich in dieser Spezifikation eine Beschreibung der zur Verfügung stehenden Schnittstellen (Abschnitt E.3).

E.2 Beschreibung des Innenlichts

In Abbildung 1 sind die Bestandteile der Innenlichtanlage dargestellt.

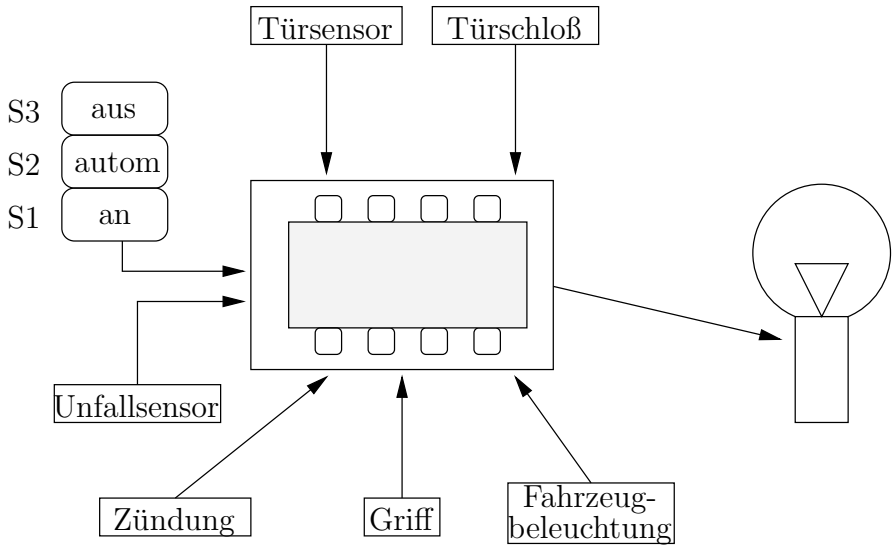

Abbildung 9: Schematische Darstellung der Innenraumbeleuchtung.

Die wesentlichen Bestandteile sind:

- Ein Sensor an jeder der Vordertüren meldet, wenn die Türe geöffnet oder geschlossen wird. Die Fondtüren bleiben unberücksichtigt und können deshalb auch das Verhalten des Innenlichts nicht beeinflussen.

- Der Unfallsensor kann einen Unfall feststellen (wie ein Airbag–Sensor). Er schickt dazu zum Zeitpunkt des Unfalls ein Signal an die Innenlicht–Steuerung.

- Ein Sensor in der Griffleiste jeder Vordertür gibt ein (gemeinsames) Signal, wenn diese angehoben wird.
- Die Zündung kann eingeschaltet oder ausgeschaltet sein.
- Schalter mit 3 Positionen: „Immer ein“, „Automatik“, „Immer aus“.
- Die Fahrzeugbeleuchtung ist entweder ein- oder ausgeschaltet.
- Mittels Zentralverriegelung kann das Fahrzeug auf- oder abgesperrt werden.
- Glühbirne, die durch die Steuerung ein- und ausgeschaltet werden kann.
- Steuerung.

Mit Hilfe des Schalters kann der Benutzer zwischen drei Betriebsarten wählen:

Stellung 1: Das Innenlicht ist immer eingeschaltet.

Stellung 2: (Mittelstellung=default:) Die Steuerung erfolgt wie unten beschrieben.

Stellung 3: Das Innenlicht ist immer ausgeschaltet. (Ausnahme: siehe unten)

Wird von Stellung 1 oder Stellung 3 nach Stellung 2 geschaltet, so ist das Licht so anzusteuern, wie der Fahrzeugstatus und dessen Historie impliziert (z.B. bewirkt ein Wechsel von Stellung 3 nach Stellung 2 2 sec nach dem Schließen beider Vordertüren ein Einschalten des Innenlichts für die verbleibenden 8 sec).

Automatische Steuerung

Wird eine Tür geöffnet, so schaltet sich das Licht ein. Das Licht ist immer eingeschaltet, solange mindestens eine Türe offen ist. Werden alle Türen geschlossen, so leuchtet die Lampe noch 10 Sekunden nach und wird danach ausgeschaltet, falls keine Komponente des Fahrzeugs ein Brennen des Innenlichts fordert oder dies vorzeitig abschaltet.

Sind alle Türen geschlossen, und ist die Zündung eingeschaltet, so brennt das Innenlicht nicht und auch ein eventuelles Nachleuchten wird vorzeitig abgebrochen. Beim Ausschalten der Zündung wird das Licht für 10 Sekunden eingeschaltet, falls auch die Fahrzeugbeleuchtung eingeschaltet ist.

Wird die Griffleiste einer abgesperrten und geschlossenen Türe (auch die Zündung ist ausgeschaltet) angehoben, so geht das Licht für 10 Sekunden an. Dies kann maximal zweimal wiederholt werden; danach wird diese Funktion bis zum nächsten Absperren der Türe deaktiviert (Sicherung gegen spielende Kinder). Wird während dieser Leuchtphase eine Griffleiste nochmals angehoben, so wird dies ignoriert.

Wird ein Unfall festgestellt, so wird die Lampe eingeschaltet, damit die Unfallhelfer bessere Sicht haben. Dies gilt auch, falls der Positionsschalter auf Stellung 3 (Immer aus) steht. Diese Sonderbetriebsart wird durch Einschalten der Zündung oder durch ein Verändern des Positionsschalters wieder abgestellt, d. h. das Innenlicht wird wieder durch die Stellung des Positionsschalters und des Fahrzeugstatus geregelt.

E.3 Die Hardware–Schnittstelle

Die Innenlichtsteuerung kommuniziert über die folgenden Ereignisse oder Bedingungen mit den Hardwarebausteinen, bzw. anderen Steuerungseinheiten wie beispielsweise die Fahrzeugbeleuchtung.

`S`x`_GEDRUECKT_E_` Event vom Innenlichtschalter zur Steuerung, wobei x für `1`, `2` oder `3` und der entsprechende Event ausgelöst wird, falls der Schalter auf die entsprechende Position geschoben wird.

`FT_GEOEFFNET_E_` Event von der Fahrertür zur Steuerung, der beim Öffnen der Fahrertür ausgelöst wird.

`BFT_GEOEFFNET_E_` Event von der Beifahrertür zur Steuerung, der beim Öffnen der Beifahrertür ausgelöst wird.

`FT_GESCHLOSSEN_E_` Event von der Fahrertür zur Steuerung, der beim Schließen der Fahrertür ausgelöst wird.

`BFT_GESCHLOSSEN_E_` Event von der Beifahrertür zur Steuerung, der beim Schließen der Beifahrertür ausgelöst wird.

`T_AUFGESPERRT_E_` Event von den Türen zur Steuerung, der beim Aufsperren einer Vordertür ausgelöst wird.

`T_ABGESPERRT_E_` Event von den Türen zur Steuerung, der beim Absperren einer Vordertür ausgelöst wird.

`Z_EINGESCHALTET_C_` Bedingung, die von der Zündung auf true bzw. false gesetzt wird, je nachdem, ob die Zündung ein- bzw. ausgeschaltet ist.

`FB_EINGESCHALTET_C_` Bedingung, die von der Lichtsteuerung auf true bzw. false gesetzt wird, je nachdem, ob die Fahrzeugbeleuchtung ein- bzw. ausgeschaltet ist.

`G_ANGEHOBEN_E_` Event zur Steuerung, der ausgelöst wird, wenn der Griff einer der vorderen Fahrzeugtüren angehoben wird.

`U_ENTDECKT_E_` Event von der Airbag–Steuerung, der eine Unfallerkennung signalisiert.

`L_EINSCHALTEN_E_` Event zum Einschalten der Innenraumbeleuchtung.

`L_AUSSCHALTEN_E_` Event zum Ausschalten der Innenraumbeleuchtung.

Nach dem Einschalten der Steuerung (Anklemmen an die Stromversorgung) sind folgende Annahmen bezüglich des Fahrzeugstatus zu machen: Die Türen sind geschlossen, und das Fahrzeug ist nicht abgesperrt. Nicht erwartete Signale in der Startphase (z.B. Event `FT_GESCHLOSSEN_E_`) sind zu ignorieren.

Da andere Fahrzeugkomponenten den Zustand des Innenlichts nicht kennen müssen, existiert auch kein von der Innenlichtsteuerung ausgehendes Signal, das beispielsweise angibt, ob das Innenlicht momentan brennt oder nicht.

Für die Simulation des Modells kann ein vorgefertigtes Panel mit zugehöriger Hardwareumgebung verwendet werden (siehe Abbildung 10. Dies ist im Projekt INNENLICHT_HW zu finden. In einem eigenen Projekt müssen also zuerst alle Elemente dieses Projektes importiert werden.

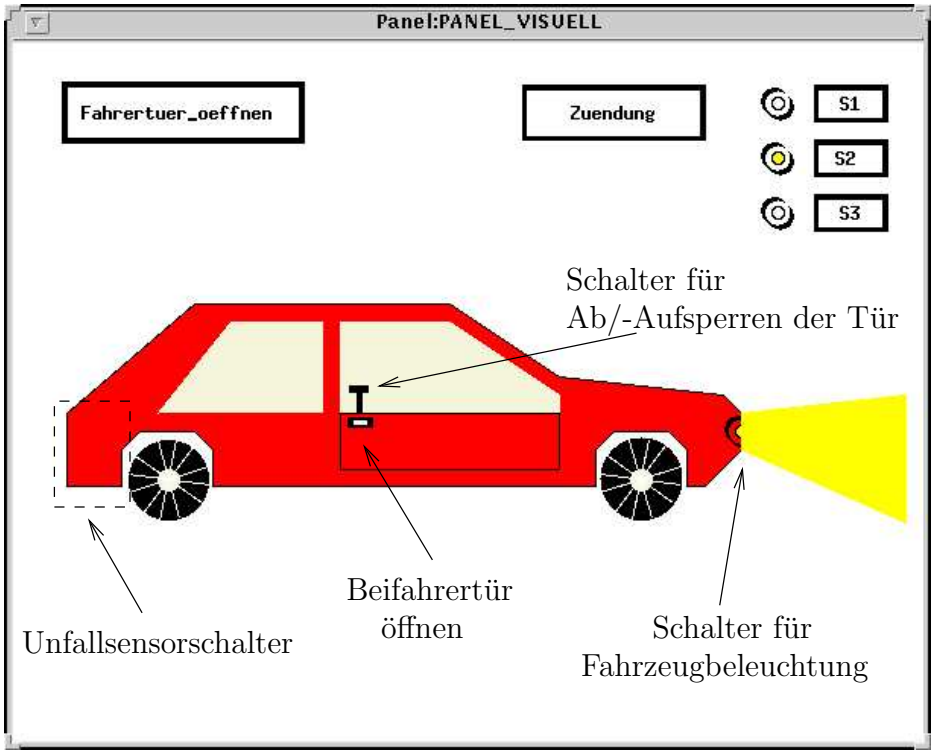

Abbildung 10: Das vorgefertige visuelle Panel für die Innenraumbeleuchtung.

Literatur

[ACL02] Acebal, C. F. und J. M. Cueva Lovelle: *A new method for Software Development: eXtreme Programming.* INFORMATIK — INFORMATIQUE, 2:5–9, 2002.

[ASRW02] Abrahamsson, P., O. Salo, J. Ronkainen und J. Warsta: *Agile Software Development Methods — Review and Analysis.* VTT Publications, 478, 2002.

[BF00] Beck, K. und M. Fowler: *Planning Extreme Programming.* Addison-Wesley, 1. Auflage, 2000.

[Coc01] Cockburn, A. (Herausgeber): *Writing Effective Use Cases.* The Crystal Collection for Software Professionals. Addison-Wesley, 1. Auflage, 2001.

[Har87] Harel, D.: *Statecharts: A Visual Formalisms for Complex Systems.* Science of Computer Programming, 8:231–274, 1987.

[Heu04] Heumesser, N. (Herausgeber): *Evolution Management and Process for Real-time Embedded Software Systems.* Framework for Requirements. EUREKA ITEA project EMPRESS, available at: http://www.empress-itea.org/, 2004. Version 3.1.

[HN96] Harel, D. und A. Naamad: *The STATEMATE Semantics of Statecharts.* ACM Transaction on Software Engineering and Methodology, 5:4:293–333, 1996.

[HP02] Houdek, F. und B. Paech: *Das Türsteuergerät — eine Beispielspezifikation.* IESE-Report 002.02/D, Fraunhofer Institut Experimentelles Software Engineering, Kaiserslautern, 2002.

[Jac95] Jacobson, I.: *Modeling with Use Cases — Formalizing Use Case Modeling.* Journal of Object-Oriented Programming (8), 3:10–14, 1995.

[JC95] Jacobson, I. und M. Christerson: *Modeling with Use Cases — A Growing Consensus on Use Cases.* Journal of Object-Oriented Programming (8), 1:15–19, 1995.

[JCJv92] Jacobson, I., M. Christerson, P. Jonsson und G. vergaard: *Object-Oriented Software Engineering — A Use Case Driven Approach.* Addison-Wesley, Workingham, England, 1992.

[KCH+90] Kang, K. C., S. G. Cohen, J. A. Hess, W. E. Novak und A. S. Peterson: *Feature-Oriented Domain Analysis (FODA) Feasibility Study.* Technical Report CMU/SEI-90-TR-21, Software Engineering Institute, Carnegie Mellon University, Pittsburgh, PA, 1990.

[KKL+98] Kang, K. C., S. Kim, J. Lee, K. Kim, E. Shin und M. Huh: *FORM: A feature-oriented reuse method with domainspecific reference architectures.* In: *Annals of Software Engineering*, Band 5, Seiten 143–168. J. C. Baltzer AG, Science Publishers, 1998.

[OTK03] Omasreiter, H. und R. Tavakoli Kolagari: *Ziel und kundenorientierte Anforderungserstellung mit Abstraktionsebenen als zentraler Erfolgsfaktor bei der Entwicklung von KFZ-Software.* In: *Proceedings of the conference CONQUEST, Automation Days, and EUROMOTIVE, ASQF*, Deutschland, 2003.

[SB02] Schwaber, K. und M. Beedle: *Agile Software Development With Scrum.* Prentice-Hall, Upper Saddle River, NJ, 2002.

[Sch95] Schwaber, K.: *Scrum Development Process.* In: *OOP-SLA '95 Workshop on Business Object Design and Implementation.* Springer-Verlag, 1995.

[Sta] *Statemate Homepage: http://www.ilogix.com/products/magnum/index.cfm.*

[TN86] Takeuchi, H. und I. Nonaka: *The New Product Development Game.* Technischer Bericht, Harvard Business Review, Jan./Feb. 1986.

[Wie99] Wiegers, K.: *Software Requirements.* Microsoft Press, 1999.

[WW03] Weber, M. und J. Weisbrod: *Requirements Engineering in Automotive Development: Experiences and Challenges.* IEEE Software, Jan./Feb. 2003.