FAKULTÄT FÜR
INFORMATIK

Thema:

Verringerung des redundanten Softwareentwicklungsaufwandes für portable Systeme

Diplomarbeit

AG Wirtschaftsinformatik - Managementinformationssysteme

Themensteller: Prof. Dr. Hans-Knud Arndt
Betreuer: Prof. Dr. Hans-Knud Arndt

vorgelegt von: Dirk Aporius

Abgabetermin: 07. Oktober 2009

Inhaltsverzeichnis

Verzeichnis der Abkürzungen und Akronyme

API	Application Programming Interface
CDC	Connected Device Configuration
CLDC	Connected Limited Device Configuration
EAST-ADL	Embedded Architecture and Software Tools Architeture Description Language
ER-Diagramm	Entity-Relationship Diagramm
FODA	Feature-Oriented Domain Analysis
Fople	Feature-Oriented Product-Line Engineering
Form	Feature-Oriented Reuse Method
Java EE	Java Enterprise Edition
Java ME	Java Micro Edition
Java SE	Java Standard Edition
JCP	Java Community Process
JSR	Java Specification Request
JVM	Java Virtual Machine
KobrA	Komponentenbasierte Anwendungsentwicklung
MAP	Mining Architectures for Productlines
MIDP	Mobile Information Device Profile
MIME	Multipurpose Internet Mail Extension
OAR	Options Analysis for Reengineering
PASTA	Process and Artifact State Transition Abstraction
PuLSE	Product Line Software Engineering
SEI	Software Engineering Institute
UML	Unifined Modeling Language

Tabellenverzeichnis

Abbildungsverzeichnis

1 Einleitung

1.1 Motivation

Auf der ganzen Welt besitzen mehrere Hundert Millionen Menschen ein Java-fähiges Mobilfunkgerät, das durch die Digitalisierung von Inhalten ein multimedialer Alleskönner geworden ist. Schätzungen gehen davon aus, dass momentan bis zu zwanzig Mal mehr Mobilfunkgeräte in Benutzung sind als Personal Computer (Vgl. Lucka (2008), S.XI). In Deutschland gibt es laut dem Bundesverband Informationswirtschaft, Telekommunikation und neue Medien bereits seit dem 1. August 2006 mehr Mobiltelefone als Einwohner (Vgl. BITKOM (2006)). Zwar ist in erster Linie das Bedürfnis nach Ortsunabhängigkeit für den Erfolg der Handys verantwortlich, jedoch nutzen immer mehr Menschen ihr Handy auch zum Musikhören, um sich Videos anzuschauen oder einfach zum Spielen. Dennoch existiert kein Standard für die Entwicklung von Anwendungen für die mobilen Endgeräte. Ziel dieser Diplomarbeit ist es ein Modell zu entwickeln mit deren Hilfe die Entwicklung für mobile Endgeräte vereinfacht wird.

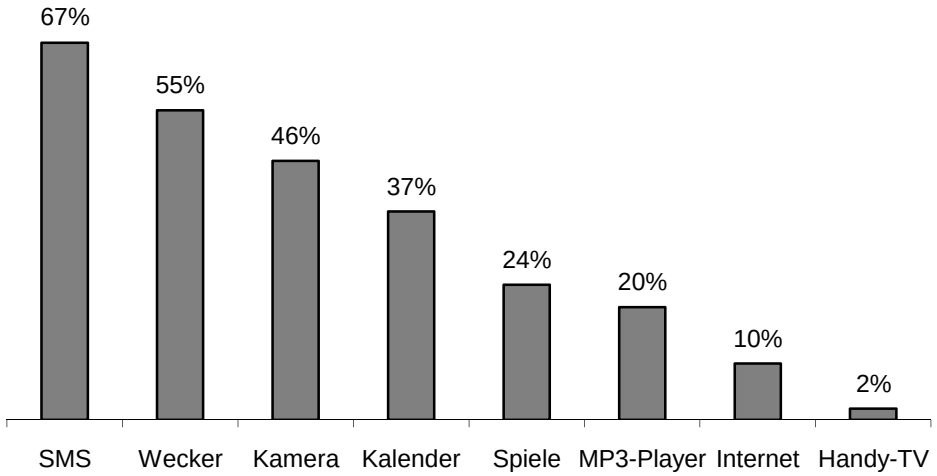
Nokia	Samsung	LG	Sony Ericsson	Motorola	RIM	Kyocera	Apple iPhone	HTC	Sharp	Other
38,6%	**16,2%**	**8,3%**	**8,0%**	**8,3%**	**1,9%**	**1,4%**	**1,1%**	**1,1%**	**1,0%**	**14,1%**

(Vgl. Abi Research, 2009)

Tabelle 1-1: Marktanteile Handys

Die Tabelle 1-1 zeigt die Marktanteile der Mobiltelefone im Januar 2009. Sie verdeutlicht, dass auf dem Mobilfunkmarkt sehr viele unterschiedliche Handyanbieter weltweit konkurrieren. Durch die verschiedenen Voraussetzungen bei der Bauweise der Mobilfunktelefone und durch den Einsatz unterschiedlicher Konfigurationen und Profile bei den einzelnen Modellen der Anbieter, kann nicht von einer einheitlichen Funktionalität aller Mobilfunkgeräte ausgegangen werden. Dies ergibt ein sehr heterogenes Umfeld, was die Entwicklung von Software erschwert. Genau an diesem Punkt setzt die Diplomarbeit an. Ziel ist es, ein Modell zu entwickeln, das auf die Probleme eingeht und sie zu lösen versucht.

Diese Diplomarbeit befasst sich mit der Entwicklung von Anwendungen für mobile Endgeräte. Mobilfunktelefone werden von jeder Altersklasse benutzt. Zum Beispiel besitzen in der Altersklasse von 18 bis 49 mehr als 90% der Menschen mindestens ein mobiles Endgerät. Aber auch bei den Personen zwischen 50 bis 69 Jahren verfügen mehr als 82% über ein Mobilfunktelefon (Vgl. Aumüller H. (2007), S.9). Da diese Erhebung auf das Jahr 2007 zurückgeht, muss davon ausgegangen werden, dass diese Werte weiter gestiegen sind. Diese Werte verdeutlichen, dass der Mobilfunkmarkt jede Altersklasse anspricht und nicht auf die jüngere Altersklasse beschränkt ist.



(Vgl. Lucka et al. (2008), S.2)

Abb. 1-1: Forsa Umfrage zur bevorzugten Handynutzung

Ein weiterer Grund für die Entwicklung eines Modells für mobile Endgeräte ist die Tatsache, dass die Mobilfunktelefone nicht nur zum Telefonieren und zum Schreiben von Kurznachrichten benutzt werden. Abbildung 1-1 veranschaulicht das multimediale Nutzungsverhalten der Handybesitzer neben dem Telefonieren. In dieser Forsa Umfrage wird deutlich, dass auch Anwendungen wie die Kamera, ein Kalender oder auch Spiele von den Mobilfunknutzern gewünscht und genutzt werden. Dieser Trend hält bis heute an. Mobilfunktelefone sind heutzutage für die Kunden mehr als nur ein Gerät zum Telefonieren.

Mit der Programmiersprache Java können sehr leicht Anwendungen für das Mobiltelefon entwickelt werden. Dabei muss es sich nicht zwangsläufig um Applikationen wie einen MP3-Player oder einen Wecker handeln. Die Forschungs- und Entwicklungsrichtung ist noch nicht ausgereizt.

Um das Problem der unterschiedlichen Konfigurationen, Profile und optionalen Paketen beherrschen zu können, existieren unterschiedliche Ansätze. Zum Beispiel gibt es die Entwicklung von Anwendungen für nur ein System bzw. für ein Mobilfunktelefon. Erfahrungen von Softwareunternehmen zeigen jedoch, dass der traditionelle Weg, Software zu entwickeln, nicht effektiv genug ist, um alle Anforderungen zu befriedigen. Unter traditioneller Softwareentwicklung werden die Entwicklungsaktivitäten im Kontext eines Entwicklungsprojektes betrachtet, bei dem der Fokus nur auf diesem Projekt liegt. Um dies zu verhindern, müssen Gemeinsamkeiten zwischen den Projekten identifiziert werden. Und genau dieser Ansatz wird mit einer Software-Produktlinie verfolgt (Vgl. Muthig et al. (2004), S.1). „Eine Produktlinie ist eine Menge von miteinander in

Beziehung stehenden Systemen, die Gemeinsamkeiten besitzen“ (Bunse et al. (2008), S.97). Dies wird seit Jahren zum Beispiel in der Automobilindustrie eingesetzt. So entwickelt die Robert Bosch GmbH Fahrerassistenzsysteme für Fahrzeuge, wie zum Beispiel den Parkpiloten und die PreCrash-Sensierung. Die große Produktvielfalt, die sich aus je nach Fahrzeugtyp verfügbarer Sensorik und geforderter Funktionalität ergibt, kann nur mithilfe einer Software-Produktlinie ökonomisch beherrscht werden (Vgl. Böckle et al. (2004), S.3f). Die Entwicklung eines einheitlichen Software-Produktlinienmodells speziell für den mobilen Gerätemarkt, vereinfacht den Umgang mit den unterschiedlichen Profilen, Konfigurationen und optionalen Paketen.

Eine weitere Motivation ist die Tatsache, dass der Anspruch und die Anforderungen von Kunden und Unternehmen an Software in den letzen Jahren rapide gewachsen sind. Neben den kritischen Kriterien wie der Minimierung der Kosten, des Arbeitsaufwandes und der time-to-market existieren gleichzeitig noch die Anforderungen, dass die Komplexität und die Größe der Produkte wachsen sollen und Kunden fordern mehr Qualität und individuelle Anpassungsmöglichkeiten für die Produkte (Vgl. Bory et al. (2001)). Durch die Entwicklung des Modells soll versucht werden, auf die neuen Ansprüche und Anforderungen der Kunden besser eingehen zu können und trotzdem die Kosten zu minimieren.

Auf dem Markt existieren bereits Software-Produktlinien für mobile Endgeräte, wie zum Beispiel MobileMedia (Vgl. Figueiredo et al. (2008)). Doch diese Produktlinien konzentrieren sich auf kleine Teilbereiche. MobileMedia beschäftigt sich mit der Anzeige und der Manipulation von Bildern, Videos und Musik für mobile Endgeräte. Anwendungen wie zum Beispiel Spiele benötigen diese Manipulationen, stellen darüber hinaus weitere Anforderungen an die Entwickler. Die Diplomarbeit hat das Ziel ein Software-Produktlinienmodell zu entwickeln, mit Hilfe deren jede Anwendung entwickelt werden kann.

1.2 Ziel der Arbeit

Das Ziel dieser Diplomarbeit ist der Entwurf eines Software-Produktlinienmodells, das eine einheitliche Entwicklung von Anwendungen für Mobilfunktelefone vereinfachen soll.

Im Mittelpunkt stehen dabei,

- die Entwicklung eines eigenen Software-Produktlinienmodells, welches unter Berücksichtigung der Vor- und Nachteile der existierenden Modelle für den Bereich der Mobilfunktelefone angepasst ist.

- die Validation des entworfenen Modells anhand eines Fallbeispiels und eines Museumsführers.

Diese Diplomarbeit hat das Ziel, neben der Entwicklung eines angepassten Modells für mobile Endgeräte, auch das Interesse an Software-Produktlinien und deren Umsetzung zu wecken. Die veranschaulichte, komplette Umsetzung einer Software-Produktlinie am Fallbeispiel kann von Anwendern und Forschern genutzt werden, um ihr Verständnis über Software-Produktlinien zu festigen und zu erweitern. Des Weiteren soll diese Arbeit dazu dienen, eine bessere Vergleichbarkeit bei der Umsetzung zu schaffen und zu zeigen wie Werkzeuge und Methoden praktikabel genutzt werden können.

1.3 Aufbau der Arbeit

Die Arbeit ist folgendermaßen aufgebaut. Als Erstes wird auf die Ziele der Arbeit hingewiesen und welche Motivation bestand, diese Diplomarbeit zu verfassen.

Das Kapitel 2 beschreibt die Grundlagen für die Vorgehensmodelle, die Entwicklung von Software bei mobilen Endgeräten und Produktlinien. Zu diesem Zweck erfolgt in Kapitel 2.1 eine Vorstellung von unterschiedlichen Vorgehensmodellen. In Kapitel 2.2 werden Mobilfunkgerätgrundlagen gelegt. Es wird der Unterschied zwischen den einzelnen mobilen Endgeräten, wie zum Beispiel Handys und Smartphones, erklärt und näher darauf eingegangen, welche Mobilfunkbetriebssysteme am Markt existieren. Darauf aufbauend wird in Kapitel 2.3 auf die Softwareentwicklung für mobile Endgeräte näher eingegangen. Es werden Grundlagen, die für die Entwicklung mit der Programmiersprache Java benötigt werden, vermittelt. Diese Grundlagen werden später in der Arbeit benötigt. Das Kapitel 2 schließt mit der Vorstellung von Produktlinien ab. Dabei wird das Referenzmodell vorgestellt, aus welchem sich die spezifischen Software-Produktlinienmodelle entwickelt haben, auf die näher in Kapitel 4 eingegangen wird.

Kapitel 3 befasst sich mit der Problemanalyse der Diplomarbeit. Dabei werden Einsatzszenarien für Software-Produktlinien in der Praxis vorgestellt, um zu verdeutlichen warum in dieser Diplomarbeit Software-Produktlinien genutzt werden. Des Weiteren wird das Fallbeispiel dieser Diplomarbeit zur Validation des zu entwickelnden Modells präsentiert und überprüft, ob es genügend Potential für die Entwicklung für mobile Endgeräte gibt.

Danach folgt die Analyse von Software-Produktlinienmodellen in Kapitel 4. Zu diesem Zweck werden als Erstes in Kapitel 4.1 die Hauptaktivitäten von Produktlinien vorgestellt. Spezifische Modelle für Software-Produktlinienentwicklungen folgen in Kapitel 4.2. Diese werden kurz vorgestellt und deren Vor- und Nachteile für die Entwicklung von Anwendungen für mobile Endgeräte miteinander verglichen, um Grundlagen zu schaffen, damit im nächsten Kapitel das zu entwickelnde Modell die Vorteile der einzelnen Modelle besitzt ohne die Nachteile mit anzunehmen. In Kapitel 4.3 werden Modellsprachen für die Umsetzung der Software-Produktlinien genannt, diese miteinander verglichen und einige kurz vorgestellt. In Kapitel 4.4 wird näher auf die Implementierungsmöglichkeiten einer Software-Produktlinie eingegangen und eine Empfehlung für mobile Endgeräte gegeben.

In Kapitel 5 wird ein Modell auf Basis der vorgestellten Modelle entwickelt, wobei die Beschränkungen von mobilen Endgeräten berücksichtigt werden.

Kapitel 6 befasst sich mit der Validation des entwickelten Modells mithilfe des Fallsbeispiels und eines schon vorhandenen Museumsführers für mobile Endgeräte.

2 Grundlagen

Im Grundlagenteil dieser Diplomarbeit werden als Erstes die Vorgehensmodelle für die Softwareentwicklung vorgestellt mit deren Hilfe sich Anwendungen entwickeln lassen. Es werden unterschiedliche Vorgehensmodelle vorgestellt und deren Vor- und Nachteile aufgezeigt.

Danach wird auf die Mobilfunkgrundlagen eingegangen. Dazu werden Handys mit Smartphones verglichen und deren Bedeutung mithilfe von Statistiken verdeutlicht. Darauf aufbauend werden die auf dem Markt existierenden Mobilfunkbetriebssysteme vorgestellt.

Dann folgen die Grundlagen, die für die Handysoftwareentwicklung von Bedeutung sind, da diese später bei der Validierung des Modells benötigt werden.

Als letztes werden Software-Produktlinien und deren Referenzmodell vorgestellt, damit in Kapitel 4 darauf aufgebaut werden kann.

2.1 Vorgehensmodelle

Vorgehensmodelle geben den ablauforganisatorischen Rahmen von Softwareprojekten vor und umfassen die notwendigen Aktivitäten zur Entwicklung von Software (Vgl. Rautenstrauch et al. (2003), S. 85).

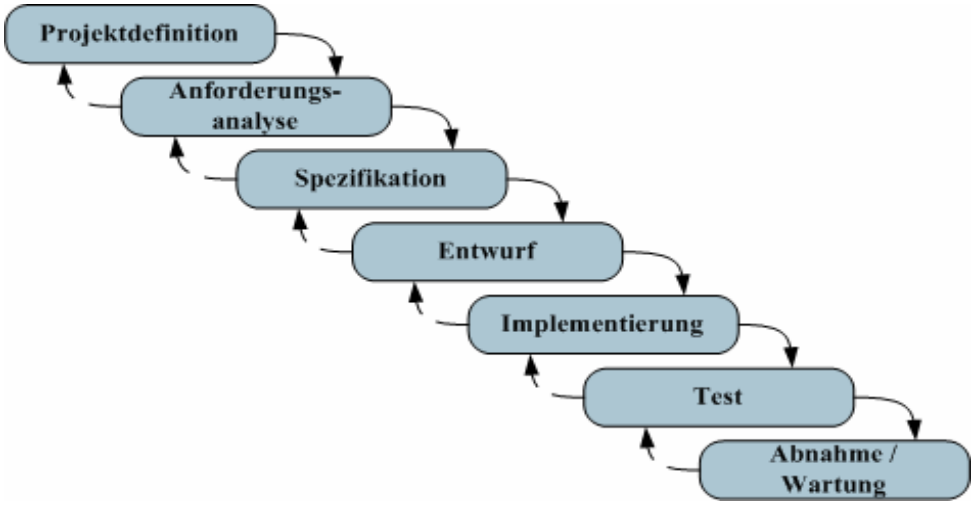
Ein Softwareprodukt durchläuft während seiner Entwicklung sieben Phasen (Vgl. Dumke (2003), S.18f). Es startet mit der *Problemdefinition*, bei der die Anforderungen an die zu entwickelnde Software definiert werden. Die *Anforderungsanalyse* überprüft die Anforderungen auf ihre Realisierbarkeit. Bei der *Spezifikation* wird das Produkt in seiner Funktionsweise beschrieben und beim *Entwurf* wird das spezifizierte Produkt auf eine konkrete Plattform umgesetzt. In der *Implementierungsphase* wird der Entwurf in Programmcode umgesetzt. Beim *Test* wird die Korrektheit der Implementierung überprüft. Die Phase der *Auslieferung* kümmert sich um die Übergabe der Software an den Auftraggeber.

Vorgehensmodelle unterteilen sich in sequentielle und nicht sequentielle Modelle.

2.1.1 Sequentielle Modelle

Sequentielle Vorgehensmodelle sehen eine strenge chronologische Abarbeitung des vorgestellten Software-Lebenszyklus während eines Entwicklungsprozesses vor. In diesem Abschnitt dient eines der bekanntesten Vorgehensmodelle, das Wasserfallmodell,

als Beispiel für das sequentielle Vorgehen. In Abbildung 2-1 ist dies Anschaulich dargestellt.



(Vgl. Royce (1970), S.2)

Abb. 2-1: Wasserfallmodell

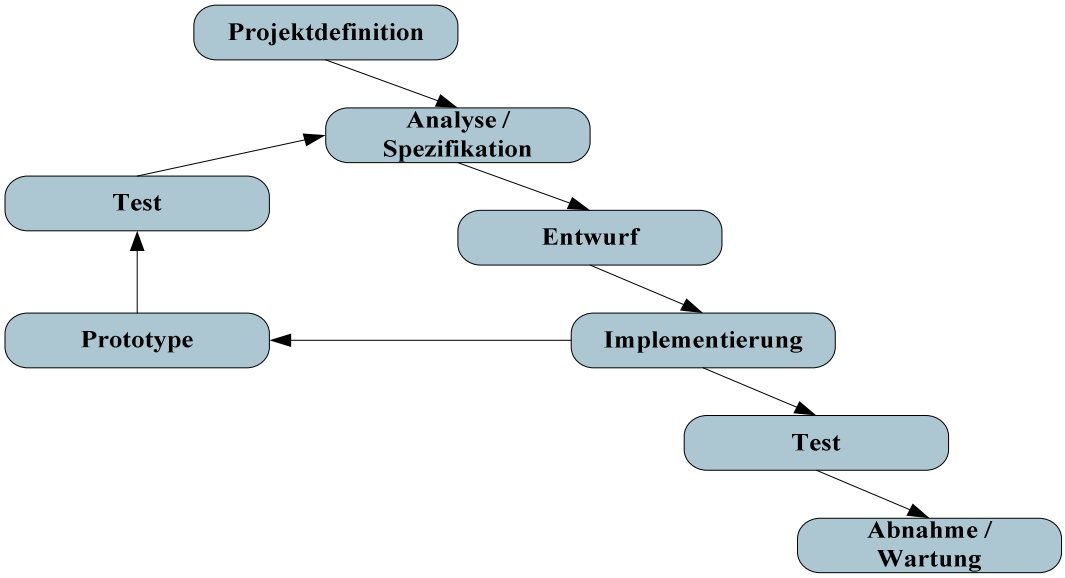
In seiner klassischen Ausprägung sieht das Wasserfallmodell eine absolut sequentielle Abarbeitung der einzelnen Phasen vor. Eine Phase wird erst begonnen, wenn die vorherige beendet wurde. Ein Rücksprung in eine abgeschlossene Phase ist nicht erlaubt. Es existieren jedoch Erweiterungen, wie in Abbildung 2-1 dargestellt, die das Zurückgehen in die vorherige Phase erlauben.

Trotzdem ist die Abfolge festgelegt und statisch. Voraussetzung für diese Art der Entwicklung ist, dass am Anfang eines Softwareprojekts eine sehr exakte und umfangreiche Planung stattfindet, die eine gute Voraussetzung für die folgenden Phasen schafft. Des Weiteren werden alle durchgeführten Schritte ausführlich dokumentiert.

Die exakte Abgrenzung der einzelnen Phasen ist vorteilhaft, da die zu erreichenden Ziele der Phasen genau definiert und somit kontrolliert werden können. Der Fortschritt des Projekts kann genau nachvollzogen werden. Außerdem ermöglicht die simple Struktur bei geeigneten Projekten eine gute Abschätzung der benötigten Zeit und der entstehenden Entwicklungskosten. Zudem ist es durch die Phasentrennung und die ausführliche Dokumentation jederzeit möglich, neue Mitarbeiter in das Projekt zu integrieren (Vgl. Munz et al. (2007), S. 71).

2.1.2 Nicht sequentielle Modelle

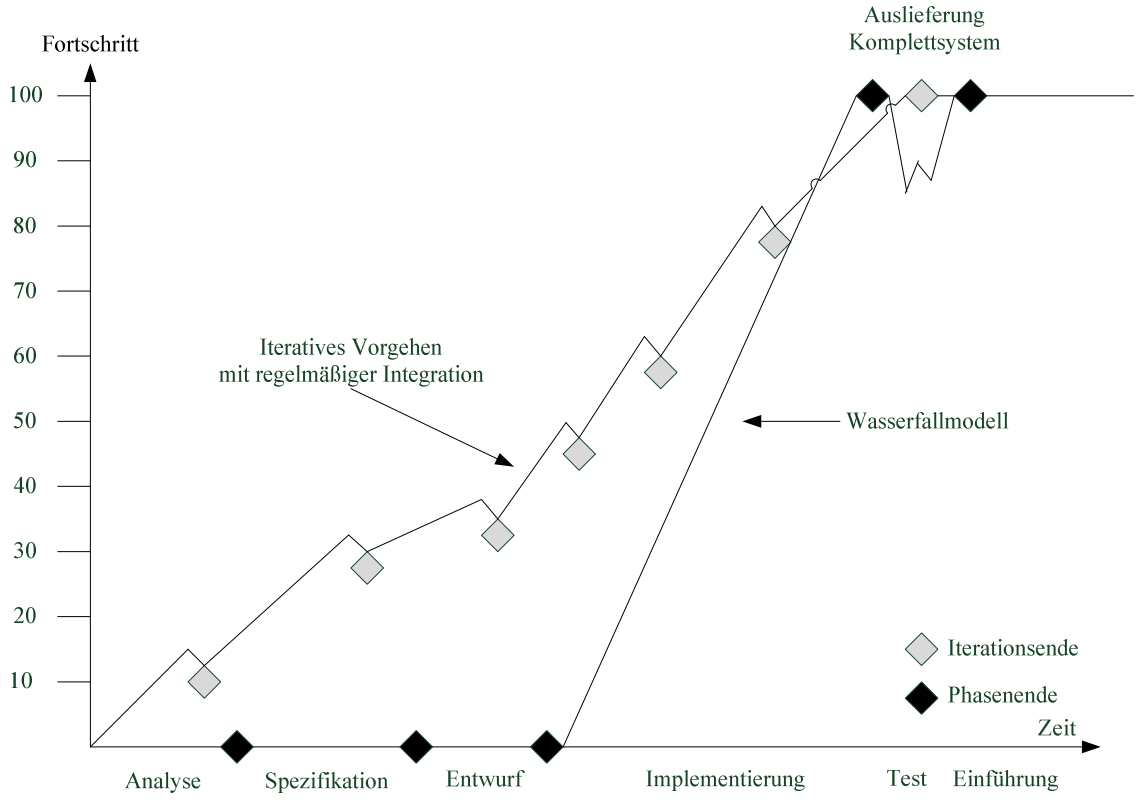
Nicht sequentielle Vorgehensmodelle rücken von der Idee ab, dass bei der Entwicklung alle Anforderungen durch den Auftraggeber von vornherein klar definiert und abgegrenzt werden können. Rücksprünge in bereits abgeschlossene Phasen werden somit ermöglicht. Ein Beispiel für ein solches Modell ist das Prototyping, wie in Abbildung 2-2 dargestellt, bei dem ein „ablauffähiges Muster (von Teilen) des endgültigen Anwendungssystems“ (Rieger et al. (2007), S.2) entsteht. Da Rücksprünge explizit erwünscht sind, ist die Umsetzung von Anforderungen, welche erst nach dem Entwicklungsbeginn hinzukommen, denkbar. Da nach jedem Zyklus aus den Phasen Analyse, Spezifikation, Entwurf und Implementierung ein lauffähiger Prototyp entsteht, ist es möglich, diesen bereits zu testen und die Testergebnisse in den nächsten Zyklus einfließen zu lassen. Ein Vorteil ist, dass durch das Vorführen des Prototyps der Auftraggeber eine frühzeitige Rückmeldung bekommt.



(Vgl. Dumke (2003), S.116)

Abb. 2-2: Beispiel des Prototyping

In Abbildung 2-3 werden die iterativen, nicht sequentiellen Vorgehen dem Wasserfallmodell bezüglich des Fortschritts in messbarem Code gegenübergestellt. Es ist gut erkennbar, dass vor jedem Iterationsende die Lines of Code durch das Testen geringfügig abnehmen. Beim iterativen Vorgehen ist ein schnellerer Fortschritt sichtbar.



(Vgl. Oesterreich et al. (2008), S.46)

Abb. 2-3: Vergleich des Wasserfallmodells mit einem iterativen Modell

2.2 Mobilfunkgerätegrundlagen

Ein Mobiltelefon ist ein tragbares Telefon, das über Funk mit einem Telefonnetz kommuniziert und daher ortsunabhängig eingesetzt werden kann. Die größten Hersteller sind, wie in Tabelle 1-1 auf Seite 1 zu sehen, Nokia, Samsung, LG, Sony Ericsson und Motorola.

Der Begriff Handy für ein Mobilfunktelefon ist nur im deutschsprachigen Raum zu finden. Im englischsprachigen Raum ist das Wort „Handy“ keine Bezeichnung für ein Mobiltelefon, sondern wird als Adjektiv mit praktisch/bequem/handlich übersetzt. Somit ist es als Scheinanglizismus für ein Mobiltelefon zu verstehen.

Neben den bekannten Handys gibt es noch weitere Sparten im Mobilfunkbereich wie zum Beispiel die Smartphones. Ein Smartphone kann für die Verwaltung von Terminen und Kontakten, zum Aufnehmen vom Sprachmemos und Aufgabenlisten und zum mobilen Internetsurfen benutzt werden. Zwar bieten heutzutage auch viele normale Mobilfunktelefone diese Möglichkeit an, aber durch das integrierte Betriebssystem in Smartphones ist es möglich, eigene Anwendungen zu entwickeln und zu installieren. Bei gängigen mobilen Endgeräten wird eine Installation einer Java-Anwendung häufig

auch unterstützt, aber die Möglichkeiten der Anwendungen sind sehr eingeschränkt. Die dürfen nur sehr klein sein und sie laufen in einer vom eigentlichen Betriebssystem abgeschotteten Sandbox, was vergleichbar mit einem Java Applet ist. Die Sandbox verhindert aus Sicherheitsgründen den Zugriff einer Java-Anwendung auf die meisten Funktionen der Betriebssysteme und anderer Anwendungen. (Vgl. Gerlicher et al. (2004), S.2)

Dies grenzt den Softwareentwickler sehr ein und somit beschränken sich viele Java-Anwendungen auf eine Weckerfunktionalität oder einen einfachen Kalender. Mit einem Betriebssystem wie dem Symbian OS, Android oder Research in Motion hat der Entwickler die Möglichkeit auf alle Betriebssystemfunktionen zuzugreifen.

Aufgrund dessen, dass nahezu jedes Mobiltelefon eine Javaunterstützung anbietet, wird in dieser Diplomarbeit nicht nur auf die Smartphones eingegangen.

2.2.1 Statistiken

Die Smartphones besitzen derzeit einen Marktanteil von knapp 10% aller mobilen Endgeräte. In dieser Arbeit wird trotzdem explizit auf sie mit eingegangen, weil sich dieser Marktanteil in den nächsten Jahren laut Prognosen stark erhöhen wird. Eine Nichtberücksichtigung dieser Geräte würde die allgemeine Aussagekraft des zu entwickelnden Software-Produktlinienmodells verkleinern.

Die Informa Telecoms & Media prognostiziert für das Jahr 2009 (Vgl. Byrne (2009)):

- Der Verkauf aller mobilen Endgeräte wird um 10,1% sinken.
- Der Absatz der Smartphones wird aber trotzdem um 35,3% steigen.
- Der Marktanteil der Smartphones wird dadurch auf 13,5% aller mobilen Endgeräte steigen.

Für das Jahr 2013 prognostiziert die Informa Telecoms & Media (Vgl. Byrne (2009)):

- Smartphones werden einen Marktanteil von mehr als 38 % besitzen
- Durch die Umstellung auf OpenSource Basis wird die Symbian Foundation seine Führung im Bereich der Smartphones weiter halten können vor Android und Research in Motion

Durch diese Zahlen dürfen bei der Entwicklung für mobile Geräte nicht nur Handys berücksichtigt werden, sondern die Smartphones rücken immer mehr in den Fokus. Nach der Prognose besitzen im Jahr 2013 vier von zehn Menschen ein Smartphone. Gründe für das starke Wachstum im Bereich der Smartphones ist die stärkere Präsenz

im Bereich der Endkunden zum Beispiel durch das iPhone. In den Jahren davor richteten sich die Smartphones an die Geschäftsleute, um ihre Termine zu koordinieren. Die Ansprüche an ein mobiles Gerät wachsen und durch die Einführung von einfach zu bedienenden Märkten, zum Tauschen und Verkaufen von Anwendungen, wächst die Nachfrage. Mit der Entscheidung, die Basis für die Softwareentwicklung OpenSource zu gestalten, werden immer mehr private Entwickler angesprochen. Durch das Zusammenarbeiten vieler namhafter Firmen, wie bei der Symbian Foundation oder Android, werden Voraussetzungen für eine einheitliche Basis geschaffen (Vgl. Byrne (2009)).

Aufgrund der ständigen Weiterentwicklung der Technik wird eine klare Abgrenzung zwischen den Smartphones und den Mobilfunkgeräten immer schwieriger. Betriebssysteme wie das Android sind nicht nur für Smartphones entwickelt worden, sondern können auch in Mobilfunktelefonen oder Netbooks verwendet werden.

2.2.2 Mobilfunkbetriebssysteme

Betriebssystem	Absatz 2008 (in Tsd.)	Marktanteil 2008	Absatz 2007 (in Tsd.)	Marktanteil 2007
Symbian OS	72933,5	52,4%	77684,0	63,5%
Research in Motion	23149,0	16,6%	11767,7	9,6%
Microsoft Windows Mobile	16498,1	11,8%	14698,0	12,0%
Mac OS	11417,5	8,2%	3302,6	2,7%
Linux	11262,9	8,1%	11756,7	9,6%
Palm OS	2507,2	1,8%	1762,7	1,4%
Andere	1519,7	1,1%	1344,0	1,2%
Total	**139287,9**	**100,0%**	**122315,7**	**100,0%**

Tabelle 2-1: Absatz und Marktanteile der Mobilfunkbetriebssysteme

In Tabelle 2 wird der Absatz in den Jahren 2007 und 2008 für die Betriebssysteme verglichen. Symbian OS ist der eindeutige Marktführer mit mehr als 50% Marktanteil. Das Research in Motion Betriebssystem ist durch das BlackBerry bekannt geworden. Der Marktanteil von über 16% im Jahr 2008 bedeutet neben Platz 2 bei den Marktanteilen auch ein Wachstum von 7% im Vergleich zum Vorjahr. Auch das Mac OS konnte im Jahr 2008 viele Prozentpunkte am Marktanteil durch das iPhone 3G gewinnen. Zwar hat Linux im Vergleich zum Jahr 2007 etwas an Marktanteilen verloren, aber durch die Einführung von Android durch die Open Handset Alliance wird es laut dem Marktforschungsinstitut Gartner bis zum Jahr 2010 zusammen mit Symbian OS einen Marktanteil von 60 bis 70 % erarbeiten (Vgl. Gartner Inc (2009)).

Im Folgenden werden kurz die Betriebssysteme im Einzelnen vorgestellt, um darzustellen, welche Firmen auf die einzelnen Betriebssysteme setzen und welche technischen Grundlagen die Betriebssysteme besitzen.

2.2.2.1 Symbian

Die Firma Symbian wurde 1998 aus der Psion Group gegründet. An dieser Firma waren neben Psion auch die damals vier größten Mobiltelefonhersteller beteiligt – Nokia, Ericsson, Motorola und Panasonic. Das erste Endgerät mit dem Symbian bzw. Symbian OS genannten Betriebssystem war der Nokia 9210 Communicator im Jahr 2001 (Vgl. Gerlicher et al. (2004), S.4ff).

Seitdem sind viele weitere Hersteller von mobilen Endgeräten in die Allianz eingetreten, wie z.B. Sony Ericsson, Samsung, Siemens und BenQ. Durch die Präsenz vieler Anbieter von Smartphones seit dem Jahr 1998, ist Symbian das erste Betriebssystem gewesen, das auf vielen Geräten eingesetzt wurde. Daher ist auch der Marktanteil von derzeit noch über 50 % zu erklären. Im Juni 2008 hat Nokia verkündet, alle Anteile der Firma Symbian zu kaufen und das Betriebssystem als Open Source zu veröffentlichen. Nokia verspricht sich mit der neu gegründeten Symbian Foundation die Schaffung einer einheitlichen Software-Plattform für Mobilgeräte (Vgl. Melzer (2008)).

Zwar stellte Symbian OS eine gemeinsame Basis für viele verschiedene Geräte bereit, die darauf aufsetzenden Oberflächen wie das S60 von Nokia und das UIQ von Sony Ericsson unterscheiden sich jedoch sehr stark. Software-Hersteller müssen deshalb unterschiedliche Varianten ihrer Anwendungen für die jeweiligen Oberflächen erstellen. Das dürfte ein Grund dafür sein, dass das Angebot an Symbian-Programmen deutlich geringer ist als das an Anwendungen für Windows Mobile.

Das Marktforschungsinstitut Gartner prognostiziert, dass Symbian OS zusammen mit Android im Jahre 2010 60 bis 70 % Marktanteile haben werden. Außerdem wird angenommen, dass sich bis zum Jahre 2015 nur noch drei offene Betriebssysteme etablieren werden und eines davon wird Symbian sein (Vgl. Gartner Inc (2009)).

Aufgrund der Unterstützung der Programmiersprachen Java und C++ ist das Betriebssystem für die Entwickler sehr interessant. In dieser Diplomarbeit wird der Fokus auf die Entwicklung und Unterstützung der Programmiersprache Java gelegt.

2.2.2.2 Android

„Android besteht aus einem Linux-Betriebssystem, einer Integrationsschicht und vorinstallierten Basisanwendungen. Google bietet für Android eine Entwicklungsplattform an, die (eine) Anwendungsentwicklung mithilfe der Programmiersprache Java ermöglicht.“ (Becker et al. (2009), S.1) Über den Sprachumfang der Java Standard Edition hinaus gibt es zahlreiche Android-spezifische Spracherweiterungen mit über 600 Klas-

sen und Schnittstellen, die der Entwicklung von Programmen für mobile Computer, insbesondere Mobiltelefonen, Rechnung tragen.

Das Betriebssystem wird von der Open Handset Alliance entwickelt und steht unter der freien Apache Lizenz 2.0. Initiator des Projektes ist das Unternehmen Google. Dessen Entwickler haben einen Großteil des Grundgerüsts und einige Anwendungen für die Plattform geschrieben. Außerdem hat Google für die Android Developer Challenge Preisgelder von insgesamt zehn Millionen US-Dollar ausgelobt, um Softwareentwickler für die Entwicklung von Android-Anwendungen zu gewinnen (Vgl. Laib (2007)).

Es basiert auf einem Linux-Kernel 2.6 und extra dafür angepasste Java- und C++-Bibliotheken. Unter diesen Komponenten sind Codecs zur Wiedergabe von diversen Medienformaten, eine auf OpenGL basierende 3D-Bibliothek, eine Browser-Engine und eine Datenbank (Vgl. Becker et al. (2009), S.2ff).

Zu der Open Handset Alliance gehören neben den Netzbetreibern wie Vodafone oder Telekom, auch Gerätehersteller wie Samsung Electronics oder Sony Ericsson und Halbleiterhersteller wie die NVIDIA Corporation, Softwareentwickler wie eBay und Vermarktungsunternehmen wie Wind River System (Vgl. Laib (2007)). Durch diese Vielzahl an unterschiedlichen Firmen und Anbietern können sich Endgerätefirmen wie Motorola und HTC künftig noch besser neue Produkte wie aus einem Baukasten zusammenstellen. Das funktioniert grundsätzlich auch heute schon, setzt aber erhebliche Anpassungsarbeiten und Modifikationen auf Seiten des Herstellers voraus und sie kosten Zeit und Geld. Mit Android ist es möglich, Handys aus Komponenten zusammenzustellen, die garantiert zueinander und mit Android kompatibel sind. Die Entwicklungszeit verkürzt sich erheblich und die Kosten werden verringert.

Derzeit gibt es in Deutschland nur wenige Smartphones, welche Android verwenden, zum Beispiel das G1. In den nächsten Monaten planen aber zahlreiche Firmen, weitere Smartphones mit dem Betriebssystem auf den Markt zu bringen.

2.2.2.3 Research in Motion

Research in Motion ist eine in Kanada beheimatete Firma, welche im Jahr 1984 gegründet wurde. Sie ist Hersteller und Vermarkter für mobile Kommunikationssysteme. Bekannt geworden ist sie im Jahr 1999 durch das BlackBerry. Das dazugehörige Betriebssystem heißt Research in Motion und besteht auf der Basis von Java und C++- Bibliotheken.

2.2.2.4 Mac OS

Mit dem Mac OS ist ein extra für das iPhone angepasste iPhone OS gemeint. Dieses Betriebssystem ist ein Ableger des Mac OS X. Die Programmierschnittstelle des iPhones ist Objective-C und unterstützt daher keine Java-Programme. Deshalb wird in dieser Diplomarbeit nicht genauer auf das Mac OS eingegangen.

2.2.2.5 Microsoft Windows Mobile

Basierend auf der Microsoft Win32 API ist das Microsoft Windows Mobile ein kompaktes Betriebssystem für mobile Geräte. Um vielen Nutzern einen schnellen Einstieg zu gewährleisten, besitzt das Windows Mobile eine Ähnlichkeit mit den Desktopversionen von Windows wie Windows XP oder Vista.

Auch werden seit Windows Mobile 5 an die Ressourcenbeschränkung angepasste Versionen von Excel, Word und Powerpoint implementiert. Weiterhin gibt es einen Windows Media Player, der von Hause aus viele Formate zum Abspielen unterstützt.

Auf vielen Smartphones mit Windows Mobile ist bereits Java installiert. Falls nicht kann dies sehr einfach per Installation nachgeholt werden, sodass die Smartphones mit Windows Mobile auch für diese Arbeit in Betracht gezogen werden können.

2.3 Softwareentwicklung beim Handy

In diesem Kapitel werden die Grundlagen für die Softwareentwicklung beim Mobilfunktelefon dargelegt. Zu diesem Zweck wird die Java Micro Edition vorgestellt und kurz auf die einzelnen Konfigurationen und Profile eingegangen. Diese Grundlagen werden später in der Arbeit benötigt.

2.3.1 Grundlagen

Auf dem Handymarkt existieren viele verschiedene Anbieter wie zum Beispiel Nokia, Sony-Ericsson, oder Motorola. Deren Anwendungssoftware ist größtenteils in C/C++ oder Java geschrieben. In dieser Diplomarbeit wird auf die Anwendungsentwicklung mithilfe von Java näher eingegangen, weil (Vgl. Breymann et al. (2008), S.17f):

- es Java auf über 1,8 Milliarden mobilen Endgeräten gibt und mehr als 80% aller neuen verkauften Handys und Smartphones als Programmiersprache Java verwenden. Daher hat die Entwicklung eines Software-Produktlinienmodells für diese Geräte einen hohen Nutzen für viele Entwickler (Vgl. SUN Microsystems Inc. ,2009).

- Java hohe Akzeptanz bei den Entwicklern erreicht hat, es daher eine sehr große Community besitzt und es viele Foren für Entwickler bei allen namhaften Herstellern gibt.

- der Java Bytecode auf allen *Java Virtual Machine* (JVM) lauffähig ist und jedes weit verbreitete Betriebssystem eine JVM besitzt. Der Aufwand einer Portierung der Anwendung ist geringer als eine Neuentwicklung.

- Java-Anwendungen in einer „Sandbox“ laufen. Eine Sandbox ist eine Umgebung, in der die Aktionen eines Prozesses anhand von Sicherheitsrichtlinien eingeschränkt werden (Vgl. Dittmann (2009)). Die JVM wacht darüber, dass ein Absturz einer Anwendung keine anderen Programme stört. Außerdem wird der Bytecode vor der Ausführung mit einem Prüfsummenverfahren verifiziert, um die Integrität des Codes sicherzustellen.

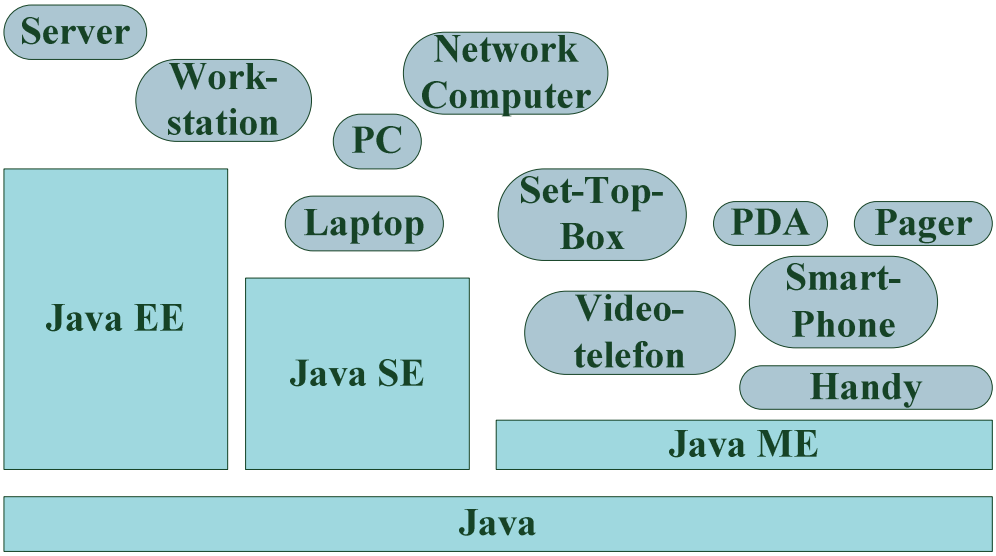
- Java einen Garbage Collector hat und somit keine Speicherlecks, wie zum Beispiel in nicht sauber geschriebenen C++ Programmen, auftreten können.

- Java ein *Application Programming Interface* (API) für sicherheitsrelevante Abläufe hat. Dazu gehören Authentifizierung, die Überprüfung von Signaturen und die Vergabe von Rechten.

- es zwar keine Standardisierung wie bei C++ (ISO 14882) gibt, aber dafür eine breitere De-facto-Standardisierung. Im *Java Community Process* (JCP) sind namhafte Firmen wie die Apache Software Foundation oder Entwickler wichtiger Softwareunternehmen vertreten. Sie entwickeln die *Java Specification Request* (JSR) und deren Spezifikationen, welche auf der Website www.jcp.org einsehbar sind (Vgl. JCP (2009)).

2.3.2 Java Micro Edition (Java ME)

Die Java ME besitzt ein breites Spektrum von Endgeräten, angefangen von Mobiltelefonen über Smartphones, PDAs und Videotelefone bis hin zu Netzwerkcomputern, wie in Abbildung 2-4 dargestellt. Aufgrund des vollkommenen Wettbewerbs unterliegen die Geräte einem permanenten Preisverfall. Dadurch sind die Anbieter gezwungen, die Geräte so günstig wie möglich zu produzieren und preiswerte Komponenten zu verbauen. Aus diesem Grund wird bei den Faktoren wie Rechnerleistung oder Speicherkapazität gespart (Vgl. Schmatz (2004), S.3).



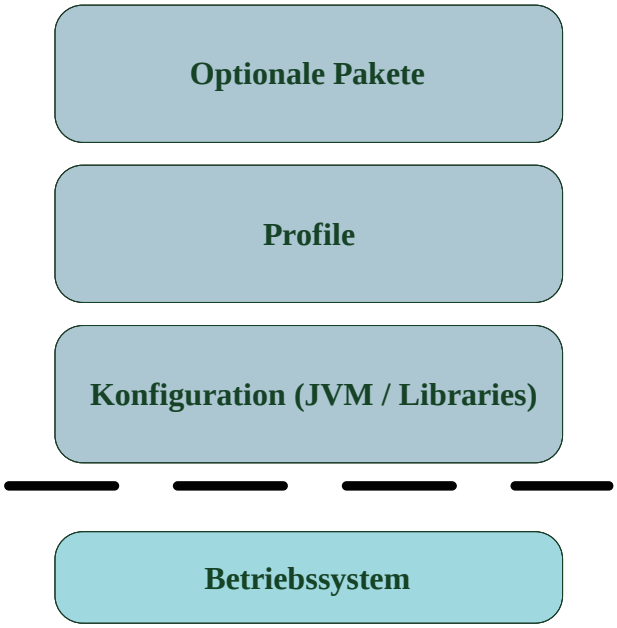
(Vgl. Schmatz (2004), S.3)

Abb. 2-4: Anwendungsgebiete der Java-Editionen

Nach dem Moore'schen Gesetz verdoppelt sich zwar die Speicherdichte auf den Chips etwa alle 18 Monate (Vgl. Moore (1965)), aber die Bibliotheken der *Java Standard Edition* (Java SE) oder der *Java Enterprise Edition* (Java EE) sind für die Speicher heutiger Mobiltelefone noch zu groß. Deshalb ist die *Java Micro Edition* (Java ME) ein angepasstes Java. Weitere Einschränkungen, die ein Mobilfunktelefon hat, sind folgende (Vgl. Breymann et al. (2008), S.19):

- Eingeschränkte Energiequelle. Je leichter und kleiner die Akkus sind, desto weniger Energie steht dem Handy zur Verfügung.
- Aus Platz-, Energiespar- und Gewichtsgründen sind die Bildschirme von Handys sehr klein im Vergleich zu Desktopmonitoren.
- Die Bedienungselemente wie Knöpfe oder die Tastatur müssen zwar klein, aber noch leicht zu bedienen sein.
- Das Handy ist nicht standortgebunden. Daher ist die Verbindung zum Netzwerk häufig unterbrochen oder nur sehr schwach.

Diese Beschränkungen führen dazu, dass die Java ME eine Menge von Spezifikationen und Technologien aufweist, die speziell auf mobile Endgeräte zugeschnitten sind.



(Vgl. Sola (2007), S.3)

Abb. 2-5: High-Level-Architektur der Java ME

Die Java ME-Architektur, siehe Abbildung 2-5, basiert auf Konfigurationen. „Eine Konfiguration definiert einen Mindeststandard an Plattformfunktionalität für Geräte mit vergleichbarer Hardware-Ausstattung“ (Foeller-Nord (2002), S.5). Die Spezifikation einer Konfiguration muss im vollen Umfang vom Hersteller der Endgeräte implementiert werden, damit die Anwendungen auf jedem Gerät die gleichen Plattformdienste vorfinden (Vgl. Foeller-Nord (2002), S.5).

Es gibt derzeit zwei Konfigurationen (Vgl. Schmatz (2007), S.6):

- Die Connected *Limited Device Configuration* (CLDC) ist für Mobiltelefone ausgelegt
- Die *Connected Device Configuration* (CDC) eignet sich für leistungsfähigere, gemeinsam genutzte, stationäre Endgeräte.

Wie in der Einleitung erwähnt, liegt der Schwerpunkt dieser Diplomarbeit auf die Entwicklung eines Software-Produktlinienmodells für Geräte mit begrenzten Ressourcen wie Mobilfunktelefonen, sodass auf die CDC nicht weiter eingegangen wird. Außerdem ist die CLDC millionenfach in Mobilfunkgeräten und Smartphones implementiert.

Ein Profil in der Java ME-Architektur erweitert die Konfiguration um Leistungsmerkmale für einen bestimmten Typ von Geräten. In einem Profil sind optionale Anteile möglich. Das wichtigste und bekannteste Profil ist das *Mobile Information Device Profile* (MIDP). Im November 2002 kam diese Spezifikation in der Version 2.0 heraus, welche heutzutage fast alle Handys unterstützen (Vgl. Schmatz (2007), S.6).

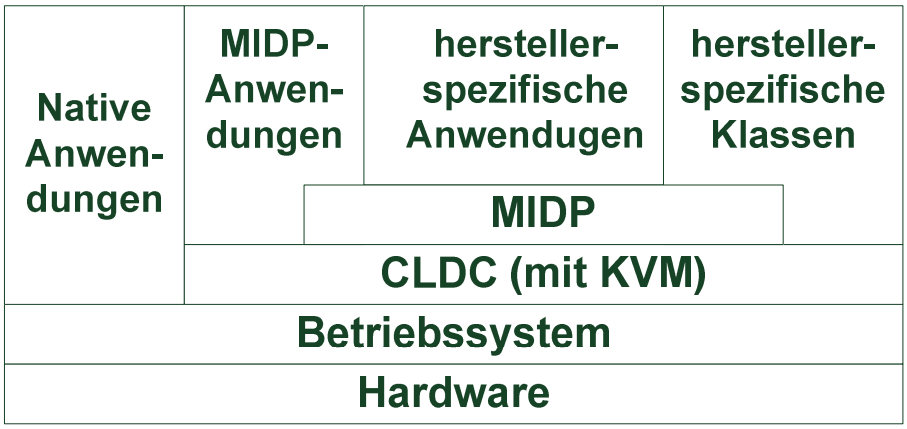
Die optionalen Packages sind für spezielle Anpassungen und Anwendungen, wie zum Beispiel eine Datenbankanbindung und die Bluetooth-Schnittstelle, gedacht. Diese werden von den JCP-Arbeitsgruppen definiert, sodass alle relevanten Hersteller in die Spezifikation eingebunden sind. Verpflichtend zur Implementierung sind die Packages jedoch nicht und außerdem sinkt durch die Nutzung einer solchen Schnittstelle die Portabilität einer Applikation (Vgl. Breymann et al. (2008), S.19; Schmatz (2007), S.6).

Das in dieser Diplomarbeit zu entwerfende Software-Produktlinienmodell wird für die Mobiltelefone entwickelt. Damit die daraus resultierenden Anwendungen auf den Handys funktionieren, müssen die Beschränkungen im Funktionsumfang berücksichtigt werden. Daher werden als nächstes die wichtigsten Konfigurationen und Profile kurz vorgestellt.

2.3.3 Konfigurationen und Profile

Durch die Vielzahl an unterschiedlichen Mobilfunkherstellern und Handys mit unterschiedlichen Eigenschaften ist die Idee entstanden, Klassen von Geräten zu definieren, die bestimmten Java-Konfigurationen entsprechen, damit eine entwickelte Anwendung auf jedem Gerät dieser Klasse laufen soll. Eine Konfiguration ist die Kombination einer Virtual Machine zusammen mit einem API. In der CLDC wird eine auf die Speicherbegrenzung zugeschnittene *Kilobyte Virtual Machine* (KVM) genutzt (Vgl. Breymann et al. (2008), S.33).

Über den Konfigurationen liegen die Profile zur Verwaltung der Benutzungsschnittstellen, der Speicherung persistenter Daten und der Steuerung des Lebenszyklus einer Anwendung.



(Vgl. Breymann et al. (2008), S. 33)

Abb. 2-6: Architektur der CLDC und MIDP

Aufgrund der Vielzahl der Endgeräte und der schnellen Entwicklung ist es nicht möglich, alle Varianten vorherzusehen und zu spezifizieren. Deshalb gibt es, wie in Abbildung 2-6 zu sehen, native Anwendungen, die diese Konfigurationen und Profile nicht nutzen, als auch direkt herstellerspezifische Java-Klassen und Anwendungen (Vgl. Breymann et al. (2008), S. 33f).

Die Abbildung 2-6 zeigt die Möglichkeiten zur Entwicklung auf. Die großen Firmen, die in Tabelle 1 erwähnt wurden, haben das CLDC und MIDP implementiert, sodass in dieser Diplomarbeit auf die spezifizierten Konfigurationen und Profilen weiter eingegangen wird.

2.3.3.1 Connected Limited Device Configuration (CLDC)

Die *Connected Limited Device Configuration* ist eine Konfiguration für Mobiltelefone, Smartphones und PDAs, wie in Abbildung 2-5 auf Seite 17 dargestellt. Durch die Ressourcenlimitierung definiert die CLDC (Vgl. Breymann et al. (2008), S.34):

- eine Java Virtual Machine mit eingeschränktem Funktionsumfang, die KVM. Die KVM ist eine ressourcensparende Version der JVM. Das wichtigste Ziel dieser Virtual Machine ist es, die wichtigsten Eigenschaften von Java zur Verfügung zu stellen und gleichzeitig mit einer Speicherkapazität von 160 bis 512 kiloByte auszukommen. Dazu wurden einige Beschränkungen eingeführt (Vgl. Schmatz (2007), S.14).

- eine sehr kleine, eigene und minimale Klassenbibliothek.

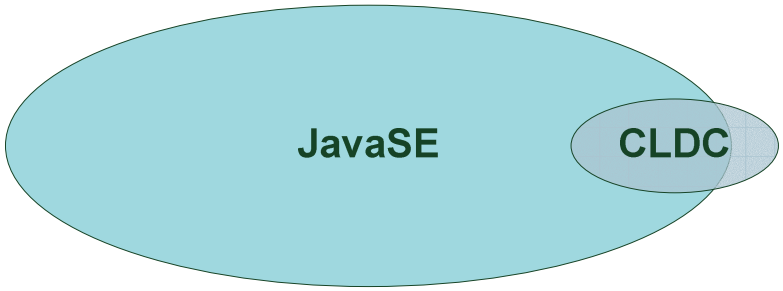
- eine kleine Teilmenge der Java Sprache mit den Kern-Bibliotheken, der Ein- und Ausgabe, Sicherheit, Netzverbindungen und Internationalisierung.

Die in der CLDC implementierte KVM hat gegenüber der normalen JVM der Java SE folgende Einschränkungen (Vgl. Schmatz (2007), S.14f.),

- Das Java Native Interface (JNI) ist nicht in der KVM vorhanden.
- Das Schreiben eigener ClassLoader wird nicht unterstützt. Der Entwickler darf nur auf die zur Verfügung gestellten ClassLoader zurückgreifen.
- Der Gargabe Collector besitzt weniger Funktionen.
- Fehlerbehandlung ist nur rudimentär vorhanden und erfordert eine implementierungsabhängige Herangehensweise.

Durch diese Vielzahl an Einschränkungen wird die Implementierung der KVM erheblich vereinfacht. Aber im Umkehrschluss bedeutet dies auch, dass eine .class-Datei unter Umständen Bytecodes enthält, die die KVM nicht verarbeiten kann. In der Java Micro Edition werden alle über den Funktionsumfang der KVM hinausreichende Bytecodes deshalb durch einen Bytecode-Verifikator abgewiesen (Vgl. Schmatz (2007), S.15f.).

Die CLDC umfasst eine Bibliothek mit grundlegenden Klassen und Interfaces. Ein Ziel der Java ME ist die Auswärtskompatibilität, das heißt Java ME-Programme sollen auch auf größeren Java-Plattformen wie Java SE und Java EE laufen können. Doch durch den starken Verbund von Java SE und Java EE gibt es nur wenige Klassen, die die Anforderungen an die Java ME erfüllen, wie zum Beispiel mit wenig Speicher auszukommen. Deshalb sind einige Java-Bibliotheken für die CLDC neu konzipiert worden. Wie in Abbildung 2-7 zu erkennen ist, lassen sich die Java ME-Bibliotheken in zwei Kategorien einteilen. Die eine stellt eine Untermenge der Java SE-Klassen dar und die andere sind CLDC-spezifische Klassen (Vgl. Breymann et al. (2008), S.34f).



(Vgl. Schmatz, 2007, S.18)

Abb. 2-7: CLDC-Klassenbibliothek

Mit der ständigen Weiterentwicklung im Bereich der mobilen Endgeräte nehmen auch die Rechenleistung und Speicherkapazität stetig zu. Es gibt derzeit zwei Versionen der CLDC-Spezifikation. Viele Anwendungen basieren noch auf der Version 1.0, die als Grundlage für die Profile MIDP 1.0, MIDP 2.0 und MIDP 2.1 ausreicht. Der Unterschied zwischen beiden Versionen ist die Aufhebung einiger Einschränkungen, wie zum Beispiel der Fließkomma-Arithmetik. Die neue Version der CLDC 1.1 erfordert aber auch eine um 32 kiloByte höhere Kapazität des Speichers (Vgl. Schmatz (2007), S.13).

Neue Mobilfunkgeräte besitzen aber die CLDC-Spezifikation 1.1, da der multimediale Markt für Handys immer größer wird. Zum Beispiel wird sich der Umsatz mit mobilen Spielen in den nächsten 4 Jahren bis 2013 von derzeit 5,4 Milliarden US-Dollar auf über 10 Milliarden US-Dollar erhöhen (Vgl. Hoden (2008)). Bei der Entwicklung von komplexen Anwendungen werden die Aufhebungen der Beschränkungen benötigt.

2.3.3.2 Mobile Information Device Profile (MIDP)

Das Mobile Information Device Profile ist das wichtigste Profil für die CLDC. Es liegt derzeit in drei unterschiedlichen Spezifikationen vor, dem MIDP 1.0, MIDP 2.0 und MIDP 2.1. Diese Versionen funktionieren mit allen vorgestellten Spezifikationen der CLDC. Das MIDP unterstützt die für eine erfolgreiche Entwicklung erforderlichen Funktionen durch eine Vielzahl von APIs und der dazugehörigen Bibliotheken. Die Funktionen betreffen die folgenden Bereiche (Vgl. Breymann et al. (2008), S.38f):

- Steuerung des Anwendungslebenszyklus
- Signierung und Sicherheitsmodell
- Bedienungsinterfaces wie zum Beispiel Bildschirm oder Tastatur
- Kommunikation
- Persistenter Speicher (RecordStore)
- Medienverarbeitung wie zum Beispiel Erzeugung und Abspielen von Tönen
- Timer
- 2D-Spiele

Um diese Funktionalitäten zu gewährleisten, erweitert die MIDP einige Bibliotheken, die die CLDC zur Verfügung stellt. Zum überwiegenden Teil handelt es sich jedoch um

neue Pakete, die zum Beispiel die Funktionalitäten wie eine Game API, eine Media API oder eine API zur Unterstützung der Implementierung von Bedienoberflächen zur Verfügung stellen (Vgl. Schmatz (2007), S.20ff).

Java-Anwendungen, die für ein MIDP geschrieben werden, heißen MIDlets. Dabei ist ein MIDlet eine gewöhnliche Java-Klasse. Diese erbt von einer abstrakten Klasse und implementiert ähnlich wie im Applet die Lebenszyklusmethoden zum Starten, Pausieren und Beenden. Im Gegensatz zu nur auf CLDC basierenden Programmen wird die Klasse nicht in der main-Methode gestartet, sondern spezifisch ähnlich wie bei einem Java-Applet über eine spezielle Methode (Vgl. Breymann et al. (2008), S.41f).

Eine oder mehrere MIDlets werden zu einer MIDlet-Suite zusammengefasst. Dabei ist eine MIDlet-Suite die kleinste installierbare Einheit auf dem portablen Endgerät. Die Zugehörigkeit eines MIDlets zu einer MIDlet-Suite hat folgende Auswirkungen (Vgl. Schmatz (2007), S.22):

- Alle MIDlets besitzen die gleichen Berechtigungen zur Verwendung sensibler Teile der Programmierschnittstellen, eine feinere Differenzierung ist nicht möglich.
- Gespeicherte Datenbestände können nur vom eigenen MIDlet wieder verändert werden. Andere MIDlets haben keinen Zugriff auf diese Daten.
- Innerhalb einer MIDlet-Suite sind alle static-Klassenvariablen von jedem MIDlet abruf- und veränderbar. Über diesem Mechanismus kann eine Kommunikation zwischen den MIDlets realisiert werden.

2.4 Produktlinien

Erst seit kurzem beschäftigen sich Software-Entwicklungsansätze mit kombinierten Software-Entwicklungsprojekten. Früher waren Softwareprodukte sehr häufig Einzelproduktionen, die parallel oder nacheinander in einem Unternehmen durchgeführt werden. Der Nachteil der Einzelproduktion liegt im geringen Wiederverwendungspotential. Nur Bibliotheken mit Teilergebnissen aus alten Projekten konnten wieder verwendet werden. Viele Unternehmen spezialisieren und fokussieren sich aber auf einzelne bestimmte Anwendungsbereiche und entwickeln keine isolierten Systeme, sondern eine Art von Familien von Systemen, so genannte Produktlinien (Vgl. Bunse et al. (2008), S.97). „Eine Produktlinie ist eine Menge von miteinander in Beziehung stehenden Systemen, die Gemeinsamkeiten besitzen“ (Bunse et al. (2008), S.97). Durch die einmalige Modellierung und Implementierung gemeinsamer Anteile der Produktlinie werden Syn-

ergieeffekte ausgenutzt, die in klassischen Ansätzen der Software-Entwicklung, wie zum Beispiel den unter Kapitel 2.1 vorgestellten Vorgehensmodellen, nicht berücksichtigt und genutzt werden (Vgl. Bunse et al. (2008), S.97).

Die Wiederverwendung einzelner Komponenten wird bei Produktlinien strategisch geplant und zielt insbesondere auf eine Leistungssteigerung ab. Mithilfe einer Komponente können mehrere Anwendungen realisiert werden. Daher wird von allen Beteiligten erwartet, dass sie das Vorgehen zur Erstellung einer Software-Produktlinie verinnerlichen und die Wiederverwendung explizit planen und nutzen (Vgl. Rietdorf et al. (2005), S.4).

Der Vorteil der konsequenten Ausrichtung auf Wiederverwendung sind Kostenersparnisse und Qualitätssteigerungen. Außerdem vermeiden sie den Nachteil der mangelnden Anpassungsmöglichkeit von Standardsoftware. Ferner ist im Automobil- oder im Mobilfunkbereich Standardsoftware in bestimmten Domänen kaum verfügbar (Vgl. Bunse et al. (2008), S.97).

Die Produktlinienentwicklung ist daher ein Ansatz zur Softwareentwicklung auf Grundlage einer gemeinsamen Plattform. Bei dieser stehen die organisierte Wiederverwendung und organisierte Variabilität im Mittelpunkt. Unter organisierter Wiederverwendung ist folgendes zu verstehen (Vgl. Böckle et al. (2004), S.4):

- Software wird für die Wiederverwendung in Form einer so genannten Plattform entwickelt.
- Einzelne Softwareprodukte werden durch Wiederverwendung aus dieser Plattform erstellt.

Ziel ist das Verlassen des reaktiven Vorgehens, wo einzelne Softwareprodukte nach Markt- und Kundenbedürfnissen entwickelt werden. Der Fokus einer Software-Produktlinie liegt auf der proaktiven Gestaltung einer gemeinsamen Plattform für eine Vielzahl von Produkten für eine bestimmte Domäne (Vgl. Böckle et al. (2004), S.4).

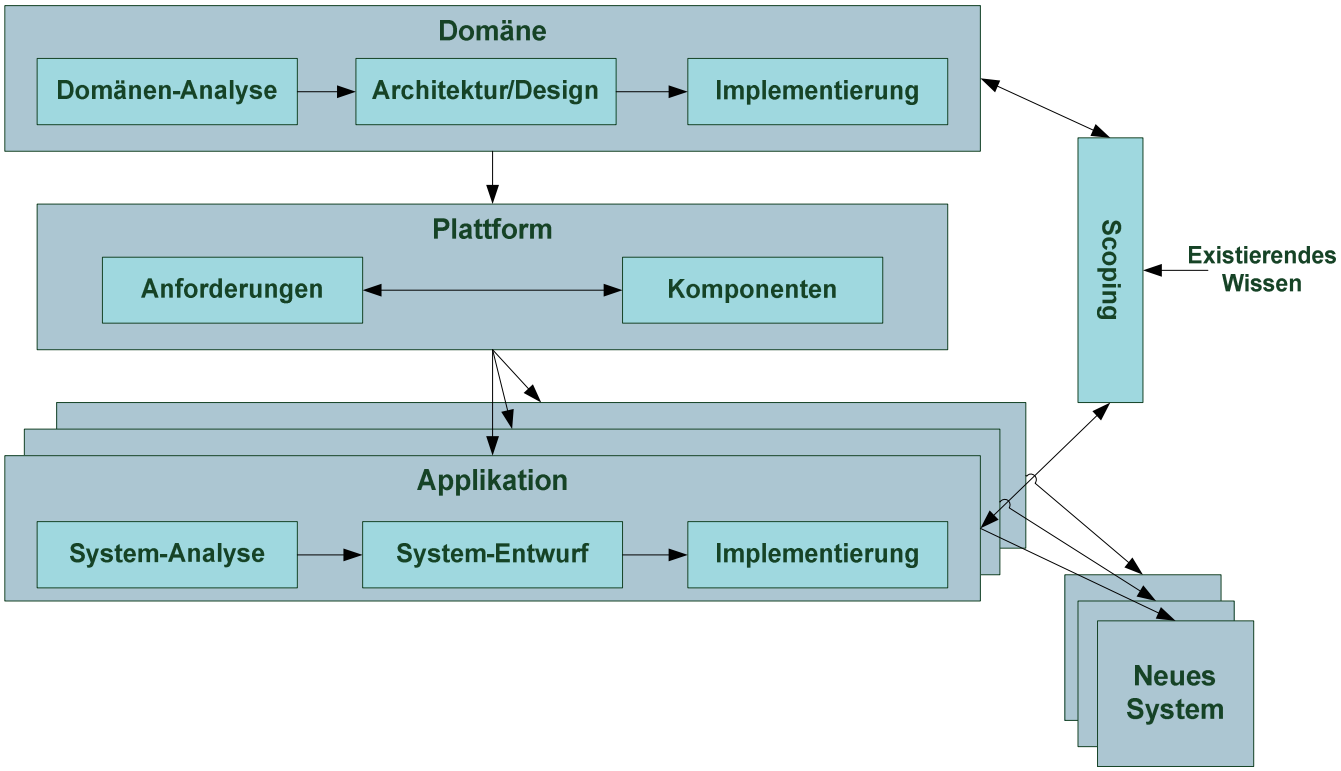
Grundlagen der Software-Produktlinienentwicklung

Die Produktlinienentwicklung für softwaregesteuerte Systeme basiert auf zwei wichtigen Grundlagen (Vgl. Böckle et al. (2004), S.4):

- Als erstes die Beschreibung der Variabilität der Produktlinie
- und als zweites die Trennung des Domänen und Application Engineerings.

Am Anfang müssen die Produkte, welche entwickelt werden sollen, identifiziert und geplant werden. Dafür müssen die wesentlichen Charakteristika dieser Produkte, die so genannten Features, zu Beginn eines Projektes festgelegt und ihre Gemeinsamkeiten und ihre Unterschiede, das heißt die Variabilität der Produktlinie, identifiziert werden.

Einen Entwicklungsprozess für Produktlinien zeigt die Abbildung 2-8. Dabei handelt es sich um den Referenzprozess der Software-Produktlinienentwicklung, der im Rahmen des CAFÉ-Projekts entwickelt wurde (Vgl. van der Linden (2002), S.41ff).

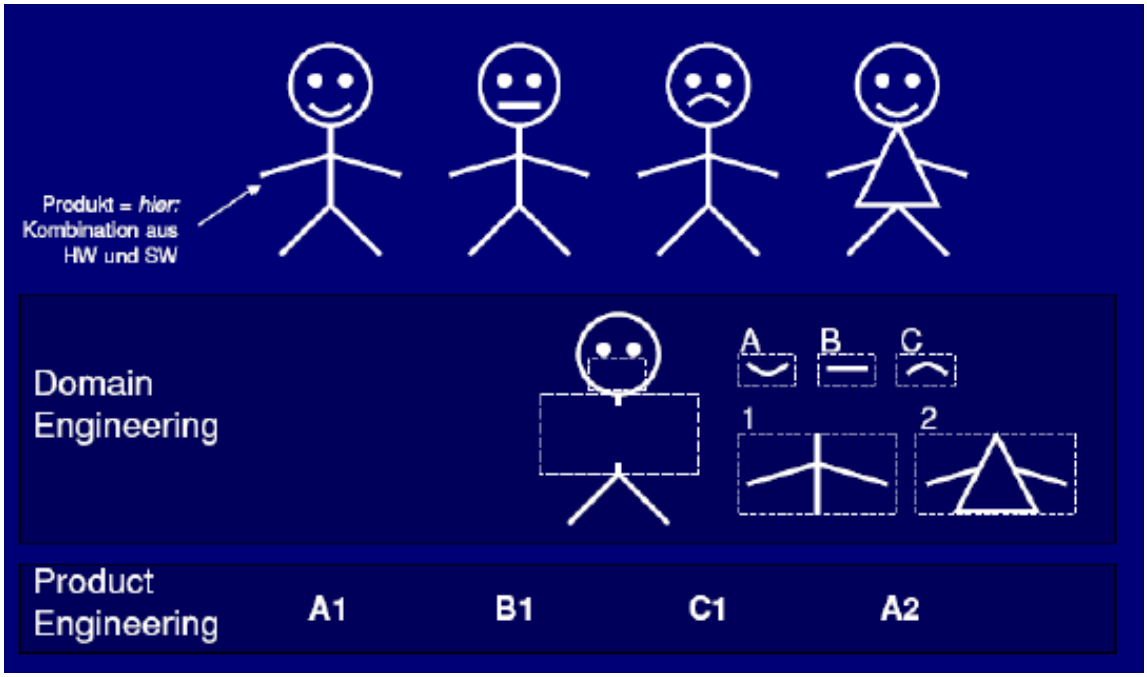


(Vgl. Bunse et al. (2008), S.98)

Abb. 2-8: Referenzmodell für die Software-Produktentwicklung

In Abbildung 2-8 wird verdeutlicht, wie ein Produkt mithilfe einer Software-Produktlinie entsteht. Als erstes muss eine Domäne identifiziert werden. „Eine Domäne ist ein zusammenhängender Teilbereich einer Produktlinie, welcher wiederverwendbare Funktionalität für die Produkte der Produktlinie enthält.“ (Böckle et al. (2004), S.5) Darauf aufbauend entsteht eine Plattform, auf Basis derer die Entwicklung der einzelnen Applikationen geschieht. Eine Applikation besitzt wie die Domäne jeweils eine Anforderungsanalyse-, eine Design- und eine Implementierungsphase, jedoch ist die Unterscheidung des Domänen und Applikation Engineering entscheidend. Die Domäne beinhaltet einen zusammenhängenden Funktionsbereich auf der Problemseite und die ei-

gentliche Anwendung ist die Umsetzung der Komponenten auf der Lösungsseite (Vgl. Böckle et al. (2004), S.5).



(Vgl. Reiser (2007))

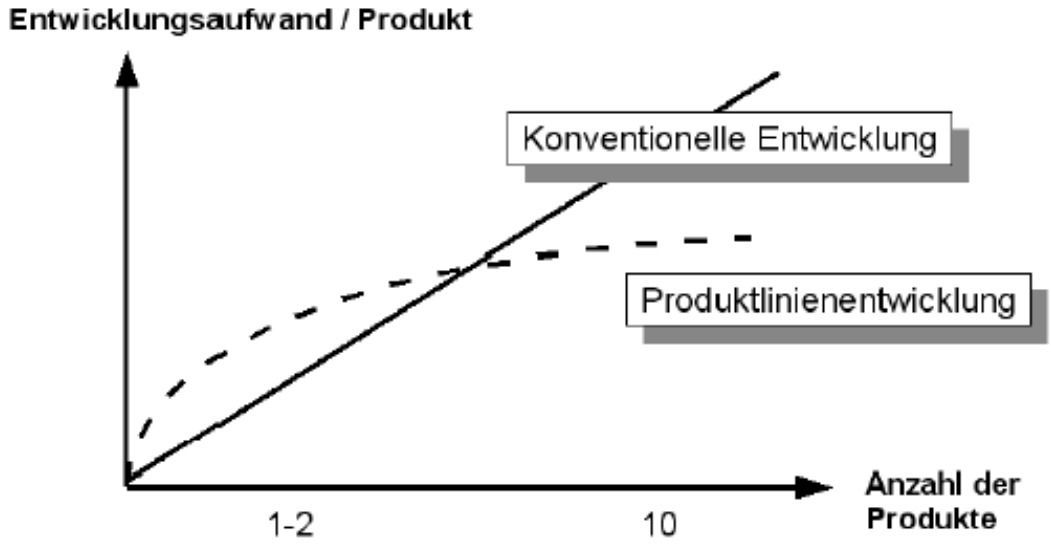
Abb. 2-9: Schematische Darstellung von Produktvariationen

Im Domänen Engineering werden die gemeinsamen und variablen Komponenten, die Bestandteile der Plattform werden, für eine oder mehrere Domänen entwickelt. In Abbildung 2-9 wurden der Mund und der Bauch als Domäne identifiziert. Diese können variabel an unterschiedliche Bedürfnisse wie zum Beispiel Geschäftsziele oder Kundenwünsche angepasst werden. Die Komponenten in der Domäne Mund können in diesem Beispiel entweder lachend, neutral oder traurig dargestellt werden. Das bedeutet, dass die Variabilität in den Anforderungen durch variable Szenarien, in der Architektur durch optionale Komponenten und in der Implementierung durch zum Beispiel bedingte Kompilierung realisiert wird (Vgl. Böckle et al. (2004), S.5).

Im Applikation Engineering (in der Abbildung 2-9 Product Engineering genannt) werden einzelne Produkte der Produktlinie abgeleitet bzw. entwickelt. Die Produkte werden aus den Komponenten der Plattform zusammengefügt. Falls eine Anpassung notwendig ist, kann diese in geringem Maße produktspezifisch geschehen. Falls zum Beispiel eine spezielle, einmalige Anforderung an das Produkt besteht, kann diese für dieses Produkt erstellt werden. In der Abbildung 2-9 entstehen dabei Produkte aus einem variablen Mund und einem variablen Körper (Vgl. Böckle et al. (2004), S.5f).

Die durch die Variabilität der im Domain Engineering zu produzierenden Produkte werden in einem Produktportfolio zusammengestellt, dass als Grundlage für die Verbesserung der Softwarewiederverwendung innerhalb der Produktlinie dient. Die Bestimmung des Produktraums wird dabei als Scoping bezeichnet. Wie in der Abbildung 2-8 auf Seite 24 dargestellt, beinhaltet das Scoping existierendes Wissen von Experten, die externe Sicht der Produktplanung und die interne Sicht der Entwicklungseigenschaften. Diese können jeweils angepasst werden, das heißt wenn aus den Produkten ersichtlich wird, dass eine Anforderung in der Domäne vergessen wurde, kann sie später noch hinzugefügt werden. Auch die Löschung unrelevanter Anforderungen kann mithilfe des Scopings erfolgen. Durch diesen ständigen Lernprozess sind die sequentiellen Vorgehensmodelle, die im Kapitel 2.1.1 vorgestellt wurden, nicht für die Software-Produktlinienentwicklung geeignet (Vgl. Böckle et al. (2004), S.7f).

Die Vorteile von Software-Produktlinien sind vielfältig. Zum einen lassen sich in kurzer Zeit Produkte auf den Markt bringen, das heißt die time-to-market kann deutlich reduziert werden. Zum anderen lässt sich das Ziel einer individualisierten Massenfertigung erreichen. Diese hängt nur von den Variabilitäten ab, welche im Domain Engineering modelliert wurden. Durch die Wiederverwendung der Produktkomponenten ist deren Qualität durch ständige Anpassung und Verbesserung höher als die vergleichbarer Einzelproduktkomponenten. Damit ist häufig auch die Qualität, der aus der Plattform abgeleiteten Produkte, höher. Weiterhin werden die Potentiale existierender Ressourcen erhöht, da diese in einer Produktlinie wesentlich besser genutzt werden können und, eingesetzt in vielen Produkten, den ROI (Return on Investment) steigern. Aber auch die Produktivität wird wesentlich gesteigert. Es können wesentlich mehr Produkte in kürzerer Zeit entwickelt werden, wenn Komponenten gemeinsam genutzt werden (Vgl. Rietdorf et al. (2005), S.12).



(Vgl. Besch (2007), S.4)

Abb. 2-10: Tendenz der Amortisierung einer Produktlinie

Doch neben den Vorteilen gibt es bei der Nutzung von Software-Produktlinien auch viele Dinge zu beachten. Der Aufbau einer gemeinsamen Infrastruktur beziehungsweise Plattform ist sehr zeit- und geldintensiv, was als Hauptkritikpunkt bei der Erstellung einer Produktlinie angesehen wird. Dies wird in Abbildung 2-10 veranschaulicht. Bei Einführung einer Software-Produktlinie ist der Entwicklungsaufwand für die ersten Produkte höher und somit auch kostenintensiver. Dieser anfängliche Nachteil durch den Aufbau der gemeinsamen Infrastruktur amortisiert sich nach einigen wenigen Produkten und wird durch den geringeren Entwicklungsaufwand für folgende Produkte zu einem wichtigen Vorteil gegenüber der konventionellen Entwicklung. Des Weiteren sollte im Falle einer Änderung der Anforderungen geprüft werden, ob nicht einzelne Produkte unkoordiniert modifiziert werden. Für jede Änderung muss eine explizite Entscheidung getroffen werden, ob die Produktlinie als Ganzes angepasst wird oder ob nur ein einzelnes Produkt modifiziert wird. Werden Entscheidungen falsch getroffen, kann es dazu führen, dass Kunden lange auf kleine Produktänderungen warten müssen, weil zunächst die Plattform aufgebaut wird oder weil Wiederverwendungspotentiale nicht ausgeschöpft werden, weil Änderungen nur an einzelnen Produkten und nicht an der wiederverwendbaren Plattform vorgenommen werden. Alle diese Aspekte sind stark miteinander verwoben, das heißt, dass eine Maßnahme an einer Stelle zu unerwarteten Auswirkungen an ganz anderer Stelle führen kann. (Vgl. Böckle et al. (2004), S.6)

3 Problemanalyse der Diplomarbeit

In diesem Kapitel werden als Erstes Praxisbeispiele für Software-Produktlinien gegeben, um zu verdeutlichen, dass Software-Produktlinien einem Unternehmen helfen können, Kosten und Ressourcen zu sparen. Danach wird das Fallbeispiel für diese Diplomarbeit vorgestellt und überprüft, ob sich eine Software-Produktlinie für das Fallbeispiel lohnt. Diese Potentialanalyse dient exemplarisch dafür, wie ein Unternehmen überprüfen kann, ob sich eine Software-Produktlinie für sie lohnt.

3.1 Praxisbeispiele für Software-Produktlinien

Es gibt viele Unternehmen, die schon erfolgreich Software-Produktlinien einsetzen und durch deren Einsatz sich Vorteile gegenüber ihren Mitbewerbern verschaffen. Im Folgenden werden einige exemplarisch kurz vorgestellt.

3.1.1 CelsiusTech

Die schwedische Firma CelsiusTech stellt die Software ShipSystem 2000 mithilfe einer Software-Produktlinie her. Das ShipSystem 2000 ist eine Marine-Kontroll-Software zur Steuerung und Überwachung von Schiffen. Durch die Einführung der Software-Produktlinie konnten Kosten- und Zeitpläne eingehalten, sowie Systemattribute wie Performanz vorausgesagt werden. Des Weiteren hat sich die Kundenzufriedenheit erhöht und einer der wichtigsten Punkte ist die Veränderung des Hardware-/Software-Kostenverhältnisses von 35:65 zu 80:20 (Vgl. Bass et al. (1998), S.2ff).

3.1.2 Market-Maker Software AG

Die in Kaiserslautern ansässige Firma Market-Maker hat im Jahr 1999 eine Software-Produktlinie zur Analyse des Aktienverlaufs im Internet mithilfe von Webservices hergestellt. Gründe für die Verwendung einer Software-Produktlinie waren die Interessen der Kostensenkung, weniger Wartungsarbeiten und eine schnellere time-to-market Situation zu schaffen. Durch die Einführung konnten die Entwicklungszeit für eine neue Software um 50% verkürzt und die Kosten um rund 70% gesenkt werden. Die Firma Market-Maker ist nur ein kleines Unternehmen, welches erfolgreich eine Software-Produktlinie eingeführt hat. Dieses Beispiel zeigt, dass Software-Produktlinien sich nicht nur bei größeren Firmen mit vielen tausend Mitarbeitern rentieren, sondern auch in kleinen und mittelständischen Unternehmen (Vgl. Gacek et al. (2001)).

3.1.3 Hewlett Packard

Das Owen Team ist ein Zusammenschluss von Entwicklungsteams, die für Hewlett Packard die Firmware herstellen. Sie produzieren die Firmware für eine große Anzahl von Druckern, Kopierern, Scannern und Faxgeräten. Diese Teams haben sich entschlossen

eine Software-Produktlinie einzuführen, da viele Kernelemente gleich sind. Seit dem Einsatz dieser Produktlinie werden nur noch 25% der Ressourcen und außerdem nur noch 1/3 der Zeit, um eine neue Firmware herzustellen, benötigt. Ein weiterer Vorteil seit Einsatz der Produktlinie ist, dass die Anzahl der Fehler um das 25fache zurückgegangen ist (Vgl. Toft et al. (2000)).

3.1.4 Nokia

Nokia produziert viele verschiedene Mobilfunktelefone. Es ist der weltweit größte Mobilfunktelefonhersteller und nach eigenen Angaben hat die eingeführte Software-Produktlinie geholfen, diese Position zu erreichen. Durch die Software-Produktlinie produziert Nokia 25 bis 30 neue Produkte pro Jahr mit unterschiedlichen Anzahlen von Tasten, unterschiedlichen Display-Größen, anderen unterschiedlichen Produktfunktionen, 58 verschiedenen unterstützten Sprachen, Kompatibilität mit früheren Produkten. Diese Vielzahl neuer Geräte mit solch variabler Funktionalitäten sind nur durch Software-Produktlinien und den damit verkürzten Zeiten zur Herstellung möglich (Vgl. Heie (2002))

3.2 Vorstellung des Fallbeispiels „GoGame“

In dieser Diplomarbeit wird ein Fallbeispiel zur Validation des Modells verwendet. Das Fallbeispiel beschäftigt sich mit einem jungen Start-Up-Unternehmen „GoGame“. Die Firma besteht aus wenigen engagierten, gerade diplomierten Studenten. Sie möchten ihr Hobby zum Beruf machen und Spiele für mobile Endgeräte herstellen. Die Mitarbeiter haben alle bereits Erfahrung im Umgang mit Java und den speziellen Anforderungen an die Spielentwicklung, wie zum Beispiel der Synchronisation zwischen dem Zeichnen und der eigentlichen Spiellogik.

Das Fallbeispiel in dieser Diplomarbeit beschäftigt sich mit einem Unternehmen, welches Spiele herstellt, da dies ein wachsender Markt ist und ein Ende des Wachstums nicht in Sicht ist. Während der Markt für mobile Spiele in Deutschland 2002 mit einem Umsatz von rund neun Millionen Euro fast nicht existierte (Vgl. Herrenbrück (2008), S.18f), zählte er 2007 schon etwa 57 Millionen. Im Jahr 2012 soll der Umsatz mit 105 Millionen Euro fast auf das Doppelte anwachsen. In Westeuropa steigen die Umsätze für mobile Spiele bis 2011 um circa ein Drittel von 1,5 auf zwei Milliarden US-Dollar. Diese Zahlen verdeutlichen, dass es sich lohnt in den Markt einzusteigen und Anwendungen dafür zu entwickeln (Vgl. Goldhammer et al. (2008), S.18ff).

Gründe für die Konzentration auf die Entwicklung von Spielen für mobile Endgeräte in diesem Fallbeispiel sind die speziellen Anforderungen, die ein Spiel an den Entwickler stellt. Während Anwendungen wie ein Kalender oder eine Uhr sich häufig mit einem

speziellen Problem, wie zum Beispiel dem Lesen und Schreiben von Dateien mit gleichzeitiger Anzeige befassen, müssen Spiele eine Vielzahl von Problemen lösen. Die Darstellung von Objekten mit einer Kollisionserkennung benötigt ein gutes mathematisches Grundverständnis. Wenn ein Spiel dreidimensional dargestellt wird, werden diese Berechnungen komplex. Der Programmierer benötigt nicht nur ein mathematisches Verständnis, sondern er muss die Lösung so effizient gestalten, dass das Spiel flüssig läuft. Ein Wegfindungsalgorithmus ist ein gutes Beispiel für dieses Problem. Während der Algorithmus für ein Objekt im Spiel akzeptabel läuft, kann es zu Problemen bei sehr vielen Objekten im Spiel kommen. Die Programmierer müssen kreativ an dieses Problem herangehen. Des Weiteren sollte ein Spiel flüssig laufen. Das heißt das Zeichnen und die ganze Spiellogik muss in einer kurzen Zeit verarbeitet werden. Dazu kann Multithreading benutzt werden oder der Programmierer muss geschickt die Logik mit dem Zeichnen verbinden. Eine weitere Herausforderung bei der Programmierung ist die Tatsache, dass die Spiele eine Hintergrundstory besitzen sollten, damit der Nutzer sein Interesse am Spiel nicht verliert. Auch eine ansprechende Grafik ist ein entscheidender Punkt für den Erfolg eines Spiels. Bei der Spielmechanik ist wieder kreative Leistung gefragt. Ein neuer Tetris-Klon wird höchstwahrscheinlich kein Verkaufsschlager. All diese aufgeführten Punkte sollen zeigen, dass die Entwicklung von Spielen ein breites Spektrum an Fähigkeiten erfordert. Deshalb möchte das Fallbeispielunternehmen „GoGame“ Spiele entwickeln.

Ein weiterer Grund für die Spezialisierung auf Spiele in dieser Diplomarbeit ist die Tatsache, dass durch die speziellen, genannten Anforderungen an die Entwickler die Spiele zu einer besonderen Anwendung machen. Spiele sind somit eine kritische Fallstudie (Vgl. Flyvbjerg (2006), S.11f), die einen besonders schweren Fall darstellen. Andere mobile Anwendungen, wie die genannte Uhr oder der Kalender, lassen sich auf die gleiche Art und Weise, wie in dieser Diplomarbeit beschrieben, umsetzen, müssen jedoch unter Umständen einige Schritte nicht in der Ausführlichkeit umsetzen. Sie können leichter und schneller umgesetzt werden.

3.3 Überprüfung des Potentials der Firma „GoGame“ für Software-Produktlinien

Als Erstes soll untersucht werden, ob Spiele die Kriterien für eine erfolgreiche Software-Produktlinie erfüllen. Die Entscheidung für die Organisation, ob sich die zu entwickelnde Anwendung mithilfe einer Software-Produktlinie darstellen lässt, kann durch folgende Kriterien überprüft werden. Deshalb werden als Erstes die Kriterien allgemeingültig genannt, mit den Kriterien für mobile Spiele verglichen und letztlich nach der Machbarkeit überprüft. Für andere mobile Anwendungen muss diese Überprüfung des Potentials mit jeweils den eigenen Kriterien der Anwendung neu stattfinden. Nur

wenn die Potentialanalyse erfolgreich war, lohnt sich der Einsatz einer Software-Produktlinie. Ansonsten muss das Unternehmen einen anderen Ansatz suchen.

An erster Stelle stehen die wesentlichen Kriterien. Diese sind notwendig und müssen zutreffen, um das Potential von Software-Produktlinien zu nutzen (Vgl. Dalgarno (2008)):

- **Mehr als ein Produkt muss entwickelt oder geplant werden**. Wenn ein Unternehmen nur ein Produkt ohne Variationen plant, sind Software-Produktlinien nicht einsetzbar. Die Firma „GoGame“ plant aber viele Spiele herzustellen.

- **Die Produkte müssen gemeinsame funktionelle Anforderungen besitzen**. Wenn die Produkte, die hergestellt werden sollen, keine gemeinsamen funktionellen Anforderungen haben, wird es für das Unternehmen unmöglich, wiederverwendbare Werte zu finden und somit sind Produktlinien nicht einsetzbar. Spiele besitzen hingegen viele funktionelle Gemeinsamkeiten, sei es die Grafikausgabe, die Spiellogik oder das Verarbeiten von Tastatureingaben. Das wird in jedem Spiel benötigt. Da die Firma „GoGame“ vor allem Puzzle-Spiele herstellen möchte, lassen sich noch mehr wiederverwendbare Teile finden.

- **Produkte müssen gleiche Qualitätsvoraussetzungen besitzen.** Neben den funktionellen Anforderungen spielen auch die Qualitätsvoraussetzungen eine wichtige Rolle. Wenn diese nicht übereinstimmen, kann keine Software-Produktlinie eingesetzt werden. Im Fallbeispiel stimmen die Qualitätsvoraussetzungen immer überein, denn selbst wenn ein Spiel sehr schnell hergestellt wird, nutzt es die gleichen Voraussetzungen wie ein Spiel, das mit Bedacht entwickelt wurde. Dieser Punkt ist für andere Produkte interessant.

Ferner sind die unterstützenden Kriterien zu beachten. Sie geben einen Hinweis darauf, dass Software-Produktlinien von Nutzen sein können, aber sind keine zwingende Voraussetzung (Vgl. Dalgarno (2008)):

- **Der gleiche Teil einer Software wird schon in mehr als in einem Produkt verwendet**. Falls ein Unternehmen schon Software wiederverwendet, kann dies als positives Anzeichen für die Verwendung von Software-Produktlinien gedeutet werden. Auch Spiele haben viele Teile, die in nahezu jedem Spiel verwendet werden können. Zum Beispiel besitzt ein Spielobjekt x, y und gegebenenfalls z Koordinaten zur Ortung seiner Position.

- **Das Unternehmen hat Qualitätsprobleme**. Einer der Vorteile des Produktlinien-Ansatzes ist die erhöhte Produktqualität und die Steigerung der Kundenzufriedenheit.

Doch der time-to-market Druck kann dazu führen, dass viele Firmen die Qualität vernachlässigen. Eine Produktlinie kann dort Abhilfe schaffen.

- **Das Unternehmen hat Komplexitätsprobleme**. Eine Software-Produktlinie kann helfen komplexe Sachverhalte besser zu verarbeiten. Durch die in Kapitel 3.1 aufgezeigte Komplexität bei der Entwicklung von Spielen, ist dies ein Punkt, der für die Entwicklung einer Software-Produktlinie steht.

- **Das Unternehmen plant sein Produkt zu variieren**. Wenn ein Unternehmen ein bestehendes Produkt auf Grund von Marktforschung und/oder Kundenwünschen zu variieren plant, dann bietet sich eine Software-Produktlinie an. Durch das wachsende Bedürfnis stärker auf den Kunden einzugehen und seine Produkte zu variieren, bietet sich die Produktlinie an, da mit ihrer Hilfe ein neues Produkt schnell und kostengünstiger auf den Markt gebracht werden kann.

Schließlich sind Ausschlusskriterien zu nennen. Diese weisen darauf hin, dass das Potential einer Produktlinie beschränkt ist (Vgl. Dalgarno (2008)):

- **Das Produkt muss in einem unstabilen Markt bestehen**. Wenn der Markt schrumpft oder sein Potential nicht bekannt ist, kann es passieren, dass die Amortisierung der Kosten für die Anschaffung einer Produktlinie nicht erreicht wird. In Kapitel 3.1 wurde aber darauf hingewiesen, dass der Markt für mobile Spiele ein steigender ist.

- **Es gibt einen technologischen Wandel**. Es wird schwierig für ein Produkt die Kernelemente herausfinden, wenn die Technologie sich ändert. Bei mobilen Spielen gibt es durch die Weiterentwicklungen schon einen technologischen Wandel, aber es werden Beschränkungen aufgehoben, sodass der Kernbereich unangetastet bleibt, erweitert werden kann und nicht komplett neu entwickelt werden muss.

- **Neue Produkte werden selten entwickelt**. Damit sich eine Software-Produktlinie amortisieren kann, ist es notwendig eine bestimmt Anzahl von verschiedenen Produkten in seinem Produktportfolio zu besitzen. Mobile Spiele werden schnell entwickelt und so kann sich eine Software-Produktlinie über die große Anzahl von Produkten amortisieren.

- **Das Unternehmen entwickelt spezifische, in Auftrag gegebene Produkte**. Eine Software-Produktlinie erfordert die Konzentration auf einen bestimmten Markt mit verschiedenen wiederverwendbaren Werten. Wenn spezifische, in Auftrag gegebene Produkte gewünscht sind, kann das Unternehmen sich nicht auf den Markt konzentrieren und seine identifizierten Features wieder verwenden. Auch ein Unternehmen

das Spiele für den mobilen Markt herstellt, kann Aufträge von Großfirmen bekommen. Falls die gewünschten Spiele sehr verschieden sind, muss geprüft werden, ob der identifizierte wiederverwendbare Teil groß genug ist, damit sich die Software-Produktlinie amortisiert. Wenn zum Beispiel immer das gleiche Genre gewünscht wird, kann sich eine Software-Produktlinie sehr wohl lohnen.

- **Software ist nur ein kleiner Teil des Produktes**. Falls das Produkt nur aus einem kleinen Teil aus Software besteht, sollte das Unternehmen prüfen, ob es sich nicht lohnt, andere Produktionsprozesse zu verbessern und nicht den Software-Entwicklungsprozess. Bei der Entwicklung von mobilen Spielen stellt sich diese Frage nicht, da die Software bei einem Spiel den größten Teil der Kosten ausmacht.

Als Fazit kann gesagt werden, dass es auf der Grundlage dieser Analyse es möglich ist, den Rückschluss zu ziehen, dass Software-Produktlinien sich für mobile Spiele eignen.

Das Fallbeispielunternehmen „GoGame“ beginnt somit seine Planung auf dem „Grüne Wiese“-Ansatz. Bei dieser Planungsart, bei der von Grund auf, ohne belastende Voraussetzungen und Rahmenbedingungen geplant werden kann, planen die Geschäftsführer gedanklich ein Unternehmen von Grund auf neu. Die Unternehmensleitung plant langfristig und möchte eine Software-Produktlinie für ihre Spiele einführen. Deshalb analysieren sie als Erstes die bestehenden Modelle, um das passende Modell zu finden und es gegebenenfalls anzupassen.

4 Analyse von Software-Produktlinienmodellen

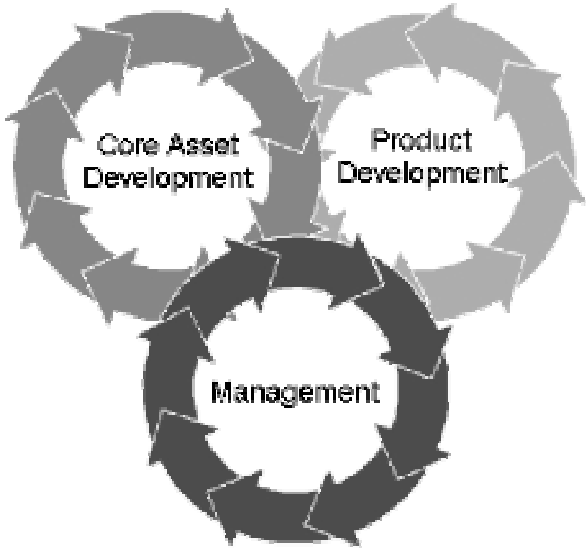
In diesem Kapitel geht es um die Vorstellung der wichtigsten Prozesse und Aktivitäten in einer Software-Produktlinie. Darauf aufbauend werden vier Modelle vorgestellt, die sich in ihrem Abstraktions- und Umsetzungsgrad des Referenzmodells unterscheiden. Danach folgen Implementierungsansätze und Methoden und Werkzeuge, die für die Umsetzung der Modelle benötigt werden. Diese werden kurz genannt und erläutert und auf Grundlage des Ziels der Diplomarbeit zur Entwicklung für mobile Endgeräte Empfehlungen erstellt, welche Werkzeuge genutzt werden sollten, damit eine Software-Produktlinie erfolgreich eingesetzt werden kann.

4.1 Die drei Gruppen von Hauptaktivitäten

Das *Software Engineering Institute* (SEI) hat drei Gruppen von Hauptaktivitäten, siehe Abbildung 4-1, klassifiziert, die für eine erfolgreiche Software-Produktlinie notwendig sind (Vgl. SEI (2009)):

- Core Asset Development
- Product Development
- Management

Diese Aktivitäten stellen einen iterativen Prozess und ein nicht sequentielles Vorgehen, siehe Kapitel 2.1.2 auf Seite 8, dar. In Abbildung 4-1 wird die iterative Herangehensweise durch die kreisrunden Pfeile verdeutlicht.



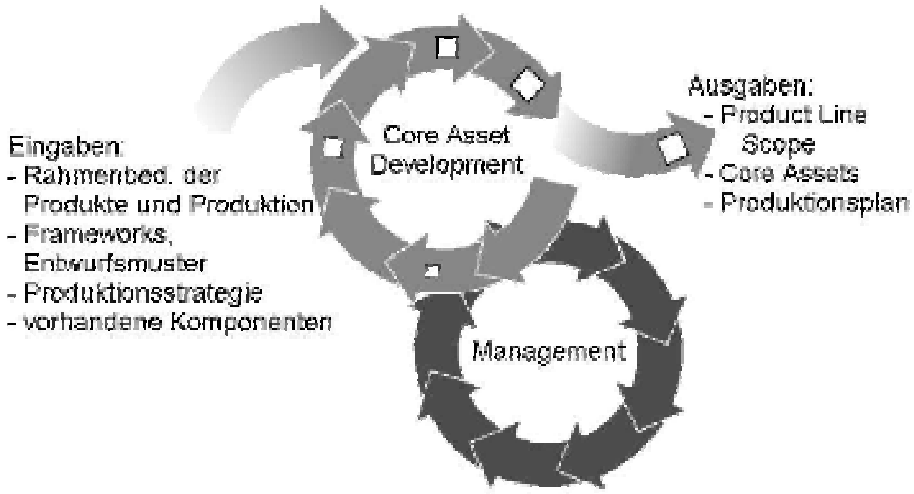
(Vgl. SEI (2009))

Abb. 4-1: Three Essential Activities

Alle drei Aktivitäten sind essentiell für die Software-Produktlinie im Unternehmen und nur eine Mischung der Aktivitäten verspricht langfristigen Erfolg. Die Umsetzung der Aktivitäten im Unternehmen kann von Firma zu Firma unterschiedlich gewichtet sein. Viele Unternehmen beginnen bei der Entwicklung der Core Assets und nehmen somit den so genannten proaktiven Ansatz. Diese definieren eine Familie von Systemen, welche ihre Software-Produktlinie darstellt. Somit wird gleich der Bereich der Produktlinie ermittelt und alle Produkte, die produziert werden sollen, werden aus diesem Bereich hergestellt. Ein anderer Ansatz ist der reaktive. In diesem beginnen die Unternehmen mit einem oder sehr wenigen Produkten, die sie schon auf dem Markt anbieten und generieren daraus die Core Assets und ihre zukünftigen Produkte (Vgl. Krueger (2001), S. 282ff).

4.1.1 Core Asset Development

Für das Core Asset Development werden Core Assets benötigt. Sie sind die Grundbausteine der Produktlinie, aus denen die jeweiligen Produkte erstellt werden. Damit umfasst das Core Asset Development alle Aktivitäten, die sich mit der Erstellung und Erweiterung der Core Assets befassen. Diese Aktivität wird in Abbildung 4-2 dargestellt.



(Vgl. SEI (2009))

Abb. 4-2: Core Asset Development

Core Assets können mithilfe einer Top Down oder einer Bottom Up Strategie identifiziert werden. Bei der Top Down Strategie werden die benötigten Funktionalitäten durch das Analysieren der geplanten Produkte ermittelt und dementsprechend Assets erstellt. Bei der Bottom Up Strategie wird mithilfe der Core Assets ermittelt, welche Produkte

hergestellt werden können. In der Praxis hat sich eine Mischung aus beiden Strategien bewährt (Vgl. Biermann (2004), S.4).

Wie in Abbildung 4-2 dargestellt verlangt das Core Asset Development eine Reihe von Eingaben und unterliegt dabei nachfolgenden Einschränkungen. Als Erstes werden die Rahmenbedingungen der Produkte konkretisiert. Diese beschreiben die Gemeinsamkeiten aller Produkte und welche Standards und welche Qualitätsanforderungen bei der Erstellung einzuhalten sind. Dabei wird in der Praxis häufig auf Frameworks, Entwurfsmuster oder Expertenwissen zurückgegriffen. Weitere Einschränkungen entstehen durch die Rahmbedingungen der Produktion, wie zum Beispiel die spezifischen Auflagen des Unternehmens bezüglich der time-to-market-Kriterien oder der vorhandenen Infrastruktur. Da eine Software- Produktlinie selten auf dem Grüne-Wiese-Ansatz entwickelt wird, schlägt die SEI vor, bereits vorhandene Komponenten im Unternehmen daraufhin zu untersuchen, ob diese zu Assets umgewandelt werden können (Vgl. Biermann (2004), S.4).

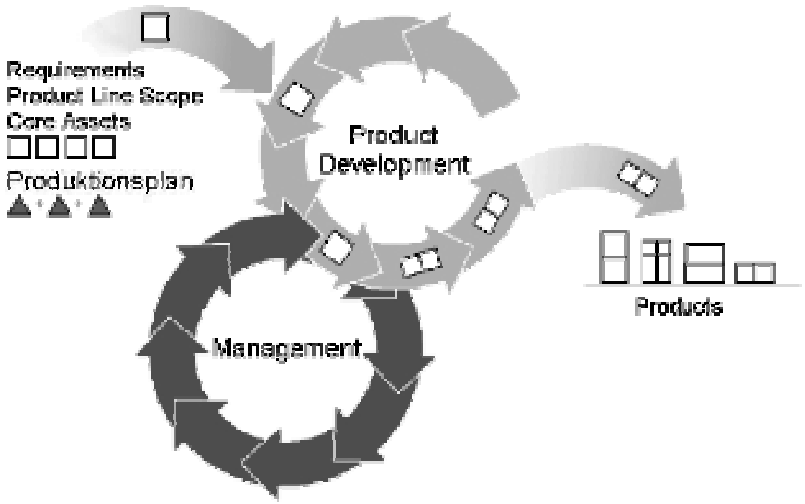
Der Output aus dem Core Asset Development sind die Core Assets, der Product Line Scope und der Produktionsplan. Der Product Line Scope enthält eine detaillierte Beschreibung der Produkte, die aus den vorhandenen Core Assets entwickelt werden können. Jedes Core Assets besitzt einen so genannten „Attached Process“, in dem dokumentiert wird, welche Aktivitäten angestoßen werden müssen, um dieses Core Asset planmäßig zu verwenden. Diese Attached Processes zusammen ergeben den Bauplan für die Produkte (Vgl. SEI (2009)).

4.1.2 Product Development

Beim Product Development werden aus den Core Assets die einzelnen Produkte hergestellt. Abbildung 4-3 veranschaulicht diesen Prozess. Die drehenden Pfeile zeigen wieder das iterative Vorgehen. Zum Beispiel kann bei der Produktion eines Produktes festgestellt werden, dass dies allgemeingültige Funktionalitäten enthält, die im Core Asset Development noch nicht erfasst wurden. In diesem Fall müssen die Core Assets überprüft werden. Der Input, welcher für die Produktentwicklung benötigt wird, ist folgender (Vgl. SEI (2009)):

- Die Produktbeschreibung für ein bestimmtes Produkt, welches häufig nur eine Variabilität der generischen Beschreibung des Product Line Scope ist.
- Der Product Line Scope, der die Praktikabilität eines Produktes innerhalb der Produktfamilie angibt
- Die Core Assets, mithilfe derer das Produkt hergestellt wird

- Der Produktionsplan, der vorgibt, wie die Core Assets zu nutzen sind



(Vgl. SEI (2009))

Abb. 4-3: Product Development

Bei der Erstellung der Produkte müssen vorhandene Mängel und Probleme dem Management gemeldet werden. Durch dieses Feedback bleibt der Core Assets Kern aktuell und entwicklungsfähig.

Der geplante Output beim Product Development ist das eigentliche Produkt.

4.1.3 Management

Der Prozess des Managements ist an allen Prozessen während der Entwicklung und Produktion beteiligt. Es wird zwischen dem technischem und organisatorischem Management unterschieden (Vgl. Biermann (2004), S.4).

Das technische Management überwacht die Entwicklung der Core Assets und der Produkte und muss sicherstellen, dass die geforderten Prozessschritte korrekt ausgeführt werden und dass die nötigen Daten geliefert werden, um den Fortschritt zu überwachen.

Das organisatorische Management ist für die Verteilung der Ressourcen verantwortlich, pflegt die Kontakte zum Kunden, ist für die allgemeine Planung zuständig und legt somit die Ziele und Strategien der Software-Produktlinie im Unternehmen fest. Das Management muss die Risiken, die die Produktlinie bedrohen, frühzeitig erkennen und versuchen entgegenzuwirken. Somit ist das Management ein kritischer Prozess der von erfahrenen und engagierten Personen zu besetzen ist (Vgl. SEI (2009)).

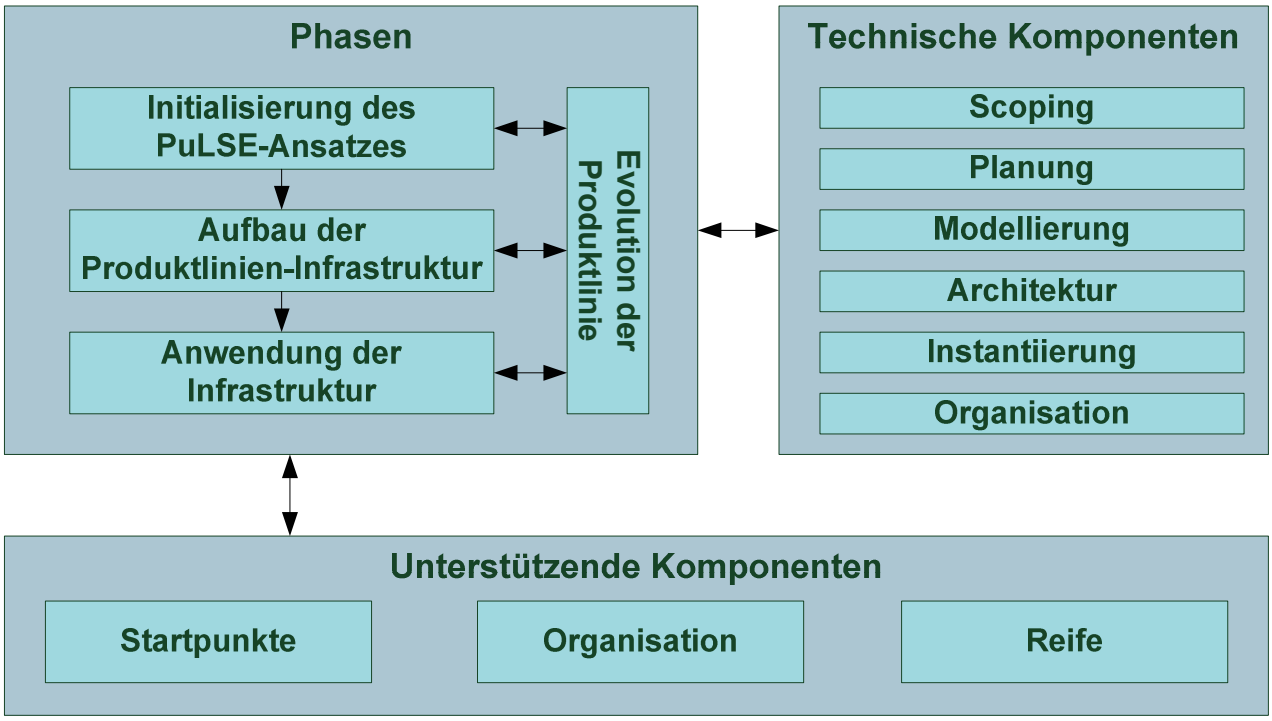
4.2 Modelle zur Software-Produktlinienentwicklung

Aus dem vorgestellten Referenzmodell haben sich über die letzten Jahre mehrere spezifische Modelle entwickelt. Vier ausgewählte Modelle werden im Folgenden kurz vorgestellt und dabei wird auf ihre Eigenheiten näher eingegangen. Die Reihenfolge der Vorstellung ist nach dem Umsetzungsgrad des Referenzmodells geordnet, beginnend mit dem Modell, welches das Referenzmodell am umfangreichsten abdeckt.

4.2.1 PuLSE

PuLSE steht für *Product Line Software Engineering* (PuLSE) und wurde von einem Fraunhofer Team entwickelt. Es deckt dabei alle Bereiche des Referenzmodells ab, vom Scoping, über das Domänen Engineering bis zum Application Engineering (Vgl. Bunse et al. (2008), S.99).

Ein Hauptziel des PuLSE-Ansatzes ist die Etablierung einer Produktlinienorganisation in einem Unternehmen, angefangen von der Initialisierung des Ansatzes bis zur praktischen Anwendung der Infrastruktur. Es bietet somit einen Ablaufplan, wie eine Produktlinie in einem Unternehmen aufgebaut und umgesetzt werden kann. Ein Vorteil des Ansatzes ist der modulare Aufbau, wie in Abbildung 4-4 dargestellt, der eine Anpassung an eigene Wünsche und Bedürfnisse erlaubt (Vgl. Bunse et al. (2008), S.99).



(Vgl. Rombach (2003), S.3)

Abb. 4-4: PuLSE-Modell

Wie in Abbildung 4-4 zu sehen, basiert PuLSE auf drei Kernelementen, den Phasen, den technischen Komponenten und den unterstützenden Komponenten.

Die Phasen bestehen aus drei Schritten, die von einer weiteren Phase durchgehend unterstützt werden. Der Initialisierung des Ansatzes, den Aufbau der Infrastruktur und die Anwendung der Infrastruktur dienen zur Umsetzung der Produktlinie, während die Evolution der Produktlinie diese drei Phasen unterstützt, das heißt sie durch Erfahrungsgewinne beständig verändern und verbessern kann.

Die technischen Komponenten unterstützen die Phasen und kommen unterschiedlich stark zum Einsatz. Im Folgenden werden die Funktionen der einzelnen Komponenten kurz beschrieben (Vgl. Bunse et al. (2008), S.99f):

- Das Scoping unterstützt die Bereitstellung wieder verwendbarer Elemente. Dabei werden vor allem ökonomische Fragen untersucht, aber auch Fragen, wie zum Beispiel ‚Welche Produkte passen in die Produktlinie?' geklärt.

- Die Planung ist eine unterstützende Komponente für die Phase der Initialisierung.

- Aufgabe der Modellierung ist die Beschreibung der Produktcharakteristika und die Definition der Mitglieder der Produktlinie.

- Die Architektur beschäftigt sich mit der Erstellung einer Referenzarchitektur für die Produktlinie.

- Die Instantiierung kommt in der Phase der Anwendung der Infrastruktur zum Einsatz und unterstützt den Produktiv-Einsatz der Produktlinie.

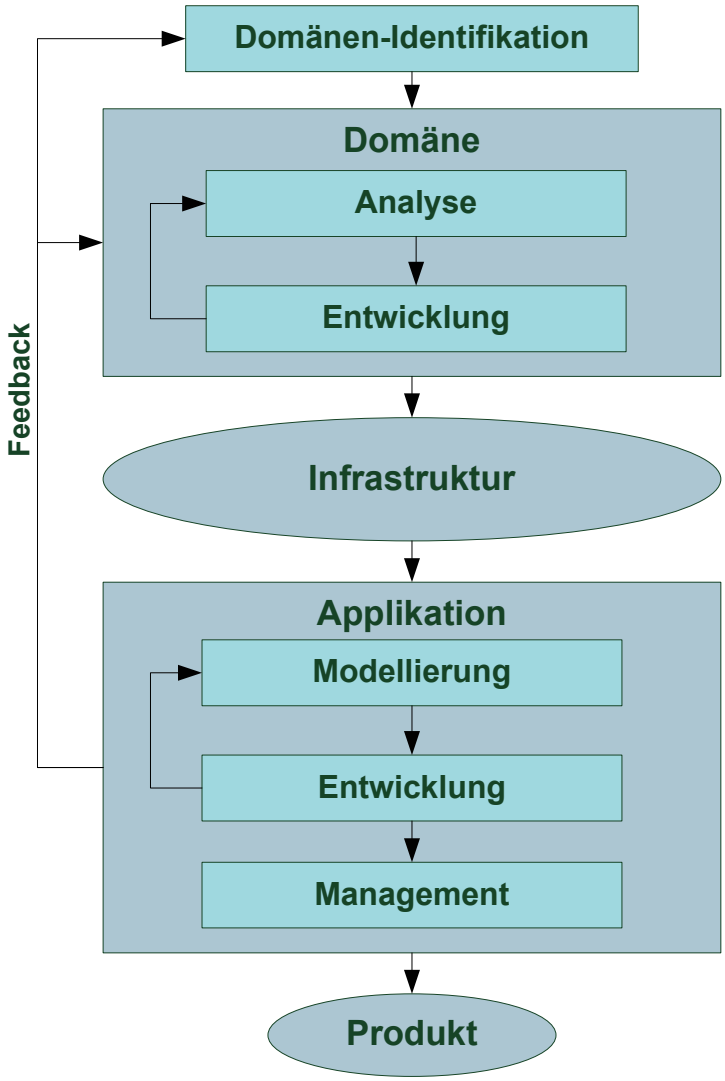
- Die Organisation unterstützt die Evolution der Produktlinie und beschäftigt sich mit Fragen der Konfigurationsverwaltung und der Evolution.

Die unterstützenden Komponenten des PuLSE-Ansatzes bestehen aus drei Komponenten und sind für die Adaption, Evolution und Bereitstellung der Software-Produktlinie entwickelt worden. Die Komponente ‚Starpunkte' befasst sich mit der Anpassung von PuLSE an vorhandene Software-Projekte Die Organisations-Komponente gibt Richtlinie vor um eine Software-Produktlinie in einem Unternehmen zu etablieren und anzupassen. Die Reife-Komponente enthält Einführungsstrategien und unterstützt damit die schrittweise Einführung des PuLSE-Ansatzes in ein Unternehmen (Vgl. Bunse et al. (2008), S.100).

Es gibt derzeit aber noch kein Werkzeug das PuLSE in seiner gesamten Breite abdeckt. Einzelne Phasen und Komponenten werden unterstützt, die anderen müssen manuell durchgeführt werden (Vgl. Bunse et al. (2008), S.103).

4.2.2 FAST

FAST steht für *Family-Oriented Abstraction, Specification and Translation* (FAST) und wurde von David Weiss vorgestellt und von der Firma Lucent entwickelt (Vgl. Harsu (2002), S.3).



(Vgl. Weiss et al. (1999))

Abb. 4-5: FAST-Modell

FAST besitzt drei zentrale Elemente, den Aktivitätsbaum, den Artefaktbaum und ein hierarchisches Rollenmodell (Vgl. Harsu (2002), S.3f).

Der Aktivitätsbaum enthält und verwaltet alle Aktivitäten, die für die Entwicklung von und mit Produktlinien von Bedeutung sind, der Artefaktbaum definiert alle dabei entstehenden Dokumente und das hierarchische Rollenmodell beschreibt die erforderlichen Rollen in FAST sowie deren Verantwortlichen. Das *Process and Artifact State Transition Abstraction-Modell* (PASTA-Modell) beschreibt die Vor- und Nachbedingungen jeder Aktivität und unterstützt somit jede Aktivität im FAST-Modell (Vgl. Bunse et al. (2008), S.108).

Abbildung 4-5 auf Seite 40 veranschaulicht das FAST-Modell. Es wird grundsätzlich in zwei Schritte unterteilt. Zum einen in die Bereitstellung der Infrastruktur und zum anderen zur Nutzung der Infrastruktur zur Entwicklung von Anwendungen (Vgl. Bunse et al. (2008), S.109).

Im ersten Schritt des FAST-Ansatzes, wie in Abbildung 4-5 dargestellt, wird die Domäne identifiziert. Dabei wird geklärt, ob die Erstellung einer Produktlinie nach ökonomischen Gesichtspunkten sinnvoll ist oder das Projekt abgelehnt werden sollte (Vgl. Bunse et al. (2008), 110).

Im zweiten Schritt folgt die Analyse und Entwicklung der Domäne. In der Domänenanalyse und -entwicklung wird ein Entscheidungsmodell entwickelt und die Gemeinsamkeiten und Unterschiede der Systeme beschrieben. Wichtiger Unterschied zu anderen Produktlinien-Modellen ist die Annahme, dass alle Variabilitäten mithilfe eines Parameters beschrieben werden können. Dadurch können Anwendungen durch eine Parameterdatei erstellt werden. Danach folgt die Implementierung der Infrastruktur, die zur Entwicklung einzelner Systeme dient (Vgl. Bunse et al. (2008), S.111).

Der zweite große Schritt des FAST-Ansatzes ist die Applikationsentwicklung. Ziel ist die Erstellung von Anwendungen unter Nutzung der entwickelten Domäne. Als Grundlage dient die, im ersten großen Schritt entwickelte, Infrastruktur. Die eigentliche Anwendungsentwicklung erfolgt iterativ. Dies entspricht dem nicht sequentiellen Vorgehensmodell aus Kapitel 2.1.2. Die Anforderungen an das System werden durch die Entwickler in Form eines Modells immer weiter spezifiziert und verfeinert. Dies wird so lange wiederholt bis das Modell genau den Wünschen des Abnehmers entspricht. Darauf aufbauend wird die Anwendung erstellt und validiert (Vgl. Bunse et al. (2008), S.112).

Neben den Entwicklungsaktivitäten besitzt das FAST-Modell Aktivitäten zur Evolution der Domäne und zur Projektverwaltung. Die Projektverwaltung beinhaltet die organisatorische Sicht auf alle Aktivitäten zur Identifizierung und Erfüllung der Anforderungen. Die Evolution der Domäne beinhaltet sowohl die Änderungen als auch Verbesserungen

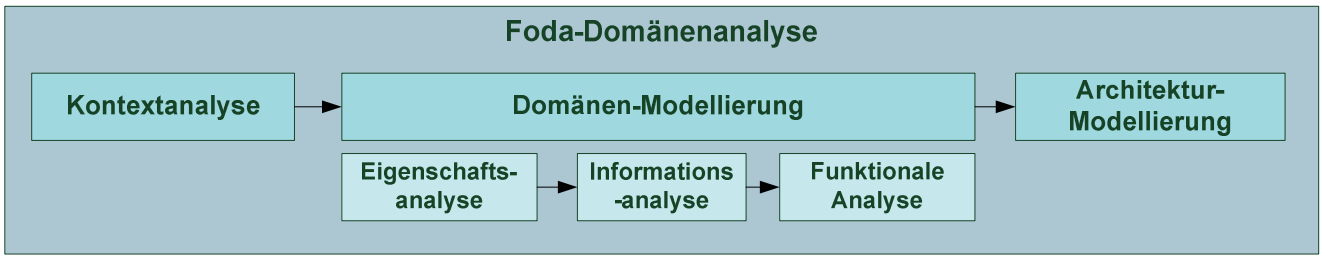
der Infrastruktur. Dazu wird das am Anfang des Abschnitts erwähnte PASTA-Modell benutzt (Vgl. Bunse et al. (2008), S.112ff).

Essentiell für die Entwicklung mit FAST ist die Werkzeugunterstützung. Der Ansatz sieht eine Eigenentwicklung der Werkzeuge vor, um eine funktionierende Infrastruktur zu etablieren (Vgl. Bunse et al. (2008), S.113).

4.2.3 FODA

FODA steht für *Feature Oriented Domain Analysis* (FODA) und wurde 1990 am SEI der Carnegie Mellon Universität entwickelt (Vgl. Kang et al. (1998), S.3ff). FODA ist heute ein Teil des Product Line Practice-Rahmenwerks der SEIs (Vgl. Bunse et al. (2008), S.104).

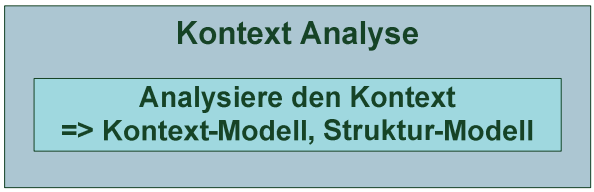
Das Hauptaugenmerk des FODA-Ansatzes ist die Domänenanalyse. Diese ist eine Komponente des Domain Engineering im Referenzmodell, siehe Abbildung 2-8 auf der Seite 24. Notwendige andere Elemente wie die Infrastruktur oder das Applikation-Engineering werden nicht untersucht und unterstützt. Um diese Lücke zu schließen, wurde FODA um zwei weitere Methoden ergänzt, der *Feature-Oriented Reuse Method* (FORM) und des *Feature-oriented Product-Line Engineering* (FOPLE). Die Erweiterungen, FORM und FOPLE, vervollständigen damit die Prozessbeschreibung. FORM (Vgl. Kang et al. (1998)) dient der Unterstützung der Architekturmodellierung. FOPLE (Vgl. Kang et al. (2002), S.3ff) unterstützt die Kontextanalyse, indem es sich mit der Durchführung einer Marktanalyse und Erstellung einer Marketing-Strategie beschäftigt (Vgl. Bunse et al. (2008), S.104f).



(Vgl. Kang et al. (1990), S.13)

Abb. 4-6: FODA-Modell

Wie in Abbildung 4-6 dargestellt, gliedert sich das FODA-Modell in drei Phasen. Die drei Phasen - Kontextanalyse, Domänenmodellierung und Architektur - werden im Folgenden näher beschrieben.

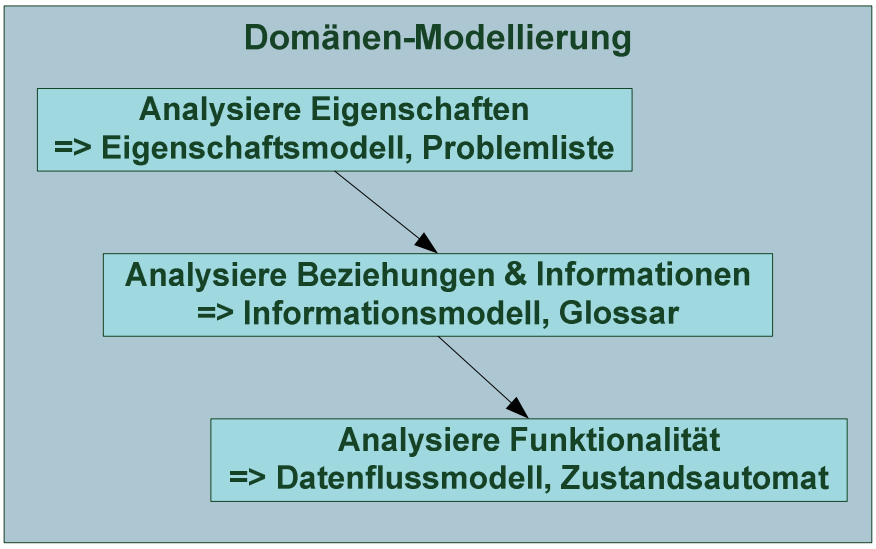


(Vgl. Bunse et al. (2008), S.105)

Abb. 4-7: FODA-Kontext Analyse

Der erste Schritt des FODA-Modells ist die Kontextanalyse. In dieser Phase wird die Domäne analysiert. Es werden Fragen nach dem Umfang und den Grenzen der Domäne geklärt und die Beziehungen zu anderen Domänen identifiziert und dokumentiert (Bunse et al. (2008), S.105).

Aus der Kontextanalyse entstehen, wie in Abbildung 4-7 abgebildet, zwei Produkte. Das Kontextmodell beschreibt den Informationsaustausch mit verwandten Domänen und das Strukturmodell gibt die Struktur aller Domänen wieder. Es positioniert zu diesem Zweck die einzelnen Domänen und fasst es in eine Struktur zusammen, wie zum Beispiel einem Blockdiagramm (Vgl. Bunse et al. (2008), S.105).



(Vgl. Bunse et al. (2008), S.106)

Abb. 4-8: FODA-Domänen Modellierung

Auf Grundlage der Kontextanalyse startet die Domänenmodellierung. Diese gliedert sich, wie in Abbildung 4-8 dargestellt, in drei Aktivitäten. Ziel der Domänenmodellierung ist die Identifikation von Gemeinsamkeiten und Variabilitäten der einzelnen Systeme.

Die erste Aktivität ist die Eigenschaftsanalyse. Dies ist einer der Schwerpunkte des FODA-Ansatzes und umfasst die Analyse der Eigenschaften, die ein System in einer Domäne bereitstellen muss beziehungsweise sollte. Die Eigenschaften werden hierarchisch in einem Eigenschaftsmodell repräsentiert (Vgl. Bunse et al. (2008), S.104).

Die zweite Aktivität ist die Informationsanalyse, deren Ziel es ist, das Domänenwissen und die Datenanforderungen als Entitäten mithilfe eines ER-Diagramms darzustellen (Vgl. Bunse et al. (2008), S.106).

Die dritte Aktivität während der Domänen-Modellierung ist die Funktionsanalyse. Das Ziel dabei sind die funktionalen Überschneidungen und Unterschiede zwischen den Anwendungen einer Domäne aufzuzeigen. Dazu werden die vorher identifizierten Informationen abstrahiert und strukturiert (Vgl. Bunse et al. (2008), S.106f).

(Vgl. Bunse et al. (2008), S.107)

Abb. 4-9: FODA-Architektur

Der letzte Schritt des FODA-Modells ist die Architektur. Dieser Schritt wird wie beschrieben nur sehr oberflächlich behandelt. Ziel der Architektur, wie in Abbildung 4-9 dargestellt, ist die Entwicklung eines Architektur-Modell mithilfe der Beschreibung der Domäne auf einem hohen Abstraktionsniveau. Durch die Erweiterung FORM kann dies näher spezifiziert werden (Vgl. Bunse et al. (2008), S.107).

Durch die langjährige Erfahrung und Unterstützung des SEIs gibt es von der SEI unterstützende Werkzeuge für die Modellierung des FODA-Ansatzes. Diese Werkzeuge unterstützen die bekannten Darstellungsformen und Notationen, wie zum Beispiel Datenflussdiagramme oder ER-Diagramme (Vgl. Bunse et al. (2008), S.107).

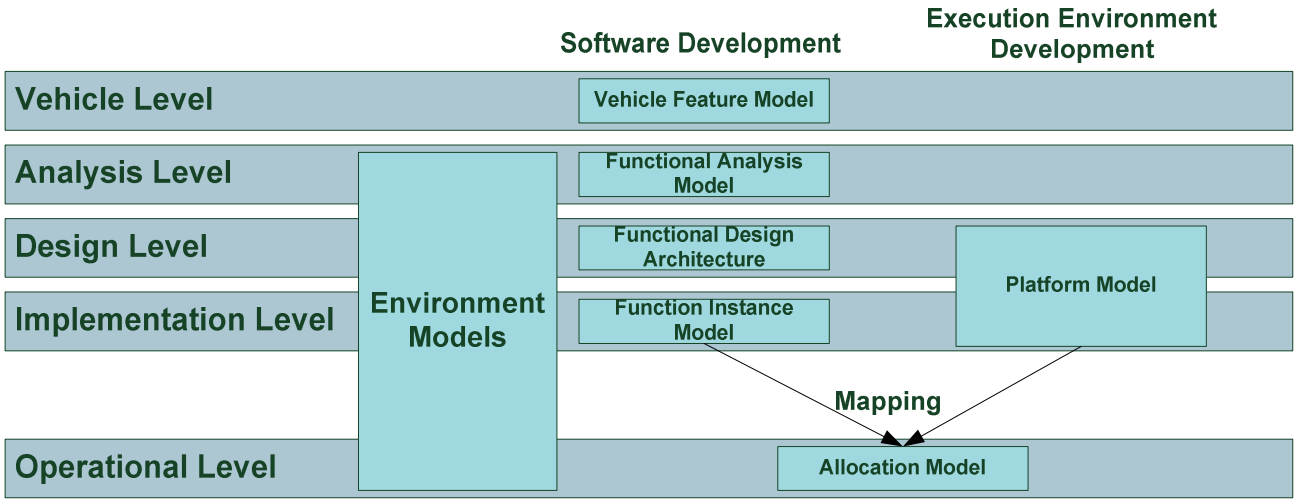
4.2.4 EAST-ADL

EAST-ADL steht für *Embedded Architecture and Software Tools Architeture Description Language* (EAST-ADL) und ist eine automobilspezifische Architekturbeschreibungssprache, welche Variabilitäten berücksichtigt. Sie zählt somit zu den Software-Produktlinienmodellen. Dieses Modell wird nur sehr kurz vorgestellt, da es nur einen kleinen Teil des Referenzmodells abdeckt und der Abstraktionsgrad des Modells sehr

gering ist. Es dient zur Veranschaulichung, dass es unterschiedliche abstrakte Modelle für Software-Produktlinien gibt.

EAST-ADL wurde entwickelt, da die Menge elektronischer Komponenten in einem Auto stark wächst. Im Jahre 2002 machten sie noch 25 Prozent aller verbauten Teile aus. Im Jahr 2015 werden es nach Schätzungen der Hersteller 40 Prozent sein. Ein Auto von heute besitzt mehr elektronische Systeme als ein Passagierflugzeug vor 20 Jahren (Vgl. Klaß et al. (2008)).

Das Ziel von EAST-ADL die Entwicklung eines industriespezifischen Systems, das mit allen anderen Standards arbeitet und diktiert, welcher Teil des Systems welche Aufgabe erledigt und sicherstellt, dass die verschiedenen Komponenten miteinander funktionieren. Dadurch wird die Kompatibilität und Wiederverwendung von Software in der heterogenen Hardwarelandschaft ermöglicht (Vgl. Lönn (2004), S.3).



(Vgl. Penzenstadler et al. (2006), S.11)

Abb. 4-10: EAST-ADL-Modell

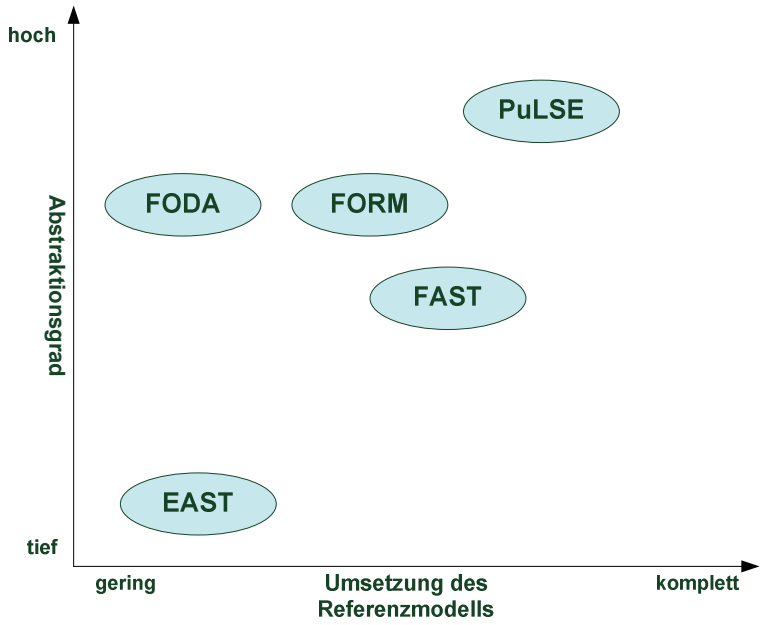
EAST ADL ist eine Sprache zur Modellierung und Entwicklung von softwarebasierten Systemen, welche die Modellierung in allen Phasen der Entwicklung der Fahrzeugfunktionen unterstützt. Dabei wird die Software- und Elektronenarchitektur um ausreichend Details ergänzt, um die Dokumentation und das Design modellieren zu können. Es beschreibt aber keinen Prozess oder eine konkrete Methode. Es definiert nur Artefakte und diese werden von jedem Unternehmen spezifisch an ihre Prozesse angepasst. In Abbildung 4-10 wird das Modell dargestellt. Die Sprache beschreibt fünf Abstraktionsebenen - das Vehicle Level, das Analysis Level, das Design Level, das Implementations Level und das Operational Level - für die elektronische Funktionalität und sieben Artefakte - die Environment Models, das Vehicle Feature Model, das Function Analysis

Model, die Functional Design Architecture, das Functional Instance Model, das Allocation Model und das Platform Model (Vgl. Lönn (2004), S.4f).

4.2.5 Vergleich und Evaluation der Modelle

Ein Problem, was derzeit alle vorgestellten Software-Produktlinien Modelle besitzen, ist, dass sie zwar das Grundverständnis für eine Software-Produktlinie ermöglichen, aber die konkrete Umsetzung mangels fehlender Beispiele nicht ausreichend beschrieben ist. Diese konkreten Beispiele von Firmen, die Software-Produktlinien schon einsetzen, sind für die Unternehmen zu wertvoll, da in den Analysen und Umsetzungen viele Unternehmensinterna zu finden sind (Vgl. Muthig et al. (2004), S.3f.). Deshalb gibt es auf dem Markt sehr wenig praxisbezogene Beispiele.

Außerdem wird der Break-Even-Point der Investitionen für eine Software-Produktlinie nicht sofort nach der Einführung erreicht. Gründe dafür sind notwendige Umstrukturierungen in der Organisation, ohne die ein reibungsloser Ablauf im Unternehmen nicht gewährleistet werden kann. Unter dem Break-Even-Point wird der Punkt verstanden, an dem die Kosten des Unternehmens oder des Produktes gleich den damit generierten Erlösen sind. Am Break-Even-Point wird folglich weder Gewinn noch Verlust erzielt.



(Vgl. Penzenstadler et al. (2006), S.28)

Abb. 4-11: Software-Produktlinienvergleich

In Abbildung 4-11 werden die Software-Produktlinien im Bezug auf den Abstraktionsgrad und den Grad der Umsetzung des Referenzmodells für Software-Produktlinien

verglichen. PuLSE besitzt einen hohen Abstraktionsgrad und kann damit in vielen unterschiedlichen Bereichen eingesetzt werden.

Der Ansatz des EAST-Modell ist zu spezifisch und die Umsetzung des Referenzmodells geschieht nur im geringen Maße. Es ist für die Automobilindustrie entwickelt worden und ist kein Prozess, an dem sich ein Unternehmen ausrichten sollte, sondern nur eine Beschreibungssprache zur Standardisierung der heterogenen Hardwarelandschaft. Deshalb ist das Modell für diese Diplomarbeit nicht weiter relevant, da ein Modell für den mobilen Endgerätemarkt entwickelt werden soll und daher die spezifischen Beschränkungen der mobilen Endgeräte berücksichtigt werden müssen und nicht die der Automobilindustrie.

Der Ansatz, den das FODA-Modell verfolgt, ist nur auf die Domainanalyse gerichtet. Da eine Software-Produktlinie aber neben der Domainanalyse auch die Architektur und die Implementierung für die Entwicklung der einzelnen Applikationen berücksichtigen muss, ist das FODA-Modell nur bedingt für diese Arbeit nutzbar. Es werden aber bei der Entwicklung des Modells für mobile Anwendungen für den Teilbereich der Domainanalyse Teile des FODA-Modells verwendet, da diese sich in der Praxis etabliert haben.

Das FAST-Modell entwickelt während der Domänenanalyse eine neue, domänenspezifische Sprache. Diese wird für das zu entwickelnde Modell nicht benötigt, da in dieser Diplomarbeit nur mobile Anwendungen, welche mithilfe der Programmiersprache Java geschrieben werden, betrachtet und untersucht werden. Außerdem setzt es, wie in Abbildung 4-11 dargestellt, nicht das komplette Referenzmodell um. Für die weitere Betrachtung und Entwicklung eines eigenen Modells wird versucht das komplette Referenzmodell abzubilden.

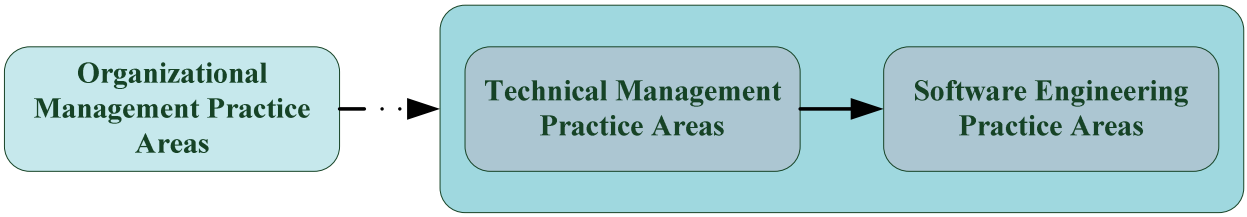
Der PuLSE-Ansatz stellt für diese Arbeit das beste Modell zur Verfügung, weil es den Zielen einer Entwicklung von mobilen Anwendungen entspricht. Es definiert die Produktlinie nach den Geschäftszielen und soll zukünftige Produktentwicklungen abdecken. Es bietet damit ein Rahmenwerk, das den Lebenszyklus einer Produktlinie einschließlich einer Infrastruktur zur Operationalisierung der Wiederverwendung abdeckt. Auch die Unterstützung der praktischen Anwendung eines Produktlinienansatzes und die Etablierung der Produktlinienorganisation im Unternehmen, sind Ziele, die das zu entwickelnde Modell abdecken soll. Trotzdem muss das Modell noch angepasst werden an die Einschränkungen und Voraussetzungen für mobile Entwicklungen. Die Komponenten des PuLSE-Ansatzes sind zwar modular aufgebaut und individuellen Wünschen und Bedürfnissen anpassbar. Aber das daraus resultierende Modell entspricht nicht mehr dem Referenzmodell des PuLSE-Ansatzes.

4.3 Modellsprachen

In diesem Kapitel geht es um die Nennung und Aufzählung der Modellsprachen, mithilfe derer eine Software-Produktlinie beschrieben und umgesetzt werden kann. Nach einer Übersicht folgt die Beschreibung und Vorstellung ausgewählter Modellsprachen, welche später verwendet werden.

4.3.1 Übersicht

Um eine Produktlinie erfolgreich zu managen, empfiehlt die SEI, dass sich alle erforderlichen Aktivitäten in 29 so genannte Practice Areas unterteilen. Diese 29 Aktivitäten sind wiederum in drei Oberkategorien eingeteilt, welche in Abbildung 4-12 dargestellt sind (Vgl. Clements et al. (2001), S.5f).



(Vgl. SEI (2009))

Abb. 4-12: Practice Areas

In den Practice Areas sind auch die Three Essential Acitivies enthalten. Diese werden in den Practice Areas noch feiner unterteilt.

Die erste Oberkategorie ist das organisatorische Management. In dieser Kategorie befinden sich alle organisatorischen und planungstechnischen Aktivitäten, die die Produktlinie betreffen. Beispiele wären das *Funding*, *Organizational Planning*, *Organizational Risk Management* und *Market Analysis*. Des Weiteren gibt es Aktivitäten für das Strukturieren des Unternehmens, wie das *Structuring the Organization*, und für die Aus- und Weiterbildung von Mitarbeitern, wie das *Training* (Vgl. Biermann (2004), S.5).

Die zweite Kategorie ist das technische Management. Es „umfasst die Aktivitäten des Managements, die leitend, unterstützend und kontrollierend auf die Entwicklung von Assets und Produkten einwirken“ (Weiß et al. (2004), S.6). Somit gehören das *Configuration Management*, *Data Collection*, *Metrics* and *Tracking*, sowie *Tool Support* zu dieser Kategorie. Aktivitäten wie zum Beispiel das *Technical Planning* und *Technical Risk Management*, sind Bestandteil des Technical Management (Vgl. Biermann (2004), S.5).

Die dritte Kategorie ist das Software Engineering. In dieser Kategorie befinden sich Aktivitäten, die sich um die Entwicklung der Core Assets kümmern. Ein Beispiel ist das *Mining Existing Assets*. Aber auch die Anforderungen an die Assets werden beim Software Engineering erarbeitet. Dazu stehen zum Beispiel die Aktivitäten des *Requirements Engineerung*, *Software System Integration* und das *Testing* zur Verfügung (Weiß et al. (2004), S.6).

Auf die einzelnen Areas wird in dieser Diplomarbeit nicht weiter eingegangen, da dies über das Thema der Arbeit hinausgeht. Die Praktiken sind im Bedarfsfall auf der Internetseite der SEI nachzulesen. Für das Fallbeispiel benötigte Praktiken und Methoden werden in den folgenden Unterkapiteln näher vorgestellt und deren Anwendung verdeutlicht.

Für die Modellierung des Features gibt es in der Praxis viele verschiedene Lösungen. Zum Beispiel das *FODA-Feature Diagramm*, 1-*Level und 2-Level Use Cases*, *erweiterte Message Sequence Charts* oder *Alternative / Options Use Cases*. Es wird in dieser Diplomarbeit näher auf das FODA-Feature Diagramm eingegangen, weil es in der Praxis das verbreitetste Modell ist und dem Nutzer einen guten Überblick verschafft. Außerdem werden noch die 1-Level und 2-Level Use Cases betrachtet, da sie im Zusammenspiel mit dem FODA-Feature Diagramm die Nachteile beider Modellsprachen beseitigen können.

4.3.2 FODA-Feature Diagramm

Das FODA-Feature Diagramm versucht mehrere verschiedene Alternativen und Optionen leicht verständlich und übersichtlich in einem Diagramm dazustellen. Es ist ein Teil des FODA-Modells, welches in Kapitel 4.2.3 Seite 42 vorgestellt wurde.

Das Beispiel in Abbildung 4-13 zeigt einen Feature-Baum mit einigen möglichen Relationen. Während das Auto einen Motor und ein Getriebe benötigt, ist die Belüftung nur ein optionaler Bestandteil. Die Belüftung wird entweder manuell oder per Klimaanlage geregelt. Das Getriebe muss ein Schaltgetriebe oder ein Automatikgetriebe sein. Der Motor unterteilt sich in einen 4, 6 und 8 Zylindermotor. Ein 4-Zylindermotor benötigt eine automatische Gangschaltung, während der 8-Zylindermotor keine automatische Gangschaltung besitzen darf.

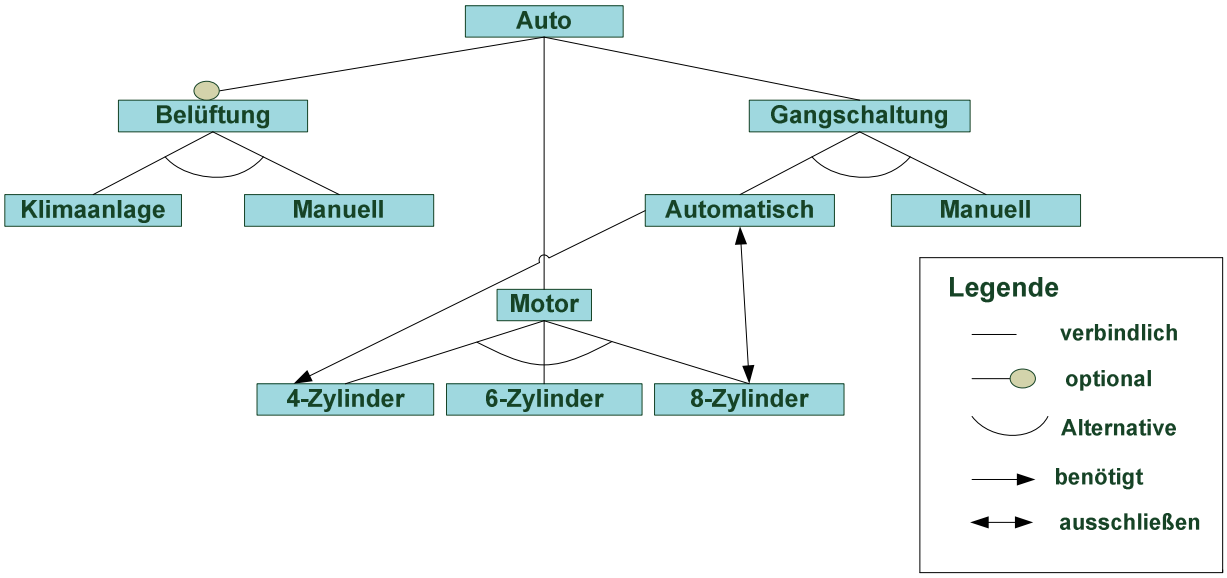


Abb. 4-13: Feature Diagramm

Bei der Produktentwicklung wird dieser Baum abgearbeitet und dabei die optionalen Features ausgewählt. Das Ergebnis ist die Feature Konfiguration. Sie ist eine Produktbeschreibung, die alle Features des Produktes beinhaltet (Vgl. Grosse (2002), S.3).

Ein Problem des FODA-Feature-Baumes kann die Unübersichtlichkeit des Diagramms sein, da bei vielen Features die textuelle Darstellung sehr viel Platz benötigt. Deshalb wird in dieser Diplomarbeit noch die 2-Level Use Cases verwendet, um dieses Problem zu minimieren. (Vgl. Grosse (2002), S. 3)

4.3.3 1-Level / 2-Level Use Cases

Beispiel eines 1-Level Use Cases am Beispiel einer Spielauswahl auf unterschiedlichen mobilen Endgeräten:

Hauptszenario:

- Light Variante eines Spiels: Das System stellt Start und Beenden dem Spieler zur Verfügung, Normal Variante eines Spiels stellt Start, Hilfe, Credits und Beenden dem Spieler zur Verfügung
- Der Nutzer wählt Start
- Das System stellt das Logo des gewählten Spiels dar
- Light Variante:
    - Auswählen drücken

    - Levelauswahl zur Verfügung stellen

- Normal Variante:
    - Auswählen drücken
    - Auswahl des Schwierigkeitsgrades
    - Levelauswahl zur Verfügung stellen
- Das System startet das Spiel und spielt, bis es vorüber ist
- …

Der 1-Level Use Case erweitert die normalen Use Case Beschreibungen um Variationen, damit das Use Case flexibler gestaltet werden kann. Diese Methodik ist sehr verbreitet, da durch die Beschreibung von Variationen keine neue Use Cases konzipiert werden müssen.

Der Vorteil der Formlosigkeit, wie die Variationen dargestellt und beschrieben werden können, ist gleichzeitig der größte Nachteil. Falls eine Variante fehlt, ist nicht erkenntlich, ob sie vergessen wurde oder ob keine Variante eingeplant war. Des Weiteren ist die Erweiterung um weitere Alternativen ein extrem aufwändiger Prozess, da an jede einzelne Stelle gesprungen und damit der gesamt Text untersucht werden muss. Wird ein Teil dabei vergessen, kann das wiederum sehr schnell zu Inkonsistenzen führen (Vgl. Grosse (2002), S.4).

Zur Lösung dieser Probleme wurde das 2-Level Use Case entwickelt. Durch die Einführung von Variablen sollen die angesprochen Probleme vermieden werden. Diese Variablen werden in drei Kategorien eingeteilt: Optional, Alternativ und Parametrisiert (Vgl. Grosse (2002), S.4):

- **Alternative Komponenten** bieten die Möglichkeit aus einer vordefinierten Liste eine spezifische Anforderung auszuwählen.
- **Parametrisierte Komponenten** besitzen einen Parameter, dessen aktueller Zustand ihr weiteres Vorgehen entscheiden.
- **Optionale Komponenten** können gewählt oder weggelassen werden. Sie sind kein zwingender Bestandteil eines Produktes.

Beispiel eines 2-Level Use Cases am Beispiel einer Spielauswahl auf unterschiedlichen mobilen Endgeräten:

Hauptaktor: [V0] Version eines Spiels

Hauptszenario:

- Das System stellt eine Liste der ([V1] verfügbaren) Menupunkte dar
- Der Benutzer drückt Start
- Das System stellt das Logo des Spiels dar
- Der Benutzer wählt das Level unter Berücksichtigung der ([V2] angebrachten) Prozedur
- Das System startet das Spiel und spielt es bis es zu Ende ist
- …

V0: Alternative

- V0: 1 – Light Variante
- V0: 2 – Normal Variante

V1: Optional

- Falls V0 = 1, dann gebe Start und Beenden aus
- Falls V0 = 2, dann gebe Start, Hilfe, Credits und Beenden aus

V2: Parametrisiert

- Falls V0 = 1, dann ProzedurA:
    - Auswählen drücken
    - Levelauswahl zur Verfügung stellen
- Falls V0 = 2, dann ProzedurB:
    - Auswählen drücken
    - Auswahl des Schwierigkeitsgrades
    - Levelauswahl zur Verfügung stellen

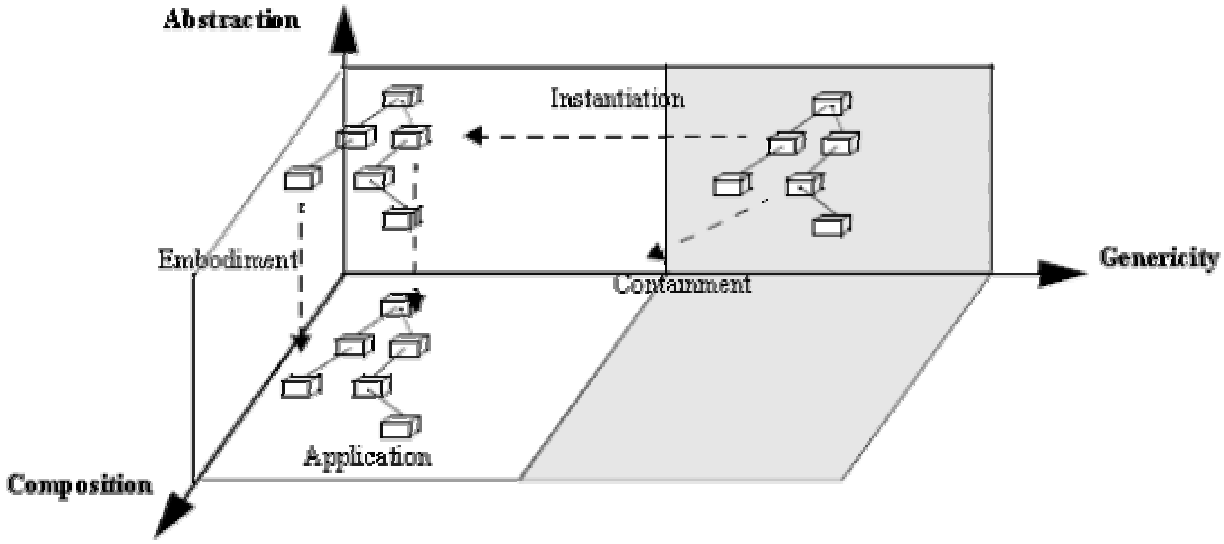
Der Vorteil dieser Methodik ist, dass die neu eingeführten zusätzlichen Alternativen, Optionen und Parameter nur an einem zentralen Ort hinzugefügt werden müssen (Vgl. Grosse (2002), S.4).

Meiner persönlichen Meinung nach ist zur Beschreibung einer Option oder Variante eine Vermengung des FODA-Feature Diagramms und des 2-Level Use Cases am übersichtlichsten, da hierbei die Struktur als auch die grafische Komponente mit ausreichender Erklärung weitläufig beschrieben werden kann.

4.3.4 KobrA

KobrA bedeutet **Ko**mponenten**b**asie**r**te **A**nwendungsentwicklung.

Ein wesentlicher Aspekt von KobrA ist die ‚Separation of Concerns'. Das heißt verschiedene Elemente der Aufgabe sollten möglichst in verschiedenen Elementen der Lösung repräsentiert werden. Die Separation of Concerns in der KobrA-Methode besteht aus den drei orthogonalen Entwicklungsdimensionen Abstraction, Genericity und Composition (Vgl. Josten (2001), S.22). Dies wird in Abbildung 4-14 verdeutlicht.



(Vgl. Atkinson et al. (2001))

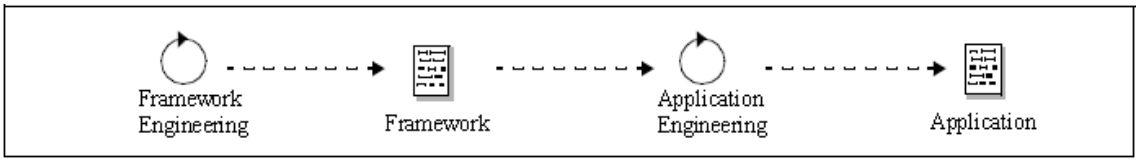
Abb. 4-14: KobrA-Entwicklungsdimensionen

Die Dimension Genericity beschreibt den Spezialisierungsgrad des Systems. Durch die Instantiierung sinkt die Spezialisierung und die Modelle beschreiben nur noch eine Produktlinie. Dabei findet durch das Containment eine Dekomposition des Systems in feinere Teile statt. Die Abstraction Dimension beschreibt den Abstraktionsgrad. Wird diese Dimension reduziert, entsteht die eigentliche Application des Systems. Entwick-

lungsaspekte, die mit dieser Dimension zu tun haben, werden auch Component Embodiment genannt (Vgl. Atkinson et al. (2001)).

KobrA führt den Begriff des ‚Komponent' (**K**obrA C**omponent**) ein. Dieser soll eine Abgrenzung zu Komponententechnologien wie JavaBeans ermöglichen. Der Begriff Komponent steht für nicht physische, logische Komponenten, welche die logischen Bausteine eines Systems darstellen (Vgl. Josten (2001), S.23).

KobrA's Product Line Engineering Prozess, wie in Abbildung 4-15 dargestellt, wird durch die beiden eigenständigen Prozesse Framework Engineering und Application Engineering realisiert und erzeugt das Framework und die Application (Vgl. Josten (2001), S.23).



(Vgl. Atkinson et al. (2001))

Abb. 4-15: KobrA's Product Line Engineering Prozess

Im Folgenden wird der Prozess des Framework Engineering kurz beschrieben (Vgl. Josten (2001), S.23f):

- Context Realization: Dabei werden eine Reihe von Modellen erzeugt, welche die Systemumgebung beschreiben.
- Komponent Specification: Diese Aktivität spezifiziert die nach außen sichtbaren Eigenschaften.
- Komponent Realization: Hierbei wird eine Beschreibung des internen Designs der Komponente erzeugt. Dazu wird die Komponente Generic Komponent Realization erzeugt, dessen Bestandteil unter anderem das Decision Model ist. Dieses wird im später entwickelten Software-Produktlinienmodell genutzt.
- Komponent Implementation: Umwandlung der erzeugten Modelle in eine Form, aus der automatisch ausführbarer Code entwickelt werden kann.

Beim Application Engineering hängen die Aktivitäten eng mit den entsprechenden Aktivitäten des Framework Engineerings zusammen. Allerdings bezieht sich Application Engineering auf eine spezielle Anwendung der Produktlinie. Aus diesem Grund müssen alle variablen Teile entfernt und durch statische ersetzt werden. Dies geschieht durch

Auflösung der Decision Models. Als Ergebnis erhält der Benutzer eine Applikation (Vgl. Josten (2001), S.25).

4.3.5 Reverse Engineering

Reverse Engineering ist ein Prozess, der das betrachtete System analysiert, um seine Komponenten und ihre Beziehungen zu identifizieren. Zweck dieses Prozesses ist es, für das analysierte System neue Abbildungen des Systems zu schaffen, meist in anderer Form und auf höherer Abstraktionsebene (Vgl. Chikofsky et al. (1990)).

Das Ziel des Reverse Engineering für Software-Produktlinien ist das Auffinden von Core Assets in bestehenden Programmen. Dabei können als wiederverwendbare Komponenten nicht nur der Quellcode, sondern auch Teile der Spezifikation, des Entwurfs, unterschiedliche Werkzeuge und Benutzerschnittstellen in Betracht gezogen und in die Produktlinie aufgenommen werden (Vgl. Bergey et al. (2000), S.1ff). Allerdings kann es zum Beispiel zu Problemen durch die fehlende Dokumentation des Quellcodes kommen oder weil die Architektur der Software während eines langen Wartungsprozesses mehrfach verändert wurde. Aus diesen Gründen ist die Altsoftware häufig schlecht strukturiert und weist zwischen den Modulen eine zu hohe Kopplung auf. Somit kann es trotz hoher Reengineeringkosten passieren, dass keine wiederverwendbaren Komponenten gefunden werden (Vgl. Simon et al. (2002), S.2ff).

Trotzdem ist das Herauslösen und Wiederverwenden von Komponenten dennoch einer kompletten Neuentwicklung vorzuziehen (Vgl. Weiderman et al. (1998)). Damit Komponenten aus vorhandener Altsoftware mit größtmöglichem Nutzen in die Produktlinie aufgenommen werden können, muss bereits zum Start der Entwicklung der Produktlinie die Wiederverwendung berücksichtigt werden. Es gibt zwei Arten der Überführung von Altsoftware in eine Software-Produktlinie, die revolutionäre und evolutionäre Entwicklung (Vgl. Weber (2004), S.1).

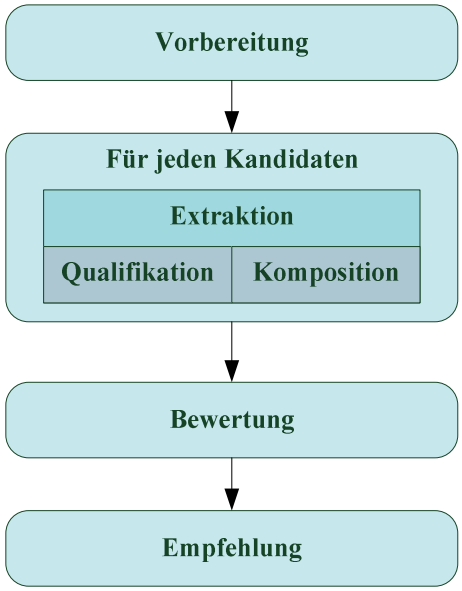
4.3.5.1 Revolutionäre Entwicklung

Bei der revolutionären Entwicklung wird die Wartung der Altsysteme gestoppt und eine neue Produktlinie entwickelt. Zu diesem Zweck wird die Altsoftware nach wiederverwendbaren Komponenten untersucht, die anschließend in die Produktlinie integriert werden (Vgl. Weber (2004), S.1).

4.3.5.1.1 Mining Architectures for Productlines

Eine Methode für die Revolutionäre Entwicklung heißt *Mining Architectures for Productlines* (MAP). Sie ist von der SEI entwickelt (Vgl. O'Brien et al., 2001), vorgestellt

und für die Identifikation von wiederverwendbaren Architekturen der Altsoftware entwickelt worden. Wie in Abbildung 4-16 dargestellt, besteht die Map-Methode aus vordefinierten Phasen, welche nacheinander ausgeführt werden. Eine Phase umfasst dabei Eingabedaten, Aktivitäten und erzeugt Ausgabedaten. (Vgl. Weber (2004), S.1f)



(Vgl. SEI II (2009))

Abb. 4-16: MAP-Aktivitäten

Nachfolgend sind die Phasen der MAP, wie in Abbildung 4-22 zu sehen, veranschaulicht:

Vorbereitung

In der Vorbereitungsphase werden die organisatorischen und technischen Fragen wie die Beweggründe für den Einsatz einer Software-Produktline gestellt und mithilfe des MAP-Teams geklärt. Des Weiteren werden die Produkt-Kandidaten mithilfe der vorhandenen Produkte bestimmt. Die identifizierten Produkte bilden die Eingabedaten für die nachfolgenden Phasen. (Vgl. O'Brien et al. (2002), S. 6)

Extraktion

In dieser Phase werden Informationen aus dem Source-Code der einzelnen Systeme extrahiert und die Daten zur späteren Manipulation gespeichert. Diese Informationen bestehen aus Funktionen, Variablen, Dateien etc. und deren Beziehungen zwischen den Elementen. Neben den statischen Informationen aus dem Quellcode werden auch dynamische Informationen wie zum Beispiel die Häufigkeit der Aufrufe der Funktionen

erfasst. Diese Daten dienen als Eingabe für die Qualifikations- und Kompositionsphase (Vgl. O'Brien et al. (2002), S. 6).

Komposition

Die Herstellung von unterschiedlichen Komponenten-Sichten ist das Ziel der Komposition. Die gesammelten Informationen aus der Phase davor werden zur Identifikation der Komponenten-Sichten abstrahiert und die Komponenten und Beziehungen zugeordnet. Dies ist der entscheide Schritt für den Erfolg der Einführung der Software-Produktlinie, da in diesem die Struktur der Produktlinie festgelegt wird (Vgl. O'Brien et al. (2002), S. 6).

Qualifikation

In dieser Phase werden bekannte Architektur- und Qualitätsmerkmale, wie zum Beispiel Sicherheit und Geschwindigkeit der Programme, auf das System und seine Komponenten abgebildet (Vgl. O'Brien et al. (2002), S. 6).

Bewertung

Bei der Bewertungsphase werden die Architektur-Kandidaten hinsichtlich ihrer Gemeinsamkeiten, Variabilitäten und anderen Eigenschaften bewertet. Nach dieser Phase liegt die Bewertung dieser Eigenschaften als Ergebnis vor (Vgl. Weber (2004), S.2).

Empfehlung

In dieser Phase werden durch das MAP-Team Vorschläge bezüglich der Aufnahme von Kandidaten in die Produktlinie gemacht (Vgl. Weber (2004), S.2).

4.3.5.1.2 Options Analysis for Reengineering

Für die Wiederverwendung auf Komponentenebene hat die SEI die *Options Analysis for Reengineering* (OAR) entwickelt (Vgl. Bergey et al. (2001)). Das Ziel der OAR ist das Auffinden von Komponenten mit der Abstimmung der unterschiedlichen Ziele und Erwartungen der Unternehmensmitglieder. Damit berücksichtigt die OAR unterschiedliche Interessengruppen, wie das Management oder die Entwickler. Während das Management zum Beispiel die maximale Kostenersparnis anstrebt, möchten die Entwickler die Entwicklung möglichst einfach gestalten (Vgl. Weber (2004), S.2).

(Vgl. Bergey et al. (2001), S.3)

Abb. 4-17: OAR-Aktivitäten

Die OAR-Methode besteht, wie in Abbildung 4-17 verdeutlicht, aus 5 Aktivitäten

In der ersten Aktivität sollen alle Teilnehmer ihre Ziele und Erwartungen festhalten und aufeinander abgestimmt werden. Des Weiteren wird das Altsystem auf mögliche Komponenten hin untersucht (Vgl. Weber (2004), S.2f).

In der zweiten Aktivität werden die Komponenten der Altsoftware untersucht, ob sie den Anforderungen der Produktlinie entsprechen. Alle Komponenten der Altsysteme, die die Anforderungen erfüllen, werden notiert. Das Ergebnis sind eine Menge von möglichen Komponentenkandidaten (Vgl. O'Brien (2002), S.8).

Bei der Analyse der Kandidaten werden die Komponenten daraufhin untersucht, welche Änderungen an ihnen vorgenommen werden müssen, um sie in die Produktlinie einarbeiten zu können. Dabei werden die Kandidaten in zwei Kategorien eingeteilt. Die Black-Box Komponenten müssen entweder sofort in die Produktlinie übernommen werden oder müssen nur gekapselt werden. Beispiel dafür sind Testfälle oder Werkzeuge. Die White-Box Komponenten müssen für die Verwendung in der Produktlinie durch Reengineering-Maßnahmen bearbeitet werden (Vgl. Weber (2004), S. 3).

Im vierten Schritt, der Planung der Alternativen, werden die Komponenten abschließend überprüft und Alternativen für die benötigen Komponenten bereitgestellt. Dabei werden die Komponenten und Alternativen bezüglich ihrer Kosten, des Aufwandes und ihres Risikos beurteilt und außerdem auch Kombinationen von Kandidaten erarbeitet (Vgl. O'Brien (2002), S.8).

In der letzten Phase, Wahl der Alternative, werden die Entscheidungskriterien festgelegt und darauf aufbauend eine Kombination ausgewählt. In diesen Entscheidungsfindungsprozess sind alle Gruppen beteiligt. Des Weiteren wird ein zusammenfassender Bericht und Begründung für die Alternative verfasst (Vgl. O'Brien (2002), S.8).

4.3.5.2 Evolutionäre Entwicklung

Bei der evolutionären Entwicklung wird die Altsoftware Schritt für Schritt in die Produktlinie überführt. Die in Kapitel 4.3.5.1 vorgestellten Methoden verfolgen den revolutionären Entwicklungsansatzes. Dieser Ansatz besitzt den Vorteil der großen Ent-

wurfsfreiheit, aber die Entwickler müssen bereits vor dem Projektstart wissen, wie die Produktlinie aufgebaut sein soll. Des Weiteren ist die Chance zum Scheitern des Projektes erhöht, da unter Umständen erst bei weit fortgeschrittener Entwicklung festgestellt werden kann, ob alle Anforderungen erfüllt werden können. Zur Verringerung dieser Nachteile wurde an der Universität Stuttgart eine Alternative zur revolutionären Einführung entwickelt. Die evolutionäre Entwicklung (Vgl. Eisenbarth et al. (2001), S.1ff) erlaubt Altsysteme Schritt für Schritt durch Restrukturierungsmaßnahmen in eine Produktlinie zu überführen. Die Vorteile dieser Strategie sind (Vgl. Weber (2004), S.6):

- Die Entwickler bekommen ein besseres Gesamtbild auf die Produktlinie, weil sie die Konzepte während der Entwicklung besser kennen lernen können.
- Das Risiko des Scheiterns wird minimiert, da die Einführung und Weiterentwicklung der Produktlinie jederzeit abgebrochen werden kann.
- Die Teilnehmer der Produktlinie kennen die Produkte und können daher bereits zu Beginn der Einführung abschätzen, ob die Anforderungen an die Produktlinie vom Produkt erfüllt werden.

Der evolutionäre Prozess besteht aus mehreren Aktivitäten. Die erste Aktivität besteht in der Identifizierung der Features im Produkt. Wenn ein Produkt sehr viele Features enthält, dann können sie nach ihrer Dringlichkeit priorisiert werden. Die zweite Aktivität ist die Schätzung des Aufwandes und der Kosten, die durch die Änderung der Features entstehen, durch den Systemarchitekten. Dafür werden vorher Grenzen festgelegt, in denen die geschätzten Werte liegen müssen. Der gesamte Prozess ist iterativ angelegt. Wie in Kapitel 2.1.2. beschrieben, können die Features beliebig oft geändert werden. Es besteht die Möglichkeit den Prozess nach jeder Iteration anzuhalten und die Produkte traditionell weiter zu entwickeln. Dabei bleiben die Vorteile durch bereits durchgeführte Änderungen erhalten (Vgl. Weber (2004), S.6).

4.4 Implementierungsmöglichkeiten

Die Referenzarchitektur einer Produktlinie legt die wichtigsten Strukturen der Systeme fest. Damit ist sie von zentraler Bedeutung für den Erfolg einer Produktlinie. Außerdem müssen die Variabilitäten berücksichtigt werden. Die Architektur muss so gestaltet werden, dass die gewünschten Varianten schnell und einfach implementiert werden können. Des Weiteren gilt es zu beachten, dass die Beschreibung der Architektur so gestaltet sein muss, dass Variabilitätsarten unterschiedlicher Komponenten nach dem gleichen Ansatz realisiert werden können (Vgl. Böckle et al. (2004), S.109f).

In diesem Kapitel wird auf die unterschiedlichen Möglichkeiten für die Implementierung näher eingegangen und deren Vor- und Nachteile kurz erläutert.

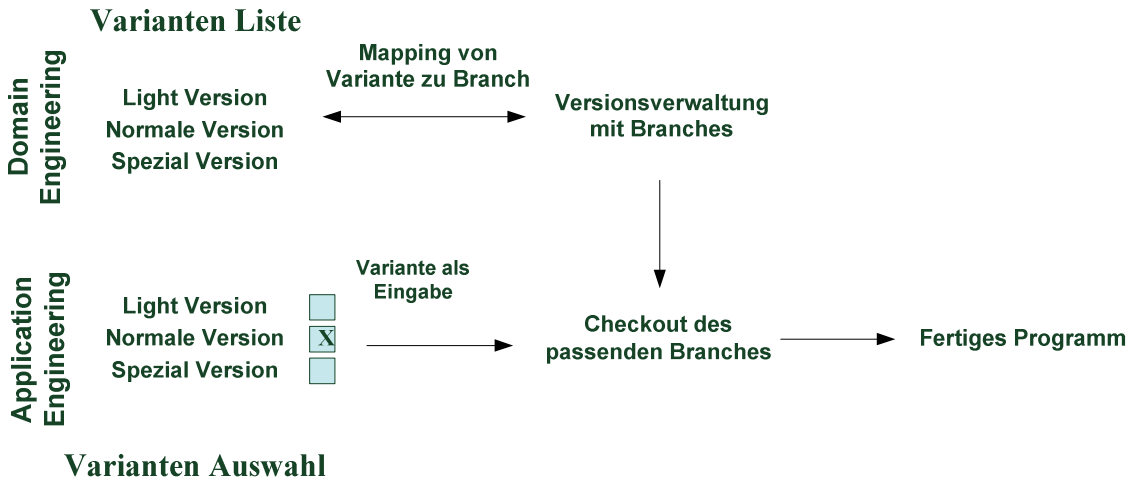
4.4.1 Versionsverwaltung

Eine mögliche Form der Implementierung einer Softwareproduktlinie sind Versionsverwaltungen. Sie dienen der Versionierung von Quelldateien und schaffen eine Plattform zur gemeinsamen Entwicklung.

Das Ziel der Versionsverwaltung ist die Speicherung der Veränderungen an einem Dokument. Somit ist es möglich einen alten Zeitpunkt der Arbeit wieder zu rekonstruieren. Des Weiteren ist eine Arbeit am gleichen Objekt von mehreren Personen möglich oder die Arbeit mit mehreren Kopien beispielsweise an unterschiedlichen Orten durch eine Person (Vgl. Leutloff (2000)).

Beispielsysteme für eine Versionsverwaltung sind CVS, SVN, Perforce und SCCS.

In der Versionsverwaltung wird zwischen Revisions, Tags und Branches unterschieden. Revision bedeutet, dass die Versionsverwaltung alle eingecheckten Files mit einer Revisionsnummer versieht. Sie ist hauptsächlich für den internen Gebrauch. Tags werden explizit von Hand hinzugefügt, um ein bestimmtes Release einer Software zu kennzeichnen. Eine extreme Anwendung von Tags ist das so genannte Branching. Hierbei wird nicht nur ein Release getaggt, sondern auch bestimmt, dass es ab diesem Release zwei verschiedene Entwicklungszweige geben soll, zum Beispiel einen, um Bugfixes für das Release zu erstellen und einem development-Zweig, um ein neues Release vorzubereiten (Vgl. Stiehler (2002), S.14).



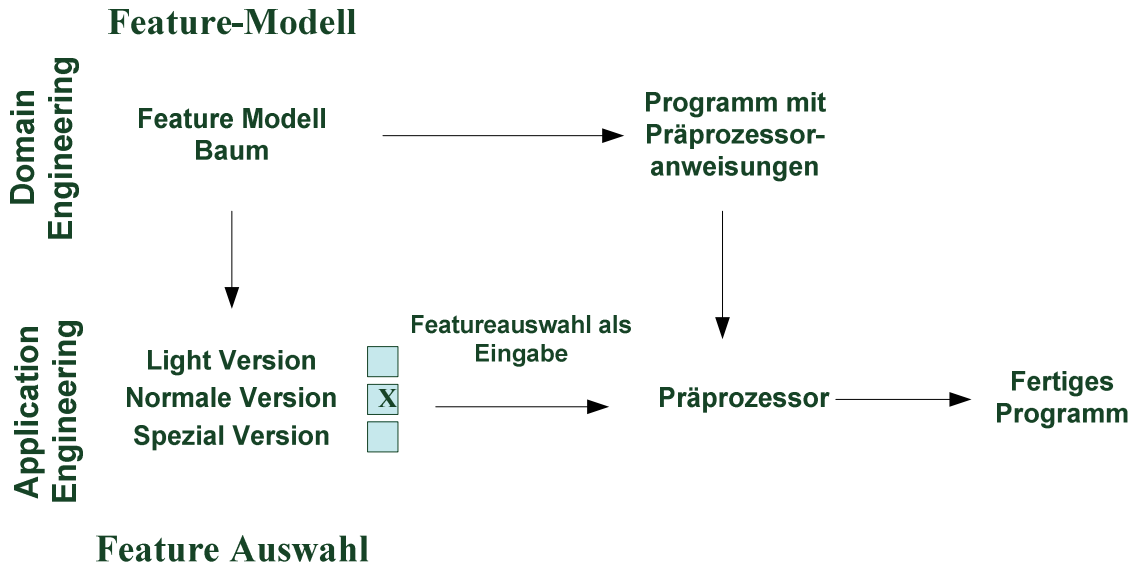
(Vgl. Apel et al. (2008), S.7)

Abb. 4-18: Produktlinien mit Versionsverwaltung

In Abbildung 4-18 ist der schematische Ablauf für eine Produktlinie mit einer Versionsverwaltung dargestellt. Als Erstes werden im Domain Engineering die Varianten analysiert. Diese werden in Branches gespeichert. Im Application Engineering wird eine Variante ausgewählt und aus dem Versionsverwaltungsprogramm ausgecheckt, und erhält damit das fertige Programm.

4.4.2 Präprozessoren

Unter einem Präprozessor wird ein kleiner Zwischenschritt verstanden, bei dem der Quelltext vor dem Compileraufruf transformiert wird. Darunter kann die Veränderung von Quellcode zu verstehen sein, indem vordefinierte Symbole oder Kommentare durch Leerstellen ersetzt werden oder das Ausführen von vordefinierten Makros. Es ist in einigen bekannten Programmiersprachen wie C, TeX oder auch PHP vorhanden (Vgl. Apel et al. (2008), S.10).



(Vgl. Apel et al. (2008), S.15)

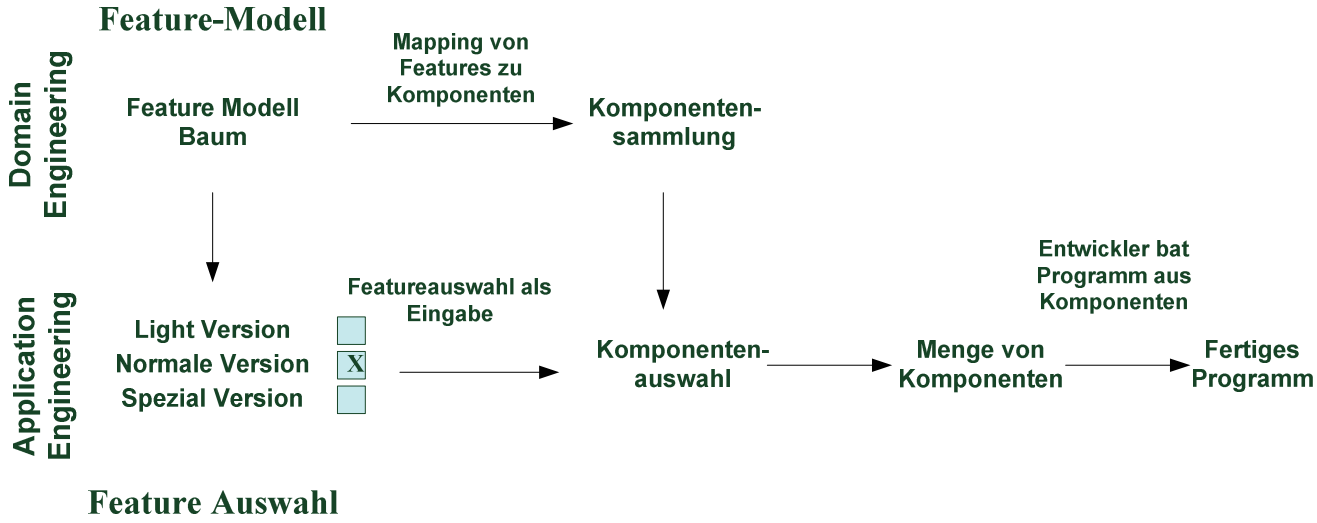
Abb. 4-19: Produktlinien mit Präprozessoren

In Abbildung 4-19 wird eine Produktlinie mit Präprozessoren dargestellt. Im Domain Engineering wird ein Feature Baum erstellt, siehe Kapitel 4.3.3. Daraus wird ein Programm mit Präprozessor-Anweisungen gebildet. Im Application Engineering sucht sich der Kunde seine Variante aus, diese wird mithilfe des Präprozessors übersetzt und ein fertiges Programm geliefert.

4.4.3 Komponenten

Eine Komponente ist eine abgeschlossene Implementierungseinheit mit einer Schnittstelle und bietet dabei ein oder mehrere Dienste an (Vgl. Apel et al. (2008), S.21).

Komponenten werden aus anderen Komponenten zusammengesetzt und ergeben zum Schluss ein Softwaresystem. Sie sind klein genug für die Erzeugung und Wartung in einem Stück, aber auch groß genug um Funktionen bieten zu können.



(Vgl. Apel et al. (2008), S.26)

Abb. 4-20: Produktlinien für Komponenten

In Abbildung 4-20 werden Produktlinien für Komponenten aufgezeigt. Als Erstes wird ein Feature Modell für die Produktlinie erstellt. Danach werden Teile der Anwendung in verschiedene Komponenten implementiert. Ein Beispiel für eine Komponente für das Fall Beispiel „GoGame“ wäre eine Entityklasse, die Funktionen wie x und y-Position, Breite, Höhe, Schnittpunkte mit anderen Entities oder Punkten berechnen kann. Durch die Feature-Auswahl werden Komponenten ausgewählt. Danach muss der Entwickler aus der Menge von Komponenten ein Programm schreiben und diese miteinander verbinden.

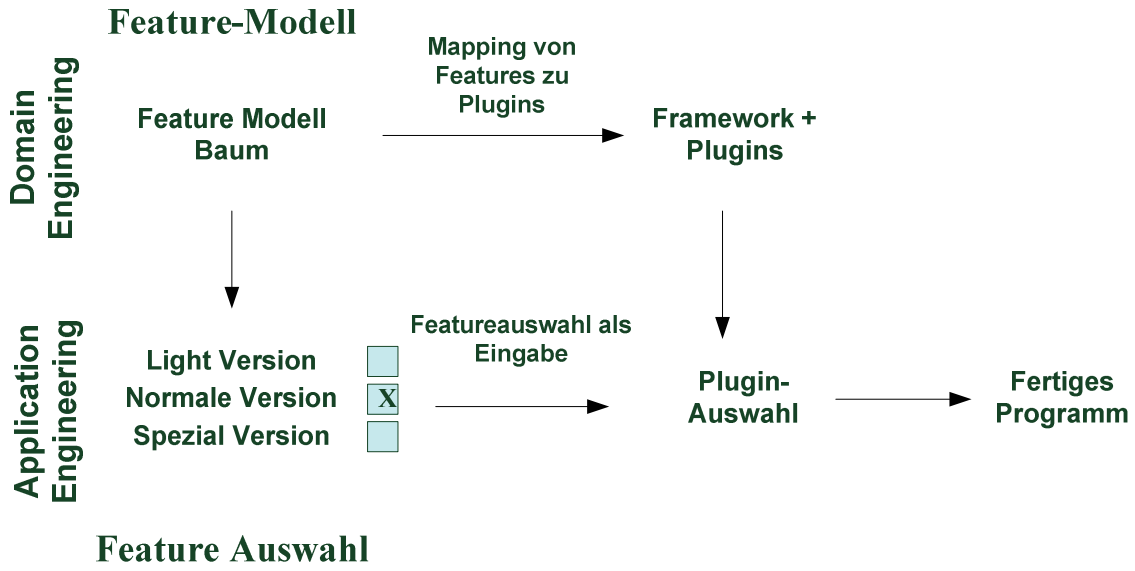
4.4.4 Frameworks

Frameworks bestehen aus einer Menge abstrakter und konkreter Klassen. Sie besitzen eine abstrakte Struktur, die für einen bestimmten Zweck angepasst werden kann. Das Framework kehrt auch die Kontrolle der Ausführungsreihenfolge um und bestimmt diese (Vgl. Apel et al. (2008), S.31f).

Ein Framework kann durch ein so genanntes Plugin erweitert werden. Dabei stellen Plugins Erweiterungen für das Framework dar, die nicht zwangsläufig benötigt werden. Spezielle Funktionen können damit bei Bedarf hinzugefügt werden. Üblicherweise werden diese getrennt kompiliert. Daher können Plugins nicht nur vom Hersteller des Frameworks erstellt werden, sondern auch später von Drittanbietern.

Ein Beispiel für ein Framework ist Eclipse. Dies ist eine Entwicklungsumgebung für verschiedene Programmier- und Modellierungssprachen. Egal welche Sprache benutzt wird, der Aufbau des Programms ist immer gleich. Plugins können sehr einfach in ein spezielles Verzeichnis kopiert werden und werden bei einem Neustart des Programms automatisch erkannt.

Andere bekannte Beispiele für Frameworks sind DirectX oder der Instant Messenger Miranda.



(Vgl. Apel et al. (2008), S.46)

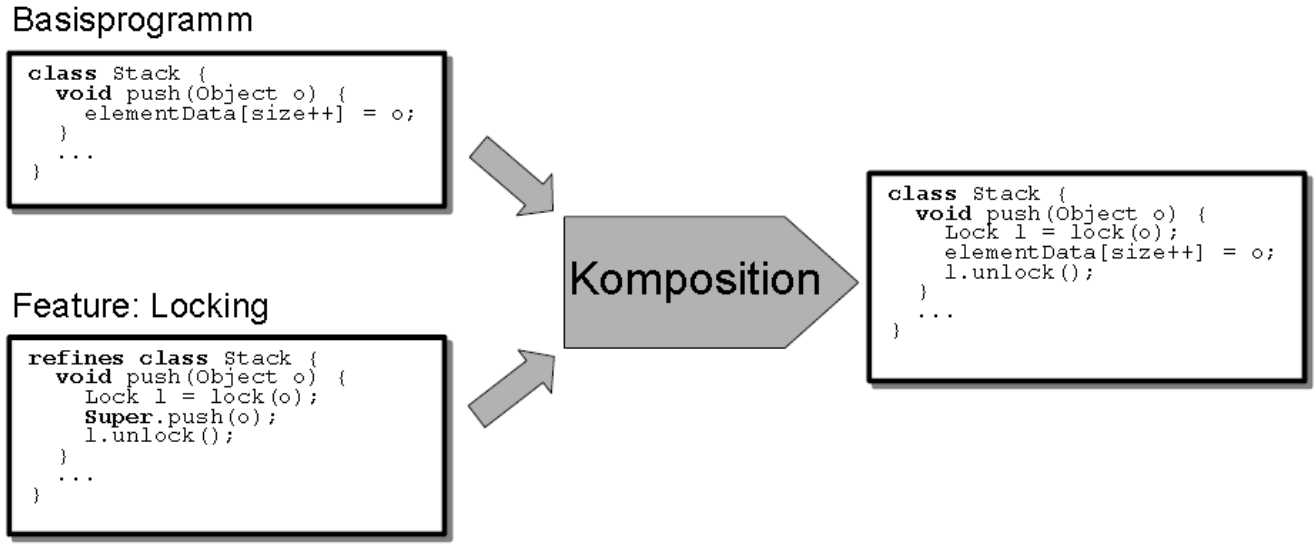
Abb. 4-21: Produktlinien für Frameworks

In Abbildung 4-21 wird eine Produktlinie für Frameworks dargestellt. Als Erstes wird ein Feature Modell erstellt. Auf Grundlage dessen werden die Features zu Plugins umgesetzt. Nach der Auswahl der Features im Application Engineering werden das Framework und die benötigten Plugins genommen und mit deren Hilfe das fertige Programm erstellt.

4.4.5 Feature Orientierung

Ziel der Feature Orientierung ist alle Features in der Produktlinie zu erkennen und diese im Code zu lokalisieren. Es wird versucht die jeweiligen Features im Modell genau einer Menge an Klassen zuzuordnen. Es wird versucht jedes Feature eigenständig zu machen, sodass später im Application Engineering nur noch gesagt werden muss, welches Feature benötigt werden und diese können dann automatisiert produziert werden. Es ist nicht wie bei den Komponenten eine Anpassung durch den Programmierer notwendig. Diese Anpassung passiert bei der Feature Orientierung schon beim Mapping der Fea-

tures zu Feature-Modulen. Eine Möglichkeit zur Umsetzung bietet der featureorientierte Generator AHEAD (Vgl. Batory (2004)). In Abbildung 4-22 wird die generelle Idee des Ansatzes illustriert.



(Quelle: Apel et al. (2009), S.3)

Abb. 4-22: Generierung von Varianten mittels Komposition von Codeeinheiten

In AHEAD werden die Features als Module implementiert. Werden Sie einem Basisprogramm hinzugefügt, dann geben sie diesem neue Pakete und Klassen bei und erweitern bestehende Pakete, Klassen und Methoden (Vgl. Apel et al. (2009), S.4).

```
1 class Stack {
2   void push(Object o) { ... }
3   Object pop() { ... }
4 }
```

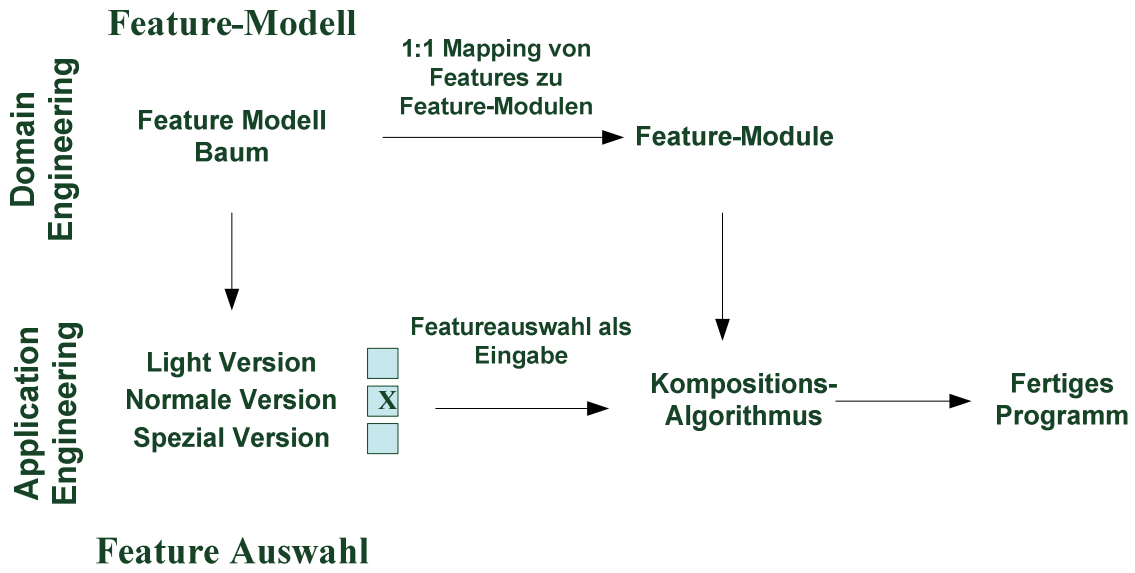
```
5 refines class Stack {
6   void backup() { ... }
7   void restore() { ... }
8   void push(Object o) { backup(); Super.push(o); }
9 }
```

(Quelle: Apel et al. (2009), S.4)

Abb. 4-23: Stackimplementierung in AHEAD

In Abbildung 4-23 wird ein einfaches Codebeispiel dargestellt. Das Basisprogramm stellt eine Klasse Stack zur Verfügung und ein separates Modul implementiert das Feature UnDo. Das Schlüsselwort *refines* in Zeile 5 verdeutlicht, dass ein Feature eine bereits vorhandene Klasse verfeinert. Diese Verfeinerung fügt zwei neue Methoden ein und erweitert die Methode push durch Überschreiben. Das Schlüsselwort *Super* in Zeile 8 verweist auf die verfeinerte Klasse. Das Basisprogramm und das Featuremodul werden mit AHEAD komponiert. Dabei werden die gewählten Module überlagert, das heißt im Beispielt entsteht eine zusammengesetzt Klasse Stack mit den Methoden push, pop,

backup und restore, wobei die Methode push um den zusätzlichen Aufruf der Methode backup erweitert ist (Vgl. Apel et al. (2009), S.4).



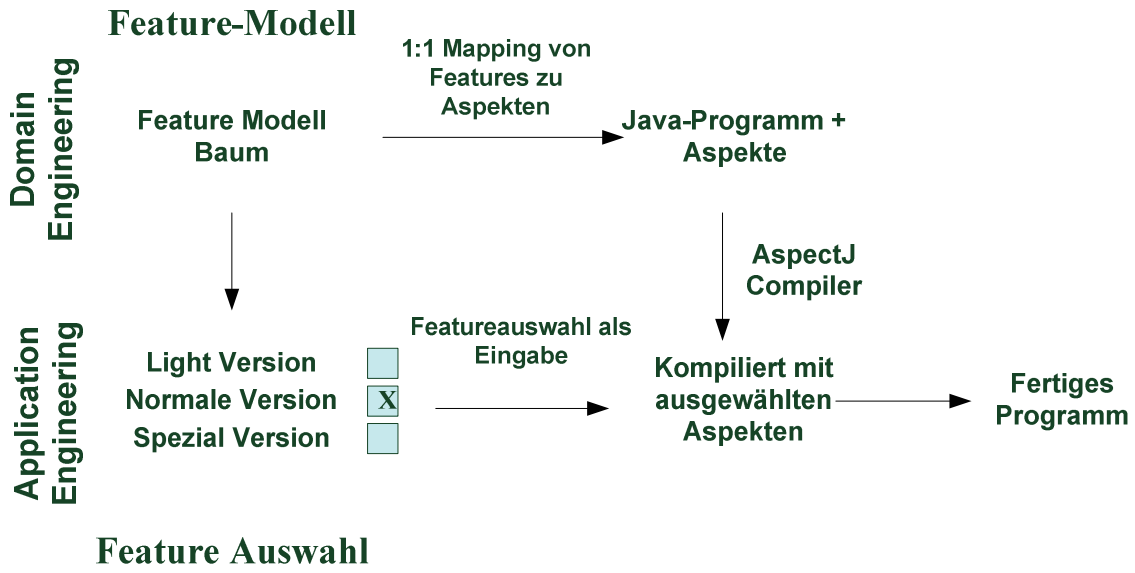
(Vgl. Kästner et al. (2008), S.9)

Abb. 4-24: Produktlinien mit Feature-Modulen

In Abbildung 4-24 wird eine Produktlinie für Feature-Module dargestellt. Als Erstes wird ein Feature Modell erstellt. Auf Grundlage dessen werden die Features zu so genannten Feature-Modulen gemappt. Nach der Auswahl der Features im Application Engineering werden mithilfe eines Kompositions-Algorithmus die Feature-Module zum fertigen Programm zusammengesetzt.

4.4.6 Aspekt Orientierung

Das Ziel der Aspektorientierung ist die Modularisierung von querschneidenden Belangen (Vgl. Kiczales et al. (1997)). Es gibt für Java eine Aspekt Orientierte Erweiterung. Dabei werden Aspekt ähnlich wie Klassen implementiert, aber es gibt neue Sprachkonstrukte, um Aspekte abbilden zu können. Der Basiscode wird aber weiterhin in Java implementiert. Die Aspekte werden später von einem speziellen Aspekt-Compiler in den Code „gewebt“ (Vgl. Kiczales et al. (2001)). Ein Aspekt in Programmiersprachen wie AspectJ kann dabei zum Beispiel Klassenhierarchien manipulieren, Methoden und Felder zu einer Klasse hinzufügen, Methoden mit zusätzlichem Code erweitern und Ereignisse wie Methodenaufrufe oder Feldzugriffe abfangen und zusätzlichen oder alternativen Code ausführen (Vgl. Kästner et al. (2008), S.15).



(Vgl. Kästner et al. (2008), S.59)

Abb. 4-25: Produktlinie mit Aspekten

In Abbildung 4-25 wird eine Produktlinie für Aspekte dargestellt. Als Erstes wird ein Feature Modell erstellt. Auf Grundlage dessen werden die Features zu Aspekten gemappt. Nach der Auswahl der Features im Application Engineering wird das fertige Programm mit ausgewählten Aspekten kompiliert und zusammengesetzt.

4.4.7 Vergleich und Evaluation der Implementierungsmöglichkeiten

Jede der vorgestellten Implementierungsmöglichkeiten besitzen Vor- und Nachteile für die Arbeit im mobilen Gerätemarkt. In folgenden wird auf die einzelnen Vor- und Nachteile kurz eingegangen, um im folgenden Kapitel eine Empfehlung zu geben, welcher Implementierungsansatz genommen werden sollte.

Für eine Versionsverwaltung gibt es sehr viele und gute Werkzeuge zur Umsetzung. Es ist sehr leicht zu verstehen und viele Firmen, wie zum Beispiel IBM, setzen es zur Versionierung ihrer Dokumente ein. Aber für eine Software-Produktlinie gibt es einige Nachteile, die gegen den Einsatz einer Versionsverwaltung im Bereich der Software-Produktlinien sprechen. Es entsteht ein hoher Aufwand für die Wartung durch das Zusammenführen unterschiedlicher Versionen. Es gibt keine geplante Wiederverwendung, sondern es muss die alte Datei erst kopiert und dann editiert werden. Außerdem ist es eine starke Mischung von Versionen und Varianten. Eine klare Abgrenzung ist nicht möglich. Deshalb ist die Versionsverwaltung für Software-Produktlinien für mobile Endgeräte zwar möglich, aber nicht sinnvoll.

Auf die Präprozessoren wird in dieser Diplomarbeit nicht näher eingegangen, weil es unter anderem in der Programmiersprache JAVA nicht vorgesehen ist, mit Präprozessoren zu arbeiten. Außerdem werden die Programme mit den Präprozessor-Anweisungen sehr komplex, weil sie beliebig geschachtelt werden können. Des Weiteren ist der Code schwer zu lesen, weil er schnell unübersichtlich wird. Aus den Erfahrungen mit den Programmiersprachen, die Präprozessoren besitzen, ist bekannt, dass ein Tool Support sehr schwierig ist.

Ein Vorteil der Komponentenimplementierung ist, dass sie in der Industrie sehr üblich ist, wie in Kapitel 3 am Beispiel von Phillips aufgezeigt wurde. Außerdem ist eine systematische geplante Wiederverwendung von Komponenten vorgesehen. Firmen können sich auf einzelne Komponenten beschränken und diese an andere Firmen und Kunden weiterverkaufen. Nachteile sind der relativ hohe Entwicklungsaufwand, weil nach der Bestimmung der Komponenten diese vom Programmierer händisch angepasst werden müssen, damit das geplante Programm herauskommt. Damit ist auch eine vollkommene Automatisierung ausgeschlossen. Ein weiteres Problem ist die Schwierigkeit, dass die Funktionalität gegebenenfalls schwer als Komponenten zu kapseln ist.

Ein positiver Punkt für die Erstellung einer Produktlinie mithilfe eines Frameworks ist die Tatsache, dass eine Vollautomatisierung möglich und es sehr praxiserprobt ist. Dagegen spricht aber, dass es schwierig zu warten ist und ein hoher Einstellungsaufwand für das Framework besteht.

Feature-Orientierung hat neben den Vorteilen der Automatisierung und der Kapselung der Features, aber auch einige Nachteile. So gibt es derzeit eine schlechte Werkzeugunterstützung, die Implementierung von querschneidenden Belangen kann in bestimmten Fällen sehr aufwändig sein und Features sind nicht immer unabhängig (Vgl. Kästner et al. (2008), S.9).

Aspekte besitzen eine gute Werkzeugunterstützung und die von den Komponenten realisierte Kernfunktionalität und die einzelnen Aspekte können isoliert betrachtet und beschrieben werden, was die Komplexität der Programmierung erheblich reduziert. Aber auch die Aspekte besitzen einige Nachteile. Zum einen können geeignete Aspekte besonders in großen Programmen schwer zu identifizieren sein. Eine nachträgliche Implementierung in ein Programm ist schwerer als das Erstellen von Programmen mit Aspekten im Hinterkopf.

4.5 Empfehlungen

Aus den resultierenden Ergebnissen durch die Vorstellung der Vor- und Nachteile ergeben sich Empfehlungen, welche Modelle, Modellsprachen und Implementierungsmög-

lichkeit für das zu entwickelnde Software-Produktlinienmodell für mobile Endgeräte genutzt werden sollten.

Wie in Kapitel 4.2 ab Seite 38 angesprochen, bietet das PuLSE Modell die beste Grundlage für das zu entwickelnde Modell, weil es den Zielen einer Entwicklung von mobilen Anwendungen entspricht. Dabei lässt es dem Nutzer viele Möglichkeiten zur Implementierung durch den hohen Abstraktionsgrad. Zur besseren Übersicht wird das zu entwickelnde Modell sich an den FAST-Modell Aufbau halten, weil dieser subjektiv am übersichtlichsten ist.

Für die Implementierung wird die Komponentenlösung genommen, da diese in der Praxis sehr verbreitet ist und die Nachteile, wie der etwas erhöhte Aufwand bei der Implementierung durch die Vorteile aufgehoben werden. Der Feature Ansatz zur Implementierung würde sich auch anbieten, jedoch ist die angesprochene schlechte Werkzeugunterstützung hinderlich. Das Fallbeispielunternehmen „GoGame“ kennt sich am besten mit der Komponentenimplementierung aus, sodass es diese anwenden wird.

Für die Umsetzung des Modells werden die in Kapitel 4.3 ab Seite 48 näher vorgestellten Modellsprachen genutzt.

5 Software-Produktlinie für Mobile Endgeräte

In Kapitel 4.5 wurden Empfehlungen für die Verwendung des zu entwickelnden Modells gegeben. Auf deren Grundlage wird in diesem Kapitel ein Modell entwickelt und vorgestellt, welches die Beschränkungen und Voraussetzungen der mobilen Entwicklung mit Java ME berücksichtigt.

5.1 Vorstellung des Modells

Das Modell heißt Software-Produktlinie für Mobile Endgeräte (SPLME) und ist ein Modell, das auf die Entwicklung einer Produktlinie mithilfe der Programmiersprache Java ME zielt. Gründe für den Bezug auf Java ME wurden in Kapitel 2.3.1. auf Seite 14 genannt.

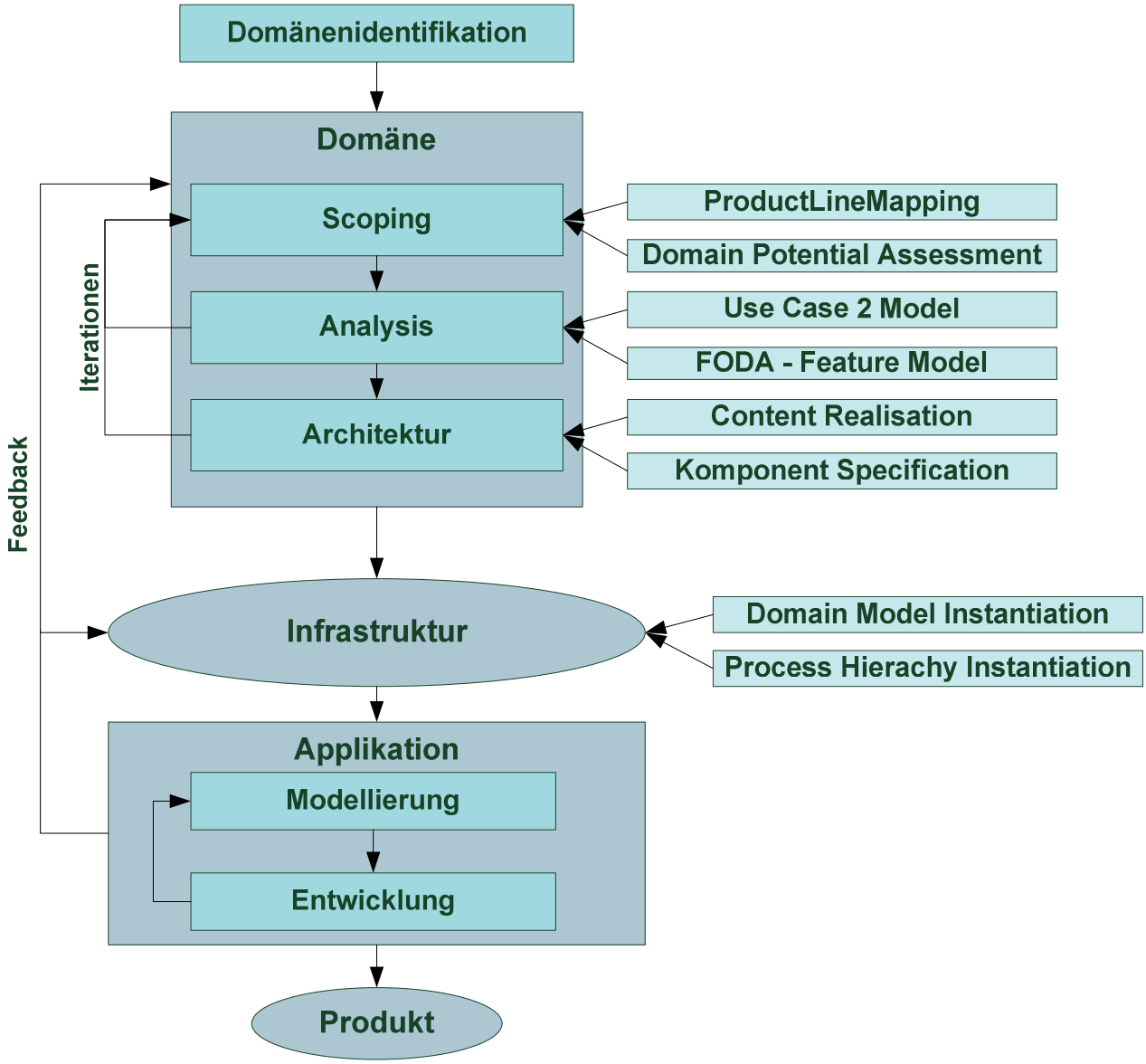


Abb. 5-1: SPLME-Modell

In Abbildung 5-1 ist das SPLME-Modell dargestellt. Es nutzt den grundsätzlichen Aufbau des FAST-Ansatzes, berücksichtigt aber neben den Beschränkungen, die bei der Entwicklung mit mobilen Endgeräten in Kapitel 2.3.2. auf Seite 16 aufgezeigt wurden, auch den PuLSE Ansatz. Das Modell schlägt weiterhin Werkzeuge für die Schaffung einer Domäne vor, die sich an das Fallbeispiel „GoPhone“ (Vgl. Muthig et al. (2004)) des Fraunhofer Institut für Experimentelles Software Engineering orientiert.

5.2 Beschreibung des Modells

Ein Unternehmen welches sich entscheidet eine Software-Produktlinie für mobile Endgeräte einzusetzen, muss als Erstes seine Domäne identifizieren. Auch im SPLME-Modell beginnt die Produktlinie mit der Domänen Identifikation. Dazu wird die geplante Produktlinie nach wirtschaftlichen Gesichtspunkten definiert und untersucht.

Nachdem die Produktlinie identifiziert ist, beginnt das Scoping. Das Scoping ist ein Teil der Domäne und hat das Ziel, der Identifikation der wirtschaftlich relevanten Domänen für die Produktlinie. Es wird durch das Product Line Mapping und des Domain Potential Assessment unterstützt.

Bei Entwicklungen für mobile Endgeräte muss beachtet werden, auf welchen Profilen und Konfigurationen die Anwendung laufen soll. Ein Unternehmen, welches Anwendungen für mobile Endgeräte entwickeln möchte, sollte schon in der Domäne festlegen, welche Mobilfunkgeräte unterstützt werden sollen. Durch ein Template kann geklärt werden, welche Konfiguration und welche Profile benötigt werden. Das Template wird in der Fallstudie genau spezifiziert.

Entweder die Organisation legt am Anfang fest, dass nur Produkte für eine spezifische Konfiguration und ein spezifisches Profil gefertigt werden und/oder sie packen es mit in die Domäne. Die Konfigurationen und Profile werden dann als variabler Ansatz für die Produkte gesehen und schränken damit teilweise die variablen Punkte für die Entwicklung ein. Auf Basis der Entscheidung der Konfiguration und des Profiles stehen andere Variabilitäten zur Verfügung. Eine Anwendung mit dem CLDC 1.0 besitzt zum Beispiel keine Fließkommazahlen.

Diese Schritte der Domänenidentifikation und des Scopings werden normalerweise nicht in einmaliger, sequentieller Form durchlaufen, Iterationen können und sollen auftreten. Das heißt nach einem Durchlauf sind nicht zwangsläufig alle Domänen erkannt worden. Durch die Risikoanalyse oder entwickelte Produkte können neue Domänen erkannt werden oder Domänen wieder abgeschafft werden, da sie nicht den Wiederverwendungswert und die Variabilität besitzen, welche sich das Unternehmen wünscht.

Nach dem Scoping folgt die Domänen Analyse. Für Produktlinien gibt es derzeit noch keine Standardisierung der Beschreibungssprache. Um alle Aspekte und Probleme beschreiben zu können, reichen herkömmliche Modelle wie das Feature Modell von FODA oder UML nicht aus, da sie allein nicht genügend Darstellungs- und Einstellungsmöglichkeiten bieten (Vgl. Grosse (2004), S.3). Daher wird in der SPLME eine Kombination aus Use Case 2 Modellierung, da Mobiltelefone sehr interaktiv benutzt werden, und dem Featuremodell aus FODA, um die Variabilitäten in der Produktlinie auszudrücken, benutzt. Es werden keine weiteren textlichen Beschreibungen benötigt, da alle funktionalen und nichtfunktionalen Anforderungen mithilfe der beiden Werkzeuge integriert sind. Die genaue Beschreibung des Ablaufs und der Funktionsweise wird im Fallbeispiel der Firma „GoGame“ erläutert.

Zur Instantiierung der Modelle wird ein Decision Model entwickelt. In tabellarischer Form wird beschrieben, wie sich Entscheidungen bezüglich der Features eines Produktes auswirken und wie die Use Cases angepasst werden müssen. Die Tabelle gibt Antworten auf Fragen, die bei der Instantiierung der Produkte entstehen.

Die Softwarearchitektur ist das Verbindungsprodukt zwischen den Anforderungen aus der Domänenperspektive und der Softwarelösung für ein spezifisches Produkt. Wenn zum Beispiel eine Software-Produktlinie zahlreiche Variabilitäten aufweist, wird Flexibilität als die Hauptanforderung an die Architektur gesehen. Ein Ansatz zu diesem Zweck ist die Komponenten-Orientierung unter Anwendung der KobrA Methode.

Im nächsten Schritt wird die Infrastruktur aus den Decision Modells abgeleitet. Dadurch entsteht für jedes Produkt das passende instanziierte Modell.

Im letzten Schritt wird aus den Modellen einzelne Applikationen modelliert und umgesetzt. Das Resultat ist das fertige Produkt.

Zu beachten gilt, dass bei jedem Schritt Erfahrungen gesammelt werden, die als Feedback wieder in die vorherigen Prozesse fließen. Das Modell ist ein lebendes Modell, das heißt es kann ständigen Veränderungen unterworfen sein. Es ist kein starres Gebilde, sondern wächst mit den Erfahrungen. Dabei gilt in diesem Zusammenhang als Wachsen auch die Verkleinerung, wenn durch Entwicklung einzelner Produkte festgestellt wird, dass die identifizierten Variabilitäten nicht den Nutzen liefern wie angenommen. Dann werden diese entweder angepasst oder entfernt.

6 Validation des Modells

In diesem Kapitel wird das in Kapitel 5.1 auf Seite 69 entwickelte Modell validiert. Dazu wird im ersten Unterpunkt die Fallstudie „GoGame“ untersucht. Dabei wird der „Grüne Wiese“-Ansatz zu Hilfe genommen, das heißt das Unternehmen kann ohne Altlasten entwickeln.

Der beschriebene Ansatz des Fallbeispiels ist in der Praxis häufig nicht gegeben, da das Unternehmen schon mit Produkten auf dem Markt vertreten ist. In diesem Fall ist ein Reverse Engineering notwendig. In Kapitel 4.3.5 auf Seite 55ff wurde das Reverse Engineering näher vorgestellt. Es wird in Kapitel 6.2 mithilfe einer existierenden Anwendung, einem Museumsführer von Sebastian König, versucht, das Reverse Engineering darauf anzuwenden. Des Weiteren soll untersucht werden, ob der Museumsführer das Potential für eine Software-Produktlinie besitzt und welche Probleme das Programm in der derzeitigen Version aufweist.

6.1 Fallstudie “GoGame”

Die Firma hat den Plan Spiele für Mobilfunktelefone mithilfe der Programmiersprache Java herzustellen. Damit wurde die Domäne identifiziert. Jetzt folgt das Scoping. Dieser beginnt, wie in Abbildung 5-1 auf Seite 69 dargestellt, mit dem Product Line Mapping. Darauf aufbauend wird das Modell abgearbeitet und versucht die Software-Produktlinie für das Fallbeispiel zu entwickeln. Danach folgt eine Auswertung, ob sich die entwickelte Software-Produktlinie für das Fallbeispiel lohnt.

6.1.1 Scoping

Das Scoping unterteilt sich wiederum in einzelne Unterschritte. Im Modell sind es das Produkt Line Mapping und das Domain Potential Assessment. Das Produkt Line Mapping unterteilt sich noch einmal in folgende Punkte (Vgl. Muthig et al. (2004), S.23ff). Als Erstes folgt das Produkt Portfolio. Danach werden die Produkte beschrieben. Nach der Beschreibung folgt ein Produktstammbaum, um planen zu können, wann welches Produkt erscheinen soll. Darauf aufbauend folgt eine Produktcharakterisierung, aus der eine Beschreibung der einzelnen Domänen entsteht. Aus den individuellen Beschreibungen der Domänen wird ein Überblick der Beziehungen zwischen den Domänen entwickelt. Nachdem die wesentliche Struktur der Domänen entwickelt wurde, dienen sie als Grundlage für zukünftige organisatorische Strukturen, welche in der Produkt Map dargestellt werden. Diese starke Unterteilung zeigt, wie wichtig das Scoping für eine Software-Produktlinie ist. Werden die Domänen nicht korrekt erkannt und unwirtschaftliche Features identifiziert, dann kann die Software-Produktlinie zu keinem Erfolg werden. Dieser Ablauf orientiert sich an dem Fallbeispiel „GoPhone“ (Vgl. Muthig et al. (2004), S.1ff), weil dieser sehr anschaulich ist, aber dennoch so abstrakt gehalten ist,

dass er für die speziellen Probleme des Fallbeispiels in dieser Diplomarbeit angepasst werden kann.

6.1.1.1 Produkt Portfolio

„Das Produkt Portfolio kann entweder durch das Marketing vorgegeben werden oder in einem systematischen Prozess entwickelt werden“ (Leichner (2005), S.3). In dieser Fallstudie wird angenommen, dass das Produktportfolio in Meetings mit allen Mitarbeitern definiert wurde. Es wurden dabei verschiedene Ideen entwickelt, für welches Spielgenre entwickelt werden soll:

- Logikspiele – Spiele, bei denen der Spieler durch geschicktes Lösen von Rätseln das Level lösen kann
- Geschicklichkeitsspiele – Spiele bei denen der Spieler durch geschicktes Ausweichen das Spiel gewinnen kann

Ein erstes Produktportfolio wurde entwickelt und dabei Spielideen aufgeschrieben. Dabei sind folgende Titel geplant worden:

- ApoDoor - Ein Logikspiel, bei welchem der Spieler Türen öffnen muss
- ApoIcejump – Ein Geschicklichkeitsspiel, bei welchem der Spieler den anderen Spieler unter Wasser tauchen muss
- ApoPolarium – Ein Logikspiel, bei welchem gleichfarbige horizontale Reihen geschaffen werden sollen

Diese Spiele sollen über einen Zeitraum von 3 Monaten entwickelt werden und danach schnell nacheinander auf den Markt gebracht werden, um einen signifikanten Marktanteil zu gewinnen. Es ist von Anfang an klar, dass kurze Zeit später weitere Spiele auf den Markt gebracht werden müssen. Für die zweite Runde plant das Unternehmen ein weiteres Logikspiel.

6.1.1.2 Produktbeschreibung

In dieser Phase werden alle geplanten Produkte mithilfe eines Templates detailliert beschrieben. In diesem Template wird im Detail beschrieben, in welchen Informationskategorien Daten gesammelt werden sollen. Diese Informationen werden für jedes Produkt separat ausgearbeitet (Vgl. Leichner (2005), S.4).

Das Template für die Firma „GoGame“ enthält folgende Informationen (Vgl. Muthig et al. (2004), S.24ff):

- **Name der Anwendung**: Wie soll das Spiel später heißen
- **System Status**: Wie ist der derzeitige Status des Spiels? Ist es hypothetisch, geplant, in Arbeit, umgesetzt oder schon in der Wartung für Verbesserungen?
- **Kurze Beschreibung**: Kurz beschreiben, um was handelt das Spiel? Die genauen Funktionalitäten kommen später.
- **Hauptfunktionen**: Hier soll in 5 bis 10 Stichpunkten beschrieben werden, was das Spiel ausmacht und welche Funktionen es besitzen soll
- **Konfiguration**: Welche Konfiguration muss das Mobilfunktelefon besitzen, damit das Spiel laufen kann? Welche CLDC wird benötigt? Oder wird CDC vorausgesetzt? Soll für unterschiedliche Konfigurationen entwickelt werden?
- **Profile**: Welches Profil muss das Mobilfunktelefon besitzen, damit das Spiel läuft. Welches MIDP-Profil benötigt das Spiel? Soll das Spiel für unterschiedliche Profile entwickelt werden?
- **Optionale Pakete**: Benötigt das Spiel optionale Pakete wie zum Beispiel die Bluetooth-Unterstützung? Welche werden benötigt?
- **Auflösungen**: Welche Auflösungen soll das Spiel unterstützen? Soll es verschiedene Versionen für die Auflösungen geben oder soll sich das Spiel an die Auflösung anpassen?
- **Marktsegment**: Dabei soll geklärt werden in welches Marktsegment das Spiel kommen soll. Soll es zum Beispiel ein kostenlosen Spiel werden, im Niedrigpreissegment angesiedelt werden oder ist es ein Titel, welcher mit einem höheren Preis angesetzt werden soll? Dazu folgt noch die Frage, welche Kundengruppen angesprochen werden sollen. Sollen vor allem jugendliche Spieler angesprochen werden oder Familien oder nur Frauen?
- **Einzigartigkeitsabgrenzung**: Was macht dieses Spiel einzigartig? Wie grenzt es sich zu bestehenden Spielen der Firma ab und wie auf dem Markt zu bestehenden?
- **Individuelles System**: Gibt es in diesem Spiel Features, welche nur in diesem Spiel vorkommen? Wenn ja wie viele?
- **Releaseplan**: Wann soll das Spiel herauskommen? Wie ist der Plan?

Dieses Template muss für jedes Spiel benutzt werden. Exemplarisch wird es nun für das ApoDoor-Spiel genutzt, um zu zeigen, wie es aussehen kann:

- **Name der Anwendung**: ApoDoor
- **System Status**: In Planung
- **Kurze Beschreibung**: In ApoDoor soll der Spieler seine Spielfigur durch geschicktes Laufen zum Endpunkt bewegen. Dabei muss er Türen öffnen beziehungsweise schließen oder aber Falltüren so geschickt betreten, dass er nicht in den Abgrund fällt.
- **Hauptfunktionen**: Spieler muss Türen öffnen, um zum Ziel zu gelangen. Spieler kann sich in alle 4 Richtungen bewegen, nach oben, unten, links und rechts. ApoDoor soll eine Highscore beinhalten, welche sich merkt, wenn ein Level gelöst wurde, wie es gelöst wurde. Es soll leichte, mittelschwere und schwere Level geben. Das Spiel soll eine Hilfe beinhalten und mit einer Menuauswahl starten, in der die einzelnen Menupunkte: Start, Hilfe, Credits und Beenden zu finden sind. Spiel soll über die Tasten 2, 4, 6, 8 oder aber optional über den Stick steuerbar sein.
- **Konfiguration**: Das Spiel benötigt die CLDC 1.1. ApoDoor wird CLDC 1.0. nicht unterstützen.
- **Profile**: Das Spiel benötigt mindestens die MIDP 2.0. Geplant ist es nicht für die Version 1.0.
- **Optionale Pakete**: ApoDoor benötigt keine optionalen Pakete. Es werden nur Funktionen und Methoden benötigt und genutzt, die von dem Profil und der Konfiguration gegeben sind, genutzt.
- **Auflösung**: ApoDoor zentriert den Bildschirm immer so, dass es wegen der Auflösung keine unterschiedliche Versionen geben muss.
- **Marktsegment**: Das Spiel soll als kostenlose Version mit nur sehr wenigen Levels angeboten werden. Eine normale Version, welche nur über normale Türen verfügt, soll im Bereich bis 1 Euro angesiedelt werden. Eine andere Version mit Falltüren, Doppeltüren, normalen Türen und mehr Levels soll im Segment bis 3 Euro pro Spiel im Markt platziert werden. Zielgruppe ist nicht auf ein Geschlecht spezifiziert, sondern richtet sich an alle casual Spieler, die gerne knobeln. Durch den Verzicht auf Gewalt soll es auch Familien und Kleinkinder ansprechen.

- **Einzigartigkeitsabgrenzung**: Spielprinzip setzt nicht auf bekannte Logikspiele, sondern ist ein neuartiges System.
- **Individuelles System**: Derzeit noch nicht absehbar.
- **Releaseplan**: Spiel soll in 3 Monaten auf den Markt kommen.

6.1.1.3 Produktstammbaum

Nach der Produktbeschreibung folgt der Produktstammbaum. Der Produktstammbaum gibt einen Überblick über die Entwicklung der Produktlinie über die Zeit. In diesem werden die Hauptgemeinsamkeiten und Unterschiede der Produktlinie dargestellt. Damit wird eine Abschätzung, ob es genügend Gemeinsamkeiten für eine Plattformentwicklung gibt, ermöglicht (Vgl. Leichner (2005), S.5).

Im Fallbeispiel sieht der Produktstammbaum folgendermaßen aus:

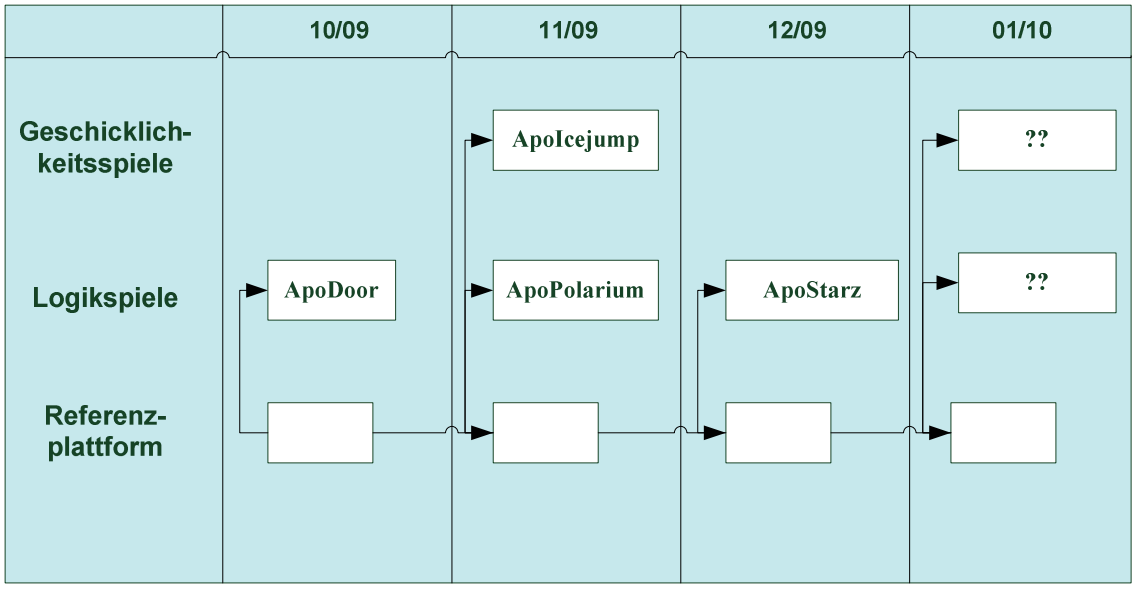


Abb. 6-1: Produktstammbaum

In Abbildung 6-1 wird der Produktstammbaum dargestellt. Der Plan sieht vor, als Erstes ApoDoor zu programmieren, danach folgt ApoIcejump und ApoPolarium und im folgenden Monat folgt dann ApoStarz. Danach kommen neue Spiele, die bis jetzt noch nicht in Planung sind. Vielleicht wird auch ein anderes Genre bedient und erweitert somit die Software-Produktlinie. Der Produktstammbaum ist wie die anderen Bereiche im Scoping nicht fest, sondern kann durch Iterationen erweitert oder auch verkleinert werden. Es ist ein sprichwörtlich ‚lebendes Modell', was sich an die Anforderungen anpassen kann.

6.1.1.4 Produktcharakterisierung

In der Produktcharakterisierung wird aus der Produktbeschreibung die erste Version der Produkt Map entwickelt. Bei der Produkt Map handelt es sich um eine Tabelle, die einen Überblick über die Produkte und deren zugehörige Funktionalitäten bereitstellt. Sie gibt Informationen über die geplanten Funktionen der Systeme der Produktlinie (Vgl. Leichner (2005), S.5).

In der folgenden Tabelle wird die Produktcharakterisierung für den Bereich der Spielobjekte gezeigt.

Area	*Feature*	*GoGame Spiele*			
		ApoDoor	ApoIcejump	ApoPolarium	ApoStarz
Spielobjekte	x-Wert	**X**	**X**	**X**	**X**
	y-Wert	**X**	**X**	**X**	**X**
	Breite	**X**	**X**	**X**	**X**
	Höhe	**X**	**X**	**X**	**X**
	Schnittpunktbrechnung mit anderen Objekten		**X**		**X**
	x-Wert für die Geschwindigkeit	**X**	**X**		**X**
	y-Wert für die Geschwindigkeit	**X**	**X**		**X**
	Farbe	**X**	**X**	**X**	
	Bild				**X**

Tabelle 6-1: Produktcharakterisierung für Spielobjekte

In Tabelle 6-1 zeigt sich, dass die Spielobjekte in den einzelnen Spielen unterschiedliche Funktionen benötigen. Unter Spielobjekten wird zum Beispiel ein Gegner bezeichnet oder der Hauptcharakter. Diese benötigen für die zu entwickelnden Spiele jeweils einen x und y-Wert. Auch eine Breite und Höhe wird benötigt. Dagegen benötigt ein

ApoStarz vorgefertigte Bilder. Die anderen Spiele zeichnen die benötigten Objekte zur Laufzeit aus Rechtecken und Ellipsen mit unterschiedlichen Farben.

6.1.1.5 Beschreibung der Domänen

Mit Hilfe aller Mitarbeiter im Fallbeispiel oder aber durch Experten werden interne Funktionalitäten identifiziert, welche wichtig für die Produkte sind. Ziel ist die Identifikation der Domänen, welche wichtige Funktionalitäten für die gesamte Produktlinie beinhalten. In den vorhergehenden Schritten bestand die Aufgabe in der Beschreibung der Produkte. Bei der Beschreibung der Domänen gilt es hinreichende Informationen herauszufinden, um Domänen zu identifizieren und abzugrenzen (Vgl. Leichner (2005), S.5).

Dazu wird wieder ein Template zu Hilfe gezogen, an welchem sich orientiert werden soll (Vgl. Muthig et al. (2004), S.33ff):

- **Name**: Wie heißt die Domäne?
- **Informationsgrundlage**: Wie und vom wem wurde die Domäne identifiziert?
- **Hauptfunktionen**: Welche Hauptfunktion besitzt die Domäne?
- **Regeln**: Nach welchen Regeln wird die Domäne innerhalb der Software-Produktlinie verwendet?
- **Funktionen**: Welche Funktionen bietet diese Domäne?
- **Daten**: Welche Daten und Objekte werden aus anderen Domänen benötigt und/oder verarbeitet?
- **System-Charakteristika**: Welche Assets können schon verwendet werden aus der Software-Produktlinie? Welche Funktionen kommen in welchem Produkt zum Einsatz?

Auf Grundlage dieses Templates wird exemplarisch für das Fallbeispiel gezeigt, wie es für die identifizierte Domäne ‚Spielobjekt' beschrieben werden kann. Dabei wird der Übersicht halber die Funktionsbeschreibung etwas gekürzt:

- **Name:** Spielobjekt
- **Informationsgrundlage**: Diese Domäne wurde durch das gesamte Team identifiziert.

- **Hauptfunktionen**: Das Spielobjekt stellt Objekte für das Spiel zur Verfügung. Es beinhaltet Angaben zur Positionierung und Ausgabe.

- **Regeln**: Die Domäne bekommt vom Nutzer Angaben zum Richtungswechsel und zur Positionierung und stellt diese der Spielkomponente zur Verarbeitung mit anderen Objekten und zum Zeichnen zur Verfügung.

- **Funktionen**: Das Spielobjekt beinhaltet den x-Wert, den y-Wert, die Breite, die Höhe, die Schnittpunktberechnungen mit anderen Objekten aus der Domäne, die Zeichenfunktionen, die Logikfunktionen, damit das Objekt sich bewegt, Funktionen, ob das Objekt angezeigt werden soll oder ob es ausgewählt wurde und vieles mehr.

- **Daten**: Die Domäne beinhaltet Bilder, Strings, float-Werte und boolean-Werte. Diese Werte bekommt die Domäne bei der Initialisierung und kann später durch Nutzereingaben und die Spielkomponente verändert werden.

- **System-Charakteristika**: Es müssen zu diesem Zeitpunkt noch alle Assets für die Domäne entwickelt werden. Die Funktionen der Positionierung werden in allen Produkten zur Verwendung kommen.

6.1.1.6 Relationsstruktur der Domänen

Aus den individuellen Beschreibungen der Domänen wird ein graphischer Überblick der Beziehungen zwischen den Domänen entwickelt (Vgl. Leichner (2005), S.8). In Abbildung 6-2 wird dies dargestellt. Das UserInterface interagiert mit dem Spielobjekt, dem Highscore und den Credits. Das sind für den Endanwender sichtbare Domänen. Die darunter liegenden Domänen, wie zum Beispiel die Spielkomponente oder die Spiellogik, sind für die interne Realisierung der Funktionalitäten zuständig.

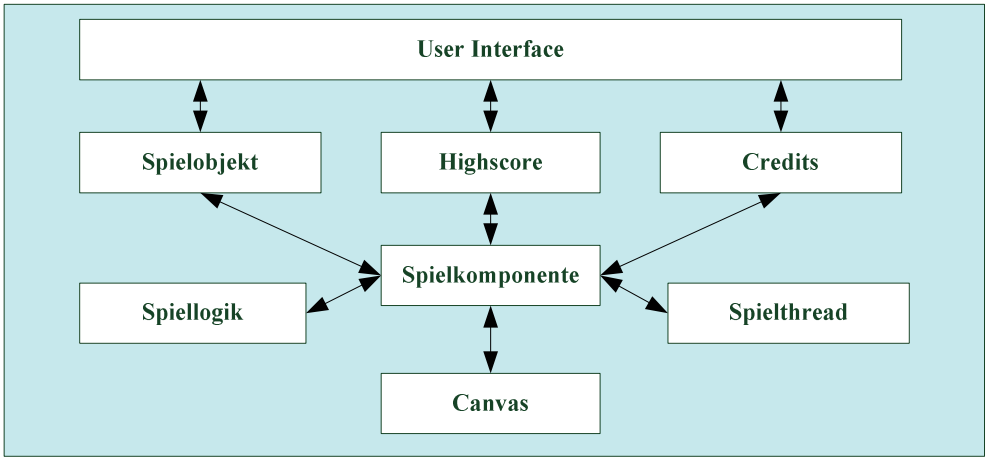


Abb. 6-2: Relationsstruktur der Domänen

6.1.1.7 Produkt Map

Nachdem die wesentliche Struktur der Domänen entwickelt wurde, werden sie als Grundlage für die zukünftigen organisatorischen Strukturen genutzt. Aus der Domänenbeschreibung wird die Produkt Map gebildet. Diese Tabelle enthält alle Informationen über die Hauptfeatures, die für die Produktlinie relevant sind (Vgl. Leichner (2005), S.8).

Die folgende Tabelle zeigt einige Beispielfeatures für die Spielobjektdomäne.

Domäne	*Feature*	*Sub-feature*	*GoGame Spiele*			
			Apo Door	ApoIcejump	ApoPolarium	ApoStarz
Spiel-objekte	Positionierung	X-Wert	**X**	**X**	**X**	**X**
		Y-Wert	**X**	**X**	**X**	**X**
		Breite	**X**	**X**	**X**	**X**
		Höhe	**X**	**X**	**X**	**X**
	Schnittpunkt-berechung	Punkt	**X**	**X**		**X**
		Objekt		**X**		**X**
		Rechteck		**X**		**X**
	Geschwindigkeit	X-Wert	**X**	**X**		**X**
		Y-Wert	**X**	**X**		**X**

Tabelle 6-2: Produkt Map

Tabelle 6-2 zeigt einen Auszug der Produkt Map für die Domäne der Spielobjekte. Es wurden Features zu Hauptfeatures zusammengefasst. So gibt es nun das Feature der Positionierung, welches die Sub-Features enthält, die schon in der Produktcharakterisierung identifiziert wurden.

6.1.1.8 Domain Potential Assessment

Im Domain Potential Assessment werden die Chancen und Risiken jeder Domäne analysiert. Dazu werden Interviews durchgeführt und somit das Potential jeder Domäne einzeln untersucht. Im Fallbeispiel werden alle Beteiligten des Unternehmens befragt.

Die Befragung gliedert sich in drei Phasen - der Vorbereitung, der Ausführung und der Analyse (Vgl. Leichner (2005), S.9).

In der Vorbereitungsphase wird die Produkt Map fertig gestellt, damit diese diskutiert werden kann. Außerdem wird ein detaillierter Zeitplan für die Interviews mit den beteiligten Personen aufgestellt (Vgl. Leichner (2005), S.9).

Die Interviews werden in der Ausführungsphase geführt. Nach Auswertung des Feedbacks werden noch einmal die Beteiligten mit den Ergebnissen konfrontiert, um weitere Informationen zu erhalten und Fehler zu entdecken. Wenn dieser Prozess abgeschlossen ist, folgt die Analysephase, in welcher ein detaillierter Bericht erstellt wird (Vgl. Leichner (2005), S.9).

Das Ergebnis der Aktivität ist eine Gesamtauswertung, ob die entdeckten Domänen für eine Software-Produktlinienentwicklung geeignet sind. In größeren Firmen müssen Faktoren wie das Marktpotential für jede Organisationseinheit getestet und untersucht werden. Da im Fallbeispiel „GoGame“ die gesamte Softwareentwicklung aus einer einzigen Organisationseinheit besteht, müssen diese Daten nur einmal erhoben worden (Vgl. Leichner (2005), S.10).

Im Fallbeispiel sind sowohl die Ergebnisse für die Gesamtproduktlinie als auch die Ergebnisse für die individuellen Domänen sehr positiv. Dennoch werden die Domänen in einer Rangliste für die Wiederverwendung ausgelistet.

6.1.2 Analyse

Durch das Scoping wurden die Domänen mit einem hohen Wiederverwendungspotential ermittelt. In der Fallstudie wird in dieser Phase der Analyse die Domäne des Spielobjektes behandelt (Vgl. Leichner (2005), S.11).

Dazu wird das FODA-Feature Diagramm zusammen mit dem 2-Level Use Case verwendet. Die Abbildung 6-3 stellt die Spielobjekt-Domäne dar. Darauf ist zu erkennen, dass die Positionierung ein optionales Feature ist. Dagegen werden die Schnittpunktberechnung und das Darstellungsfeature immer benötigt. Für eine Berechnung des Schnittpunktes zwischen dem Spielobjekt und anderen Spielobjekten wird das Positionierungsfeature und davon mindestens der X und der Y-Wert benötigt.

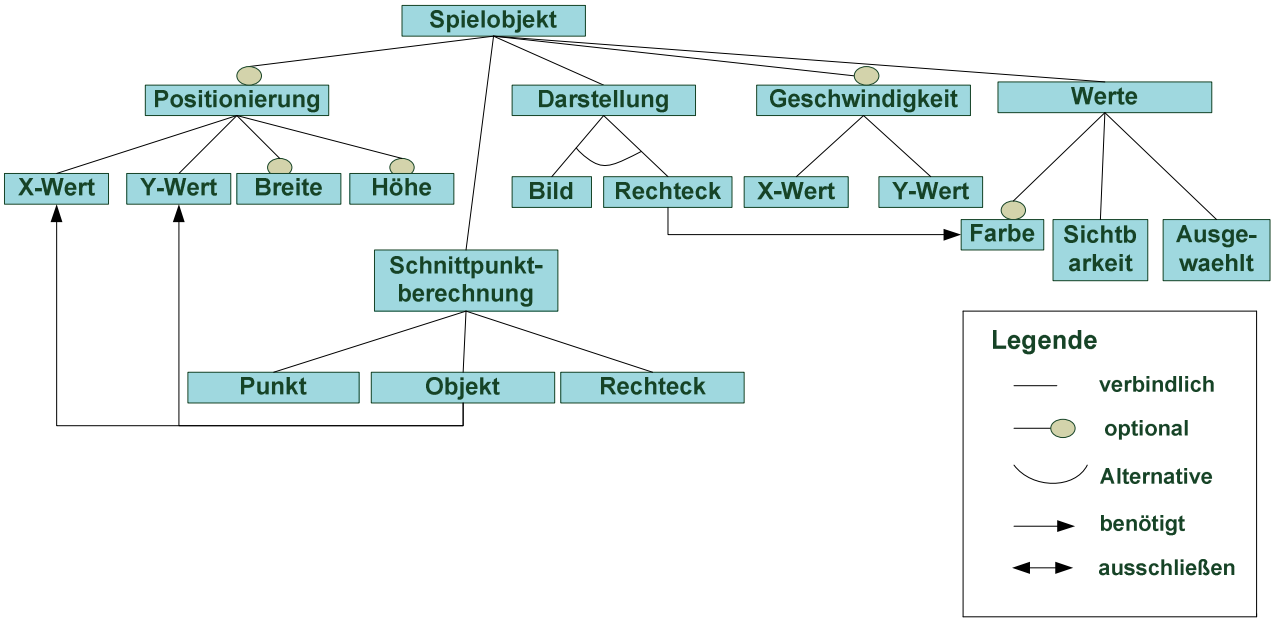


Abb. 6-3: Feature Diagramm für die Spielobjekt-Domäne

Ein 2-Level Use Case für die Schnittpunktberechung sieht folgendermaßen aus:

- **Use Case Name**: [V] Schnittpunktberechnung
- **Nutzer**: Mobilfunkbenutzer
- **Scope**: Spielobjektdomäne
- **Interessen**: Spieler möchte wissen, ob er mit einem anderen Objekt kollidiert
- **Bedingung**: Spieler oder Objekt hat sich bewegt
- **Hauptszenario**:
  - Der Nutzer hat das Spiel gestartet
  - Der Nutzer oder ein Objekt haben sich bewegt
  - Schneiden sich die beiden Objekte? <OPT> Der Spieler schneidet einen Punkt, ein Objekt oder ein Rechteck
  - Spieler schneidet den Spieler? <ALT 1> Spieler schneidet Punkt (Produkte: ApoDoor, ApoStarz, ApoIcejump), <ALT 2> Spieler schneidet Objekt (Produkte: ApoStarz, ApoIcejump), <ALT 3> Spieler schneidet Rechteck (Produkte: ApoStarz, ApoIcejump)

6.1.3 Architektur

In nächsten Schritt wird die Softwarearchitektur der Produkte erstellt. Sie ist das Verbindungsprodukt zwischen den Anforderungen aus der Domänen Perspektive und der Softwarelösung für ein spezifisches Produkt. Da in diesem Fallbeispiel „GoGame“ zahlreiche Variabilitäten existieren, wird eine flexible Struktur als Hauptanforderung an die Architektur gesehen. Ein Ansatz zu diesem Zweck ist die Komponenten-Orientierung unter Anwendung, der in Kapitel 4.3.4. auf Seite 53 vorgestellten KobrA-Methode (Vgl. Leichner (2005), S.15).

Gestartet wird mit der Umgebungsbeschreibung des Systems. Dazu wird ein Enterprise Modell und eine Prozess Hierarchie erstellt.

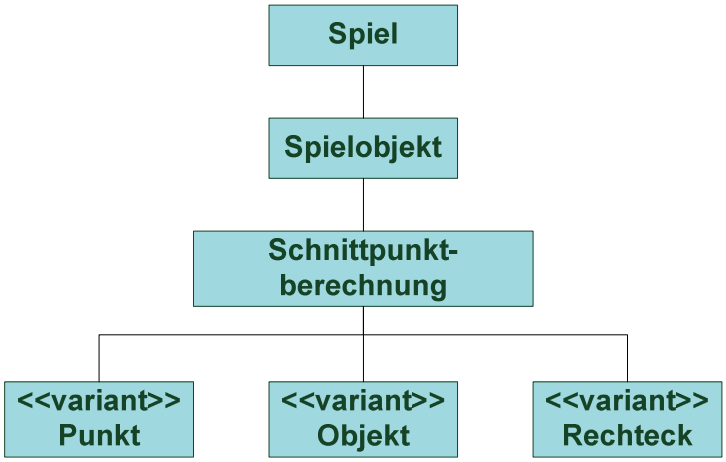


Abb. 6-4: Prozess Hierarchie

Während das Enterprise Modell die fundamentalen Rollen und Entities zeigt, analysiert die Prozess Hierarchie die unterschiedlichen Prozesse. Das Ziel dieser Aktivitäten ist die Entwicklung einer ersten Beschreibung des Systems (Vgl. Leichner (2005), S.15).

In Abbildung 6-4 wird ein Ausschnitt der Prozess Hierarchie für ein Spiel gezeigt.

Mithilfe der vorangegangenen Schritte wird die erste Komponente, die Spielkomponente, entwickelt. Für jede Komponente wird ein Spezifikationsmodell modelliert. Dieses Spezifikationsmodell verdeutlicht die Sichtweise von außen auf die Komponente. Die genaue Implementation der Funktionen wird in den Realisierungsmodellen beschrieben (Vgl. Leichner (2005), S.16).

6.1.4 Infrastruktur

In dieser Phase werden die einzelnen, konkreten Produkte abgeleitet. Für jedes Produkt entsteht ein passendes instanziiertes Modell (Vgl. Leichner (2005), S.17).

Um das Modell zu instanziieren wird das Domain Modell aus dem Domain-Analysis-Prozess übernommen. Als Beispiel soll das ApoDoor-Spiel fungieren. Gemäß der Produkt Map (siehe Kapitel 6.1.1.7) verfügt dieses Spiel zum Beispiel über keine Schnittpunktberechung für Objekte und Rechtecke. Aus diesem Grund wird in diesem Modell dieses Feature nicht implementiert werden.

6.1.5 Applikationsentwicklung

Nachdem die Modelle für die einzelnen Produkte entwickelt wurden, beginnt die Umsetzung der Modelle. Dazu werden die einzelnen Komponenten angepasst. Zum Beispiel wurde in ApoIcejump unter anderem das Credits-Feature implementiert.

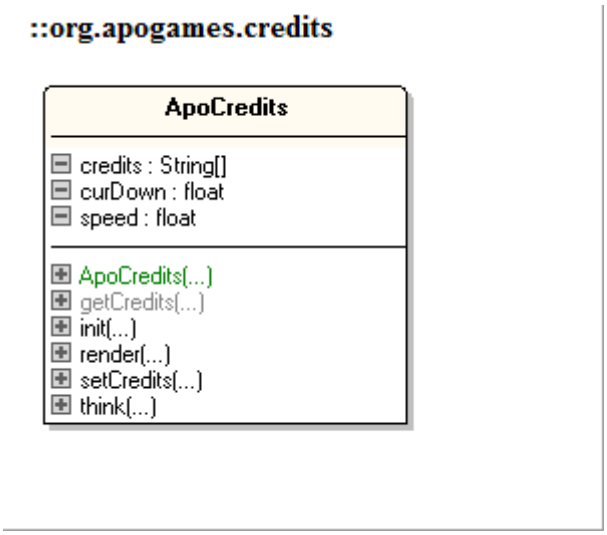


Abb. 6-5: UML-Diagramm des Credits-Features

In Abbildung 6-5 wird ein UML-Diagramm des Credits-Features dargestellt. Diese Komponente besitzt drei Variablen - credits, curDown und speed. Die grün dargestellte Methode ist die Konstruktor-Methode der ApoCredits-Klasse. Dazu besitzt es noch einige Methoden zur Darstellung und zur Verarbeitung der Logik.

In der Abbildung 6-6 wird das komplette UML-Diagramm für das ApoIcejump Spiel dargestellt. Durch die Größe ist es bildlich nicht gut zu erkennen, aber alle benötigten Entwicklungen für das Spiel basieren auf der entwickelten Software-Produktlinie. Auf der beiliegenden CD sind der Sourcecode der Spiele und das UML-Diagramm in Originalgröße zu finden.

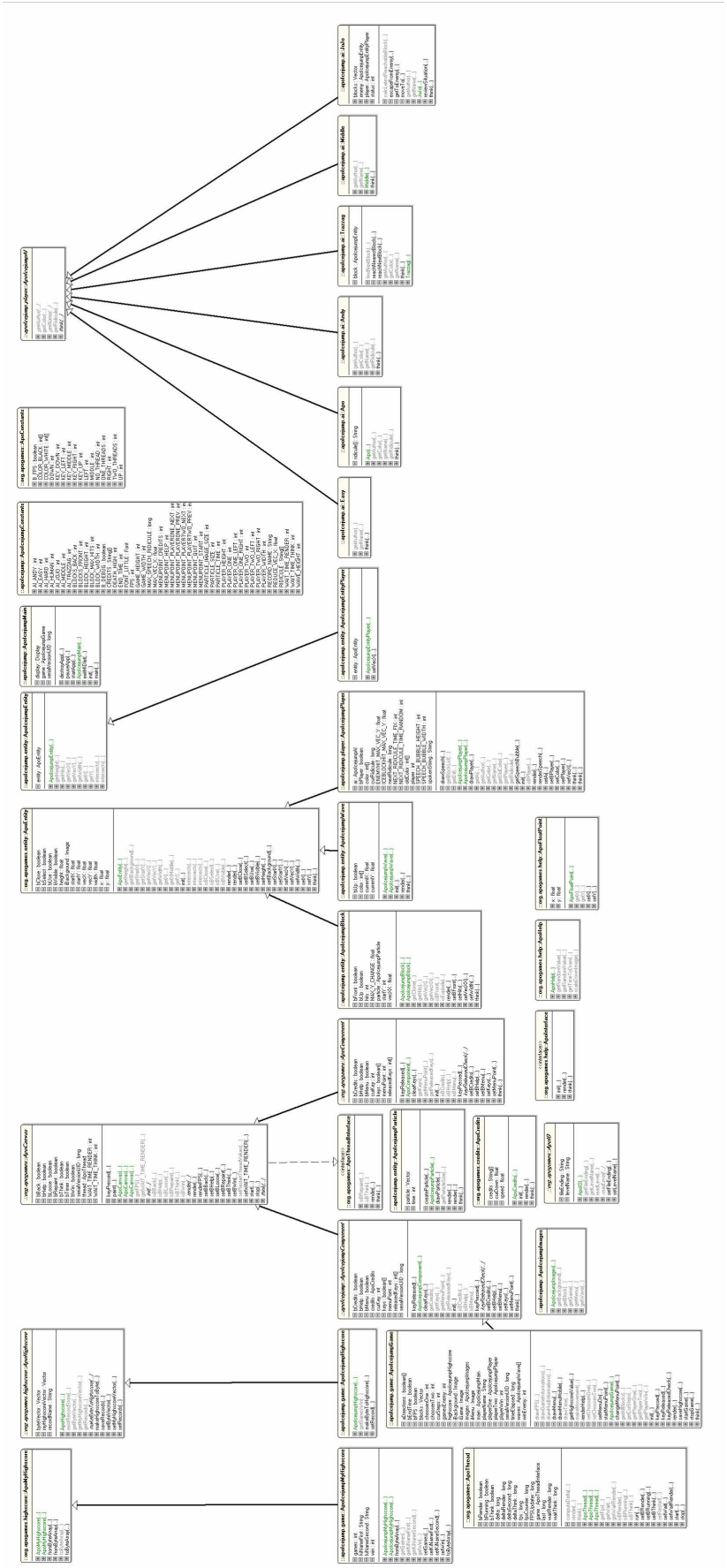


Abb. 6-6: UML Diagramm von ApoIcejump

6.1.6 Auswertung

Es gibt einige Faktoren die zur Messung, ob die Software-Produktlinie Vorteile gegenüber einer Einzelentwicklung besitzt, eingesetzt werden können.

Als Erstes ist, wie in Abbildung 2-10 auf Seite 27 verdeutlicht, die Verringerung des Produktionsaufwandes zu nennen. So können Kosten und Zeit durch die Software-Produktlinie eingespart werden. Im Fallbeispiel „GoGame“ müssen durch die Verwendung der Komponentenbasierten Implementierung, wie in Abbildung 4-20 auf Seite 62 dargestellt, der Entwickler die Komponenten noch anpassen, aber es werden Zeit und Kosten eingespart. Eine Anpassung benötigt weniger Zeit als eine komplette Neuentwicklung und spart damit auch Kosten ein. Nach Entwicklung der Software-Produktlinie für das Fallbeispiel konnten die Spiele in weniger als einer Woche pro Spiel entwickelt werden. Diese kurze Entwicklungszeit ist ohne eine Software-Produktlinie nicht möglich gewesen.

Ein weiterer Vorteil ist die erhöhte Qualität der Software. Durch die Wiederverwendung und Anpassung bestehender Komponenten besitzt die Anwendung ein geringeres Risiko Fehler zu beinhalten. Die Komponenten werden in der Software-Produktlinie ständig weiterentwickelt, angepasst und Fehler beseitigt, sodass die Entwickler eine qualitativ hochwertige Grundlage bei der Entwicklung besitzen. Die Messung der Qualität kann mithilfe von Interviews der Kunden durchgeführt werden. Im Fallbeispiel konnten die Spiele fehlerfrei programmiert werden. Dazu wurden die Spiele auf unterschiedlichen Mobilfunktelefonen installiert und getestet. Jedes entwickelte Spiel funktionierte tadellos und ist nicht ein einziges Mal angestützt.

6.1.7 Screenshots der Spiele

In diesem Kapitel werden einige Screenshots der entwickelten Spiele dargestellt. Sie zeigen, dass der Aufbau der Spiele im Menu sehr ähnlich ist und die Logikspiele mithilfe eines 2-dimensionalen Arrays einen sehr ähnlichen Aufbau besitzen. Durch diese identifizierten Gemeinsamkeiten, konnten die Spiele schneller entwickelt werden und besitzen eine durchgehend hohe Qualität, weil nicht alles neu programmiert werden musste.

Abb. 6-7: ApoDoor Menu

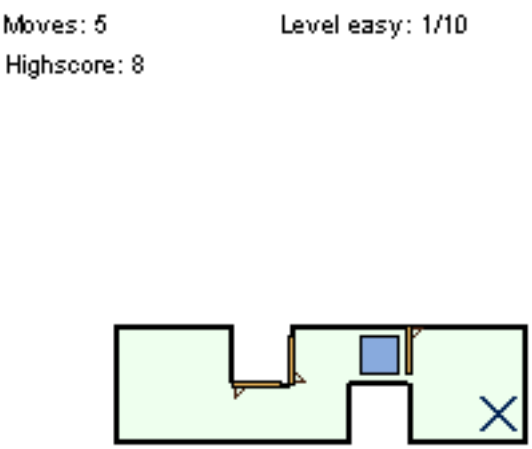


Abb. 6-8: ApoDoor Level

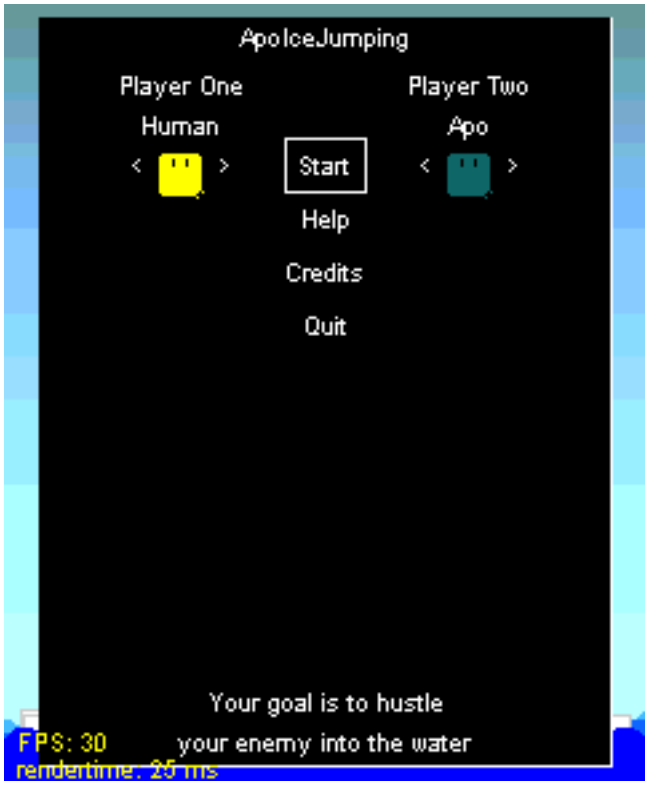


Abb. 6-9: ApoIcejump Menu

Abb. 6-10: ApoIcejump ingame

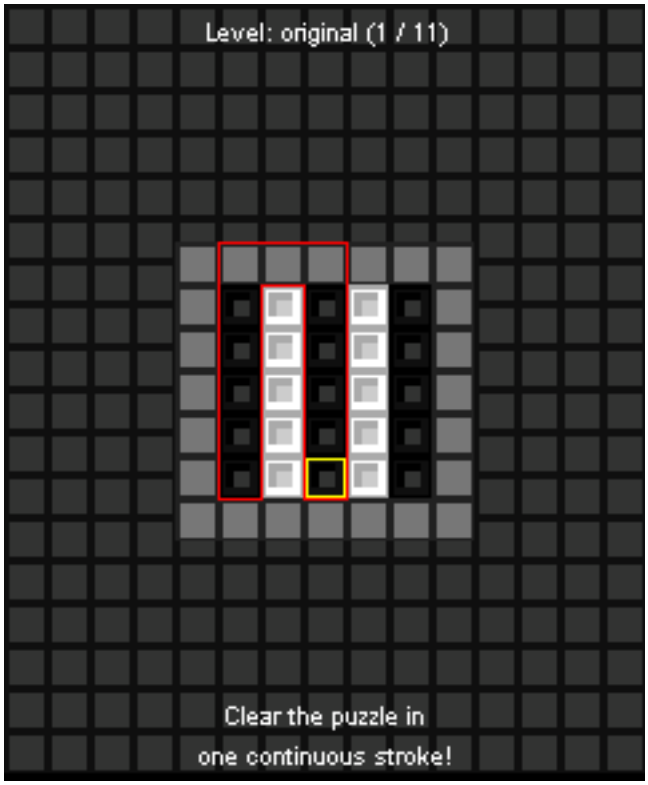


Abb. 6-11: ApoPolarium ingame

Abb. 6-12: ApoPolarium Level

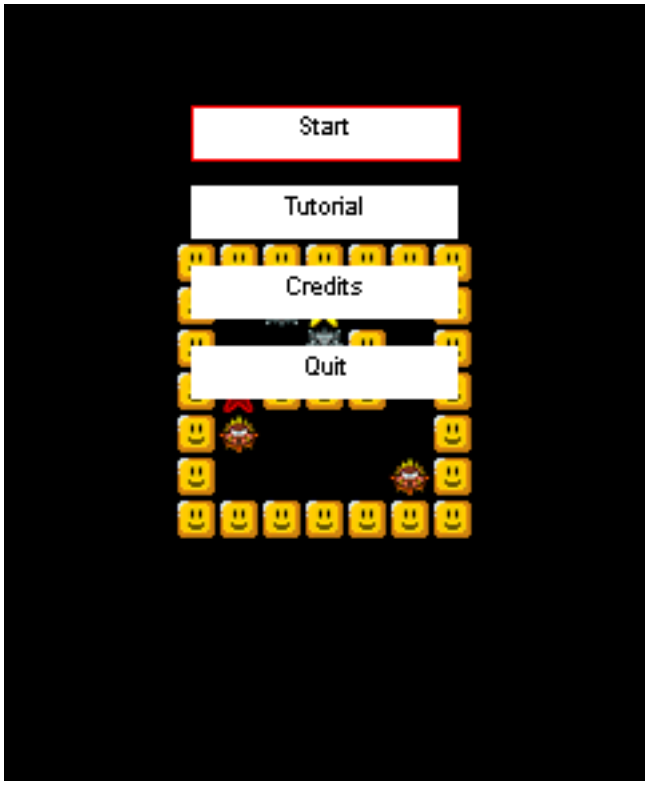


Abb. 6-13: ApoStarz Menu

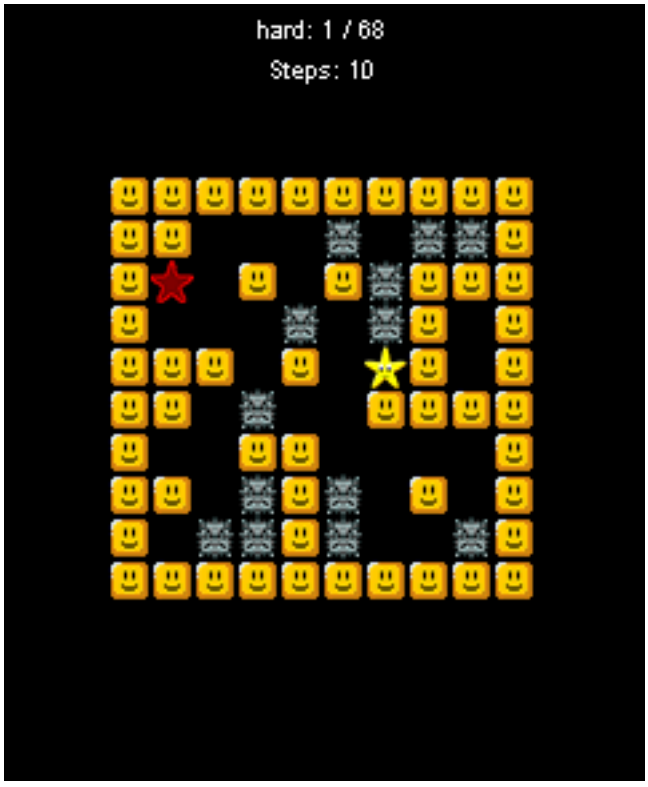


Abb. 6-14: ApoStarz Level

6.2 Validation mithilfe des Museumsführers

Ein Museumsführer hat die Aufgabe eine Person oder eine Gruppe von Personen durch Museen zu leiten. Dabei informiert der Museumsführer die Besucher über die einzelnen, ausgestellten Exponate. Die Führungen finden in der Regel stündlich statt, sodass Besucher, die zu spät angekommen sind, eine lange Wartezeit in Kauf nehmen müssen.

Neben den personellen Museumsführern gibt es auch Druckerzeugnisse, die relevante Informationen zu allen Objekten im Museum beinhalten. Diese können entweder als Heft erworben werden oder sie stehen als Hinweise vor dem Ausstellungsobjekt. In den letzten Jahren haben sich auch elektronische Pendants entwickelt. So gibt es z.B. im Deutschen Museum in München vor einigen Exponaten einen Knopf. Wird dieser betätigt, liest eine Damenstimme einen Text mit Informationen vor. Aber nicht jedes Exponat hat eine derartige Unterstützung. Viele Ausstellungsstücke stehen ohne weitere oder nur mit sehr spärlichen Informationen an ihrem Platz.

6.2.1 Vorstellung des Museumsführers von Sebastian König

In dem entwickelten, mobilen Museumsführer von Sebastian König bekommt der Nutzer zu Beginn seiner Museumsführung das Programm auf sein Mobilfunktelefon gespielt. Nach Starten des Programms versucht das Programm mithilfe der Bluetooth-Schnittstelle einen Server zu finden und sich zu verbinden. In jedem Raum mit Exponaten soll ein Server stehen, dass mit dem Programm kommunizieren kann. Jedes Exponat besitzt eine Nummer, die der Nutzer im Museumsführer eingeben kann. Wird diese Nummer gefunden, öffnet das Programm ein neues Fenster und gibt dem Nutzer mehrere Möglichkeiten zur Auswahl. Zum Beispiel kann der Nutzer Texte zu dem Exponat bekommen, Bilder, Videos oder wenn vorhanden auch Sprachdateien. Diese Daten werden, wenn sie aufgerufen werden, mithilfe der Bluetooth-Schnittstelle an das Mobilfunktelefon gesendet und der Nutzer kann die Multimediadateien öffnen. Der Vorteil zu bestehenden Museumsführern für Mobilfunktelefone ist die Tatsache, dass die anderen Anwendungen alle Daten im Programm gespeichert haben und dadurch nur eine beschränkte Anzahl an Exponaten vorstellen können. Die Beschränkungen kommen durch die in Kapitel 2.3 auf Seite 14f vorgestellten Einschränkungen für Mobilfunktelefone. So darf das eigentliche Programm zum Beispiel nicht größer als zwei Megabyte sein. Damit konnten keine Sprachnachrichten oder Videos implementiert werden, da diese zu groß gewesen wären. Der Museumsführer von Sebastian König hebt diese Einschränkung durch die Übertragung nur der gerade benötigten Daten auf.

Des Weiteren ist auch ein einfaches Bezahlmodell implementiert. Der Kunde bekommt an der Kasse einen Code, der es dem Nutzer erlaubt, die Anfragen an die Exponate zu stellen und Antworten zu erhalten. Die Kommunikation mit dem Server funktioniert

mithilfe von XML. Auch ist ein einfaches Speicherprogramm implementiert. So werden schon einmal ausgewählte Aktionen notiert und die Daten über das Handy nach einmaligem Aufrufen beim ersten Start auch gespeichert.

Falls kein Server gefunden wurde, gibt das Programm auch Feedback, welches in Abbildung 6-15 dargestellt ist.

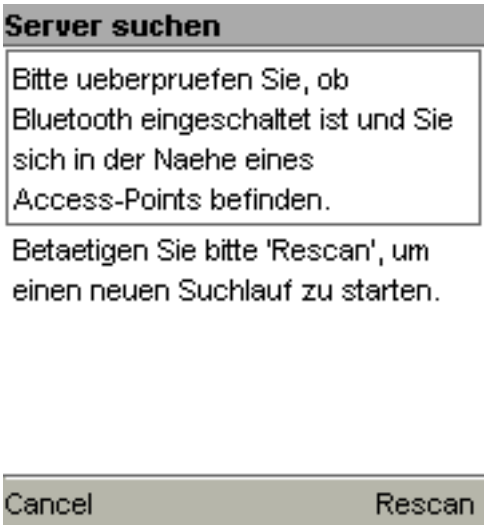


Abb. 6-15: Museumsführer (Server suchen)

Durch das neue Konzept der Übertragung von Daten soll im Folgenden untersucht werden, ob sich eine Software-Produktlinie auf Basis des Museumsführers lohnt. Das Konzept könnte in anderen Anwendungen ebenfalls implementiert werden. Deshalb werden kurz die Probleme im Museumsführer aufgezeigt und danach versucht einzelne Komponenten zu identifizieren, auf deren Basis die Software-Produktlinie entwickelt werden könnte.

6.2.2 Probleme

Die Idee mit der Übertragung nur der Daten, welche benötigt werden, ist ein erfrischender Ansatz. Aber es wurde mithilfe eines bestimmten Mobilfunktelefons entwickelt. So kommt es zu Problemen auf anderen Mobilfunkgeräten. Ein Ziel eines mobilen Museumsführers ist aber die große Verbreitung und das Laufen auf vielen verschiedenen Systemen. Diese Ziele können durch die Anwendung des entwickelten Software-Produktlinienmodells untersucht und verwirklicht werden. Ziel ist es neben der Beseitigung des angesprochenen Problems, auch das Identifizieren implementierter Featurekomponenten, welche wiederverwendet werden könnten für neue Produkte ähnlicher Art.

6.2.3 Implementierung des Museumsführers in das Software-Produktlinienmodell

Für die Analyse des Museumsführers wird die revolutionäre Entwicklung des Reverse Engineering, siehe Kapitel 4.3.5.1 Seite 55, genommen, da die Wartung des Altsystems gestoppt und die Produktlinie neu entwickelt wird. Die Altsoftware wird nach wiederverwendbaren Komponenten untersucht, die anschließend in die Produktlinie integriert werden. Dazu wird, die in Kapitel 4.3.5.1.2 auf Seite 57 vorgestellte OAR-Methode verwendet, weil sie einfach zu verstehen und umzusetzen ist.

Die OAR-Methode gliedert sich in 5 Aktivitäten. In der ersten Aktivität, Überblick verschaffen, sollen alle teilnehmenden Interessengruppen in den Prozess und in die Entscheidungsfindung mit einbezogen werden. Im Beispiel des Museumsführers ist es das Ziel Komponenten zu entdecken und zu schauen, ob sie in die Software-Produktlinie des Fallbeispielunternehmens „GoGame“ oder in eine neue Software-Produktlinie implementiert werden können.

In der zweiten Aktivität werden die Anforderungen an die Komponenten der Altsoftware mit den Anforderungen der Software-Produktlinie verglichen. Dazu müssen die Komponenten der Altsoftware identifiziert werden. Dabei werden der Quellcode und die Software näher betrachtet. Im Beispiel des Museumsführers gibt es nach Sichtung des Quellcodes folgende Komponenten, die die Anforderungen für eine Software-Produktlinie erfüllen:

- die Bluetoothkomponente: Diese Komponente dient zur Kommunikation mithilfe von Bluetooth zu einem Server.
- die Loginkomponente: Die Loginkomponente umfasst das Anmelden am System und die Implementierung eines einfachen Bezahlmodells.
- die Speicherkomponente: Die Speicherkomponente dient zum Speichern von relevanten Daten
- die Multimediakomponente: Mithilfe dieser Komponente können Bilder oder Videos dargestellt werden. Zusätzlich können Sprachdateien abgespielt werden.

Die nächste Aktivität ist die Analyse der Kandidaten. Dabei werden diese Kandidaten daraufhin untersucht, wie sie in die Software-Produktlinie eingearbeitet werden können. Im Fallbeispiel des Museumsführers sind keine Black-Box Komponenten enthalten, sondern nur White-Box Komponenten. Da die einzelnen Komponenten stark miteinander in verschiedenen Klassen verzahnt sind, müssen die Komponenten umgeschrieben werden, damit sie genutzt werden können. Dabei muss abgeschätzt werden, ob sich die

Kosten, die Risiken und der Aufwand der Bearbeitung der Kandidaten lohnen oder die Kosten für eine Neuentwicklung niedriger sind. Die Hauptklasse des Museumsführers besitzt mehr als 3000 Codezeilen und ist daher sehr schwer nachvollziehbar. Auch weitere Klassen sind durch ihre Größe nicht gut lesbar. Dennoch können durch Umschreiben des Codes und Aufteilen in neue Klassen die Komponenten relativ schnell gekapselt, der Code somit stark verkleinert und die Lesbarkeit erhöht werden.

Im vierten Schritt, der Planung der Alternativen, werden mögliche Kombinationen von Alternativen von Kandidaten erarbeitet. Exemplarisch werden im Folgenden mögliche Sub-Komponenten für die Multimediakomponente aufgezeigt:

- Anzeigen von einfachen Texten
- Anzeigen von Bildern
- Abspielen von Videos
- Abspielen von Sprachdateien

Abschließend werden die Entscheidungskriterien für die Wahl der Kandidatenkombination festgelegt und schließlich eine Kombination ausgewählt. In diesem Schritt wird die Zeitplanung der Software-Produktlinienentwicklung aktualisiert. Im Beispiel des Museumsführers wird ein Zeitplan festgelegt, wann welche Komponente umgesetzt wird.

Nach Analyse des Quellcodes und Identifizierung der Komponenten wurden einige neue Probleme entdeckt. Der Museumsführer konnte in der Ursprungsform nicht ausgeführt werden, da er offenbar mit der Entwicklungsumgebung „Netbeans“ geschrieben wurde. Es gibt im Code importierte Klassen aus der Netbeansbibliothek. Damit wurde der Einstieg stark erschwert, da der Autor der Diplomarbeit mit der Eclipse-Entwicklungsumgebung arbeitet und die importierten Klassen nicht besaß. Nach dem Schreiben eigener Klassen, um dieses Problem zu beheben, konnte das Programm erstmals emuliert werden. Nach Rücksprache mit dem Autor des Museumsführers wurden auch die Anforderungen klar. Es sollten nur Mobilfunktelefone unterstützt werden, die mindestens eine CLDC 1.1 und MIDP 2.0 oder aufwärts besitzen. Außerdem wird das Bluetooth-Package benötigt. Diese Voraussetzungen sollten neue Mobilfunktelefone seit 2004 besitzen. Der neue angepasste Museumsführer liegt auf der beiliegenden CD dabei. Er kann nur in Verbindung mit einem Bluetooth-Server genutzt werden.

Eine Anpassung des Museumsführers in eine Software-Produktlinie ist möglich, aber es müssen weitere Anwendungen geplant werden, damit sich der erhöhte Arbeitsaufwand zur Identifizierung der Komponenten rentiert.

7 Fazit

Die Motivation der Diplomarbeit war die Schaffung eines Software-Produktlinienmodells für mobile Endgeräte. Dieses wurde unter Zuhilfenahme vorhandener und in der Praxis etablierter Modelle erledigt und durch die Fallstudie „GoGame“ veranschaulicht. Es wurde gezeigt, dass eine praktische Anwendung des Modells möglich ist. Ein weiteres Ziel der Diplomarbeit war es beim Leser Interesse an Software-Produktlinien zu wecken. Nach subjektiver Meinung des Autors der Diplomarbeit wurde das Ziel erreicht, da nicht nur die Theorie betrachtet wurde, sondern durch das Fallbeispiel ein praktisches Beispiel geliefert wurde, wie eine Software-Produktlinie mit dem entwickelten Modell aussehen könnte. Des Weiteren sind die Ergebnisse des Fallbeispiels ein Hinweis darauf, dass das Vorhaben erfolgreich war. Die entwickelten Spiele laufen auf allen Mobilfunktelefonen, welche über die geforderten Vorrausetzungen, mindestens CLDC 1.1 und MIDP 2.0, verfügen. Auch die kurze Entwicklungszeit von weniger als einer Woche pro Spiel ist ein Hinweis, dass die identifizierten Domänen mit ihren Features der entwickelten Software-Produktlinie sehr gut wieder verwendbar waren und somit Zeit und Kosten bei der Entwicklung gesenkt werden konnten. Dabei ist noch zu erwähnen, dass bei der Entwicklungszeit von einer Woche ein Großteil in das Leveldesign und die Grafiken gefallen ist. Somit muss davon ausgegangen werden, dass die reine Programmierzeit noch wesentlich kürzer war und bei neuen Spielen dieser Art nicht mehr als 3 Tage in Anspruch nehmen wird.

Doch natürlich hat auch dieses Modell seine Grenzen. Im Fallbeispiel wurde nicht näher auf die Auflösungen der Mobilfunkgeräte eingegangen, sondern die Anwendung hat sich selber angepasst, indem es die Objekte entweder kleiner gestaltet bei kleineren Auflösungen und/oder die Kamera zentriert, so dass der Spieler alles Wichtige im Blick hat. Bei Anwendungen, bei denen die Objekte nicht skaliert werden sollten, muss darauf speziell geachtet werden und viele Versionen für die verschiedenen Mobilfunktelefone entwickelt und veröffentlicht werden.

Auch wurde das Modell nur in einem sehr kleinen Rahmen benutzt. Das Fallbeispielunternehmen „GoGame“ besteht nur aus ein paar wenigen Mitarbeitern. Wie das Modell in einem großen Unternehmen umgesetzt werden kann, konnte nicht validiert werden.

Es wurden auch nicht spezielle Anpassungen für die Smartphones vorgenommen. Zum Beispiel basiert Android zwar auf Java, aber ist ein eigener Dialekt mit neuen Sprachkonstrukten und Befehlen. Deshalb müsste der Code hierfür speziell angepasst werden. Doch das lässt sich auch im entwickelten Modell verwirklichen. Dadurch wird die Software-Produktlinie noch mächtiger und es bedarf einer guten Organisation, dass alles weiterhin gut verläuft. Auch die Steuerung im Fallbeispiel orientierte sich an normalen

Mobilfunktelefonen. Die Steuerung der neuen Smartphones funktioniert sehr häufig mithilfe der TouchScreen-Tastatur. Auch diese Anpassung wurde in dieser Diplomarbeit nicht betrachtet.

Der Autor der Diplomarbeit wird das Modell in der Zukunft weiter entwickeln und versuchen, durch Verbesserungen und Anpassungen des Fallbeispielunternehmens, die Spiele Android- und IPhone-tauglich zu gestalten und sie auf dem jeweiligen Markt dafür zu verkaufen. Dazu soll als Nächstes untersucht werden, ob eine Featureimplementierung den Entwicklungsprozess noch vereinfachen und verkürzen kann. Außerdem soll die Steuerung angepasst werden und extra für die TouchScreen-Tastatur neue Spiele entwickelt werden, die für die neue Steuerungsform optimiert sind. Die Anpassungen an das Modell und die entwickelte Software-Produktlinie sollen weiterhin schriftlich erfasst werden, um die Forschungen in diesem Bereich weiter voranzutreiben.

Anhang

Literaturverzeichnis

Alves, V.; Gheyi, R.; Massoni, T.; Kulesza, U.; Borba, P.; Lucena, C. (2006): Generative Programming And Component Engineering. In Proceedings of the 5th international conference on Generative programming and component engineering, Portland, Oregon, S. 201-210.

Apel, S.; Kästner, C.; Lengauer, C. (2009): Vergleich und Integration von Komposition und Annotation zur Implementierung von Produktlinien. In Software Engineerung 2009 – Fachtagung des GI-Fachbereichs Softwaretechnik (Kaiserslautern, Germany), volume P-143 of Lectures Notes in Informatics, S. 101-112. Gesellschaft für Informatik(GI).

Apel, S.; Kästner, C.; Saake, G. (2008): Erweiterte Programmierkonzepte für maßgeschneiderte Datenhaltung – Teil 4 Präprozessoren, Komponenten, Frameworks. http://wwwiti.cs.uni-magde-burg.de/iti_db/lehre/epmd/2008/slides/04_FrameworksPraeprozessor.pdf. Stand: 05.10.2009.

Atkinson, C.; Bayer, J.; Bunse, C.; Kamsties, E.; Laitenberger, O.; Laqua, R.; Muthig, D.; Paech, B.; Wüst, J.; Zettel J. (2001): Component-Based Product Line Engineering with UML. Addison-Wesley.

Aumüller, H. (2007): Handynutzung in der Bevölkerung: Marktabdeckung und Zielgruppen-Potentiale. Mifm München – Institut für Marktforschung GmbH. http://www.mifm.de/downloads/studien/mifm_handynutzung_2007.pdf. Stand: 05.10.2009.

Bass, L.; Clements, P.; Kazman R. (1998): Software Architecture in Practice. Addison-Wesley Professional.

Batory, D.; Sarvela J.N.; Rauschmayer, A. (2004): Scaling Step-Wise Refinement. In Internation Conference on Software Engineering, Proceedings of the 25th International Conference on Software Engineering, S.187-197, Portland, Oregon.

Becker, A.; Pant, M. (2009): Android – Grundlagen und Programmierung, dpunkt. Verlag GmbH, Heidelberg.

Bergey, J.; O'Brien L.; Smith, D. (2000): Mining Existing Software Assets for Software Product Lines, Technical Report.

Bergey, J.; O'Brien L.; Smith D. (2001): Options Analysis for Reengineering (OAR): A Method for Mining Legacy Assets. Technical Report, Pittsburgh.

Biermann, S. (2001): Seminar Produktlinien – Einführung und Überblick. Seminararbeit, Institut für Softwaretechnologie, Universität Stuttgart.

BITKOM (2006): Mehr Handys als Einwohner in Deutschland. http://www.bitkom.org/46624_40990.aspx. Stand: 05.10.2009.

Böckle, G.; Knauber, P.; Pohl, K.; Schmid, K. (2004): Software-Produktlinien - Methoden, Einführung und Praxis, dpunkt. Verlag GmbH, Heidelberg.

Bory, M.; Hartkopf, S.; Kohler, K.; Rombach, D. (2001): Combining Software and Application Competencies, IEEE Software. S.93-95.

Byrne, G. (2009): Android OS smartphone sales to surpass OS X iPhone by 2012. Analyst, Informa Telecoms & Media.

Chikofsky, E. J.; Cross, J. H. (1990): Reverse Engineering and Design Recovery: A Taxonomy. IEEE Software, S.13-17.

Clements, P.; Northrop, L. (2001): Software Product Lines: Practices and Patterns, Addison Wesley Pub Co. 1st edition.

Dalgarno, M. (2008): Mobile Games and Software Product Lines. http://www.software-acumen.com/industries/mobile/mobile-games/. Stand: 05.10.2009.

Dittman, J. (2009): Sichere Systeme (Vorlesungsfolien Sommersemester 09). http://omen.cs.uni-magdeburg.de/itiamsl/cms/upload/lehre/sommer09/vorles-all-sisys-3-sw.pdf. Stand: 05.10.2009.

Dumke, R. (2003): Software Engineering. Vieweg Verlag, Braunschweig.

Eisenbarth, T.; Koschke, R.; Simon, D. (2003): Herleitung der Feature-Komponenten-Korrenpondenz mittels Begriffsanalyse. Proceedings of the 1. Deutscher Software.Produktlinien Workshop. Kaiserslautern.

Eisenbarth, T.; Simon, D. (2001): Guiding Feature Asset Mining for Software Product Line Development. Proceedings of the International Workshop on Product Line Engineering: The Early Steps: Planing, Modeling and Managing. Erfurt. Germany. Fraunhofer IESE.

Figueiredo, E., Cacho, N.; Sant'Anna, C.; Monteiro, M.; Kulesza, U.; Garcia, A.; Soares, S.; Ferrari, F.; Khan, S.; Filho, F.; Dantas, F. (2008): Evolving Software Product Lines with Aspects: An Empirical Study on Design Stability. In Proc. of International Cnference on Software Engineering (ICSE).

Flyvbjerg, B. (2006): Five Misunderstandings About Case-Study Research, Aalborg University, Denmark. In Qualitative Inquiry, No. 2, S.219-245.

Foeller-Nord, M. (2002): Mobile Drahtlose Informationsverarbeitung – Grundlagen. Institut Embedded And Mobile Computing, Hochschule Mannheim.

Frei, M.; Wittwer R.; Studer B. (2004): J2ME – Tutorial. Hochschule für Technik und Wirtschaft.

Gacek, C.; Knauber, P.; Schmid, K.; Clements, P.(2001): Successful Software Product Line Development in a Small Organization. A Case Study. Technical Report. Fraunhofer Institut for Experimental Software Engineering (IESE).

Gerlicher, A.; Rupp S. (2004): Symbian OS – Eine Einführung in die Anwendungsentwicklung, dpunkt. Verlag GmbH, Heidelberg.

Gernert, C. (2003): Agiles Projektmanagement – Risikogesteuerte Softwareentwicklung. Carl Hanser Verlag München Wien.

Goldhammer, Dr. K.; Wiegand, Dr. A.; Becker, D.; Schmid, M. (2008): Goldmedia Mobile Life Report 2012: Mobile Life in the 21st century – Status quo and outlook. Goldmedia GmbH, Media Consulting & Research. Berlin.

Grosse, M. (2004): Requirements Engineering. Seminararbeit, Universität Stuttgart. Stuttgart.

Harsu, M. (2002): FAST Product-line Architecture Process. Report 29. Institute of Software Systems. Tempere University of Technology. http://practise2.cs.tut.fi/pub/papers/fast.pdf. Stand: 05.10.2009.

Heie, A. (2002): Global Software Product Lines and Infinite Deversity. The Second Software Product Line Conference. San Diego. California.

Herrenbrück, S. (2008): Medienboard News 2.08. Medienboard Berlin-Brandenburg GmbH. Potsdam-Babelsberg.

Holden, W. (2008): Mobile Games – Subscription & Download, 2008-2013, Fifth Edition. Juniper Research.

Horn, T. (2009): Technische Kurzdokumentationen. http://www.torsten-horn.de/techdocs/index.htm . Stand: 05.10.2009.

JCP (2009): Java Community Process. http://www.jcp.org/en/home/index. Stand: 05.10.2009.

Josten, S. (2001): Komponentenbasierte Software-Entwicklung: Adaption der KobrA-Methode für eingebettete Systeme. Diplomarbeit, IESE Siegelbach.

Kang, K.C.; Cohen, G.; Hess, J.A.; Novak, W.E.; Spencer Peterson, A. (1990): Feature-Oriented Domain Analysis (FODA). Technical Report. SEI Pittsburgh.

Kang, K. C.; Kim, S.; Lee, J.; Kim, K.; Shin, E.; Huh, M. (1998): FORM: A Feature-Oriented Reuse Method with Domain-Specific Reference Architectures. Annals of Software Engineering.

Kang, K.C.; Lee, J.; Donohoe, P. (2002): Feature-Oriented Product Line Engineering. IEEE Software, S.58-65.

Kästner, C.; Apel, S.; Saake, G. (2008): Erweiterte Programmierkonzepte für maßgeschneiderte Datenhaltung. http://wwwiti.cs.uni-magdeburg.de/iti_db/lehre/epmd/2008/slides/. Stand: 05.10.2009.

Kiczales, G.; Lamping, J.; Mendhekar, A.; Maedam C.; Lopes, C.V.; Loingtier J.-M., Irwin, J. (1997): Aspect-Oriented Programming. In Proc. Europ. Conf. On Object-Oriented Programming (ECOOP).

Kiczales, G.; Hilsdale, E.; Hugunin, J.; Kersten, M.; Palm, J.; Griswold, W.G. (2001): An Overview of AspectJ. In Proc. Europ. Conf. On Object-Oriented Programming (ECOOP).

Klaß, C; Ihlenfeld, J. (2008): EAST-ADL2 - Programmiersprache für das Auto. Klaß & Ihlenfeld Verlag GmbH. http://www.golem.de/0803/58115.html Stand: 05.10.2009.

Krueger, C. (2001): Easing the Transition to Software Mass Customization, Proceedings of the 4th International Workshop on Software Product Family Engineering. Bilbao, Spain. New York, NY: Springer.

Laddad, R. (2003): AspectJ in Action – Practical Aspect-Oriented Programming. Manning Publications.

Laib, L. (2007): Android - Das freie Smartphone-Betriebsystem. http://www.android-os.de/. Stand: 05.10.2009

Leichner, M. (2005): PuLSE Anwendungsbeispiel GoPhone. Seminararbeit, Entwicklung verteilter eingebetteter Systeme, TU Berlin.

Leutloff, C. (2000): Kurzanleitung zur Versionsverwaltung mit CVS. http://www.oche.de/~leutloff/cvs/t1.html. Stand: 05.10.2009.

Lucka, T. (2008): Mobile Games – Spieleprogrammierung für Handys mit Java ME. Carl Hanser Verlag München. München.

Melzer, R. (2008): Symbian Foundation – S60 und UIQ werden vereinheitlicht. http://www.areamobile.de/news/9403-symbian-foundation-s60-und-uiq-werden-vereinheitlicht. Stand: 20.09.2009.

Munz, S.; Soergel, J. (2007): Agile Produktentwicklung im Web 2.0. Boizenburg.

Muthig, D.; John, I.; Anastasopoulos, M.; Forster, M.; Dörr, J.; Schmid, K. (2004): Go-Phone – A Software Product Line in the Mobile Phone Domain. IESE-Report. Version 1.0.

O'Brien, L.; Smith, D. (2002): MAP and OAR Methods: Techniques for Developing Core Assets for Software Product Lines from Existing Assets, Technical Note.

O'Brien, L.; Stoermer, C. (2001): MAP: Mining architectures for Product Line Evaluations. Proceedings of the Third Working IFIP Conference on Software Architecture, S.35-44, Amsterdam, Netherlands.

Oestereich, B.; Weiss C. (2008): APM – Agiles Projektmanagement, dpunkt.Verlag GmbH, Heidelberg.

Penzenstadler, B. (2006): Evaluierung und Erprobung von Ansätzen beim Modellbasierten Design in Software-Produktlinien. Universität Passau.

Pichler, R. (2008): Scrum – Agiles Projektmanagement erfolgreich einsetzen, dpunkt.Verlag GmbH, Heidelberg.

Rautenstrauch, C.; Schulze, T. (2003): Informatik für Wirtschaftswissenschaftler und Wirtschaftsinformatiker. Springer Berlin.

Rieger, B. (2007): Einführung in die Wirtschaftsinformatik – Anwendungsentwicklung (Vorlesungsfolien Wintersemester 07/08). http://sansibar.oec.uni-osnabrueck.de/uwdwi2/EiEDV/VLFolien/IV_VL_Rieger_08.pdf. Stand: 05.10.2009.

Rietdorf, U.; Egin, C. (2005): Produktlinienentwicklung – Scoping, Domain Analysis & Application Analysis. Seminararbeit, Hochschule Mannheim.

Rombach, D. (2003): PuLSE™ - Produktlinen für Software-Systeme. Fraunhofer-Institut für Experimentelles Software Engineering IESE. Kaiserlautern. http://www.iese.fraunhofer.de/Products_Services/pulse/pulse_d.pdf. Stand: 05.10.2009.

Royce, W. (1970): Managing the development of large software systems. In Proceedings, IEEE WESCONS. The Instutue of Electrial and Electronics Engineers

Schmatz, K.-D. (2004): J2ME - Entwicklung mobiler Anwendungen mit CLDC und MIDP, dpunkt. Verlag GmbH, Heidelberg.

Schmatz, K.-D. (2007): Java Micro Edition – Entwicklung mobiler JavaME-Anwendungen mit CLDC und MIDP, dpunkt. Verlag, Heidelberg.

SEI (2009): A Framework for Software Product Line Practice. http://www.sei.cmu.edu/productlines/frame_report/PL.essential.act.htm. Stand: 05.10.2009.

SEI II (2009): Software Product Lines: MAP - Mining Architectures for Product Line Evaluation. http://www.sei.cmu.edu/productlines/products_services/map.html. Stand: 05.10.2009.

Simon, D.; Eisenbarth T. (2002): Evolutionary Introduction of Software Product Lines. Proceedings of the 2nd Software Product Line Conference. San Diego. CA. S.272-282.

Sola, A. (2007): Building Mobile Games Using Java ME. http://ls5-www.cs.tu-dortmund.de/export/sites/ls5/download/javaME_happyhour.pdf. Stand: 05.10.2009.

Stiehler, J. (2002): Versions-Verwaltung mit CVS. http://www.cis.uni-muenchen.de/sysinfo/cvs/vortrag/. Stand: 05.10.2009.

SUN Microsystems Inc. (2009): Sun Welcomes SavaJe Technologies. http://www.sun.com/software/savaje/index.xml. Stand: 05.10.2009.

Toft, P.; Coleman, D.; Ohta, J. (2000): A Cooperative Model for Cross-Divisional Product Development for a Software Product Line. In Patrick Donohoe (ed.): Proceedings SPLC1. Kluwer Academic Publishers. Denver, Colorado, S.111-132.

Tuckman, Bruce W. (1965): Developmental Sequence in Small Groups. Psychological Bulletin, Vol. 63.

Van der Linden, F. (2002): Software Product Families in Europe: The Esaps & Café Projects. IEEE Software, Los Alamitos, CA, S.41-49.

Weber, J. (2004): Reverse Engineering Techniken. Seminararbeit, Institut für Softwaretechnologie, Universität Stuttgart.

Weiderman, N.; Bergey, J.; Smith, D.; Tilley, S. (1998): Can Legacy Systems Beget Product Lines?. In Lecture Notes in Computer Science, Vol. 1429, London, UK, S.123-131.

Weiss, D. M.; Lai, C. T. R. (1999): Software Product Line Engineering: A Family Based Software Engineering Process. Addison-Wesley.

Weiß, P.; Zaher, Z.; Pothmann, C.; Amare, S. (2004): Software-Produktlinien. Seminararbeit, Computergestützte Informationssysteme, Technische Universität Berlin.

Wilde, N.; Scully, M. C. (1995): Software Reconnaissance: Mapping Program Features to Code. Journal of Software Maintenance: Research and Practice.

Abschließende Erklärung

Ich versichere hiermit, dass ich die vorliegende Diplomarbeit selbständig, ohne unzulässige Hilfe Dritter und ohne Benutzung anderer als der angegebenen Hilfsmittel angefertigt habe. Die aus fremden Quellen direkt oder indirekt übernommenen Gedanken sind als solche kenntlich gemacht.

Magdeburg, den 07. Oktober 2009