Hildesheimer Informatikberichte

herausgegeben von:

Andreas Birk, SWPM
Ludger Fiege, Siemens AG
Klaus Schmid, Universität Hildesheim
Ramin Tavakoli Kolagari, Volvo AB

Tagungsband der PIK 2009
Produktlinien im Kontext

25. Mai 2009 in Berlin

Informatikbericht 1/2009
SSE/02/09-D

Software Systems Engineering • Institut für Informatik
Universität Hildesheim • Marienburger Platz 22 • D-31134 Hildesheim

Kurzfassung

Produktlinien sind heute in vielen Bereichen der Software-Industrie vertreten, von eingebetteten Systemen bis zu betrieblichen Informationssystemen. Sie ermöglichen höhere Produktivität, steigern die Qualität und verbessern die strategische Position der Unternehmen. Zugleich ist die Produktlinienentwicklung noch eine relativ junge Technologie, die für viele Unternehmen noch bedeutende Herausforderungen und Risiken birgt.

Die Tagung PIK 2009 beleuchtete aktuelle Erfahrungen mit der Software-Produktlinien-Entwicklung und förderte den Dialog zwischen Praxis und anwendungsorientierter Forschung. Im Mittelpunkt stand das Wechselspiel zwischen technischen Fragestellungen und den geschäftlichen, organisatorischen und Prozessaspekten. Darüber hinaus wurden auch neue technologische Entwicklungen vorgestellt und diskutiert.

Inhaltsverzeichnis

Kurzfassung **2**

Inhaltsverzeichnis **3**

Abbildungsverzeichnis **5**

1 PIK2009: Produktlinien im Kontext—Überblick **9**

1.1 Schwerpunktthemen 9

1.2 Zusammenfassende Diskussion 10

2 Mini-Tutorial: 15 Jahre Software-Produktlinien—Einführung und aktueller Stand **12**

2.1 Überblick 12

2.2 Gliederung und Schwerpunktthemen 12

2.3 Was ist eine Software-Produktlinien (SPL)? 12

2.4 Historische Entwicklung des SPL-Gebietes 13

2.5 Konzepte und Verfahren der Produktlinienentwicklung 13

2.6 SPL-Fallstudien 14

2.7 Kernfragen aus SPL-Forschung und -Praxis 14

3 Lösungsorientierte Software Produktlinienentwicklung in heterogenen Systemlandschaften **15**

3.1 Zusammenfassung 15

3.2 Einleitung 15

3.3 Lösungsorientierte Entwicklung kundenindividueller Lösungen 16

3.3.1 Lösungsarchitektur und heterogene Systemlandschaften 16

3.3.2 arvato Entwicklungsprozess 18

3.4 Angepasster SPL Entwicklungsprozess 19

3.4.1 Rahmenbedingungen für arvato SPL Entwicklung 19

3.4.2 Angepasster SPL Entwicklungsprozesses 20

3.5 Anforderungen an die SPL Entwicklung 21

3.6 Ergebnis und weitere Arbeit 22

3.7 Literaturverzeichnis 23

4 Requirements-Engineering für Produktlinien in der Automobilelektronik **24**

4.1 Multi-Level Konzept 24

4.2 Configuration Links 24

4.3 Literaturverzeichnis 25

5 A model-driven product line approach for HMI development **26**

5.1 Abstract 26

5.2 HMI development process 26
5.3 The model-driven approach 27
5.4 Product line support 29
5.5 Conclusion 33
5.6 References 33

6 UML und Softwareproduktlinien – Wie, Weshalb und Wann? 34
6.1 Produktlinien im Kontext 34
6.2 Wie sehen Integrationen von UML und SPL aus? 35
6.3 Wann kann UML in Kombination mit SPL eingesetzt werden? 37
6.4 Fazit 37
6.5 Literaturverzeichnis 37

7 Merkmalmodellierung im Industrieeinsatz—ein Erfahrungsbericht 38
7.1 Zusammenfassung 38
7.2 Einleitung 38
7.3 Ausgangssituation 38
7.4 Wünsche 39
7.5 Werkzeugauswahl 40
7.6 Auswahlverfahren 40
7.7 Datenmodellierung 41
7.8 Zwei Jahre Pause 41
7.9 Neuer Anlauf 41
7.10 Versuch einer Umsetzung 42
7.11 Merkmalmodellierung 42
7.12 Offene Fragen 43
7.13 Literaturverzeichnis 43

Anhang A: Vortragsfolien 44
A.1 15 Jahre Software-Produktlinien—Einführung und aktueller Stand 46
A.2 Lösungsorientierte Software Produktlinienentwicklung in heterogenen Systemlandschaften 66
A.3 Requirements-Engineering für Produktlinien in der Automobilelektronik 76
A.4 A model-driven product line approach for HMI development 92
A.5 UML und Softwareproduktlinien – Wie, Weshalb und Wann? 101

Abbildungsverzeichnis

Abbildung 1. Lösungsarchitektur: Zusammenhang Service Systeme und Lösungen. 17
Abbildung 2. arvato Lösungs-Entwicklungsprozess. 18
Abbildung 3. SPL Architektur zur Lösungsentwicklung. 21
Abbildung 4. Abhängigkeiten zwischen Lösungs- und Service-Plattform(en). 22
Abbildung 5. Screen flow model. 28
Abbildung 6. Abstract screen model. 28
Abbildung 7. Simulation of screen flow model. 29
Abbildung 8. Variant model derivation. 30
Abbildung 9. Screen flow family model with variability point. 31
Abbildung 10. Mapping features to screen flows in family models. 31
Abbildung 11. Variant feature model derivation. 32
Abbildung 12. Variant screen flow model. 32
Abbildung 13. SysML Modell mit Variationspunkt. 35
Abbildung 14. Zustandsdiagramm mit Bedingungen für Features. 36
Abbildung 15. Zustandsdiagramm einer Variante. 36

Organisatoren

Andreas Birk

Software.Process.Management
Stuttgart, Deutschland

Ludger Fiege

Siemens AG
Corporate Technology
Architecture
München, Deutschland

Klaus Schmid

Universität Hildesheim
Institut für Informatik
Hildesheim, Deutschland

Ramin Tavakoli Kolagari

Volvo Technology Corporation
Mechatronics & Software
Göteborg, Schweden

Autoren

Jon Ashley

DENSO Sales UK Ltd
Coventry, UK

Danilo Beuche

pure-systems GmbH
Magdeburg, Deutschland

Andreas Birk

Software.Process.Management
Stuttgart, Deutschland

Marc Born

ikv++ technologies ag
Berlin, Deutschland

Omar Ekine

ikv++ technologies ag
Berlin, Deutschland

Ludger Fiege

Siemens AG
Corporate Technology
Architecture
München, Deutschland

Dirk Janzen

metadoc GmbH
Birkenfeld, Deutschland

Mark-Oliver Reiser

Technische Universität Berlin
Gruppe Softwaretechnik
Berlin, Deutschland

Klaus Schmid

Universität Hildesheim
Institut für Informatik
Hildesheim, Deutschland

Enrico Schnepel

b+m Informatik AG
Berlin, Deutschland

Ramin Tavakoli Kolagari

Volvo Technology Corporation
Mechatronics & Software
Göteborg, Schweden

Thomas von der Maßen

arvato services
Bertelsmann
Gütersloh, Deutschland

Matthias Weber

Carmeq GmbH
Berlin, Deutschland

Andreas Wübbeke

Software Quality Lab
Universität Paderborn
Paderborn, Deutschland

Programmkomitee

Danilo Beuche
pure-systems

Samuel Fricker
Fuchs Informatik und Universität Zürich

Eva Geisberger
TU München

Birgit Geppert
Avaya

Lars Geyer-Blaumeiser
Robert Bosch

Paul Grünbacher
Universität Linz

Andreas Helferich
Universität Stuttgart

Gerald Heller
SE Consulting

Frank Houdek
Daimler

Dirk Janzen
metadoc GmbH

Olaf Kath
ikv++

Michael Kircher
Siemens

Peter Knauber
Hochschule Mannheim

Andreas Kreß
HOOD-Group

Klaus Marquardt
Dräger Medical

Thomas von der Maßen
Arvato/Bertelsmann

Dirk Muthig
Fraunhofer IESE

Klaus Pohl
Universität Duisburg-Essen

Mark-Oliver Reiser
TU Berlin

Reinhard Stoiber
Universität Zürich

Ramin Tavakoli Kolagari
Volvo Technology Corporation

Axel Uhl
SAP

Martin Verlage
Vereinigte Wirtschaftsdienste

Markus Völter
Völter Ingenieurbüro

Matthias Weber
Carmeq

1 PIK2009: Produktlinien im Kontext—Überblick

Andreas Birk, Ludger Fiege, Klaus Schmid, Ramin Tavakoli Kolagari

Produktlinien sind heute in vielen Bereichen der Software-Industrie vertreten, von eingebetteten Systemen bis zu betrieblichen Informationssystemen. Erfahrungen belegen das Potenzial von Produktlinien: Sie ermöglichen höhere Produktivität, steigern die Qualität und verbessern die strategischen Positionen der Unternehmen.

Dennoch bergen Produktlinien für viele Unternehmen noch bedeutende Herausforderungen und Risiken. Auch etablierte Produktlinien sehen sich im Laufe ihres Lebenszyklus immer wieder neuen Herausforderungen gegenüber. Die Gründe liegen teilweise im technischen Bereich. So sind viele Technologien für den breiten Einsatz in der Praxis noch nicht genügend ausgereift und miteinander integriert. Die wohl größten Herausforderungen stellen sich an der Schnittstelle zwischen den technischen Verfahren und den Prozessen sowie dem organisatorischen und geschäftlichen Kontext der Produktlinienentwicklung. Wie müssen die technologischen Ansätze an den Kontext angepasst werden? Welche Potenziale bieten neue technologische Entwicklungen in verschiedenen Einsatzgebieten? Wie ist insgesamt mit den Prozess- und organisatorischen Herausforderungen in der Produktlinienentwicklung umzugehen (bspw. bei einer Methodeneinführung)?

Die Tagung „Produktlinien im Kontext“ beleuchtete aktuelle Erfahrungen mit Produktlinien und förderte den Dialog zwischen Praxis und anwendungsorientierter Forschung. Im Mittelpunkt stand das Wechselspiel zwischen technischen Fragestellungen und den geschäftlichen, organisatorischen und Prozessaspekten. Daneben wurden auch neue technologische Entwicklungen vorgestellt und diskutiert.

1.1 Schwerpunktthemen

Im Mittelpunkt der Tagung standen die Themen:

- Produktlinien für Individuallösungen und Massenprodukte
- Variantenmanagement in der modellbasierten Entwicklung
- Einführung und Etablierung von Produktlinien

Eingerahmt wurden diese Themenblöcke durch ein einführendes Mini-Tutorial in Software-Produktlinien und eine abschließende Diskussion.

Das einführende Mini-Tutorial „15 Jahre Software-Produktlinien: Einführung und aktueller Stand“ (Andreas Birk und Klaus Schmid) gab einen Überblick über die wesentlichen Konzepte, den Stand der Praxis und die aktuellen Themenschwerpunkte der Produktlinien-Entwicklung.

Dem Themenblock „Produktlinien für Individuallösungen und Massenprodukte“ widmeten sich zwei Erfahrungsberichte. Der Vortrag von Matthias Weber und Mark-Oliver Reiser beleuchtete die Entwicklung von Massenprodukten, speziell das Requirements-Engineering für Produktlinien in der Automobilindustrie.

Der Beitrag von Thomas von der Maßen und Andreas Wübbeke stellte einen Ansatz für die lösungsorientierte Software-Produktlinienentwicklung in heterogenen Systemlandschaften vor. Da die Autoren kurzfristig verhindert waren, präsentierten die Tagungsorganisatoren die zentralen Inhalte des Beitrags im Überblick. Im Anschluss entwickelte sich unter den Tagungsteilnehmern eine rege Diskussion, die auch in der abschließenden Diskussion am Ende der Tagung wieder aufgenommen wurde.

Die Vorträge zum Themenblock „Variantenmanagement in der modellbasierten Entwicklung“ gaben einen Einblick in den aktuellen Stand der modellbasierten Produktlinienentwicklung und zeigten die verschiedenen Zusammenhänge zwischen Varianten, UML und modellbasierter Entwicklung auf.

Modellbasierte Verfahren sind ein wichtiger Ansatz zur Entwicklung von Produktlinien. Mit der Anzahl der Varianten steigt aber auch schnell die Komplexität der Modelle. Aktuelle Fragen, die sowohl aus Sicht der Forschung als auch aus Sicht der Praxis relevant sind: Wie stehen die verschiedenen modellbasierten Verfahren zueinander? Welche konkreten Vorteile bieten sie im Verlauf von Software-Projekten? Wie kann man die besonders hohe Komplexität der modellbasierten Entwicklung von Produktlinien im Griff behalten und erfolgreich gestalten?

Marc Born, Omar Ekine und Jon Ashley stellten eine Systemumgebung vor für die modellbasierte Entwicklung und Simulation interaktiver grafischer Benutzeroberflächen. In einer anschließenden Vorführung der Software wurden die Vorteile und das Potenzial der modellbasierten Produktlinienentwicklung deutlich. Unter anderem beschleunigt das Verfahren die Entwicklung und bietet durch frühe Simulation der zu erstellenden Software-Funktionen neue Ansätze zur Qualitätssicherung.

Den Einsatz von UML und modellgetriebener Entwicklung im Rahmen von Software-Produktlinien analysierten Danilo Beuche und Enrico Schnepel. Sie stellten die drei Methoden einander gegenüber und diskutierten Potenzial und Schwierigkeiten verschiedener Methodenkombinationen. Die vorgestellten Kriterien geben Software-Entwicklern Orientierung beim Einsatz von UML und domänenspezifischen Sprachen in der Produktlinienentwicklung.

Der dritte und abschließende Themenblock widmete sich den Erfahrungen bei der Einführung von Software-Produktlinien. Dirk Janzen schilderte einführend anhand eines Beispielfalls, dass es sich durchaus schwierig gestalten kann, bis sich in einer Organisation das Bewusstsein über Produktlinienentwicklung und ihre besonderen Herausforderungen verbreitet. Wenn dieses Bewusstsein einmal vorhanden ist, kann es jedoch durchaus gelingen, neue Methoden wie die Merkmalsmodellierung von Produktvarianten in sehr kurzer Zeit einzuführen und so eine wichtige Grundlage für den Erfolg der Produktlinienentwicklung zu legen.

1.2 Zusammenfassende Diskussion

Rund um die Frage der Einführung von Produktlinien entwickelte sich eine rege Diskussion im Plenum, bei der auch insbesondere Aspekte von Produktlinien für die Entwicklung von Individuallösungen angesprochen wurden. Ein weiterer Diskussionsschwerpunkt war der schrittweise Übergang von einer projektorientierten Entwicklung hin zur Produktlinienentwicklung.

Als wichtige Vorkehrungen für die erfolgreiche Einführung von Produktlinienentwicklung identifizierten die Tagungsteilnehmer die folgenden Prinzipien und Verfahren:

- Das richtige Zeitfenster für die Einführung von Produktlinienentwicklung abwarten: Wenn zu großer Projektdruck besteht, kann die Organisation Neuerungen nicht aufnehmen. Günstig ist ein Zeitpunkt, wenn eine neue Produktgeneration konzipiert wird und Schwierigkeiten mit der bisherigen Einzelproduktentwicklung offenbar sind.
- Schrittweise, geplante Migration: Produktlinienentwicklung erfordert vielfältige Umstellungen und Anpassungen in Vorgehen und Zusammenarbeit innerhalb der Organisation und gegenüber dem Umfeld (z.B. Kunden und Lieferanten). Die Änderungen sollten in kleinen stabilen Schritten erfolgen, die durch eine längerfristige Strategie geleitet werden. Das Management muss sich klar zu dieser Strategie bekennen.

- Champions: Bei den ersten Schritten zum Aufbau einer Produktlinienentwicklung sollten vor allem erfahrene Mitarbeiter („Champions“) einbezogen werden, die der Produktlinienidee offen und unterstützend gegenüber stehen. Sie sind die beste Voraussetzung, dass unvermeidliche Schwierigkeiten gut gemeistert werden. Später fungieren sie als Meinungsführer und leiten die Kollegen an, die neu zur Produktlinienentwicklung dazu stoßen.

Neben allen wichtigen technischen Vorkehrungen im Rahmen der Produktlinienentwicklung (z.B. Methoden, Software-Werkzeuge, Vorgehensweisen) sind auch der Faktor Mensch und organisatorische Aspekte von entscheidender Bedeutung für den Erfolg einer Produktlinieneinführung. Die Mitarbeiterinnen und Mitarbeiter müssen sich in die Änderungsprozesse aktiv einbringen können. Mögliche Vorbehalte müssen sorgfältig adressiert werden und ihre Anregungen müssen aufgenommen werden.

Zu den wichtigen organisatorischen Aspekten gehört, dass die Infrastrukturkosten zum Betrieb einer Produktlinienentwicklung erkann werden müssen. Das Management muss die Kosten in Relation zum erwarteten Nutzen bewerten und ihren Umfang akzeptieren. Außerdem muss das Zusammenspiel zwischen Plattform- und Anwendungsentwicklung effizient und reibungslos gestaltet werden. Wird all dies beachtet, dann sind wichtige Voraussetzungen erfüllt, um Produktlinienentwicklung erfolgreich einzuführen und ihren Nutzen zu etablieren.

2 Mini-Tutorial: 15 Jahre Software-Produktlinien— Einführung und aktueller Stand

Andreas Birk, Klaus Schmid

2.1 Überblick

Das Mini-Tutorial führt die grundlegenden Konzepte von Software-Produktlinien und ihrer Anwendung ein. Ausgehend von der historischen Entwicklung von Produktlinien und Wiederverwendungskonzepten im Software-Engineering zeigt das Tutorial den heutigen Stand der Praxis und die aktuell relevanten Fragestellungen auf.

Personen, die zum ersten Mal Kontakt mit dem Thema Software-Produktlinien haben, erfahren, was sie von Produktlinienentwicklung erwarten dürfen. Personen, die bereits mit Produktlinienentwicklung vertraut sind, erhalten Hinweise auf weiterführende Themen und den Umgang mit möglichen Herausforderungen beim Einsatz von Produktlinien.

2.2 Gliederung und Schwerpunktthemen

Das Mini-Tutorial gliedert sich in die folgenden Teile:

- Was ist eine Software-Produktlinien (SPL)?
- Historische Entwicklung des SPL-Gebietes
- Konzepte und Verfahren der Produktlinienentwicklung
- SPL-Fallstudien
- Kernfragen aus SPL-Forschung und -Praxis

Die folgenden Abschnitte stellen die Inhalte dieser Punkte jeweils genauer da. Weitere Details enthalten die beigefügten Auszüge aus dem Vortragsfolien des Mini-Tutorials.

2.3 Was ist eine Software-Produktlinien (SPL)?

Das Vorgängerkonzept für unser heutiges Verständnis von Software-Produktlinien (SPL) ist die Idee der Software-Produktfamilie, wie sie 1976 von Parnas formuliert wurde. Die heute verbreiteteste Definition stammt von Clements und Northrop (2002). Sie betont, dass eine Produktlinie auf systematische, geplante Weise unter Verwendung eines Satzes von wiederverwendbaren *Core Assets* entwickelt wird. Bei diesen Core Assets kann es sich um Software-Code handelt, genauso aber auch um Modelle, Architekturprinzipien, Testfälle oder Dokumentationsbausteine.

Der Begriff der Produktlinie ist in Marketing und Software-Entwicklung unterschiedlich belegt. Marketing bezieht sich auf die gemeinsamen Eigenschaften unterschiedlicher Produkte, unabhängig von der Entwicklungsseite. Beim Begriffsverständnis in der Software-Entwicklung ist wichtig, dass die Produkte aus einer gemeinsamen Basis (Core Assets) heraus entwickelt werden – aus der Marketingperspektive können die einzelnen Produkte zu verschiedenen Produktlinien gehören.

Produktlinienentwicklung ist letztendlich ein Wiederverwendungsansatz. Von traditionellen Wiederverwendungsverfahren mit sogenannter „opportunistischer“ Strategie unterscheidet sie

sich durch den betont systematischen Ansatz. Die Systematik drückt sich dabei vor allem in einer klar zielgerichteten Entwicklung von Software für Wiederverwendung aus. Wenn eine Organisation die unvermeidlichen Komplexität der Produktlinienentwicklung erfolgreich meistert, kann sie sowohl die Produktqualität steigern, die Kosten besser kontrollieren oder gar senken, als auch die Entwicklungszeit wesentlich verkürzen. Dies belegen eine Vielzahl von Fallstudien.

2.4 Historische Entwicklung des SPL-Gebietes

Das Gebiet der Software-Produktlinien hat sich in den vergangenen ca. 15 Jahren schrittweise entwickelt. Dabei hat es wichtige Impulse von verschiedenen Wiederverwendungsansätzen aufgenommen und hängt eng zusammen mit anderen modernen Software-Technologien, wie beispielsweise Softwarearchitekturen, Framework-Entwicklung, domänenspezifische Entwicklung, modellbasierte Verfahren und Software-Generierung.

Wesentliche Fortschritte wurden durch eine Reihe von großen europäischen Forschungsprojekten und ein Programm des amerikanischen Software Engineering Institute erzielt. Beide Aktivitätsstränge brachten eigene Konferenzserien hervor, die heute zur zentralen international Software Product Line Conference zusammengeführt sind.

Den Nutzen und das Potenzial der Software-Produktlinienentwicklung belegen eine Vielzahl von Erfahrungen aus der Industrie. Die Erfolgsberichte kommen aus großen wie aus kleinen Unternehmen und den unterschiedlichsten Branchen. Vor allem hat sich gezeigt, dass sowohl die Entwicklung von Informationssystemen als auch von eingebetteten Systemen von diesem Ansatz profitieren können. Relevant sind die Verfahren insbesondere, wenn der Markt eine Vielzahl von Produktvarianten fordert. Dies können sowohl kundenspezifische Varianten sein, Produktvarianten um unterschiedliche Marktsegmente zu adressieren, oder auch eine Kombination beider Fälle. Eine große Zahl von Varianten findet sich insbesondere in vielen technischen Bereichen und für eingebettete Systeme. Doch auch bei Anwendungssoftware nimmt die Variantenvielfalt stetig zu. In dieser Situation findet die systematische Produktlinienentwicklung immer weitere Verbreitung.

2.5 Konzepte und Verfahren der Produktlinienentwicklung

Zur erfolgreichen Produktlinienentwicklung gehören Konzepte und Verfahren aus den unterschiedlichsten Bereichen von Technologie, Prozessen, Business und Organisation.

Wichtige Säulen der Produktlinienentwicklung sind:

- Eine Integration der Produktlinienentwicklung in die Geschäftsstrategie, was zu wechselseitigen Abhängigkeiten führt. (Die Geschäftsstrategie bestimmt die Produktlinienentwicklung, aber die Produktlinienentwicklung beeinflusst auch die Geschäftsstrategie.)
- Systematisches Management der Produktvariabilität. Dies umfasst die Identifikation der benötigten Variabilität ebenso wie das kontinuierliche Management der Variabilität, sowohl innerhalb der gesamten Entwicklung als auch über die Zeit hinweg.
- Architektur-zentrierte Software-Entwicklung. Dies bedeutet dass die Etablierung einer einheitlichen Referenzarchitektur und ihre übergreifende Berücksichtigung von zentraler Bedeutung sind.

- Ein Software-Lebenszyklus-Modell, das die zwei Bereiche Plattformentwicklung (Core Assets) und Produktvarianten-Entwicklung unterscheidet (das sogenannte Zwei-Lebenszyklus-Modell)

Innerhalb dieses Rahmens gibt es unterschiedliche Spielarten von Software-Produktlinien (z.B. Produktsuiten und kundenspezifisch angepasste Produkte). Bei deren Entwicklung können recht unterschiedliche Verfahren, Technologien und Organisationmodelle zum Einsatz kommen. Ein Beispiel sind die verschiedenen Arten, mit denen man die Teams innerhalb der Entwicklungsorganisation gliedert und ihnen die verschiedenen Bestandteile einer Produktlinienarchitektur als Entwicklungsaufgaben zuweist. Dabei kann das jeweils zweckmäßigste Verfahren auch mit dem Etablierungsgrad der Produktline und mit der technischen Durchdringung der Produktlinienproblematik im Laufe der Zeit wechseln.

2.6 SPL-Fallstudien

Es gibt heute eine Vielzahl von Erfahrungsberichten und Fallstudien die die Machbarkeit und den Nutzen der Software-Produktlinienentwicklung belegen. Anhand zweier unterschiedlicher Fälle illustriert das Mini-Tutorial Vorteile und Kennzeichen der Produktlinien-Entwicklung.

Das Beispiel der Market Maker Software AG belegt, dass auch kleine Unternehmen Produktlinien erfolgreich nutzen und wichtige Vorteile am Markt erschließen können. Die Umsetzung von Produktlinien in großen Organisationen veranschaulicht der Fall von Philips Medical Systems. Interessanterweise berichten die beiden sehr unterschiedlichen Unternehmen vergleichbare Nutzenausprägungen. So ist in beiden Fällen die Entwicklungsdauer der Produkte auf 50 Prozent gesunken, also doppelt so schnell wie vor der Produktlinieneinführung. Auch die Wartungskosten sanken in beiden Fällen um etwa 60 Prozent des vorangegangenen Wertes.

2.7 Kernfragen aus SPL-Forschung und -Praxis

Trotz aller Vorteile und Erfolge von Software-Produktlinien bietet das Verfahren noch eine Reihe von Herausforderungen. Dazu gehören zurzeit insbesondere:

- Wie kann komplexe Produktvariabilität, wie sie in der Praxis sehr häufig auftritt, am effektivsten dargestellt und verwaltet werden?
- Wie kann man Produktlinienentwicklung mit anderen Verfahren am Besten integrieren, zum Beispiel mit agiler Software-Entwicklung?
- Wie kann man innerhalb einer Organisation das Bewusstsein für die besonderen Erfordernisse der Produktlinienentwicklung am Besten etablieren?
- Wie gestaltet man den Übergang von traditioneller Nicht-Produktlinienentwicklung hin zur Produktlinienentwicklung am flüssigsten?

3 Lösungsorientierte Software Produktlinienentwicklung in heterogenen Systemlandschaften

Thomas von der Maßen, Andreas Wübbeke

3.1 Zusammenfassung

Die Entwicklung von Software Produktlinien (SPL) bietet die Möglichkeit durch gezielte Wiederverwendung ähnliche Produkte schneller und qualitativ hochwertiger zu erstellen. Neben der aus der Literatur bekannten Anwendung des SPL-Entwicklungsvorgehens im Bereich der klassischen Produktentwicklung (z. B. im Bereich der eingebetteter Systeme), kann das Entwicklungsparadigma auch bei kundenindividuellen Projekten, basierend auf heterogenen Systemlandschaften, seine bekannten Vorteile präsentieren. In diesem Beitrag sollen die besonderen Eigenschaften von SPL-Entwicklung für Lösungen in diesem Bereich am Beispiel von arvato services aufgezeigt und daraus Anforderungen für die Adaption des Entwicklungsprozesses für diese spezielle Form der SPL Entwicklung abgeleitet werden.

3.2 Einleitung

Software Produktlinien (SPL), gemäß der Definition von [CN01], bieten für die industrielle Entwicklung von gleichartigen Softwareprodukten für eine bestimmte Domäne Vorteile. Sie unterstützen die Produktentwicklung innerhalb einer Domäne, indem sie die kontrollierte Wiederverwendung von Entwicklungsartefakten ermöglichen. Diese Wiederverwendung wird durch die explizite Modellierung des Schlüsselmerkmals Variabilität ermöglicht. Einführende Informationen zu SPL finden sich in [PBL05] und [CN01]. Die dort motivierten Entwicklungsvorgehen für Software Produktlinien fokussieren meist auf die Entwicklung von klassischen Produkten für eine bestimmte Domäne basierend auf einer gemeinsamen Plattform. Hierbei ist der Bereich der eingebetteten Systeme (vor allem der Automobilbereich) ein oft genanntes Beispiel. In diesem Kontext werden die genannten Produkte entwickelt und anschließend als solche vertrieben.

Neben dem Bereich der eingebetteten Systeme ist auch die Domäne der Informationssysteme ein Gebiet, auf dem ein SPL-Vorgehen Vorteile bei der Entwicklung bieten kann.

Arvato services bietet in diesem Bereich kundenindividuelle Lösungen aus den Domänen Kundenbindung, Loyalty und Fulfilment an. Damit existieren bei der Entwicklung von Lösungen für die jeweilige Domäne viele Gemeinsamkeiten. Aus diesem Grund wurde mit der Einführung des Software Produtklinien-Entwicklungsansatz begonnen. Erste Erkenntnisse für den Teilbereich der Variabilitätsmodellierung werden in [MW08] beschrieben. Im Rahmen der SPL Initiative und dabei im Speziellen bei der Definition des arvato spezifischen Entwicklungsvorgehens zeigten sich in vielen Punkten Besonderheiten im Vergleich zu klassischen Plattform bzw. Produktentwicklungen, wie sie in der Literatur beschrieben sind. Das arvato-spezifische Entwicklungsvorgehen macht an dieser Stelle daher ein angepasstes Vorgehen notwendig, um die Potentiale des SPL-Entwicklungsvorgehens nutzbar zu machen.

In diesem Beitrag sollen daher die Besonderheiten der Lösungsentwicklung bei arvato im Hinblick auf eine mögliche SPL Entwicklung beschrieben werden. Im Anschluss daran wird eine angepasste SPL Entwicklungsarchitektur skizziert, welche den arvato spezifischen Besonderheiten Rechnung tragen soll.

Der Beitrag gliedert sich dabei wie folgt: In Abschnitt 3.3 wird zunächst die existierende arvato Entwicklungsarchitektur mit ihren Besonderheiten beschrieben. Daraus werden im Abschnitt 3.4.1 zunächst Rahmenbedingungen für eine mögliche SPL Entwicklung beschrieben und darauf folgend in Abschnitt 3.4.2 wird eine erste Version eines angepassten SPL-Entwicklungsprozesses skizziert. In Abschnitt 3.5 werden die aus dem angepassten Entwicklungsvorgehen entstehenden Anforderungen beschrieben. Der Abschnitt 3.6 fasst den Beitrag zusammen und beschreibt das weitere Vorgehen, um die Anforderungen umzusetzen.

3.3 Lösungsorientierte Entwicklung kundenindividueller Lösungen

In diesem Abschnitt werden grundlegende Begriffe zum aktuellen, arvato-spezifischen Entwicklungsvorgehen geklärt. Dabei wird zunächst auf die Architektur von kundenindividuellen Lösungen eingegangen. Darauf folgt die Beschreibung des aktuellen Entwicklungsprozesses.

3.3.1 Lösungsarchitektur und heterogene Systemlandschaften

Der Aufbau eines kundenindividuellen Informationssystems geschieht bei arvato nicht auf Basis eines einzelnen, monolithischen Systems. Vielmehr werden mehrere Teilsysteme, genannt Service Systeme, im Rahmen der Entwicklung verbunden, um die kundenindividuellen Anforderungen zu erfüllen.

Diese Service Systeme haben dabei Eigenschaften, die bei der Entwicklung einer Kundenindividuellen Lösung berücksichtig werden müssen:

- Service Systeme adressieren einen funktional geschlossenen Bereich, wie z.B. Finanzbuchhaltung, Prämienlogistik, CRM oder Briefdruckerei
- Service Systeme basieren auf unterschiedlichen technologischen Umsetzungen (z. B. Java oder C#)
- Die Service Systeme sind heterogen da aufgrund des abgedeckten funktionalen Bereichs und der gewählten technologischen Umsetzung hierbei von individueller Entwicklung, über anpassbare Standardsoftware, bis hin zu (konfigurierbarer) Standardsoftware jede Form der Umsetzung der Anforderungen an das jeweilige Service System vertreten ist.
- Die Entwicklung der Service Systeme wird typischerweise von unterschiedlichen organisatorischen Einheiten durchgeführt

Die Erstellung eines kundenindividuellen Informationssystems auf Basis der Service Systeme wird in Form von *Projekten* organisiert. Das Ergebnis des Projekts wird dann (kundenindividuelle) *Lösung* genannt. Eine wichtige Eigenschaft von arvato Lösungen ist, dass arvato nicht nur die Erstellung der Software, sondern auch den Betrieb und den Support übernimmt.

In Abbildung 1 wird die Architektur zweier CRM-Lösungen auf Basis von Service Systemen skizziert. Im oberen Teil der Abbildung ist eine Auswahl an Service Systemen abgebildet. Diese Service Systeme werden nun für zwei Lösungen eingesetzt. In *Lösung A* (gepunktet in der Abbildung) sind dies u. a. das *Prämien-Management* und das *Operative CRM*. Ersteres bietet die Funktionalität für die Verwaltung und den Versand von Prämienartikeln an. Das operative CRM dient als zentraler Ablage für die Informationen über Teilnehmer der CRM-Lösung. Das operative CRM-Service System wird nicht nur in *Lösung A* sondern auch in der gekästelt skizzierten *Lösung B* eingesetzt. Dabei handelt es sich in jeder Lösung um eine eigene Instanz des operativen CRM-Service. Dies liegt darin begründet, dass für viele Service Systeme im Rahmen des Einsatzes in einer bestimmten Lösung individuelle Anteile entwickelt werden müssen. Dies ist in der Abbildung durch die diagonal schraffierten (Lösung A) und schwarz gefüllten (Lösung B)

Rechtecke dargestellt. Die individuell zu entwickelnden Anteile in jedem Service System hängen zum einen von der eingesetzten Technologie des jeweiligen Systems ab (Individualentwicklung, angepasste Standardsoftware, Standardsoftware) und zum anderen von der Art der Kundenanforderungen ab. In Lösung B wird weiterhin ein Website Service System eingesetzt, welches nicht Bestandteil von Lösung A ist. Dieses Service System ist also Teil genau einer Lösung.

Im Beispiel aus Abbildung 1 gibt es also Service Systeme die in allen Lösungen eingesetzt werden (hier z. B. der *operative CRM*-Service, schwarzer Pfeil) und solche, die nur in manchen Lösungen eingesetzt werden (hier beispielsweise das Website Service System, schraffierter Pfeil). Darüber hinaus gibt es auch Service Systeme die als eine einzige Instanz in mehreren Lösungen genutzt werden. Hierfür ist der Service *Kampagnenmanagement* in der Abbildung ein Beispiel. Bei diesem Service werden die individuellen Anteile der Lösungen in ein und dieselbe Instanz des Service implementiert. Diese Form der gemeinsam genutzten Services Stellen eine besondere Herausforderung in den jeweiligen Projekten dar. Die Auswahl von passenden Kandidaten für ein solches Service System beschreiben wir in [As09].

Die einzelnen Service Systeme werden im Rahmen des jeweiligen Projektes mit Hilfe von definierten Schnittstellen verbunden, um die vom Kunden geforderten Funktionalitäten (Geschäftsprozesse) erfüllen zu können. In Abbildung 1 wird dabei am Beispiel der Schnittstelle zwischen dem operativen CRM und dem Debitoren-management deutlich, dass auch Schnittstellen Teil mehrerer Lösungen sein können. Beide Service Systeme sind Teil beider Lösungen und damit existieren auch bei den beiden Instanzen der Schnittstelle potentielle Gemeinsamkeiten.

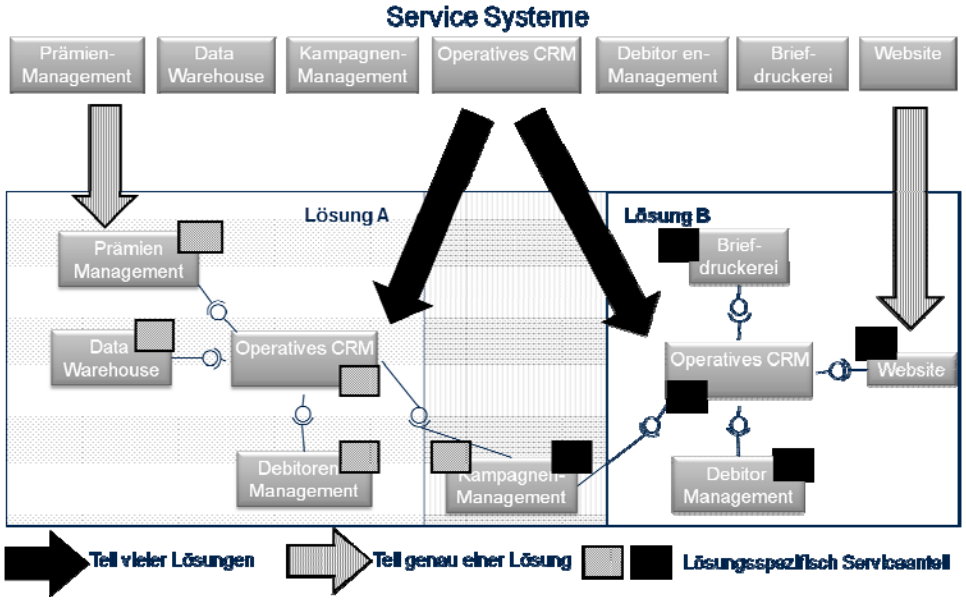


Abbildung 1. Lösungsarchitektur: Zusammenhang Service Systeme und Lösungen.

Zum aktuellen Zeitpunkt wird den in dieser Architektur vorhandenen Gemeinsamkeiten hinsichtlich der Service Systeme und der Schnittstellen zwischen diesen wenig Aufmerksamkeit in Bezug auf die potentielle Optimierung des Entwicklungsprozesses geschenkt.

3.3.2 arvato Entwicklungsprozess

Die im letzten Abschnitt erläuterte Lösungsarchitektur auf Basis der arvato eigenen heterogenen Service System-Landschaft macht ein spezielles Entwicklungsvorgehen notwendig. Als Basis für das Entwicklungsvorgehen dient der *Rational Unified Process* [Kr99]. In diesem Abschnitt wird dieses Vorgehen erläutert, um darauf folgend gemeinsam mit den Erkenntnissen aus Abschnitt 3.3.1 Potentiale für die SPL Einführung abzuleiten.

In Abbildung 2 ist der Entwicklungsprozess für eine kundenspezifische Lösung dargestellt. Hierfür wird wieder die *Lösung A* aus Abbildung 1 als Beispiel verwendet. Die Entwicklung startet mit der Modellierung der Geschäftsprozesse (1). Hier werden die dem Geschäft zugrunde liegenden Prozesse und Geschäftsregeln in Form von textuellen Anwendungsfällen bzw. im Falle der Geschäftsregeln rein textuell beschrieben. Das Vorgehen orientiert sich dabei an [Co01] und [Ja92]. Der zweite Schritt ist die Entscheidung, welche Service Systeme sich am besten für die Umsetzung der Lösungsanforderungen eignen.(2). Das Problem der Identifikation der am besten geeigneten Service System ist nicht Teil dieses Beitrags. Im Rahmen dieser sogenannten Lösungs-Bestimmung wird also die Entscheidung getroffen, welche Service Systeme sich für die Umsetzung der Lösung eignen (2.1) und welche Schnittstellen zwischen den Teilsystemen existieren müssen, um die Geschäftsprozesse zu ermöglichen (2.2). Die einzelnen Service Systeme besitzen, wie bereits in Abschnitt 3.3.1 erläutert, unterschiedliche Mengen an gemeinsamer Funktionalität. Hierbei ist die Funktionalität gemeint, die Teil von mehreren Lösungen sein kann.

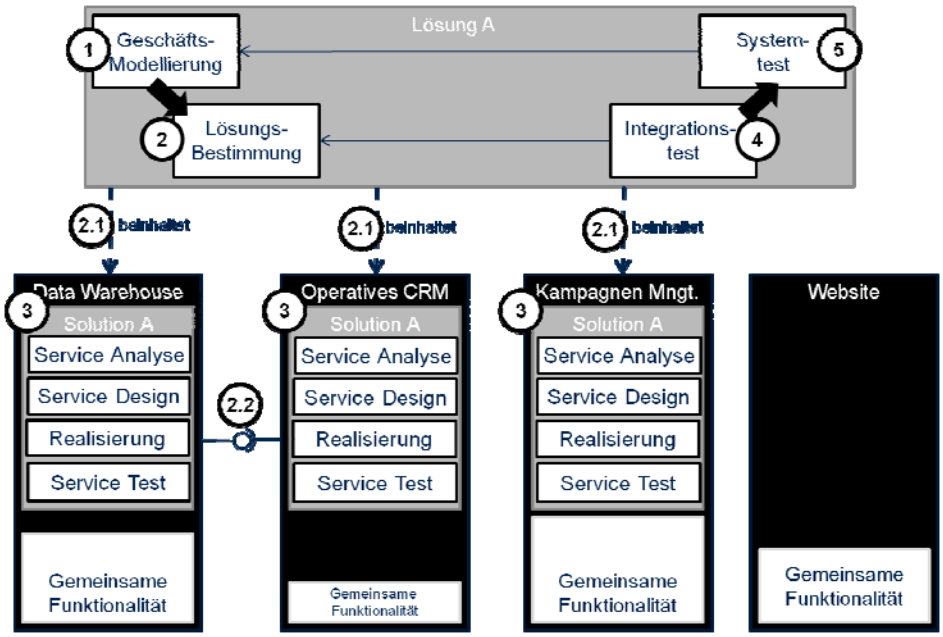


Abbildung 2. arvato Lösungs-Entwicklungsprozess.

Nach der Entscheidung für die jeweiligen Service Systeme wird die Umsetzung der Anforderungen in diesen Systemen durchgeführt. Dazu wird im ersten Schritt eine *Service Analyse* durchgeführt, in dessen Rahmen zunächst die für das Service System relevanten Teile der Geschäftsmodellierung identifiziert werden. Anschließend werden diese Teile detailliert modelliert. In [MW08] sind weitere Informationen über den Zusammenhang zwischen Geschäfts- und Service Modellierung zu finden. Im Anschluss an die Modellierung der service-spezifischen Anforderungen werden das *Service Design*, also die Festlegung der service-eigenen Architektur, sowie die *Implementierung* und der *Service Test* durchgeführt.

Auf Basis der Lösungs-Bestimmung wird anschließend ein Integrationstest durchgeführt (4), der die in (2.2) definierten Schnittstellen gegenüber ihrer Spezifikation validiert. Im letzten Schritt des Entwicklungsprozesses wird auf Basis der Geschäftsanalyse der Systemtest durchgeführt (5). Hierbei werden die Geschäftsprozesse und die Geschäftsregeln validiert.

Wie bereits in [MW08] beschrieben, existieren bei der Entwicklung von unterschiedlichen Lösungen viele Gemeinsamkeiten. Diese Gemeinsamkeiten betreffen zum einen die Modellierung der Gesamtlösung und der dazugehörigen Qualitätssicherung (grauer Kasten in Abbildung 2) und zum anderen immer wieder ähnliche Verwendung der einzelnen Service Systeme in ähnlichen Lösungen.

Die vorliegende arvato Lösungsarchitektur und der dazugehörige Entwicklungsprozess bieten zusammengefasst weitere Potentiale für die Verbesserung der Lösungsentwicklung.

- Es existieren viele Gemeinsamkeiten in verschiedenen Lösungen. Die Entwicklung einer jeden Lösung unabhängig von den anderen bedeutet, dass diese Gemeinsamkeiten immer wieder neu entwickelt werden. Hier existiert ein Potential zur Wiederverwendung, welches den Entwicklungsaufwand verringern kann.
- Jedes Service System für sich gesehen, wird in verschiedenen Lösungen in immer ähnlicher Weise benutzt. Hier existiert ein Potential diese dabei existierenden Gemeinsamkeiten bei der Nutzung von Service Systemen in verschiedenen Lösungen bei der Lösungsentwicklung zu berücksichtigen. Dadurch kann der Entwicklungsaufwand für jedes einzelne Service System und schließlich auch für die Entwicklung von Lösungen verringert werden.

Das Konzept der SPL-Entwicklung und die dadurch mögliche organisierte Wiederverwendung der existierenden Gemeinsamkeiten bietet die Voraussetzungen die genannten Potentiale für arvato nutzbar zu machen. Im Folgenden soll das SPL-Paradigma daher für den arvato Entwicklungsprozess adaptiert werden. Zunächst werden jedoch die Rahmenbedingungen für dieses mögliche SPL-Entwicklungsvorgehen betrachtet.

3.4 Angepasster SPL Entwicklungsprozess

Die in der Lösungsentwicklung existierenden Gemeinsamkeiten, welche in Abschnitt 3.3 beschrieben wurden, erfordern ein spezielles Vorgehen, um sie effizient in der Entwicklung wiederverwenden zu können. In diesem Abschnitt sollen zunächst die Rahmenbedingungen für ein solches Vorgehen entwickelt werden und anschließend diese Bedingungen in der Anpassung des SPL-Entwicklungsprozesses für arvato berücksichtig werden.

3.4.1 Rahmenbedingungen für arvato SPL Entwicklung

Bei den vorliegenden Gemeinsamkeiten müssen zunächst zwei Ebenen im Rahmen der Lösungsentwicklung unterschieden werden: Die erste Ebene bezeichnen wir als Lösungsebene (grauer Kasten in Abbildung 2). Auf dieser Ebene sind die unterschiedlichen Lösungen hinsichtlich ihrer Geschäftsmodellierung und der Lösungsbestimmung ähnlich, da die Lösungen immer für die gleiche spezifische Domäne Entwickelt werden (siehe Abschnitt 3.2). Auf der Ebene der Entwicklung einer spezifischen Lösung kommt es zur Wiederverwendung von Service Systemen. Dabei existieren wiederum Gemeinsamkeiten bzw. Ähnlichkeiten im Einsatz eines einzelnen Service Systems (Service System-Ebene) in verschiedenen Lösungen. Diese zwei Ebenen haben im Zusammenhang mit dem in Abschnitt 3.3 beschriebenen Lösungs-Entwicklungsprozess Auswirkungen auf die Ausgestaltung des Entwicklungsprozesses bei

arvato. Es ergeben sich dadurch konkret folgende Rahmenbedingungen für ein mögliches SPL Entwicklungsvorgehen:

- Berücksichtigung der Gemeinsamkeiten auf Lösungsebene.
- Berücksichtigung der Gemeinsamkeiten auf Service System-Ebene.
- Berücksichtigung des Aufbaus der Lösungsebene. Sie besteht aus der Geschäftsanalyse, der Qualitätssicherung und mehreren Service System Ebenen.
- Berücksichtigung der organisatorische Trennung von Service Systemen. Jedes Service System ist eine eigene Geschäftseinheit.

Auf Basis dieser Rahmenbedingungen wird nun im folgenden Abschnitt eine spezielle SPL Entwicklungsarchitektur skizziert.

3.4.2 Angepasster SPL Entwicklungsprozesses

Unter Berücksichtigung der Rahmenbedingungen, die im letzten Abschnitt definiert wurden, wird eine Mehrplattform-Strategie für den Entwicklungsprozess von arvato gewählt. Diese Strategie bietet die Möglichkeit, die gesetzten Rahmenbedingungen möglichst gut zu erfüllen. In Abbildung 3 ist die aus dieser Idee resultierende SPL-Architektur skizziert. Zunächst wird nun der Entwicklungsablauf anhand der Abbildung erklärt. Dabei entspricht die Lösung aus Sicht des SPL-Paradigmas dem Produkt. Nach der Beschreibung des Entwicklungsvorgehens wird auf die Erfüllung der Rahmenbedingungen eingegangen.

Die Grundsätzliche Idee mehrere Plattformen zu verwenden wird in der Abbildung deutlich. Oben in Abbildung 3 befindet sich die sogenannte *Lösungsplattform*. In dieser befinden sich gemeinsamen Elemente der Geschäftsmodellierung, der Lösungsbestimmung und deren System- bzw. Integrationstestfälle. Bei der Erstellung einer neuen Lösung auf Basis dieser Plattform werden zunächst die passenden Elemente der Plattform-Geschäftsanforderungen selektiert und in die Lösung (hier wieder Lösung A) überführt (1). In diesem Kontext wird die in der Plattform vorhandene Variabilität für die Lösung konkretisiert, d. h. gebunden. Darüber hinaus können auf Ebene der Lösung die lösungsindividuellen Geschäftsanforderungen modelliert werden. Im zweiten Schritt wird die Auswahl der passenden Service Systeme im Rahmen der Lösungsbestimmung durchgeführt (2). Im Rahmen der Lösungsbestimmung werden also die Service Systeme ausgesucht, die zur Realisierung der Geschäftsanforderungen eingesetzt werden sollen. Hierbei existieren in der Lösungsplattform bereits Informationen über die jeweiligen Service Systeme und ihre Eigenschaften. Diese Informationen unterstützen die Auswahl der Systeme für die jeweilige Lösung.

Jedes Service System besitzt seine eigene Plattform (Abbildung 3 unten), die sogenannte *Service System Plattform*. Im Rahmen der Lösungsentwicklung werden nun die gemeinsamen Elemente der Service System Plattformen in die Lösung A übernommen, die zu den bereits gewählten Geschäftsanforderungen passen. In unserem Beispiel geschieht dies für die Service Systeme *Data Warehouse*, *operatives CRM* und *Kampagnenmanagement* (3). Die zuvor in den Geschäftsanforderungen modellierten individuellen Anteile müssen anschließen auch in den jeweiligen Service Systemen berücksichtig werden.

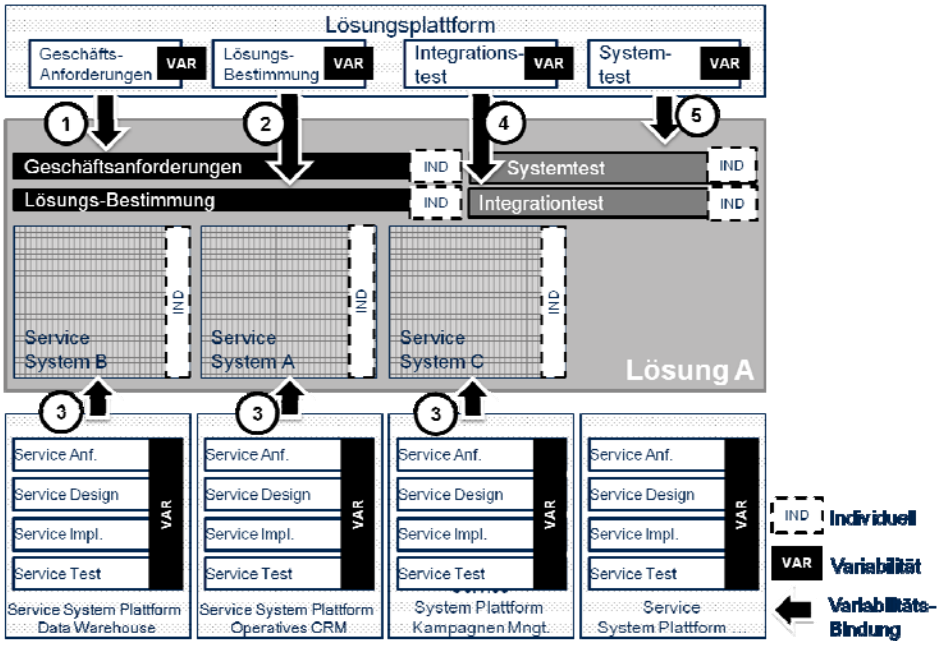


Abbildung 3. SPL Architektur zur Lösungsentwicklung.

Nach Abschluss der Service Entwicklung werden die zur Lösung A passenden Integrationstests aus der Lösungsplattform abgeleitet (4) und mit lösungsspezifischen Anteilen ergänzt. Analog zu diesem Vorgehen wird auch im Fall der Systemtests verfahren (5).

Die vorgestellte SPL-Architektur und das anhand der Architektur motivierte Entwicklungsvorgehen berücksichtigen die in Abschnitt 3.1 geforderten Rahmenbedingungen. Die Gemeinsamkeiten auf den zwei Ebenen Lösung und Service System wurden mit Hilfe von unterschiedlichen Plattformen berücksichtigt. Dabei existiert für jedes Service System eine eigene Plattform. Dies ist darin begründet, dass zum einen jedes Service System eine eigene organisatorische Einheit darstellt und zum anderen mit der Kapselung eines bestimmten funktionalen Bereichs aus Basis einer bestimmten Technologie in einem jeden Service System (siehe Abschnitt 3.3.1).

Durch diese Ausgestaltung der SPL-Architektur entstehen neue Herausforderungen, die im folgenden Abschnitt näher betrachtet werden sollen.

3.5 Anforderungen an die SPL Entwicklung

Der in diesem Beitrag vorgestellte Ansatz für eine SPL-Architektur zur Realisierung von Lösungen auf Basis einer heterogenen Systemlandschaft bringt neue Anforderungen auf verschiedenen Ebenen des Entwicklungsprozesses und der Organisation mit sich. Das Charakteristikum Variabilität ist hierbei das Schlüsselmerkmal.

Eine erste wichtige Anforderung in diesem Zusammenhang ist die Abhängigkeit bezüglich der Variabilität zwischen den Plattformen. Abbildung 4 zeigt beispielhaft eine Lösungsplattform und zwei Service System Plattformen. Hierbei ist die Berücksichtigung der Variabilität zwischen den verschiedenen Ebenen wichtig, damit die beiden Plattform-Ebenen ineinander greifen können. D. h. beispielsweise, dass eine variable Anforderung in den Geschäftsanforderungen Auswirkungen auf die Service Anforderungen zweier Service Systeme hat (im Beispiel Service System Plattformen des Data Warehouse und des operativen CRM). Diese Form der Abhängigkeit wurde initial für das Service System operatives CRM in [MW08] vorgestellt. Weiterhin hat

die Variabilität im Schritt *Lösungsbestimmung* Abhängigkeiten zu der Variabilität in den Schnittstellen zwischen den Service Systemen. Zusätzlich stellt sich die Frage, ob sich analog zu den Geschäfts- und Service-Anforderungen Abhängigkeiten bezüglich der Variabilität in den Systemtestfällen zu den Testfällen des jeweiligen relevanten Service Systems ergeben. Weiterhin ist eine Verfolgbarkeit der Variabilität zwischen Anforderung und Testfällen derselben Plattform wichtig, damit auf Basis der für eine bestimmte Lösung gewählten Anforderungen, analog durch Bindung der Variabilität die passenden Testfälle gewählt werden können (Abbildung 4).

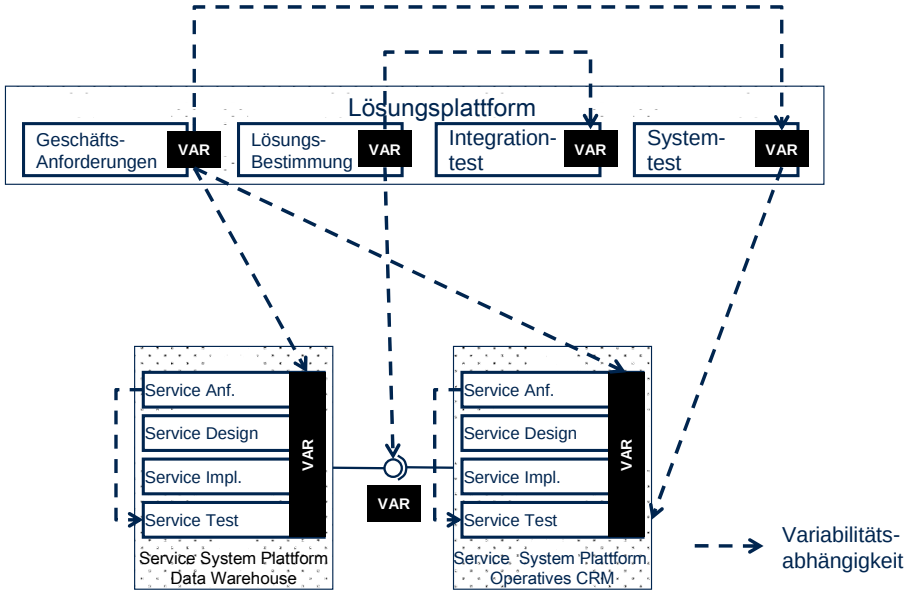


Abbildung 4. Abhängigkeiten zwischen Lösungs- und Service-Plattform(en).

3.6 Ergebnis und weitere Arbeit

Aufgrund des besonderen Charakters der Lösungsentwicklung in einer heterogenen Systemlandschaft bei arvato services besteht die Möglichkeit der Verbesserung des Entwicklungsprozesses durch Einführung des SPL-Entwicklungsvorgehens. Dazu wurde in diesem Beitrag zunächst die aktuelle arvato Entwicklungsarchitektur beschrieben und im Anschluss daraus Potentiale zur Verbesserung abgeleitet (Abschnitt 3.3). Auf dieser Basis wurde eine angepasste SPL-Architektur vorgestellt. Dieser Architektur ist eine Konkretisierung der klassischen SPL Architektur- und Entwicklungsprozess-Schaubilder aus der Literatur (vgl. [Ma07] und [PBL05]). Im Rahmen der Konkretisierung wurden zunächst die Rahmenbedingungen für die Gestaltung der Architektur und des Entwicklungsprozesses bestimmt. Im Anschluss wurde die konkretisierte SPL-Architektur beschrieben und schließlich wurden weitere Anforderungen an diese Architektur bestimmt.

Diese Anforderungen definieren weitere, notwendige Arbeiten zur Ausgestaltung der arvato SPL-Architektur. Dabei soll ein besonderes Augenmerk auf die Abhängigkeit von Variabilität zwischen den verschiedenen Plattform-Ebenen gelegt werden. Ein weiterer interessanter Aspekt ist die Modellierung von Variabilität in den verschiedenen Service Systemen, da hier unterschiedliche Mengen an Gemeinsamkeiten und verschiedenartige Technologien zum Einsatz kommen.

3.7 Literaturverzeichnis

[As09] Assmann, M.; Engels, G.; von der Maßen, T.; Wübbeke, A.: Identifying Software Product Line Component Services, angenommen zur Veröffentlichung, ENASE, 2009.

[CN01] Clements, P.**;** Northrop, L.: Software Product Lines: Practices and Patterns, Addison-Wesley Professional, 2001.

[Co01] Cockburn, A.: Writing Effective Use Cases, Addition-Wesley, 2001.

[Ja92] Jacobson, I.; Christerson, M.; Jonsson, P.; Øvergaard, G.: Object-Oriented Software Engineering – A Use Case Driven Approach, Addision-Wesley, 2. Auflage, 1992.

[Kr99] Kruchten, P.: Der Rational Unified Process - Eine Einführung, Addison-Wesley, 2. aufl. edition, 1999.

[Ma07] von der Maßen, T.: Feature-basierte Modellierung und Analyse von Variabilität in Produktlinienanforderungen, Dissertation RWTH Aachen, 2007.

[MW08] von der Maßen, T.; Wübbeke, A.: Modellierung von Variabilität in der Geschäftsanalyse – eine industrielle Fallstudie. In: W. Maalej and B. Brügge (eds.): Produktlinien im Kontext: Technologie, Prozesse, Business und Organisation (PiK2008). Gesellschaft für Informatik (GI) (München), Lecture Notes in Informatics, vol. 122, pp. 285-296, 2008.

[PBL05] Pohl, K.**;** Böckle, G.; van der Linden, F. J.: Software Product Line Engineering: Foundations, Principles and Techniques, Springer, 2005.

4 Requirements-Engineering für Produktlinien in der Automobilelektronik

Matthias Weber, Mark-Oliver Reiser

Die Idee der produktlinienorientierten Entwicklung von Software und Software-/Hardware-Systemen ist nicht neu und es wurden bereits eine Vielzahl an Methoden und Prozessen vorgeschlagen, um ein solches Vorgehen in der Entwicklung zu unterstützen. Bei der Anwendung des Grundgedankens der Produktlinienentwicklung auf sehr komplexe Entwicklungsgegenstände und vor dem Hintergrund hochkomplexer organisatorischer Rahmenbedingungen, bspw. einem globalen Automobilkonzern, ergeben sich jedoch immer noch äußerst schwierige Herausforderungen, die mit den bislang verfügbaren Entwicklungskonzepten nicht vollständig beherrschbar sind. Im Bereich der Automobilsoftware bedeutet dies, daß die Entwicklung einzelner, relativ abgeschlossener Teilsysteme durch Zulieferer bereits recht gut in den Griff zu bekommen ist, daß jedoch die Integration vieler solcher Teilsysteme zu einem System auf Gesamtfahrzeugebene und damit zu einer Gesamtproduktlinie auf Seite des Herstellers noch erhebliche methodische Fragen aufwirft.

Der Vortrag beschäftigt sich mit diesen Herausforderungen und stellt aktuelle Arbeiten zu zwei Lösungsansätzen vor, die in den letzten Jahren an der Technischen Universität Berlin erarbeitet wurden und derzeit in Zusammenarbeit mit der Carmeq GmbH in Berlin weiterentwickelt werden. Im folgenden sollen diese beiden Lösungsansätze kurz skizziert werden.

4.1 Multi-Level Konzept

Beim Einsatz produktlinienorientierter Entwicklungsmethoden im Kfz-Bereich stellt sich zunächst die Frage, auf welcher Ebene der Produktpalette die Produktlinieninfrastuktur angesetzt werden soll. So ließen sich beispielsweise einzelne Fahrzeugmodelle – wie Polo, Golf, Passat, etc. im Falle von Volkswagen – als jeweils unabhängige Produktlinieninfrastrukturen entwickeln oder die gesamte Produktpalette des Herstellers wird als eine einzige, globale Produktlinieninfrastruktur realisiert – im Beispiel wäre dies „Volkswagenfahrzeug“. Ersteres hat den Nachteil, daß gerade die Vorteile einer produktlinienorientierten Entwicklung zwischen den untergeordneten Linien wie Polo, Golf, etc. verloren gehen; letzteres hat den Nachteil einer enormen, derzeit nicht beherrschbaren Komplexität.

Das Mulit-Level-Konzept (Reiser, Weber 2007) strebt nun einen pragmatischen Kompromiß zwischen diesen beiden Alternativen an: die untergeordneten Produktlinien wie Polo und Golf können zunächst unabhängig voneinander mit ihrer jeweils eigenen Produktlinieninfrastruktur organisiert und gepflegt werden um die Komplexität beherrschbar zu halten; das Multi-Level-Konzept ermöglicht dann eine strategische Planung und Steuerung der Gemeinsamkeiten und Unterschiede auf der Ebene der globalen, übergeordneten Produktlinie.

4.2 Configuration Links

Hierbei handelt es sich zunächst um ein allgemeines, technisches Ausdrucksmittel, mit dem sich eine Konfigurationsbeziehung zwischen zwei (oder mehreren) Featuremodellen definieren läßt (Weber, Reiser 2008). Dabei wird festgelegt, wie das eine Featuremodell (Ziel-Featuremodell) konfiguriert werden muß in Abhängigkeit von einer gegebenen Konfiguration des anderen Modells (Start-Featuremodell). So läßt sich beispielsweise ausdrücken: „wähle im Ziel-

Featuremodell das Feature ‚RegenSensor' aus, wenn im Start-Modell das Feature ‚Komfortpaket' gewählt ist".

Eine solche Konfigurationsbeziehung läßt sich im Kontext komplexer Produktlinieninfrastrukturen äußerst flexibel einsetzen um orthogonale Sichten auf die Variabilität zu realisieren (z.B. Endkundensicht) oder die Variabilitätsbeschreibung in Teilen der Infrastruktur unabhängig zu machen von der globalen Variabilität.

4.3 Literaturverzeichnis

Reiser, M.-O. and Weber, M., *Multi-level feature trees – a pragmatic approach to managing highly complex product families*. Requirements Engineering Journal, 12(2):57-75, Apr 2007.

Weber, M. and Reiser, M.-O., *Product lines in automotive electronics*. In: Nicolas Navet and Francoise Simonot-Lion (editors), *Automotive Embedded Systems Handbook*, Chapter 7, CRC Press Inc., Dec 2008.

5 A model-driven product line approach for HMI development

Marc Born, Omar Ekine, Jon Ashley

5.1 Abstract

The aim of the initiative introduced in this paper is to better control the creation process of infotainment HMI software for automotive OEM customers of DENSO. This is achieved by a combination of a Model Driven Development strategy together with a Software Product Line (SPL) approach. Model Driven Development offers more precise modelling especially of screens and screen flows, early simulation, model validation, automatic transformation and code generation. That speeds up the development process and increases the quality. The SPL approach enables systematic re-use of existing code and the HMI design (including graphics) among multiple projects for different customers and brands. A product derivation from a product line model based on a feature model is realized to significantly reduce the delivery time to the customer once a few projects have been completed. The combination of these two powerful approaches is a significant challenge and the result will radically change the way HMI software development is executed by DENSO.

5.2 HMI development process

In the automotive industry, suppliers are interested to re-use large parts of their implementations when delivering products to different customers. Only with a high degree of re-use will they be able to survive in the hard competition. This holds especially for products with a relatively large part of common behavior like navigation systems. On the other hand, the OEM has a strong interest in the realization of an own look & feel of the HMI as a differentiator to other OEMs. Therefore, the supplier cannot deliver exactly the same navigation system to different OEMs. It is necessary to adapt the HMI to the OEMs needs and consider OEM HMI design guidelines. In future it will be even more important to integrate with an overall HMI concept of the OEM which does not only include the navigation system but all interactions of the driver with the car, e.g. for infotainment as well as for the control and configuration of car functions. This may lead to the requirement that the OEM actively participates in the HMI design process or even does large parts of the HMI design by himself. A tool support is definitely necessary for the development as such with advanced technologies like simulation and code generation but also to manage the different variants a supplier provides for its customers. That's why we decided to combine model-driven development with a product line approach and provide tool support for both aspects.

In order to obtain the requirements, we did an in-depth analysis of the current development process. The observed development process at DENSO was roughly organized as follows:

- The navigation systems core is developed in DENSO Japan and is re-used for different customers. It is also relatively stable in terms of features.

- The HMI is different and designed specifically for each customer by DENSO, with EU customers handled by DENSO UK. For HMI skin design of EU customers DENSO UK use their own TiGRiX™ toolset, which is a very powerful screen definition tool.

- The screen flow specification is mainly developed in Japan using MS Visio diagrams, Excel sheets and explanatory text.

- The realization of the screen flow (in software) is done by manual coding, as well as the connection to the developed graphical skin definitions.

With the advanced tool support delivered by this initiative, the following changes to the development process are anticipated:

- The screen flow specification is provided as a model with clear semantics and can be connected to different skin designs. To realize this, an abstract screen specification is provided as a model. The ambiguity of screen flow specifications is reduced and automatic consistency checks and other means of model validation are available.

- The screen flow model can be simulated to in order to review it and to detect problems in the flow at an early stage. Furthermore the simulation can use the available skin designs (done with the integrated TiGRiX™ toolset) to show up a realistic HMI. This will improve the understanding and communication between development engineers and customers.

- The connection of the HMI software to the navigation core software will be autogenerated out of the models. For that purpose an abstract task model is part of the screen flow models to specify the HMI's need to access the navigation core and to define the purpose of that access.

- The screen flow model will ultimately be linked with a feature model. This will allow the feature based configuration and derivation of customer specific variants to be created much more easily.

We describe the modeling support for the HMI in more detail in Section 5.3. The variant modeling and the variant model derivation is detailed in Section 5.4.

5.3 The model-driven approach

The realization of the model-driven development of the HMI is by the introduction of the following models:

- an abstract screen model that defines the structure of the screens of the HMI and the possible actions which can be triggered form each particular screen,

- a screen flow model which models the transitions between screens; the transitions originate from actions which can be triggered from the screens,

- a task model which contains tasks which are expected to be executed by the underlying core software (e.g. the navigation core).

All those models are described by meta models and dedicated graphical editors are implemented supporting the notation of these models. Relations between the different models are defined as relations between the elements of the corresponding meta models. They are used to express the various mappings mentioned before.

Abbildung 5 shows an example for a screen flow model. It can be seen that the notation is similar to UML activities, which are indeed used as a basis for the flow model. The nodes in the flow diagram are either task invocations (e.g. "determine location"), screen invocations (e.g. "Show Main destination entry screen") or invocations of other screen flows (e.g. "Enter POI"). The pins are used to model the actions which that trigger a transition to be taken. In contrast to UML activities, a node is left only via exactly one of the pins. To embed a screen flow in its invoking context, parameter nodes are used (e.g. "Start action") which map to pins in the invok-

ing context. Another concept not shown on this figure is alternative screen flows modelled by decision nodes.

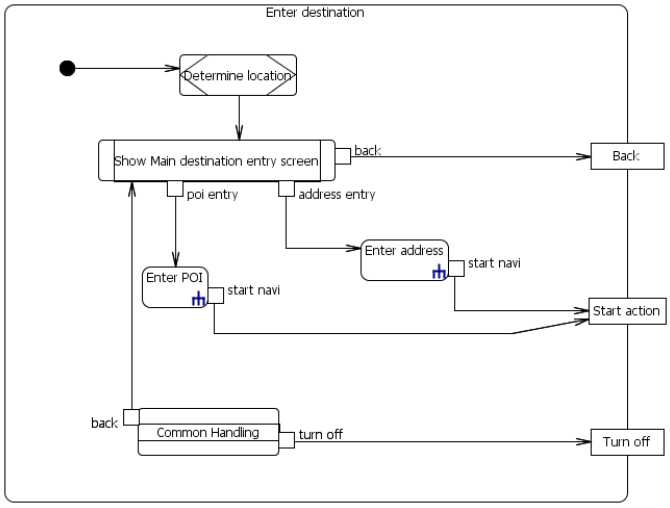


Abbildung 5. Screen flow model.

The abstract screen model is shown in Abbildung 6. It can be seen that the screen definitions contain the potential actions which can be triggered when the screen is invoked. The “Destination entry screen” contains two actions and inherits two others. In a screen flow model, the actions correspond to pins in the screen invocation.

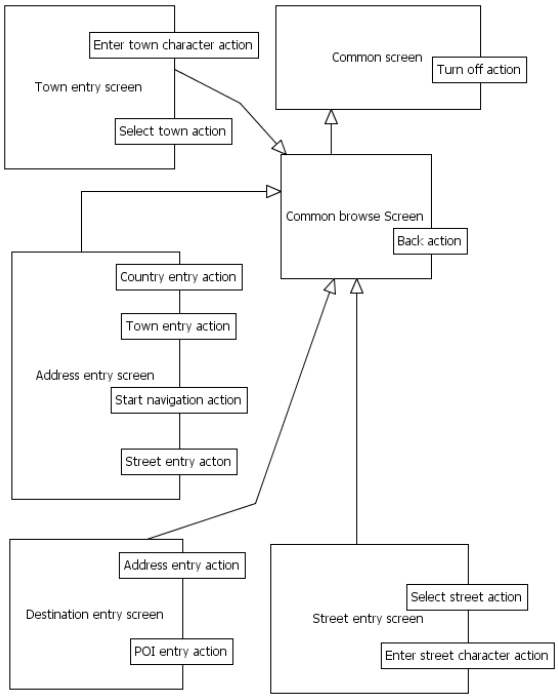


Abbildung 6. Abstract screen model.

In order to fulfil the requirement for rapid development and improve the communication with customers, screen flows may be simulated. Simulation means to execute the activity model according to the execution semantics as defined for the activity meta model. In a certain screen invocation the simulation engine receives events which then trigger transitions. Such events

may originate from the screen modelling tool, if it has been previously mapped to the abstract screen model, but also other simulation event sources can be integrated including the target device or the flow model editor. Abbildung 7 shows such a simulation in the tool developed by us. It can be seen that the real screen (skin) is displayed, when the execution reaches the corresponding screen invocation node. Based on the mapping that is defined between the real screen, the abstract screen in the model and the invocation of the screen in the flow model, the user may select the button on the real screen and the simulator will execute the correct transition and bring up the screen which follows according to the flow model.

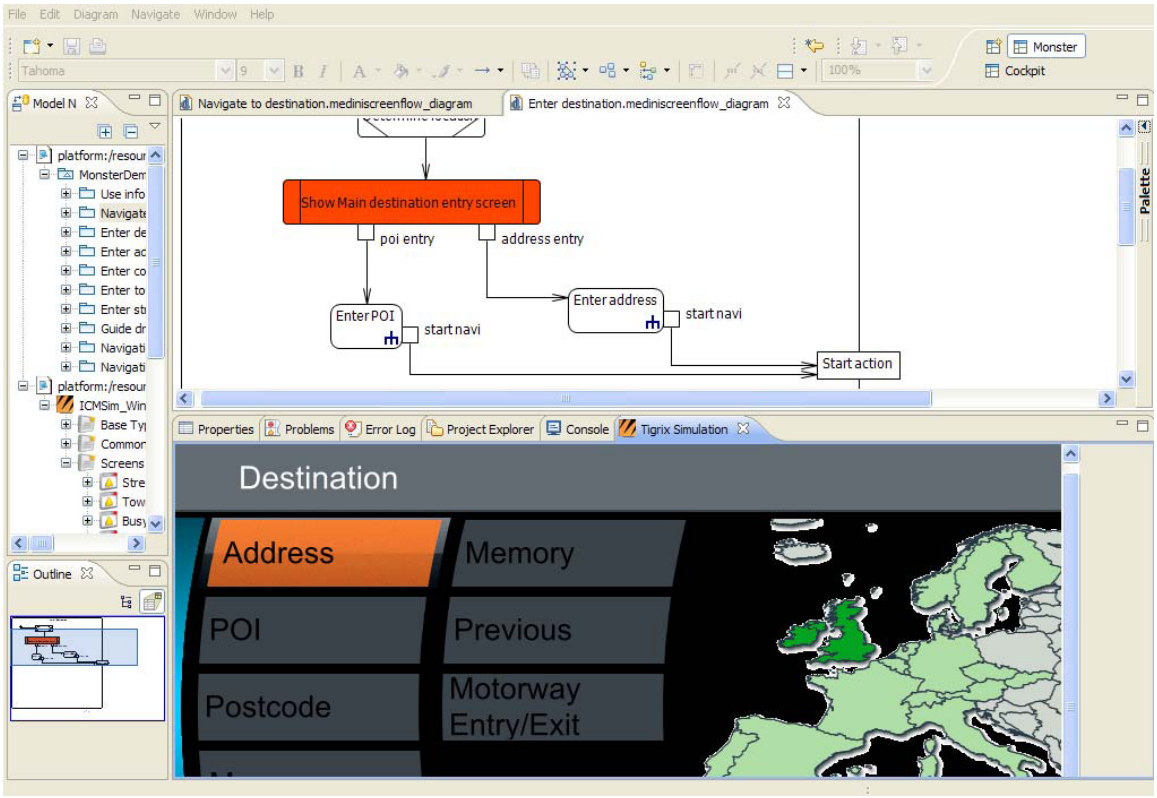


Abbildung 7. Simulation of screen flow model.

5.4 Product line support

Model-driven development as introduced in the previous section already offers a significant productivity enhancement. However, the problem of supporting different variants for different customers is not completely solved. Another strategy is needed and the models have to be extended accordingly. We decided to apply a product line approach for variant support. Assuming, that we have the possibility to define a family screen flow model which implicitly contains all possible variants, we can use feature models to derive concrete variant screen flow models out of such a family screen flow model. The general approach is visualized in Abbildung 8. For the feature modelling, we provided an appropriate meta model and graphical notation which supports mandatory and optional sub-feature relations as well as logical dependencies between features. Features can be related to screen flows by a feature mapping.

Since the screen flow model is the main model, we restrict ourselves to consider variability in screen flow models. If the derivation of a variant model out of a family model should lead to a consistent and complete variant model, we need to define all aspects of all variants in the family model. To specify such variability we need to express at least:

1. Screen flows, which are present in some variant models and not present in others (optional flows)

2. Invoked screen flows which are differently realized in some variants (variant flows).

The possible variants are explicitly modelled in the family model - for that purpose, a new modelling concept is included in the screen flow model. At so called variation points which are inserted in a transition, alternative screen flows can be invoked. Assuming that we want to model the fact, that a different "setting" screen flow is used for variants delivered to different markets. The family model may look like in Abbildung 9. The settings action has an outgoing transition leading to a variation point. From the variation point different screen flows are the target of the transition. Which of the screen flows is included in a variant model is determined during the feature selection process for the variant model derivation.

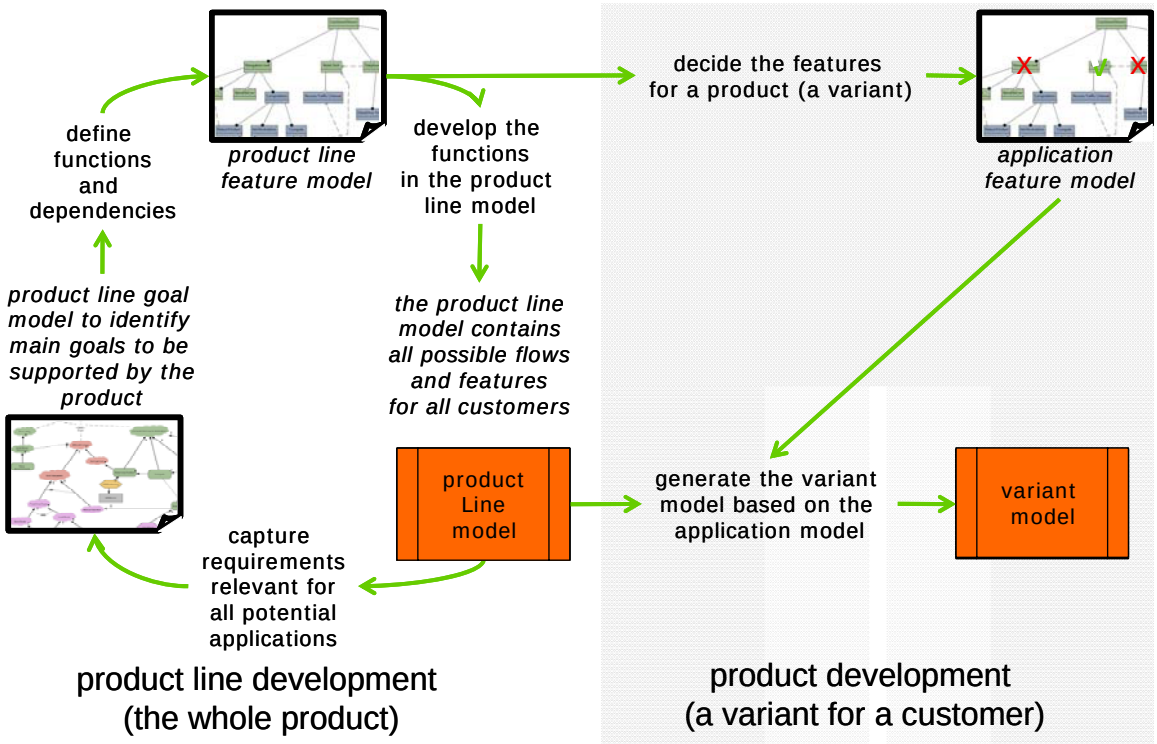


Abbildung 8. Variant model derivation.

To enable the variant model derivation, two pre-conditions have to be fulfilled:

1. the family feature model needs to be mapped to the family screen flow model and

2. the feature decision has to be made and the feature variant model has to be derived.

Regarding the first aspect, we see in Abbildung 10 the association of features to screen flows via a simple mapping table. This mapping needs to be done for all screen flows that are targets of the variability point.

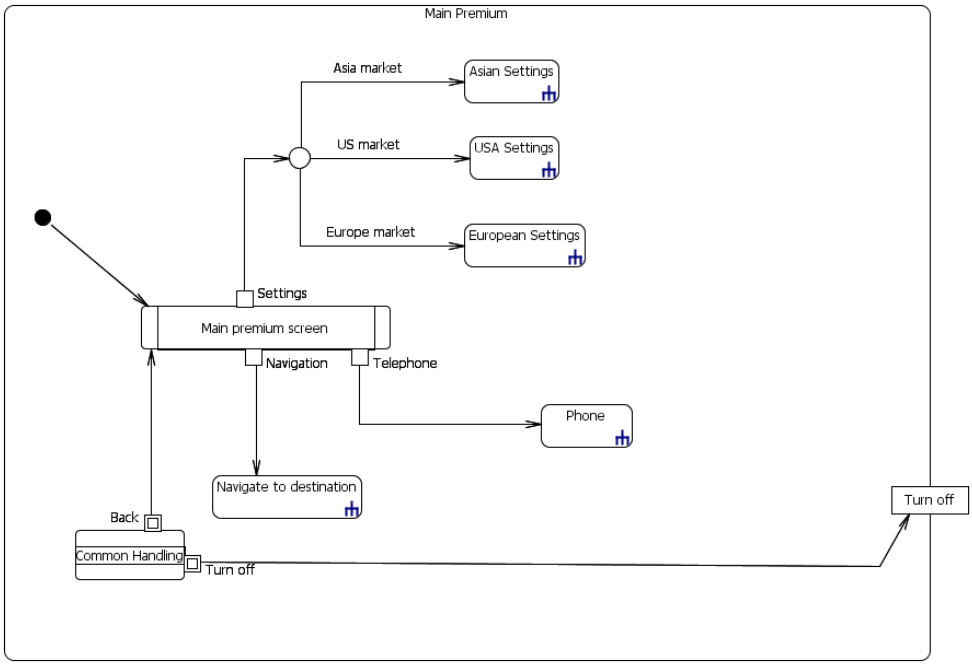


Abbildung 9. Screen flow family model with variability point.

For the second aspect, the graphical editor for the feature definition also allows feature selection. The feature selection observes dependencies and logical relations between features, so that the resultant variant feature tree is always correct wrt. the feature model.

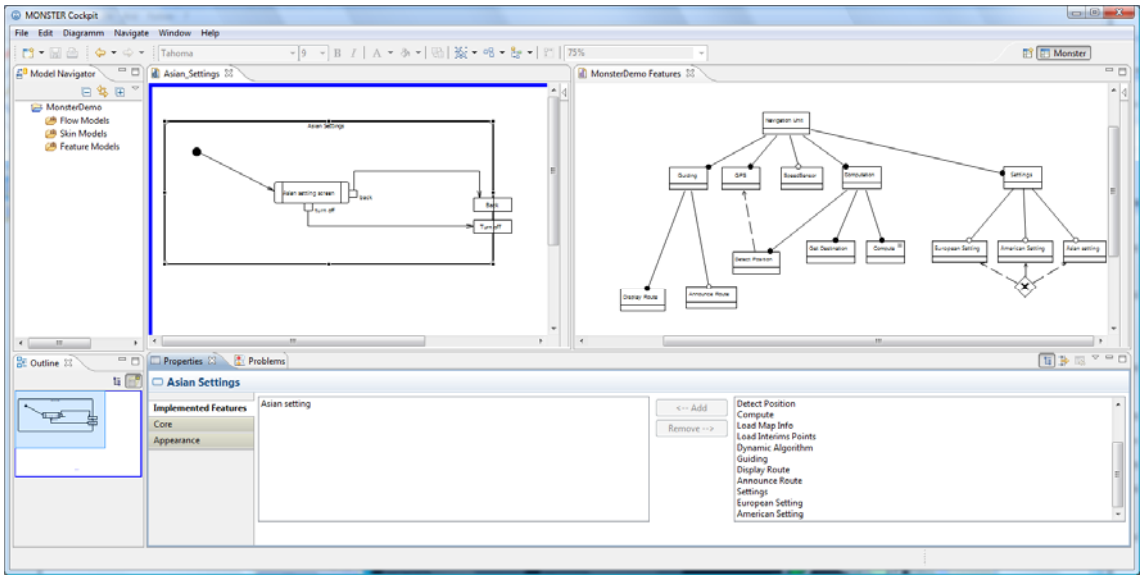


Abbildung 10. Mapping features to screen flows in family models.

For the derivation of the variant feature tree we use the Query View Transformations (QVT) relational model transformation as to be seen in Abbildung 11.

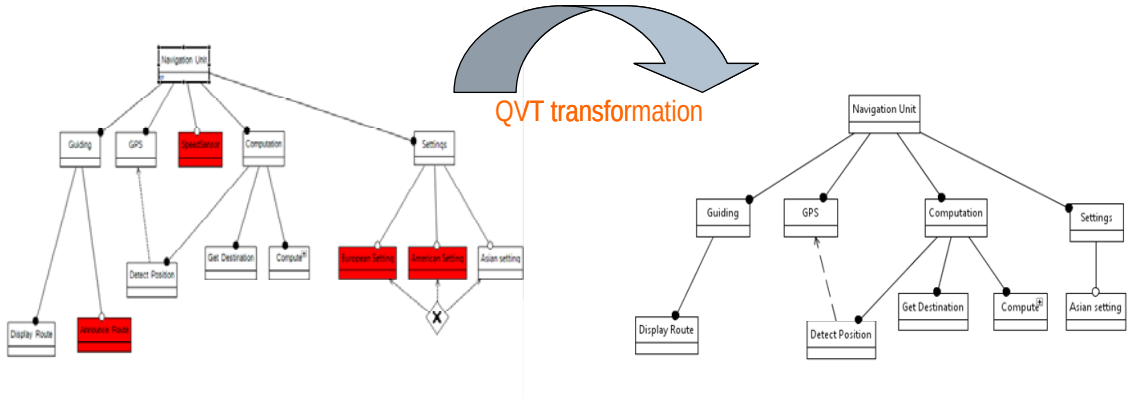


Abbildung 11. Variant feature model derivation.

After the variant feature model is derived, the QVT engine is used again to derive also the variant screen flow model. It uses therefore the mapping of the features to screen flow definitions to decide for each variation point, which of the flow invocations has to be included in the variant screen flow model. After the transformation process, the derived variant screen flow model is validated to not allow a not valid variant model to be generated. In the above example, the derivation would lead to a screen flow with only the Asian setting available. This is depicted in Abbildung 12.

With the variability points in screen flow models we can express both optional and variable flows. A variable flow is expressed like in the example above. For an optional flow, different screen definition variants are modelled which differ in the actions they define (e.g. the main infotainment screen of a premium variant offers telephone functionality while that of the standard variant doesn't). The screens are invoked in different screen flow definitions which are used in screen flow invocations connected to a variability point.

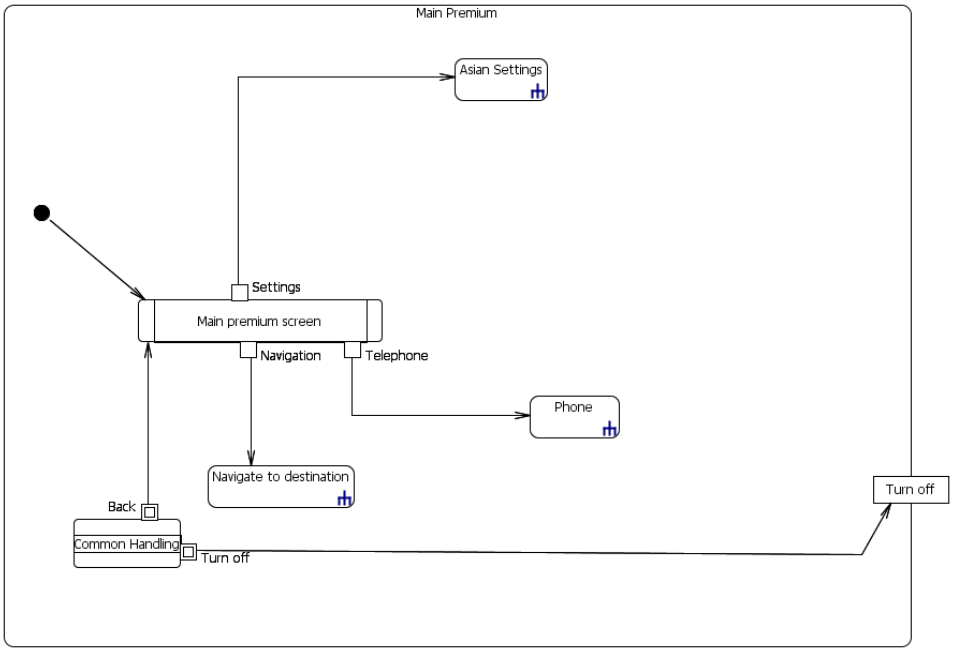


Abbildung 12. Variant screen flow model.

5.5 Conclusion

With the tool described herein, we have shown that model-driven development and product line approach can be combined in a useful way to take advantage of both approaches - the productivity advantage can be realized due to the integration of both approaches: flexibility and automation as well as high degree of re-use. The usage of the QVT relational engine for the variant model derivation offers the possibility to benefit from the backward transformation capability of QVT. When a variant model is adapted or extended, and it is obvious, that such extensions or adaptations are useful for other customers, they can be backward propagated into the family model without extra effort.

Our next step is to apply the approach in multiple DENSO customer contexts in order to measure efficiency improvements as well as the impact of each project carried out on the expected effort for subsequent projects. Furthermore, we will extend the variant generation to also cover the connection to the navigation core API.

5.6 References

[1] Object Management Group, MOF 2.0 Query / Views / Transformations RFP, OMG document ad/02-04-10, April 2002.

[2] Object Management Group, MOF 2.0 Core Proposal, OMG document ad/2002-09-14, Sept, 2002.

[3] Object Management Group, Model Driven Architecture (MDA), ormsc/01-07-01, July 2001.

[4] Object Management Group, UML 2.0 Superstructure FTF convenience document, OMG document ptc/04-10-02

[5] Extensible Markup Language (XML) 1.0, 2nd edition. W3C Recommendation, 2000, Document REC-xml-20001006.

[6] Object Management Group (Hrsg.), XML Metadata Interchange (XMI), 2002, OMG-Document formal/00-11-02

[7] ikv^{++} technologies ag, medini QVT engine, http://www.ikv.de/

6 UML und Softwareproduktlinien – Wie, Weshalb und Wann?

Enrico Schnepel, Danilo Beuche

Softwareproduktlinien (SPL) und die Unified Modelling Language (UML) haben mit der modellgetriebenen Softwareentwicklung (MDSD) ein gemeinsames verwandtes Themengebiet, sind jedoch selbst sehr verschieden. Die UML kann in Kombination mit Profilen als Eingangsmodell für die MDSD dienen und mit den Modellen einer SPL lässt sich ebenfalls modellgetrieben Software entwickeln. Gerade in der Kombination UML und SPL tauchen verschiedene Fragen auf: Wie sehen Integrationen von UML und SPL aus? Und wann kann UML in Kombination mit SPL eingesetzt werden?

6.1 Produktlinien im Kontext

Eine große Produktvielfalt, wie sie z.B. in der Automobilindustrie zu finden ist, hängt meistens eng mit Produktlinien zusammen. Anders könnten die komplexen Anforderungen und deren vielfältigen Abhängigkeiten auch nicht verwaltet werden. Auch in der Informatik werden „Software-“Produktlinien (SPL) eingesetzt. Hier geht es sogar soweit, dass eine vollständige Software, durch die im Rahmen einer SPL erstellten Konfiguration, modellgetrieben generiert und ausgeliefert werden kann.

Modellgetriebene Softwareentwicklung

Modelle bilden die Basis einer jeden Softwareentwicklung. Manchmal existieren sie nur als gedankliches Konzept. Wenn sie jedoch vollständig und formal digitalisiert werden, besteht die Möglichkeit mit den Modellen mehr anzufangen, als sie nur zur Dokumentation zu verwenden. Im Rahmen der modellgetriebenen Softwareentwicklung lässt sich ein Regelwerk erstellen, mit dem solche Modelle zu einer vollständigen und funktionsfähigen Software oder zumindest zu wichtigen Artefakten dieser Software verarbeitet werden können. Gerade im Kontext von SPL lassen sich mit den hier bereits existierenden Variantenmodellen Synergien erreichen.

Die Unified Modelling Language

Die Object Management Group (OMG) hat mit der Unified Modelling Language (UML) eine universelle Sprache geschaffen, die sich durch Profile an die unterschiedlichsten Anforderungen anpassen lässt. Im Zusammenhang mit der modellgetriebenen Softwareentwicklung lassen sich mit Profilen domänenspezifische Sprachen entwickeln, sowie die für die MDSD so wichtigen Modelle erstellen.

Die Integration von UML in SPL wird in den folgenden Abschnitten beschrieben.

6.2 Wie sehen Integrationen von UML und SPL aus?

Im Kontext von Produktlinien lässt sich die UML an verschiedenen Stellen einsetzen. Zum einen ist hier mit

„SysML“ ein Profil zu nennen, mit dem komplexe Systeme ganzheitlich, d.h. über alle Lebenszyklen, beschrieben werden können. Von Stahl/Weilkiens wurde hierzu die Erweiterung „SysML/RE“ zur Definition von Varianten diskutiert (siehe Abbildung 13). Sie ermöglicht es UML als Werkzeug für das Anforderungsmanagement (Requirements Engineering, RE) im Kontext von Produktlinien zu verwenden.

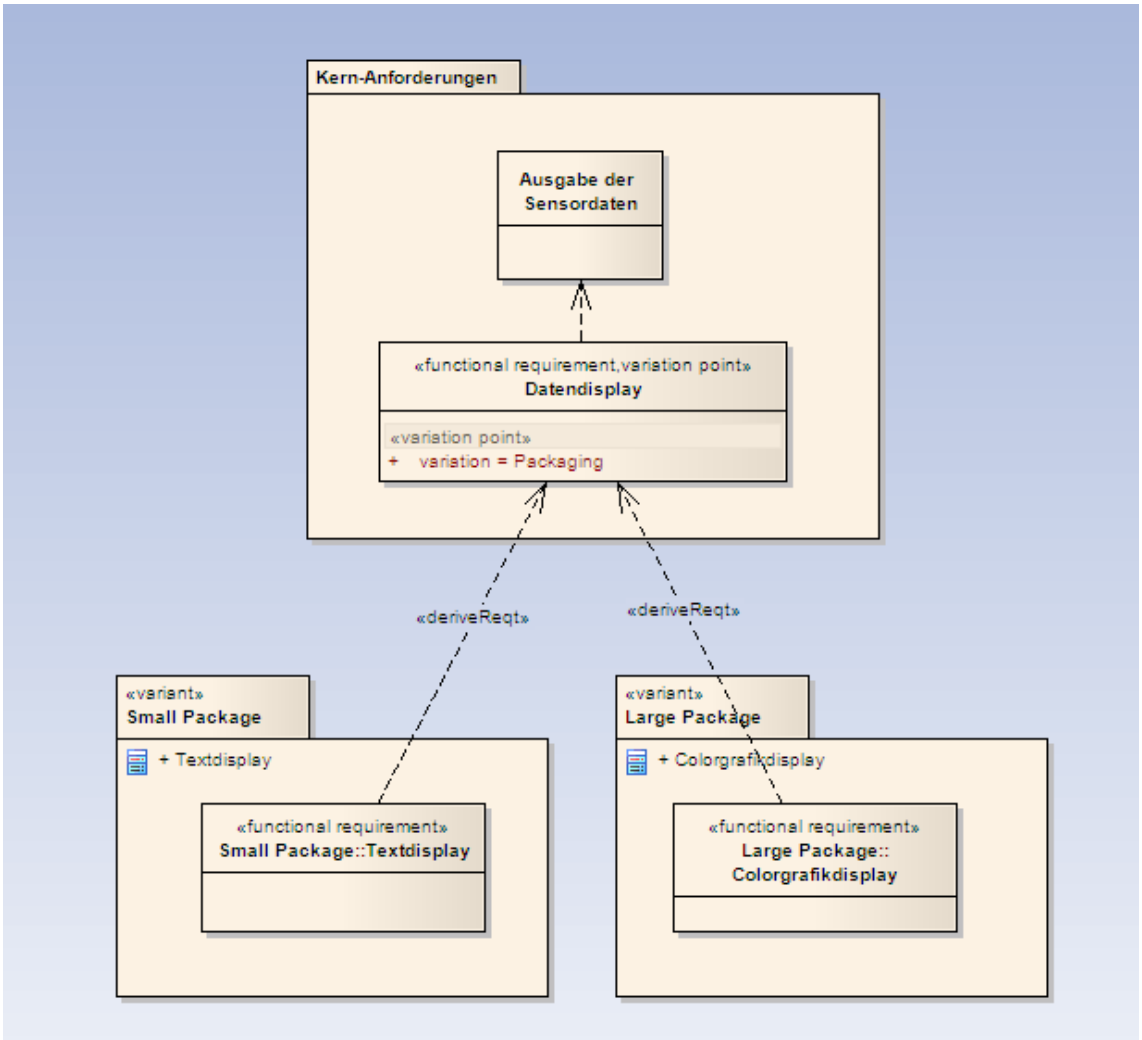


Abbildung 13. SysML Modell mit Variationspunkt.

Eine andere Herangehensweise ist das Konzept von „Conditional UML“. Hier werden in „normalen“ UML-Diagrammen Bedingungen in Form von Annotationen eingesetzt, um Abhängigkeiten zu Features der Produktlinie zu definieren (siehe Zustandsdiagramm in der Abbildung 14).

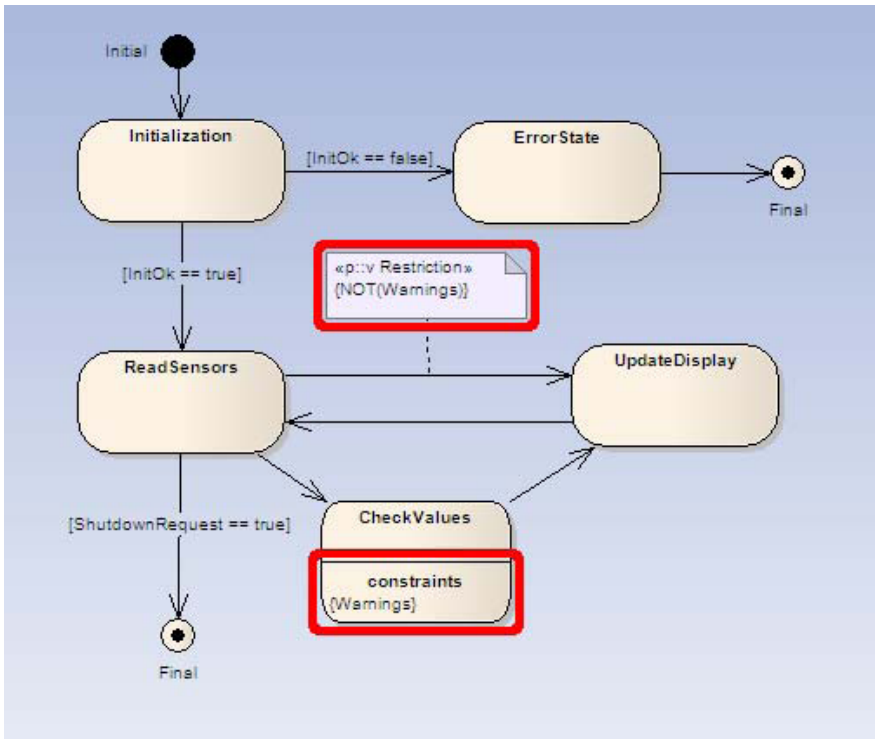


Abbildung 14. Zustandsdiagramm mit Bedingungen für Features.

Bei der Instanziierung anhand einer Variantenbeschreibung werden die annotierten Modellelemente abhängig von der evaluierten Bedingung bzw. den in der Variante (in)aktiven Features aus dem Modell entfernt (Abbildung 15). Dies entspricht dem „#ifdef“-Konzept einiger textueller Programmiersprachen.

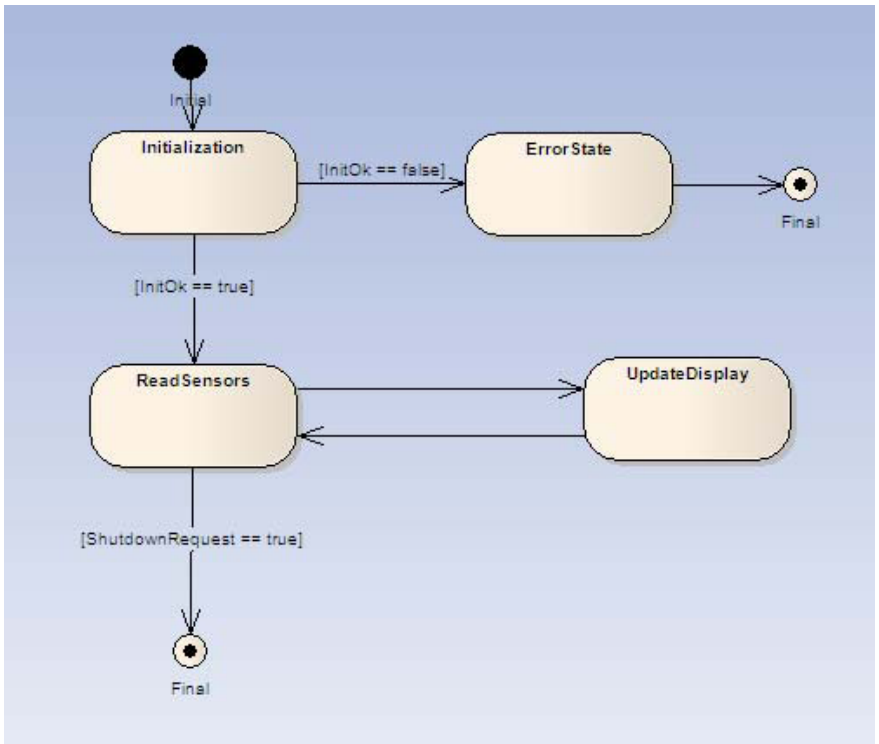


Abbildung 15. Zustandsdiagramm einer Variante.

Eine Einschränkung dieses Ansatzes ist, dass nur negative Variabilität modelliert werden kann. Ein Diagramm in einer Produktlinie enthält immer alle Varianten und damit alle Modellelemente, die evtl. mit Varianten-spezifischen Bedingungen versehen wurden. Für diese Modelle können jedoch nicht alle durch das UML-Tool geprüften semantischen Bedingungen angewendet werden. Hintergrund ist, dass z.B. in bestimmten Varianteninstanzen eines Zustandsdiagramms Zustände existieren können, die keine ein- oder abgehenden Transitionen haben. Im annotierten Modell würde das UML-Tool keine Fehler erkennen und Bedingungen müssen für jede relevante Variante separat geprüft werden.

6.3 Wann kann UML in Kombination mit SPL eingesetzt werden?

Wird UML (bzw. SysML) zur Beschreibung von Anforderungen komplexer System eingesetzt, ergibt sich oft der Bedarf Varianten effizient beschreiben zu können um den Wiederverwendungsgrad von Anforderungen zu erhöhen. Mit „SysML/RE“ existiert ein Profil, das UML um Funktionalitäten für das Anforderungsmanagement erweitert. Die Modelle können durch die beschriebenen Konzepte um Variantenbeschreibungen erweitert werden.

Ebenso ist in einem Umfeld, in dem bereits viel mit UML gearbeitet wird, oft sinnvoll, auch für die Entwicklung einer SPL (profiliertes) UML statt domänenspezifischer Modellierungssprachen zu verwenden. Dabei kann auf die vorhandenen Erfahrungen der Anwender mit den vorhandenen UML Werkzeugen aufgesetzt werden.

Ein weiterer Anwendungsfall ist die Dokumentation von Produktlinien. Sinnvoll ist dies vor allem dann, wenn UML-Diagramme aus anderen Artefakten generiert werden können. Für den zu dokumentierenden Teil des Systems sollte ein passender Diagrammtyp oder ein passendes Profil genutzt werden.

6.4 Fazit

Im Gegensatz zur Anwendung von domänenspezifischen Modellierungssprachen für die Produktlinienentwicklung ist aus aktueller Sicht das Thema UML nur eine Randerscheinung, unter anderem auch wegen fehlendem Toolsupport. Mit den jetzt vorhandenen Möglichkeiten zur Nutzung von UML in Kombination mit Produktlinienwerkzeugen wird die weitere Forschung und Entwicklung zum Thema „UML und Produktlinien“ zeigen, wie gut hier Synergien erzielt werden können.

6.5 Literaturverzeichnis

Beuche, Danilo & Schnepel, Enrico (2009). UML und Softwareproduktlinien. Produktlinien im Kontext 2009. Vortrag 10.03.2009

Stahl, Thomas & Weilkiens, Tim. SysML/DSL für Anforderungen mit Varianten. REConf 2009. Vortrag 25.05.2009

7 Merkmalmodellierung im Industrieeinsatz—ein Erfahrungsbericht

Dirk Janzen

7.1 Zusammenfassung

Dieser Erfahrungsbericht wird in chronologischer Abfolge über die Voraussetzungen, Bedingungen und Ereignisse während der Einführung einer Merkmalmodellierung im Anforderungsmanagement eines Haushaltsgeräteherstellers berichten. Vorgestellt wird die Ausgangssituation und der zeitliche Ablauf bis zur Einführung der Merkmalmodellierung in der Elektronikentwicklung. Erfolgsfaktoren und besondere Herausforderungen sollen als Diskussionsgrundlage dienen.

7.2 Einleitung

Die Einführung der Merkmalmodellierung stößt in der Industrie auf einige Vorbehalte und Herausforderungen. Dieser Beitrag bietet einen Erfahrungsbericht als Fallstudie und soll als Diskussionsgrundlage dienen.

Akteure der Fallstudie sind die Mitglieder der Elektronik-Entwicklungsabteilung des Herstellers, die ihre Geräte mehr als auswendig kennen, ein Mitglied der IT-Abteilung des Herstellers, der für die Auswahl des Requirements-Management (RM) Werkzeugs verantwortlich ist und ein unabhängiger Berater mit einigen Jahren Erfahrung im Anforderungsmanagement, in der Embedded-Systems- und in der Produktlinienentwicklung. Berichtet wird aus der Sicht des Beraters, Namen und Orte fehlen in der Fallstudie, da sie nichts Wesentliches beitragen würden.

7.3 Ausgangssituation

Ein Hersteller produziert eine Reihe von elektrischen Haushaltsgeräten. Die Geräte werden mit unterschiedlichen Funktionen sowie in unterschiedlichen Ausführungen, Länder- und Ausstattungsvarianten auf mehreren Kontinenten angeboten. Die Funktion der Geräte wird durch Mikrocontroller geregelt, gesteuert und überwacht. Die Elektronik der Geräte umfasst die Bedienelemente für die Benutzerinteraktion, einen oder mehrere Mikrocontroller, Sensoren zur Erfassung des Systemzustandes sowie Leistungsteile zur Steuerung der elektrischen Lasten. Die Elektronikkomponenten sowie die Software für die Mikrocontroller wird von Zulieferern entwickelt und hergestellt. Die Elektronikkomponenten sind für jedes Gerät auf ein oder mehrere Steuergeräte verteilt, die geräteintern durch ein Bussystem verbunden sind.

In unregelmäßigen zeitlichen Abständen wird, angefordert von der Marketing-Abteilung, eine neue Produktgeneration entwickelt. Eine Produktgeneration umfasst in der Regel etwa ein Dutzend Geräte, die in unterschiedlichen Varianten hergestellt werden. Für eine neue Produktgeneration müssen also mehrere Dutzend Geräte entwickelt werden.

Die Anforderungen an die Elektronikkomponenten werden von einer internen Elektronik-Entwicklungsabteilung erstellt und verwaltet. Die Beschreibungen der Regel- und Steueralgorithmen erfolgt in MS-Word-Dateien, teilweise in natürlicher Sprache, teilweise semiformal. Die Zuordnung von Funktionen und Komponenten zu einzelnen Geräten wird in MS-Excel ta-

bellarisch verwaltet. Die Versionierung der Dateien erfolgt durch Zeitstempel im Dateinamen. Ein Versionskontrollsystem wird nicht eingesetzt.

Durch die große Anzahl der Geräte, Komponenten und Funktionen hat sich mit der Zeit eine große Anzahl Dateien angesammelt, die nur schwer zu überblicken ist. Es fällt zunehmend schwerer, die für die Zulieferer der Steuergeräte relevanten Informationen zusammenzustellen.

7.4 Wünsche

Was ist die Kano-Methode?

Die Kano-Methode ist ein Verfahren, um den Einfluss von Produkteigenschaften auf die Zufriedenheit der Benutzer zu ermitteln. Dabei werden die einzelnen Produkteigenschaften in unterschiedliche Merkmalgruppen eingeteilt. Die Merkmalgruppen sind Basis-, Leistungs-, Begeisterungs-, Unerhebliche und Rückweisungsmerkmale. Die Methode wurde von Prof. Noriaki Kano 1978 vorgestellt.

Für die empirische Messung hat Prof. Kano eine bipolare Befragung entwickelt, bei der die Befragten für jede Produkteigenschaft zwei Fragen gegensätzlich formulierte Fragen beantworten sollen. Die Fragen sind einmal positiv formuliert (funktional) und einmal negativ formuliert (dysfunktional). Zur Beantwortung sind fünf Antwortmöglichkeiten von „Das würde mich sehr freuen.“ bis zu „Das würde mich sehr stören.“ vorgegeben.

Hinweis: Die „Merkmale“ der Kano-Methode haben nichts mit den „Merkmalen“ der Merkmalmodellierung in den folgenden Abschnitten zu tun.

Wie so oft soll ein neues Werkzeug die Rettung bringen. Die IT-Abteilung des Herstellers hat einen der führenden Werkzeughersteller eingeladen, seine Lösung zu präsentieren. Die Präsentation kann die Elektronik-Entwicklungsabteilung nicht überzeugen. Die Entwicklungsabteilung kann aus der Vorstellung des Werkzeugs keine Verbesserung der Situation erkennen. Ratlosigkeit macht sich breit.

Der Leiter der IT-Abteilung sucht nach einer Lösung und wird dabei auf den Berater aufmerksam. In längeren Telefonaten versucht letzterer von ersterem zu erfahren, wo denn das Problem liegt. Wie so oft sucht ersterer eine Lösung für ein Problem, dass er nicht in einer Art und Weise beschreiben kann, dass letzterer ihm eine Lösung anbieten kann. Man einigt sich auf ein Treffen, bei dem das Problem detailliert erläutert werden soll.

Das Treffen findet statt, der Berater besucht den Hersteller und schon beim Blättern in den ausliegenden Produktprospekten im Empfang wird offensichtlich, dass der Hersteller eine Produktlinie entwickelt. Während des Treffens stellt die Elektronik-Entwicklungsabteilung eine Unmenge großer MS-Word- und noch größerer MS-Excel-Dateien vor, mit der sie versucht, der Datenflut Herr zu werden. Dabei wird u. a. die Anforderung gestellt, dass Grafiken, die Boolsche Ausdrücke repräsentieren, die in den MS-Word-Dokumenten verschiedene Parameter der Regelalgorithmen verknüpfen, je nach Zulieferer, für den das Anforderungsdokument bestimmt ist, automatisch angepasst werden sollen. Auf Nachfrage bestätigt die Entwicklungsabteilung mehrfach die Wichtigkeit dieser Anforderung. Nun ist das automatische Erzeugen von parametrisierten Grafiken für den Berater kein unlösbares Problem, er wird aber dennoch von nicht leisen Zweifeln beschlichen, ob es das ist, was der Hersteller wirklich will. Fragen des Beraters, wo sich denn in diesen ganzen Dokumenten die offensichtliche Produktli-

nie und die offensichtlichen Varianten widerspiegeln, werden von der Entwicklungsabteilung mit ihrerseits fragenden Blicken erwidert.

Der Hersteller möchte ein Anforderungsmanagement-Werkzeug. Der Vorschlag, den Entwicklungsprozess zu beleuchten und gemeinsam zu untersuchen, wie denn die Produktlinie effizient entwickelt werden kann, stößt auf äußerste Zurückhaltung.

Des Menschen Wille ist sein Himmelreich: Man einigt sich darauf, einen Workshop zur Auswahl eines Anforderungsmanagement-Werkzeugs zu veranstalten.

7.5 Werkzeugauswahl

An dem Workshop nehmen Vertreter der IT- und der Elektronik-Entwicklungsabteilung des Herstellers teil. Der Berater stellt die Kano-Methode vor, mit der die Wichtigkeit der einzelnen Eigenschaften der RM-Werkzeuge bewertet werden soll. Die Methode findet gefallen.

7.6 Auswahlverfahren

Mittels der Kano-Methode werden etwa 120 Eigenschaften von RM-Werkzeugen von den Mitgliedern der Elektronik-Entwicklungsabteilung im Interview bewertet. Der Berater erläutert dabei die Bedeutung der Eigenschaften und notiert die Bewertung. Die Bewertung als Interview und nicht als Fragebogen durchzuführen erscheint hier wesentlich, da die Mitglieder der Entwicklungsabteilung nicht mit spezifischen Problemen und der Terminologie des RM vertraut sind und daher viele Beschreibungen der Werkzeugeigenschaften mißgedeutet hätten. Aus dem Umfang der notwendigen Erläuterungen darf der Wert von Bewertungen, die nur einen Fragebogen einsetzen, als nahezu unbrauchbar eingeschätzt werden. Ergebnis des Interviews ist eine Tabelle der Wichtigkeit der Eigenschaften.

Die Kombination dieser Wichtigkeit der einzelnen Werkzeugeigenschaften mit den Erfüllungsgraden der jeweiligen Werkzeuge ergibt eine Aussage darüber, in wie weit die einzelnen Werkzeuge die Anforderungen der Entwicklungsabteilung erfüllen. Daraus ergibt sich eine Rangfolge der betrachteten RM-Werkzeuge. Interessanterweise sind die Unterschiede zwischen den einzelnen Werkzeugen jedoch nur marginal. Nicht unbeachtet soll dabei bleiben, dass Anforderungen, die von der Entwicklungsabteilung für wichtig gehalten werden, von keinem bekannten RM-Werkzeug erfüllt werden können.

Werkzeuge für das Anforderungsmanagement in Produktlinien

Die Arbeitsgruppe „Requirements Management Tools for Product Line Engineering“ der Gesellschaft für Informatik hat sich mit Anforderungen an RM-Werkzeugen für Produktlinen beschäftigt und 2006 ihren Abschlussbericht vorgelegt [TvdM2006].

Ein Ergebnis der Arbeitsgruppe ist eine Liste mit etwa 120 Anforderungen an RM-Werkzeuge, wobei jede Anforderung mit einer relativen Wichtigkeit im Produktlinenkontext versehen ist.

Ein weiteres Ergebnis ist eine Bewertung von verschiedenen RM-Werkzeugen, in wie weit das jeweilige Werkzeug eine Anforderung aus der Anforderungsliste erfüllt. Bewertet wurden hierbei die Werkzeuge Telelogic DOORS 7.1; Borland Caliber RM V2005; IRqA sowie der Forschungsprototyp RequiLine der RWTH Aachen.

Die Bewertung der relativen Wichtigkeit und der Erfüllungsgrade wurde von Praktikern aus der Industrie durchgeführt, die mehrjährige Erfahrung in der Produktlinienentwicklung und mit dem Einsatz des jeweiligen Werkzeugs haben.

Damit war ein Werkzeug ausgewählt und somit der ursprüngliche Wunsch erfüllt. Und wie so oft wird dem Wünschenden nach der Erfüllung des Wunsches klar, dass die Erfüllung gar nicht dasjenige war, was er wirklich wollte.

7.7 Datenmodellierung

Das Werkzeug war nun ausgewählt, aber die Mitglieder der Entwicklungsabteilung können sich, verständlicherweise, gar nicht vorstellen, wie sie damit nun ihr eigentliches Problem lösen sollen.

Man einigt sich auf einen Workshop zur Datenmodellierung, in dem die bestehenden Artefakte analysiert, und daraus eine Datenstruktur gewonnen werden soll.

Mehrere Workshops finden statt, da ein Workshop nicht ausreichte, alle bestehenden Artefakte auch nur annähernd zu sichten. Aus den Workshops entsteht ein Datenmodell, das nicht nur die Anforderungen umfasst sondern eher ein Produktdatenmodell widerspiegelt. Aus den Workshops wird deutlich, dass das eigentliche Problem nicht so sehr im Anforderungsmanagement sondern in der Verwaltung der Produktvarianten, ihrer Teile und in der Zuordnung der Teile zu den Varianten liegt.

Ein umfangreiches Datenmodell ist entstanden, das aber nach Meinung der Beteiligten den Aufbau der Produkte sehr gut wiedergibt. Das Datenmodell umfasst etwa 23 Entitäten und 35 Relationen.

7.8 Zwei Jahre Pause

Die Entwicklungsabteilung wird durch neue Produktgenerationen derart in Anspruch genommen, dass das Projekt zwei Jahre nicht mehr weiterentwickelt wird.

7.9 Neuer Anlauf

Nach zwei Jahren ein neuer Anlauf. Der Hersteller möchte das Datenmodell aktualisieren und endlich ein RM-Werkzeug einführen.

Man einigt sich auf einen Workshop, in dem das bestehende Datenmodell den veränderten Anforderungsdokumenten angepasst werden soll. Aus den Anforderungsdokumenten sind mittlerweile die Grafiken für die Boolschen Ausdrücke verschwunden und durch semiformale Texte ersetzt. Das Datenmodell wird aktualisiert, danach folgt die Umsetzung in einem RM-Werkzeug.

7.10 Versuch einer Umsetzung

Die Umsetzung soll in demjenigen RM-Werkzeug erfolgen, das vor zwei Jahren ausgewählt wurde. Die Werkzeuge haben sich in zwei Jahren nicht so wesentlich verändert, als das dies nicht gerechtfertigt wäre.

Das Datenmodell ist schnell umgesetzt, wobei deutlich wird, dass das ausgewählte Werkzeug einige unschöne Beschränkungen auferlegt. Die Beschränkungen werden allerdings als nicht so schwerwiegend empfunden, als das sie eine Umsetzung unmöglich machen würden. Nach der Umsetzung des Datenmodells werden exemplarisch die Daten für einige Geräte erfasst, um Erfahrungen mit der Umsetzung in der täglichen Arbeit zu sammeln. Die Mitglieder der Entwicklungsabteilung sind von der ihrer Meinung nach eher umständlichen Arbeitsweise nicht sehr angetan. Sie können sich nicht recht vorstellen, wie das RM-Werkzeug ihre tägliche Arbeit wesentlich erleichtern kann.

In dieser Situation schlägt der Berater vor, den Ansatz der Merkmalmodellierung einmal kurz vorzustellen. Der Berater ist der Meinung, dass die Merkmalmodellierung gut geeignet ist, die Abhängigkeiten der Produkteigenschaften untereinander darzustellen und systematisch zu verwalten. Man einigt sich darauf, beim nächsten Treffen ein bis zwei Stunden auf dieses Thema zu verwenden.

Was ist Merkmalmodellierung?

Merkmalmodellierung ist ein Verfahren zur Domänenanalyse, bei der die Produktmerkmale und deren Beziehungen zueinander modelliert werden. Merkmale sind dabei Produkteigenschaften, die für eine definierte Nutzergruppe wesentlich sind.

Die Beziehungen zwischen den Merkmalen werden in Domänen- und Abhängigkeitsbeziehung unterteilt. Die Domänenbeziehungen und die Merkmale beschreiben den hierarchischen Aufbau der Produkte. Die Arten der Domänenbeziehungen legen die mögliche Variabilität fest. Folgende Domänenbeziehungen können modelliert werden:

- notwendig-Beziehnung
- oder-Beziehnung
- alternativ-Beziehnung
- optional-Beziehung

Abhängigkeitsbeziehungen schränken die Variantenbildung ein. Abhängigkeitsbeziehungen können zwischen je zwei beliebigen Merkmalenbestehen bestehen. Abhängigkeitsbeziehungen stellen entweder einen Ausschluss zwischen zwei Merkmalen oder eine Implikation von einem Merkmal zu einem anderen Merkmal dar.

7.11 Merkmalmodellierung

Der Berater stellt die Grundlagen der Merkmalmodellierung nach FODA (Feature Oriented Domain Analysis) [KCK1990], mit einigen Erweiterungen [TvdM2007], vor und demonstriert die Umsetzung in einem Add-On zu einem RM-Werkzeug. Die Präsentation wird überraschend positiv aufgenommen. Die Mitglieder der Elektronik-Entwicklung können sich vorstellen, mit dem Ansatz der Merkmalmodellierung einen Teil ihrer Probleme zu lösen. Entgegen den ursprünglichen Planungen wird der Rest des Treffens dazu verwendet, exemplarisch ein Merkmalmodell der eigenen Produkte zu erstellen. Am Ende des Treffens umfasst das Modell bereits mehr als 120 Merkmale.

Es ist an dieser Stelle bemerkenswert, wie sehr die Mitglieder der Entwicklung von der Demonstration der Umsetzung der eigenen Produkte im Merkmalmodell überzeugt wurden. Das Konzept der Domänen- und Abhängigkeitsbeziehung zeigt sich als gut verständlich und wird

schnell aktiv genutzt. Dass innerhalb eines knappen Tages ein Merkmalmodell mit mehr als 120 Merkmalen aufgebaut werden konnte, zeigt die Praktikabilität des Ansatzes.

Der Übergang von einer informalen textuellen Darstellung zu einer formalen Modellierung bietet Möglichkeiten zur Konsistenzprüfung des Modells. Diese Konsistenzprüfungen werden von den Mitgliedern der Entwicklungsabteilung sehr positiv bewertet. Konfigurationsfehler, die in den bisherigen MS-Excel-Tabellen verborgen blieben, lassen sich sehr schnell aufdecken und visualisieren. Diese Prüfmöglichkeiten bestehen sowohl für das gesamte Merkmalmodell als auch für die daraus abgeleiteten Varianten.

Die Einführung formaler Beschreibungsverfahren oder formaler Modelle trifft oft auf den Widerstand der Anwender, die sich in ihrer Ausdrucksfähigkeit gegenüber informalen Darstellungen eingeschränkt fühlen. Auch finden die notwendigen Schulungen nicht immer den Beifall der Anwender. Im hier vorgestellten Fall werden die wesentlichen Aspekte der Merkmalmodellierung nach einer kurzen Vorstellung (< 1 h) von den Mitgliedern der Entwicklungsabteilung verstanden und aktiv eingesetzt. Die relativ kleine Anzahl der Modellierungselemente trägt sicher zur schnellen Erlernbarkeit der Methode bei.

7.12 Offene Fragen

Ist der lange Weg zur Lösungsfindung ein notwendiger Lernprozess in der Industrie – oder gibt es Abkürzungen?

Wie kann die anfänglich sehr ablehnende Haltung gegenüber der Produktlinienentwicklung und der Merkmalmodellierung aufgelöst werden?

Wie lassen sich die Vorteile einer Produktlinienentwicklung vermitteln?

7.13 Literaturverzeichnis

[KCK1990] Kyo C. Kang, Sholom G. Cohen, James A. Hess, William E. Novak, A. Spencer Peterson, Feature-Oriented Domain Analysis (FODA) Feasibility Study, Software Engineering Institute; Carnegie Mellon University; Pittsburgh, Pennsylvania 1990 CMU/SEI-90-TR-21; ESD-90-TR-222

[TvdM2006] Danilo Beuche, Andreas Birk, Heinrich Dreier, Andreas Fleischmann, Heidi Galle, Gerald Heller, Dirk Janzen, Isabel John, Ramin Tavakoli Kolagari, Thomas von der Maßen (Ed.), Andreas WolframReport of the GI Work Group “Requirements Management Tools for Product Line Engineering” Aachener Informatik Berichte, Aachen 2006, ISSN 0935-3232

[TvdM2007] Thomas von der MaßenFeature-basierte Modellierung und Analyse von Variabilität in Produktlinienanforderungen, Shaker Verlag, Aachen 2007, ISBN 978-3-8322-6027-9

Anhang A: Vortragsfolien

Im folgenden werden die gezeigten Vortragsfolien in der Reihenfolge der Präsentation während der Tagung abgedruckt:

- 15 Jahre Software-Produktlinien—Einführung und aktueller Stand
  Andreas Birk, Klaus Schmid
  Ein Mini-Tutorial über Software-Produktlinien mit einem Überblick über die wesentlichen Konzepte, den Stand der Praxis und die aktuellen Themenschwerpunkte. Die Teilnehmer erfahren, was sie von Produktlinienentwicklung erwarten dürfen.

- Lösungsorientierte Software Produktlinienentwicklung in heterogenen Systemlandschaften
  Thomas von der Maßen, Andreas Wübbeke
  Die Entwicklung von Software Produktlinien (SPL) bietet die Möglichkeit durch gezielte Wiederverwendung ähnliche Produkte schneller und qualitativ hochwertiger zu erstellen. Neben der aus der Literatur bekannten Anwendung des SPL-Entwicklungsvorgehens im Bereich der klassischen Produktentwicklung (z. B. im Bereich der eingebetteter Systeme), kann das Entwicklungsparadigma auch bei kundenindividuellen Projekten, basierend auf heterogenen Systemlandschaften, seine bekannten Vorteile präsentieren. In diesem Beitrag sollen die besonderen Eigenschaften von SPL-Entwicklung für Lösungen in diesem Bereich am Beispiel von arvato services aufgezeigt und daraus Anforderungen für die Adaption des Entwicklungsprozesses für diese spezielle Form der SPL Entwicklung abgeleitet werden.

- Requirements-Engineering für Produktlinien in der Automobilelektronik
  Matthias Weber (carmeq) und Mark-Oliver Reiser
  Dieser Vortrag stellt eine Reihe von Problemen vor, auf die bei der Einführung von Produktlinienansätzen in der Automobilelektronik geachtet werden muss. Dabei wird eine Erweiterung der Featuremodellierung motiviert und näher erläutert, mit Hilfe von der die im Lebenszyklus einer Funktions- bzw. Steuergeräteentwicklung auftretende Komplexität der Variabilitätsmodellierung insbesondere beim Requirements Engineering besser handhabbar gemacht werden kann. Der Vortrag präsentiert und diskutiert das Konzept anhand einer Fallstudie. Darüberhinaus wird eine DOORS-Erweiterung vorgestellt, mit der das Konzept in neue bzw. laufende Projekte eingeführt werden kann.

- A model-driven product line approach for HMI development
  Marc Born, Omar Ekine, Jon Ashley
  The aim of the initiative introduced in this paper is to better control the creation process of infotainment HMI software for automotive OEM customers of DENSO. This is achieved by a combination of a Model Driven Development strategy together with a Software Product Line (SPL) approach. Model Driven Development offers more precise modelling especially of screens and screen flows, early simulation, model validation, automatic transformation and code generation. That speeds up the development process and increases the quality. The SPL approach enables systematic reuse of existing code and the HMI design (including graphics) among multiple projects for different customers and brands. A product derivation from a product line model based on a feature model is realized to significantly reduce the delivery time to the customer once a few projects have been completed. The combination of these two powerful approaches is a

significant challenge and the result will radically change the way HMI software development is executed by DENSO.

- UML und Softwareproduktlinien – Wie, Weshalb und Wann?
  Danilo Beuche, Enrico Schnepel
  Dieser Vortrag geht die 3 W's in einer eher, pragmatischen Weise an. Ausgehend von einem Ansatz, wie UML-Modelle zur Modellierung von Softwareproduktlinien aussehen können, wird über die Schwierigkeiten und Vorteile einer Nutzung von UML zur Modellierung von SPL diskutiert. Hierbei wird insbesondere Bezug auf "echte" domänenspezifische Sprachen und "modellose" Ansätze genommen. Keiner der beiden Sprecher ist ein UML-Evangelist, aber genausowenig soll im Vortrag UML verteufelt werden. Auch deshalb sind die Teilnehmer eingeladen, den Vortrag mit eigenen Erfahrungen zu bereichern und die Vortragenden herauszufordern!

A.1 15 Jahre Software-Produktlinien—Einführung und aktueller Stand

Software.Process.Management
software processes that fit your business needs

Software Systems Engineering

Stiftung Universität Hildesheim 2003

15 Jahre Software-Produktlinien: Einführung und aktueller Stand

Mini-Tutorial

Dr. Andreas Birk (Software.Process.Management), Prof. Dr. Klaus Schmid (Universität Hildesheim)
PIK2009, Berlin
25. Mai 2009

PIK '09
Produktlinien im Kontext

Agenda

What is a Software Product Line (SPL)?

History of the SPL discipline

Concepts & approaches to SPL development

SPL case study

Key questions of SPL research & practice

Copyright © 2009, Dr. Andreas Birk und Prof. Dr. Klaus Schmid

Terminology: Product Family

We consider a set of programs to constitute a **family**, whenever it is worthwhile to study programs from the set by …
first studying the common properties of the set and
then determining the special properties of the individual family members.

(David L. Parnas, 1976)

Copyright © 2009, Dr. Andreas Birk und Prof. Dr. Klaus Schmid

Terminology: Product Line

A **software product line (SPL)** is …
a set of software-intensive systems
that share a common, managed set of features
satisfying the specific needs of a particular market segment or mission and
that are developed from a common set of core assets in a prescribed way.

(Clements and Northrop, SEI, 2002)

Copyright © 2009, Dr. Andreas Birk und Prof. Dr. Klaus Schmid

Terminology: Product Line

***Marketed* (Software) Product Line:**
A set of products that are marketed together as sharing a common set of concepts or features.

***Engineered* (Software) Product Line:**
A set of products that are engineered together so as to share major parts of their implementation.

Copyright © 2009, Dr. Andreas Birk und Prof. Dr. Klaus Schmid

Reuse

Build something

from existing or pre-produced items

Copyright © 2009, Dr. Andreas Birk und Prof. Dr. Klaus Schmid

Advantages of Reuse

Quality

Productivity

Time

Copyright © 2009, Dr. Andreas Birk und Prof. Dr. Klaus Schmid

Reuse Items

Test Cases

Architecture

Estimates

Models

Code

Requirements

Subsystems

Documentation

Rules

Copyright © 2009, Dr. Andreas Birk und Prof. Dr. Klaus Schmid

But…

..most traditional software reuse approaches actually failed?

So what is new?

opportunistic
vs.
systematic

explicit differentiation in:

- development ***for reuse***
- development ***with reuse***

Copyright © 2009, Dr. Andreas Birk und Prof. Dr. Klaus Schmid

Agenda

What is a Software Product Line (SPL)?

History of the SPL discipline

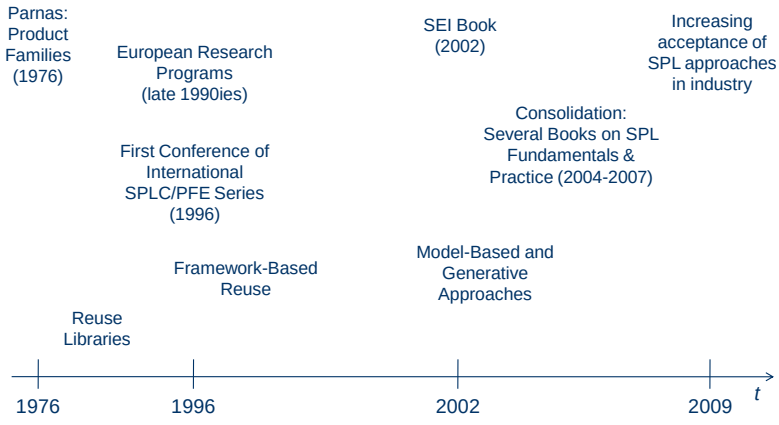
Concepts & approaches to SPL development

SPL case study

Key questions of SPL research & practice

Copyright © 2009, Dr. Andreas Birk und Prof. Dr. Klaus Schmid

History of the SPL discipline



Copyright © 2009, Dr. Andreas Birk und Prof. Dr. Klaus Schmid

Agenda

What is a Software Product Line (SPL)?

History of the SPL discipline

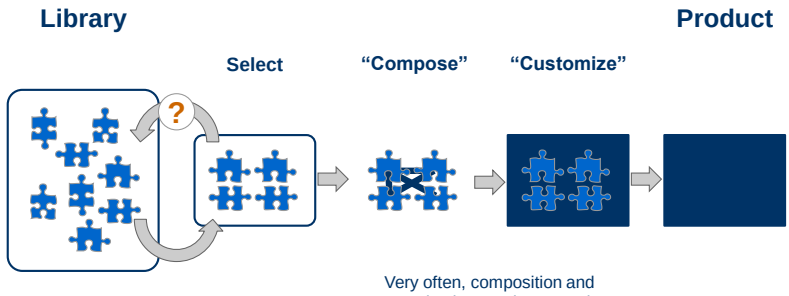
Concepts & approaches to SPL development

SPL case study

Key questions of SPL research & practice

Copyright © 2009, Dr. Andreas Birk und Prof. Dr. Klaus Schmid

Reuse: The Old Way



Very often, composition and customization require extensive modification of „reused“ items. This reduces or even outweighs the expected reuse benefits significantly.

Copyright © 2009, Dr. Andreas Birk und Prof. Dr. Klaus Schmid

Reuse, the Old Way: Shortcomings

Library did not contain needed items

Selection tedious

Mostly adaptation is needed

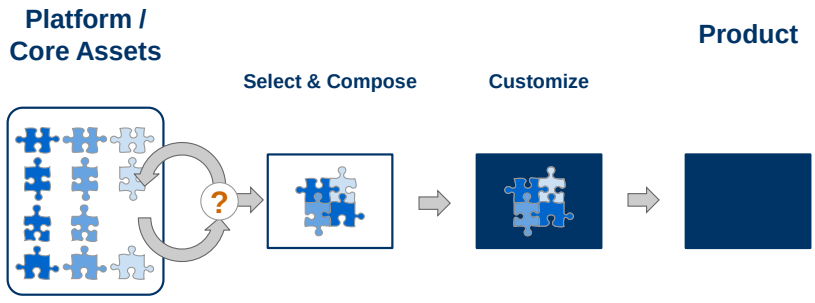
Too little composition

Too much customization

People tend to build from scratch

Copyright © 2009, Dr. Andreas Birk und Prof. Dr. Klaus Schmid

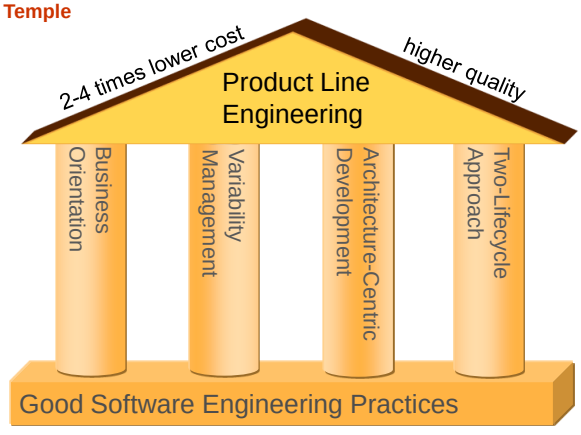
Reuse: Product Lines



25. Mai 2009 Copyright © 2009, Dr. Andreas Birk und Prof. Dr. Klaus Schmid 15

Product Lines: Technical Challenges

PLE Temple



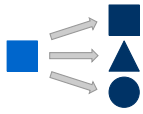
25. Mai 2009 Copyright © 2009, Dr. Andreas Birk und Prof. Dr. Klaus Schmid 16

SPL Type: Business Rationale

Product Suites

- Existing applications are integrated over time
- Customer uses more than one product from SPL
- **SPL motivation: Reduced cost of ownership**
- Example: Microsoft Office

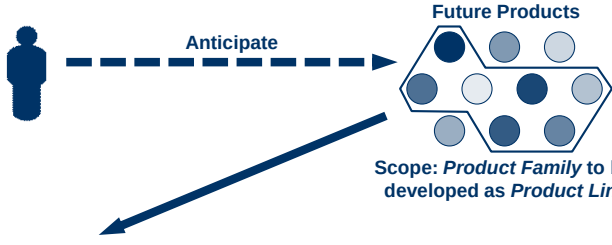
Customized Products



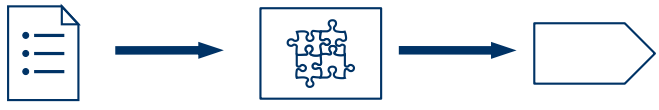
- New applications are instantiated from the framework
- Customer uses only one product from SPL
- **SPL motivation: Delivery efficiency**
- Examples: Cummins, Bosch, Danfoss

Copyright © 2009, Dr. Andreas Birk und Prof. Dr. Klaus Schmid

Setting up a Product Line

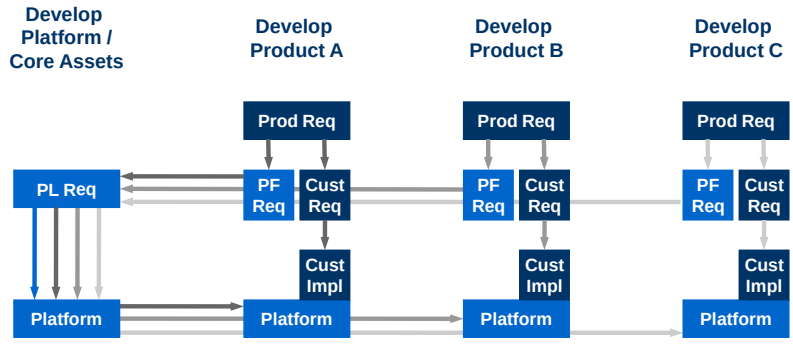


Define Requirements & Domain Architecture → **Develop Platform / Core Assets** → **Establish SPL Processes & Practices**



Copyright © 2009, Dr. Andreas Birk und Prof. Dr. Klaus Schmid

Operating a Product Line



Legend: PL = Product Line; PF = Platform; Prod = Product; Cust = Custom; Req = Requirement; Impl = Implementation

Copyright © 2009, Dr. Andreas Birk und Prof. Dr. Klaus Schmid

Challenges of SPL: Requirements

Anticipate Requirements

Describe Commonality & Variability

Maintain large requirements base over very long lifetime

Split requirements into platform and variant

Derive variant- & version-specific requirements views

Copyright © 2009, Dr. Andreas Birk und Prof. Dr. Klaus Schmid

Variability Management

Core Idea of Product Line Engineering

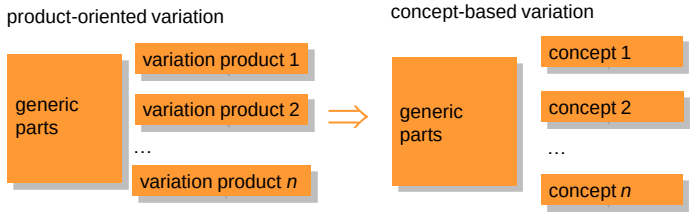
- Product 1
- Product 2
- Product 3

Core idea: Similarity of Products =
Commonality + (reg.) Variability + Product-specific parts

develop once | make selectable | single development

Copyright © 2009, Dr. Andreas Birk und Prof. Dr. Klaus Schmid

Variability Management



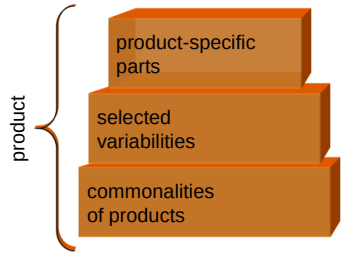
Product Definition
product 1: concept 1,2, ,4,5, ,…
product 2: concept 1, ,3,4, ,6,…

Copyright © 2009, Dr. Andreas Birk und Prof. Dr. Klaus Schmid

Variability Management

Definitions:

- **Product-specific:**
  an artifact (or characteristic) that is specific to a product
- **Variability:**
  an artifact (or characteristic) that exists in some, but not all, products of the product line
- **Commonality:**
  an artifact (or characteristic) that exists in all products of the product line



Copyright © 2009, Dr. Andreas Birk und Prof. Dr. Klaus Schmid

SPL Types: Product Instantiation

Constructive

- Customization based on framework, library, or shared components

Compositional

- Fixed building blocks with parametrization
- Parametrization during build process or during installation; never exposed to end user

Generative

- Model-driven development with artefact generation

Combination

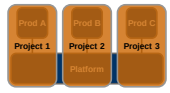
- For instance, generation of shared components with subsequent customization

Copyright © 2009, Dr. Andreas Birk und Prof. Dr. Klaus Schmid

Mapping Organization and Architecture

Separate platform team

Platform under control of product engineering projects



Mixed

Phased combination

Copyright © 2009, Dr. Andreas Birk und Prof. Dr. Klaus Schmid

Product Lines: Technical Challenges

Anticipate scope & requirements

Define & maintain domain architecture

Develop & maintain platform / core assets

Establish mature software engineering practices

Copyright © 2009, Dr. Andreas Birk und Prof. Dr. Klaus Schmid

Challenges of SPL: Organization

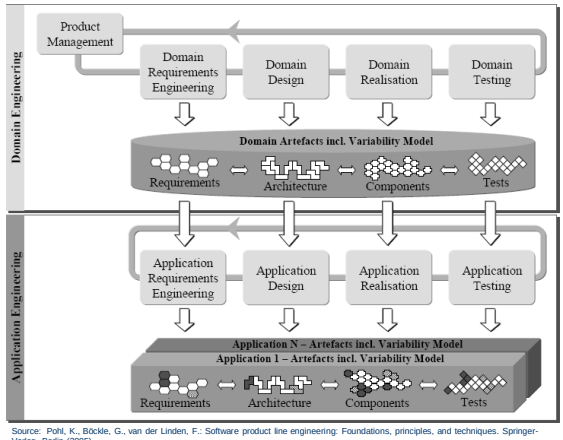
Very many stakeholders

Diverging interests

Changes of requirements & products

Copyright © 2009, Dr. Andreas Birk und Prof. Dr. Klaus Schmid

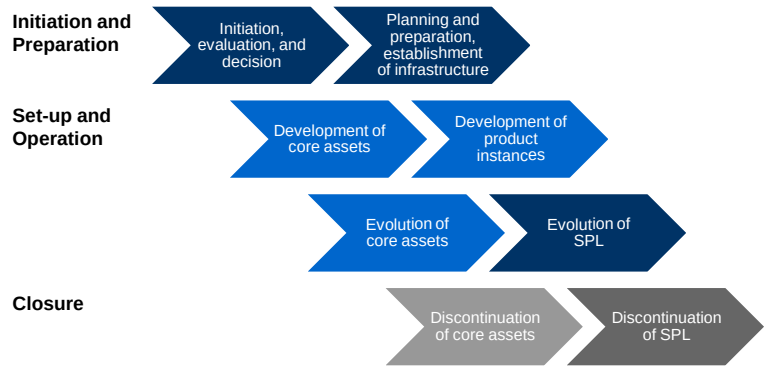
Product Line Framework



Source: Pohl, K., Böckle, G., van der Linden, F.: Software product line engineering: Foundations, principles, and techniques. Springer-Verlag, Berlin (2005).

Copyright © 2009, Dr. Andreas Birk und Prof. Dr. Klaus Schmid

Process: Overall SPL Lifecycle



Different life cycle phases require different organizational structures and processes.

25. Mai 2009 Copyright © 2009, Dr. Andreas Birk und Prof. Dr. Klaus Schmid 30

Agenda

What is a Software Product Line (SPL)?

History of the SPL discipline

Concepts & approaches to SPL development

SPL case study

Key questions of SPL research & practice

25. Mai 2009 Copyright © 2009, Dr. Andreas Birk und Prof. Dr. Klaus Schmid 31

Case Study: Small

Market Maker Software AG

Small organization (< 25 developers total; ~50 people total)
Company started in 1984 with a focus on developing stock market software
Since then development of various products (product line)

- Desktop / PC-based systems
- Delphi-/C++-based development

Large range of customers (from consumer-market to large banks)

New product line (i* product line) started in 1999

- Web-based products
- Java-development
- Development started from scratch

25. Mai 2009 Copyright © 2009, Dr. Andreas Birk und Prof. Dr. Klaus Schmid 32

Case Study: Small

Market Maker: Product Line Engineering Results

New product line highly successful

– Running for more ~ten years now
– Large number of products

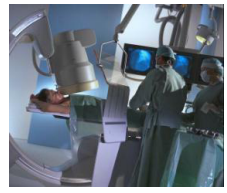
\+ Reduction of Time-to-Market: 2-4
\+ Break-Even: after about five products
\+ Reduction of maintenance cost: ~60%
\+ Reduced cost of quality (reliability in the field)
– Increase of issue resolution time

25. Mai 2009 Copyright © 2009, Dr. Andreas Birk und Prof. Dr. Klaus Schmid 33

Case Study: Large

Philips Medical Systems

- Large organization (> 1000 developers)
- Focus on medical imaging systems, e.g.,
  - Acquisition: X-Ray,
    Magnetic Resonance Imaging (MRI)
    …
  - Image processing, viewing & storing
- World-wide development
- Development structured around product groups
  - cross product group reuse
  - core functionality: imaging products (storing, retrieving, processing)



Copyright © 2009, Dr. Andreas Birk und Prof. Dr. Klaus Schmid

Case Study: Large

Product Line Engineering Results

- Successful transition
- Integration of additional companies

+ Reduction of Effort: 2-4
+ Time-to-Market reduction: ~50%
+ Product defect density: <50%
+ Reduction of maintenance cost: ~60%
+ Common look and Feel
+ Better product planning and use of roadmaps

Copyright © 2009, Dr. Andreas Birk und Prof. Dr. Klaus Schmid

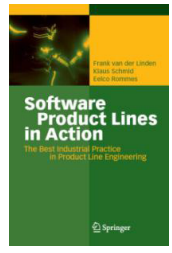
Some Literature

Further Material

- Product line engineering from a practitioner perspective
- Families Evaluation Framework
- Many industrial case studies!

`http://www.spl-book.net`

Linden, Schmid, Rommes
Product Lines in Action
Springer, 2007



Gleichheit in Vielfalt –
Produktlinien die Zukunft der industriellen Softwareentwicklung

iX-Archiv, 5/2008, Seite 110

25. Mai 2009 Copyright © 2009, Dr. Andreas Birk und Prof. Dr. Klaus Schmid 36

Agenda

What is a Software Product Line (SPL)?

History of the SPL discipline

Concepts & approaches to SPL development

SPL case study

Key questions of SPL research & practice

25. Mai 2009 Copyright © 2009, Dr. Andreas Birk und Prof. Dr. Klaus Schmid 37

Key Questions of SPL Research & Practice

How to model complex real-world variability?

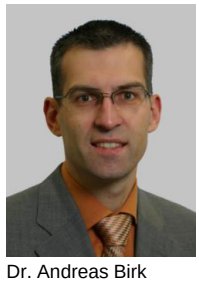
How to integrate SPL with other methodologies?

How to create awareness of SPL demands?

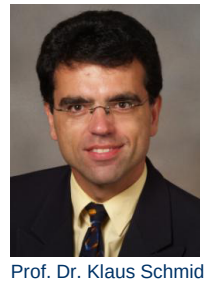
How to switch over from non-SPL development to SPL development?

Copyright © 2009, Dr. Andreas Birk und Prof. Dr. Klaus Schmid

Kontakt



Dr. Andreas Birk
Software.Process.Management
info@swpm.de
http://www.swpm.de



Prof. Dr. Klaus Schmid
Universität Hildesheim
schmid@sse.uni-hildesheim.de
http://www.sse.uni-hildesheim.de

Copyright © 2009, Dr. Andreas Birk und Prof. Dr. Klaus Schmid

A.2 Lösungsorientierte Software Produktlinienentwicklung in heterogenen Systemlandschaften

Lösungsorientierte Software Produktlinienentwicklung in heterogenen Systemlandschaften

Andreas Wübbeke
Dr. Thomas von der Maßen

Workshop Produktlinien im Kontext 2009
25.05.2009

Agenda

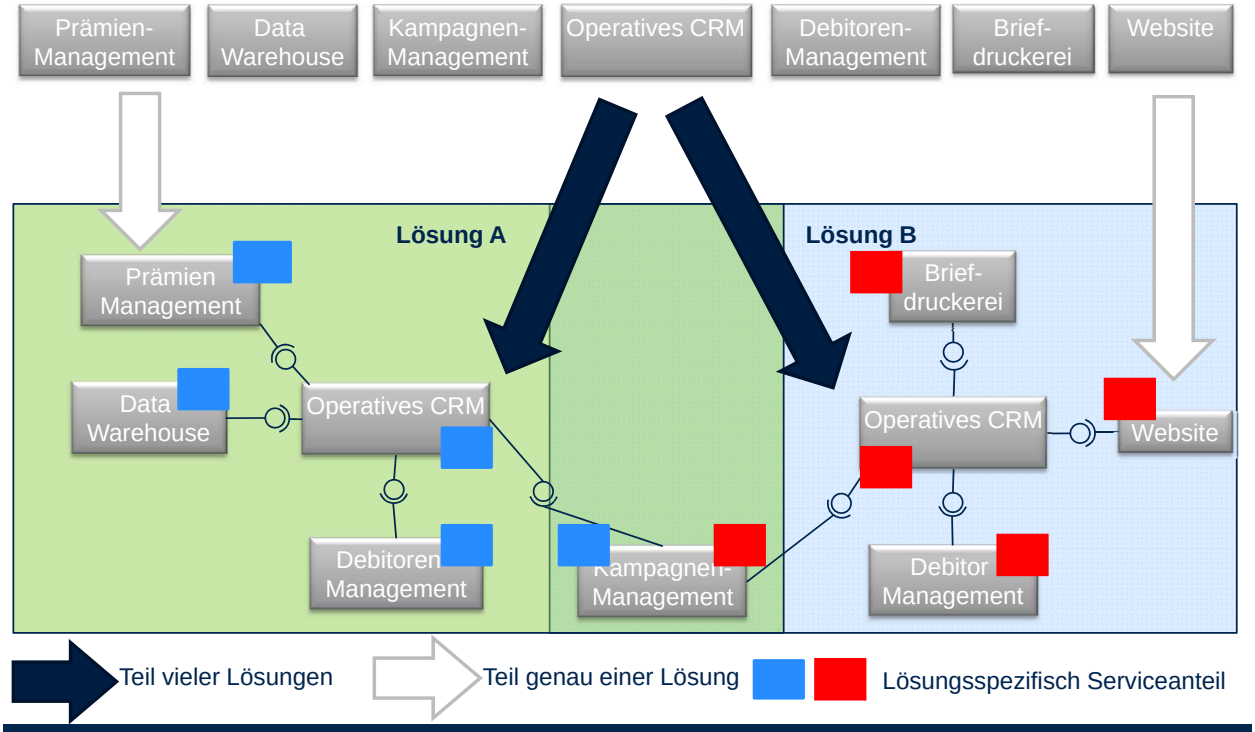
- Ausgangssituation:
  - arvato Lösungsarchitektur und Lösungsentwicklungsprozess
  - Rahmenbedingungen für arvato SPL Entwicklungsarchitektur
- Ziel:
  - Lösungsorientiertes SPL Entwicklungsvorgehen
  - Anforderungen an das SPL Entwicklungsvorgehen
- Zusammenfassung und Ausblick

arvato Services

- Domänen:
  - Kundenbindung
  - Loyalty
  - Fulfilment
  - …
- Entwicklung ist projektorientiert
- Ergebnis ist eine kundenindividuellen IT-Lösung
- arvato übernimmt auch Betrieb und Support der Lösung

arvato Lösungsarchitektur (1/2)

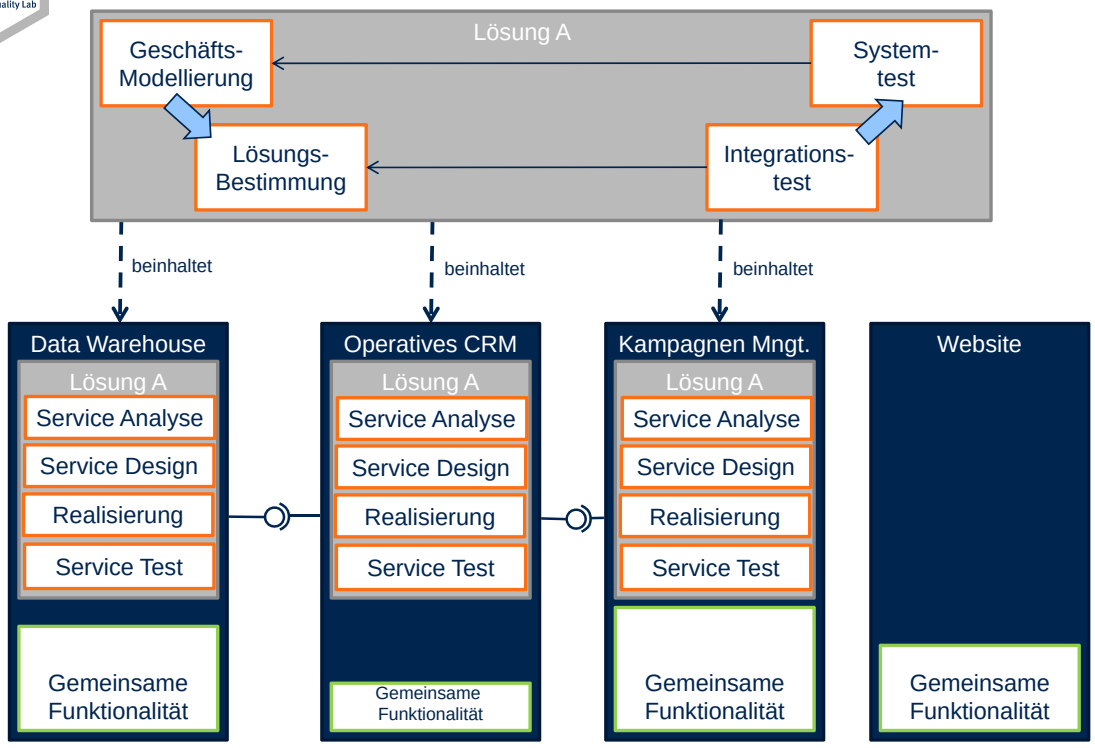
Service Systeme



arvato Lösungsarchitektur (2/2)

- Service Systeme
  - adressieren einen funktional geschlossenen Bereich (z. B. Finanzbuchhaltung)
  - basieren auf unterschiedlichen technologischen Umsetzungen (z. B. Java oder C#)
  - sind heterogen (von Individual- bis Standardsoftware)
  - sind typischerweise organisatorische Einheiten

arvato Lösungs-Entwicklungsprozess



Wiederverwendungspotentiale

- Es existieren viele Gemeinsamkeiten bei der Modellierung der Lösungsgeschäfte (Lösungsebene)
- Service Systeme werden in verschiedenen Lösungen in immer ähnlicher Weise benutzt (Service System-Ebene)

- Bisher geringe Berücksichtigung von Gemeinsamkeiten in
  - der Geschäftsmodellierung
  - Service Systemen
  - Schnittstellen

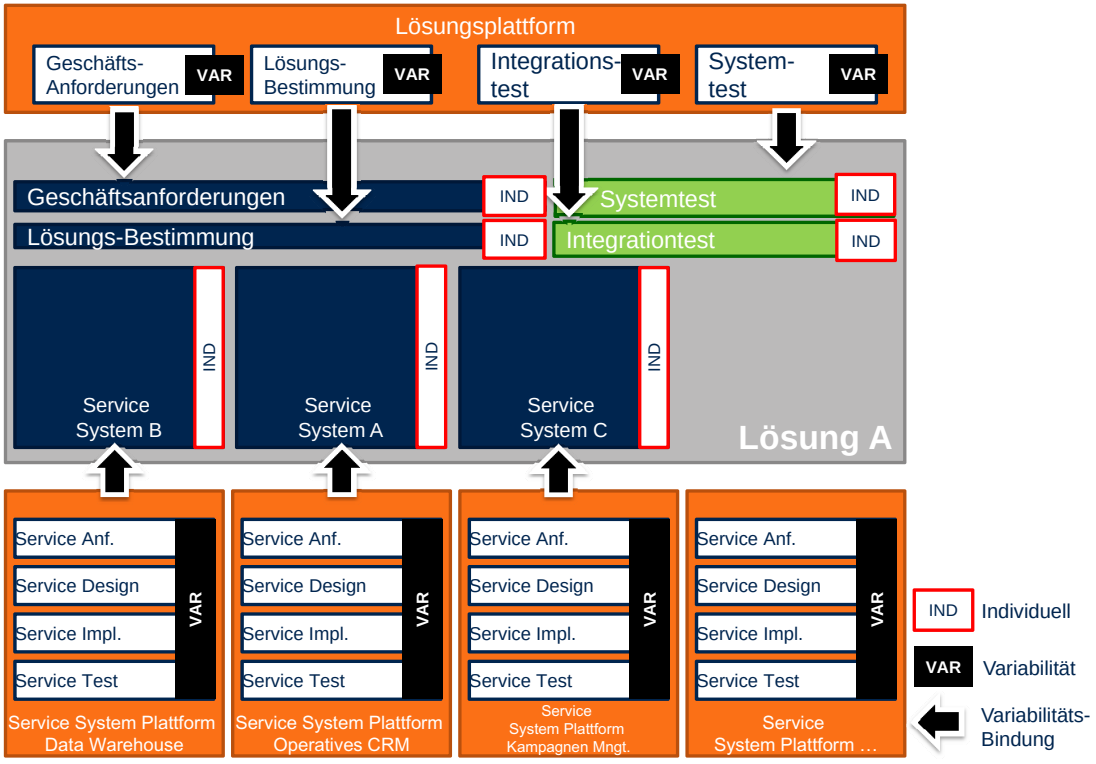
Rahmenbedingungen für SPL

- Berücksichtigung der Gemeinsamkeiten auf Lösungsebene
- Berücksichtigung der Gemeinsamkeiten auf Service System-Ebene
- Berücksichtigung des Aufbaus der Lösungsebene
  - Geschäftsanalyse
  - Qualitätssicherung
  - mehrere Service System Ebenen
- Berücksichtigung der organisatorischen Trennung von Service Systemen

Agenda

- Ausgangssituation:
  - arvato Lösungsarchitektur und Lösungsentwicklungsprozess
  - Rahmenbedingungen für arvato SPL Entwicklungsarchitektur
- Ziel:
  - Lösungsorientiertes SPL Entwicklungsvorgehen
  - Anforderungen an das SPL Entwicklungsvorgehen
- Zusammenfassung und Ausblick

Lösungsorientiertes SPL Entwicklungsvorgehen



Multiplattform SPL Entwicklung

- Organisation der Gemeinsamkeiten für die Ebenen
  - *Lösung* und
  - *Service Systeme*

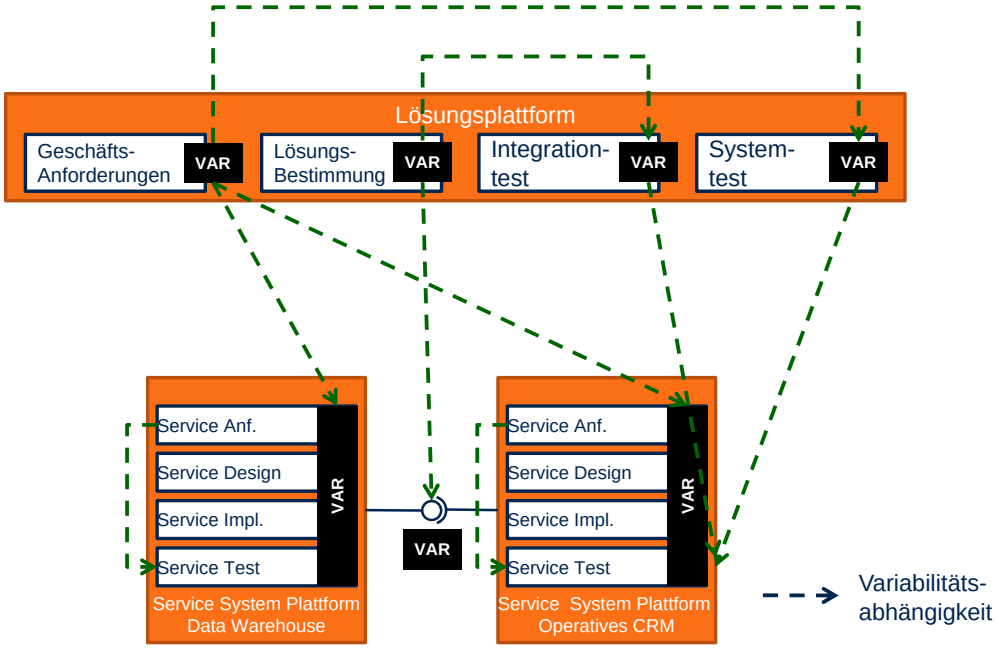
  mit Hilfe von eigenen Plattformen
- Jedes Service System hat eine eigene Plattform, da
  - es eine organisatorische Einheit darstellt,
  - einen funktionalen Bereich kapselt und
  - auf Basis einer bestimmten Technologie entwickelt wird

Anforderungen an das SPL Entwicklungsvorgehen

- Umgang mit Abhängigkeiten zwischen den Plattformen hinsichtlich
  - Variabilität
  - Organisation

  → Verfolgbarkeit der Variabilität auf Plattformebene und zwischen Plattformen
- Ableitung von konkreten Lösungen auf Basis konkreter Service Systeme

Abhängigkeiten zwischen Lösungs- und Service-Plattform(en)



Zusammenfassung

- Definition von Rahmenbedingungen für arvato SPL Entwicklungsvorgehen
  - Berücksichtigung von organisatorischen, technologischen und domänenspezifischen Besonderheiten

→SPL Architektur mit Mehrplattformstrategie

- Definition von Anforderungen an die skizzierte Lösung

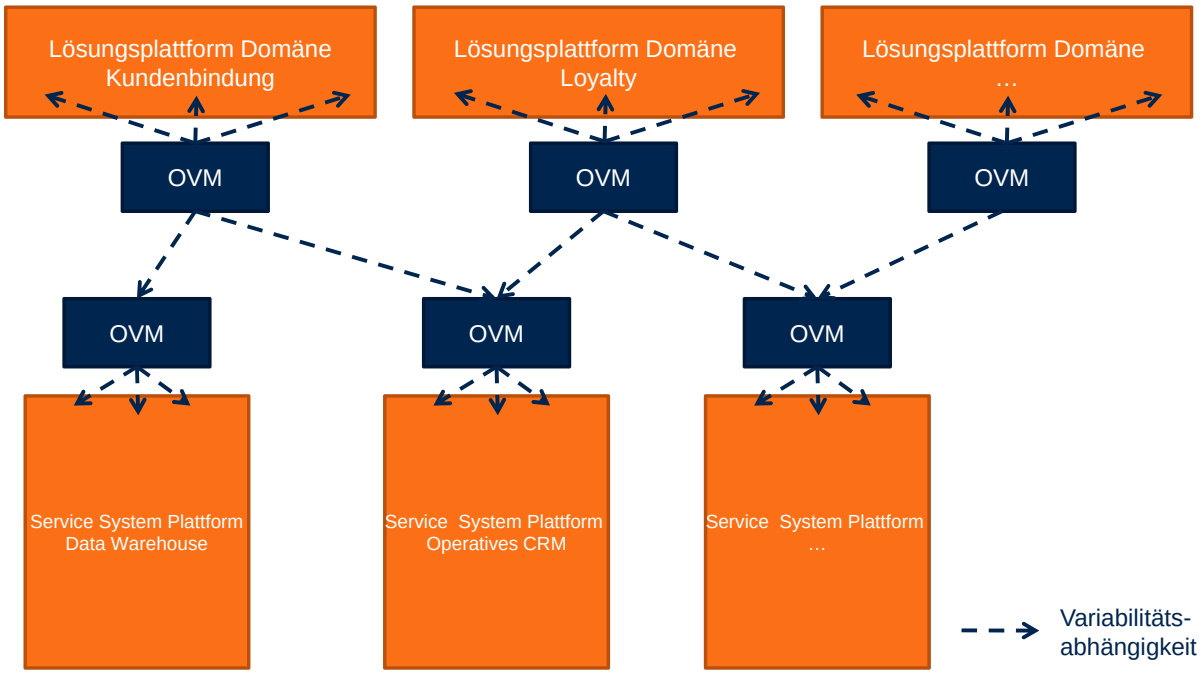
Ausblick

- Definition passender Variabilitätsmodellierungsvorgehen für unterschiedliche Typen von Service Systemen (Individualsoftware vs. Standardsystem)
- Auswahl eines Variabilitätsmodells mit Berücksichtigung von Variabilitätsabhängigkeiten innerhalb
  - einer Plattform
  - zwischen Plattformen

Vielen Dank für Ihre Aufmerksamkeit

Fragen?

Mehrplattformarchitektur für verschiedene Domänen



A.3 Requirements-Engineering für Produktlinien in der Automobilelektronik

carmeq
software & systems

Requirements-Engineering für Produktlinien in der Automobilelektronik

Matthias Weber (Carmeq GmbH)
Mark-Oliver Reiser (TU Berlin)

PIK, Berlin, 25.5.09

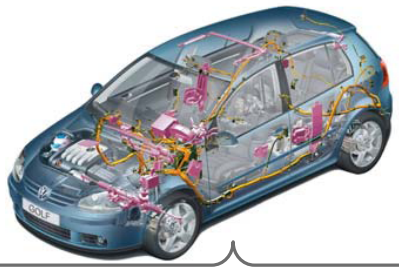
carmeq
software & systems

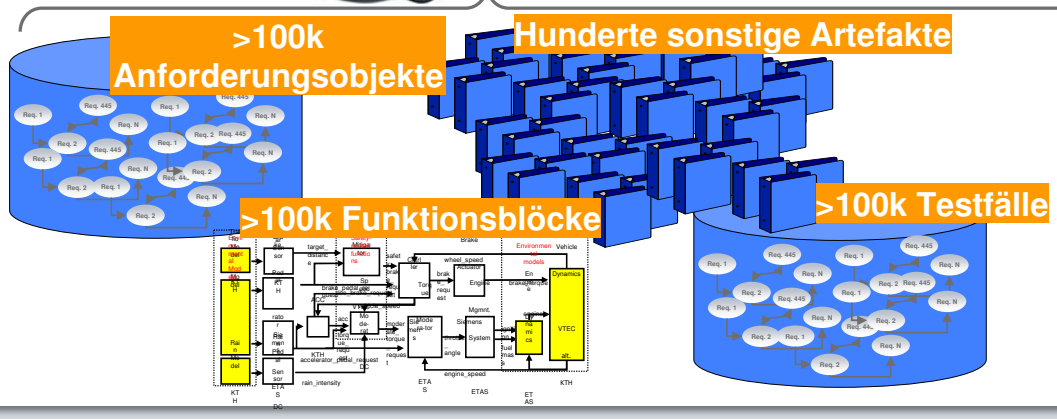
Agenda

- Motivation
- Grundlagen
- Variabilität in Anforderungen
- Strukturierung von Featuremodellen
- Subscoping von Anforderungen

carmeq
software & systems

Entwicklungsobjekte einer Elektronikplattform





carmeq
software & systems

Wiederverwendung im Automobilbereich

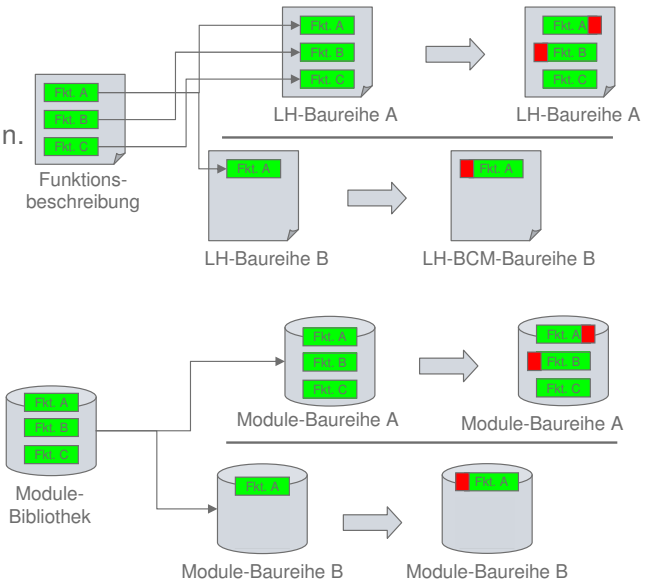
- **hohe Variabilität**
  - Baureihen
  - verschiedene Märkte
  - Ausstattungsvarianten / Optionen
  - gesetzliche Vorgaben (geographisch, zeitlich)
  - wechselnde Verfügbarkeit von Bauteilen (geographisch, zeitlich)
  - Technologiewechsel
  - Kostendruck
- **sehr heterogene Laufzeitumgebung (Hardware & Software)**
- **sehr lange Produktlebenszyklen (ca. 5+20 Jahre)**
- **divergierende Lebenszyklen**

→ ***Wiederverwendung schwierig aber sehr wichtig !!***

Wiederverwendung für Lastenhefte und Modelle

Ist-Situation:

- Funktionsorientierte Entwicklung, d.h. funktionale Anforderungen werden in techn. Bauteilfunktionen strukturiert
- Funktionen im Lastenheft besitzen den gleichen Namen und gleiche Dekomposition.
- Zwischenzeitlich treten projektspezifische Änderungen auf.
- Differenzen müssen nachträglich berücksichtigt werden

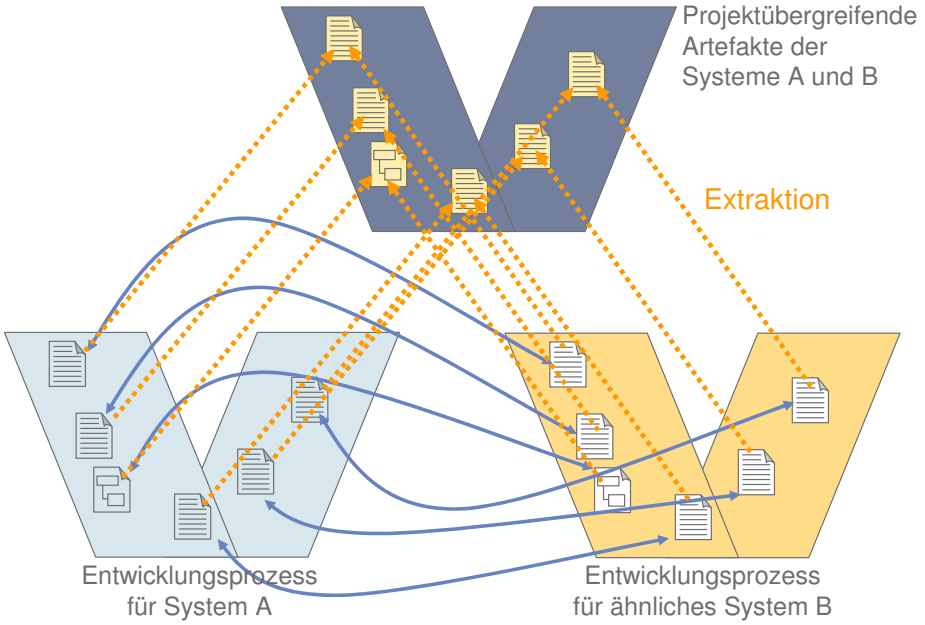


Dankeschön an H.Kleinwechter für die Folie!

5

- M. Weber, M.-O. Reiser, PIK'09, Berlin -

Integration paralleler Entwicklungen / Einführung der Produktlinienentwicklung



6

- M. Weber, M.-O. Reiser, PIK'09, Berlin -

carmeq
software & systems

Agenda

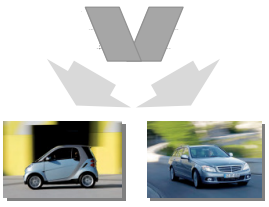
- Motivation
- **Grundlagen**
- Variabilität in Anforderungen
- Strukturierung von Featuremodellen
- Subscoping von Anforderungen

carmeq
software & systems

Grundlagen

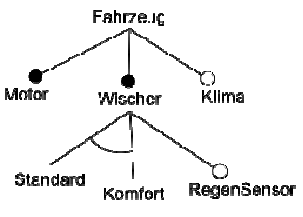
Produktlinienorientierte Entwicklung

- gemeinsame Entwicklung ähnlicher Produkte
- Koordination von Variabilität & Wiederverwendung



Featuremodellierung

- Variabilitätsmodellierung
  = Modellierung von Gemeinsamkeiten & Unterschieden
- graphisch, hierarchisch (idR Baumstruktur)

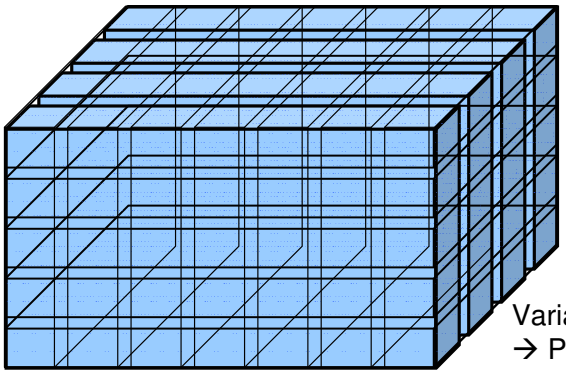


- M. Weber, M.-O. Reiser, PIK09, Berlin -

8

carmeq
software & systems

Mehrdimensionale PL-Dekomposition



Artefakte
→ Artefaktlinien

Variabilität
→ Produkt-**Teil**linien

Teilsysteme/-komponenten
→ Teilproduktlinien

carmeq
software & systems

Agenda

- Motivation
- Grundlagen
- **Variabilität in Anforderungen**
- Strukturierung von Featuremodellen
- Subscoping von Anforderungen

carmeq software & systems

Darstellung von Variabilität in Anforderungen

Main Specification Text	*Additional Attributes*	*Selection Condition*
The car shall have a light control system based on ….	status = mature, rationale = „compatibility to system …“	**ECE AND (Standard-Light-System OR …)**
<light control system specification object 1>		**selection condition object 1**
<light control system specification object 2>		**selection condition object 2**
<light control system specification object 3>		**selection condition object 3**
<light control system specification object 3.1>		**selection condition object 3.1**

Nachteil:
- keine explizite Darstellung von Variationspunkten.

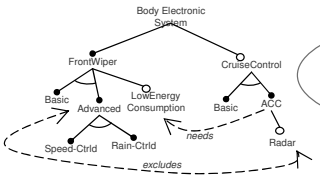
Vorteil:
- Reihenfolgeunabhängigkeit bezüglich Variabilität
- Werkzeugunterstützung (Filter-/Sichtenkonzepte etc.)

- M. Weber, M.-O. Reiser, PIK'09, Berlin -

11

carmeq software & systems

Konfiguration von Anforderungen



(Domain-Specific) Requirements Feature Model

{Radar, SpeedControl, RainControl, …}

Selected Features

	ID	Condition
of the gas pedal or d its active control.	38	SpeedControl && RainControl
able to actively	39	RainControl
actively control	40	Radar & (Country = US)
controlling the he driver.	42	Radar & ~(Country = US)

Requirements Configuration

	ID	Condition
of the gas pedal or d its active control.	38	true
able to actively /h.	39	true
o actively control	40	Country = US
/ controlling the the driver.	42	~(Country = US)

Cross-Platform Requirements Library

Common Requirements
Variable Requirements
Selection conditions
(logical combinations of features)

(Partially) Configured Requirements

Common Requirements
Instantiated Req. variabilities
Remaining Req. variabilities

- M. Weber, M.-O. Reiser, PIK'09, Berlin -

12

carmeq
software & systems

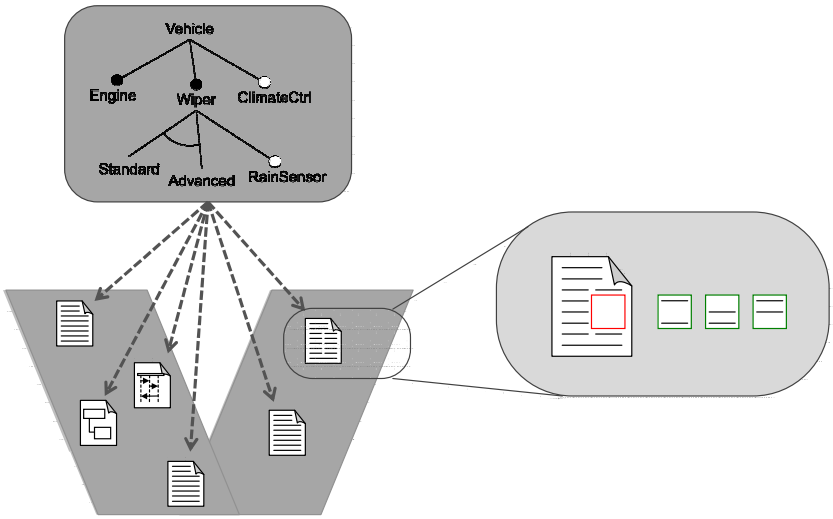
Agenda

- Motivation
- Grundlagen
- Variabilität in Anforderungen
- **Strukturierung von Featuremodellen**
- Subscoping von Anforderungen

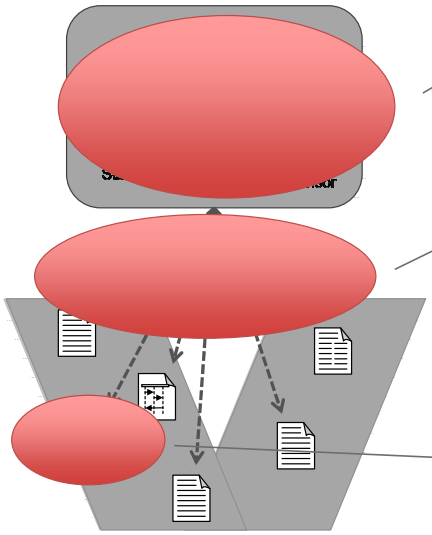
carmeq
software & systems

Strukturierung von Featuremodellen

Bislang:



Problemstellungen



ungeeignet für Endkunden
- Komplexität (>10.000 Features)
- technische Perspektive

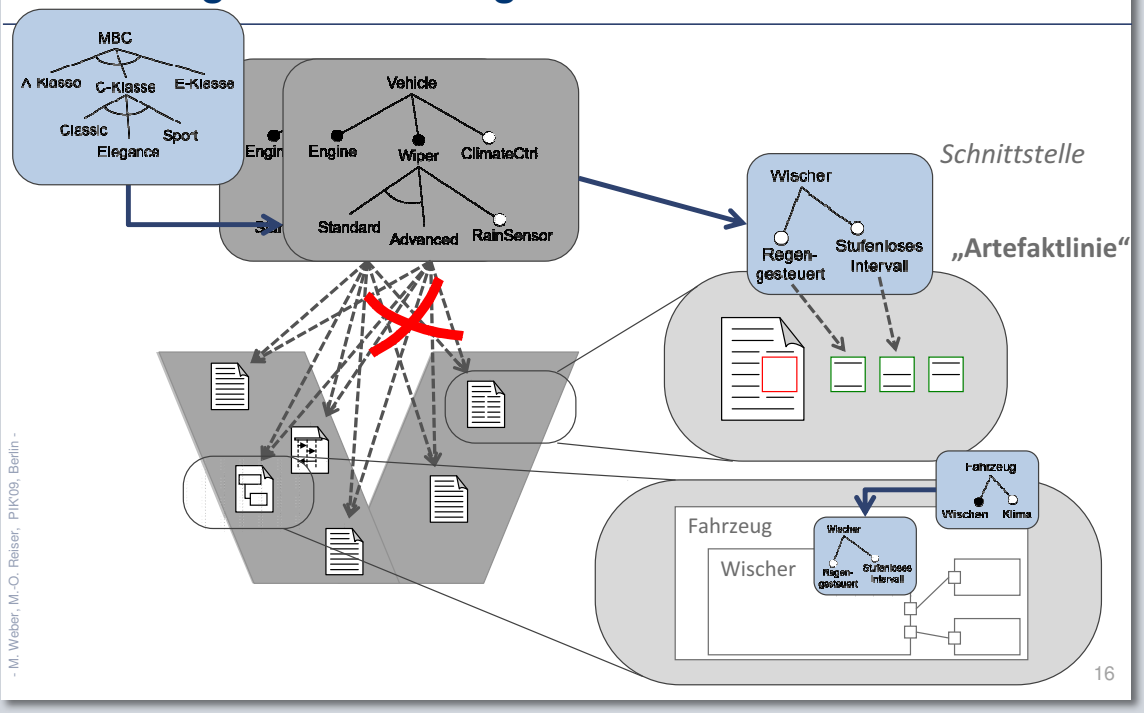
direkte Anbindung zu starr
- Vielzahl von Akteuren
- divergierende Lebenszyklen und Geltungsbereiche
- Heterogenität (Methoden,Tools,...)

Anbindung je Artefakt zu grobgranular
- manche Artefakte sehr umfangreich
- obige Probleme innerhalb eines Artefakts !

- M. Weber, M.-O. Reiser, PIK'09, Berlin -

15

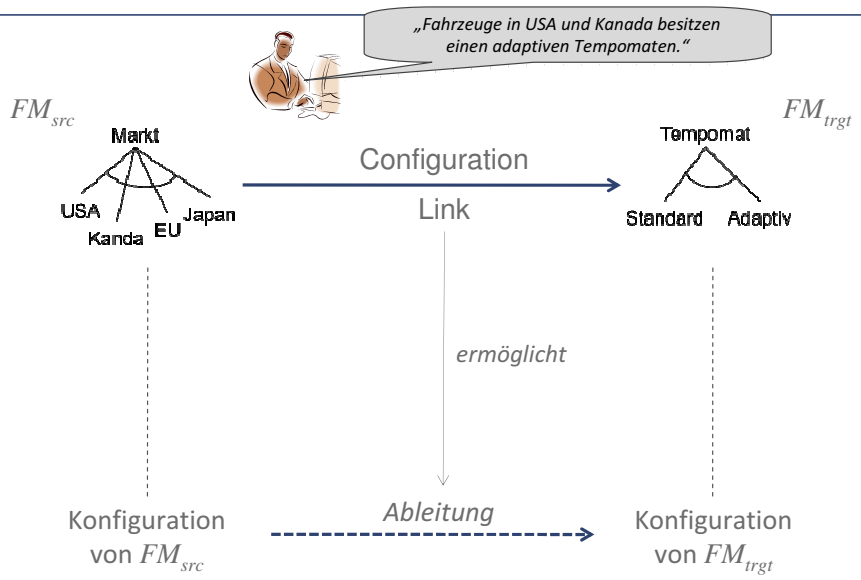
Lösungsansatz: Configuration Links



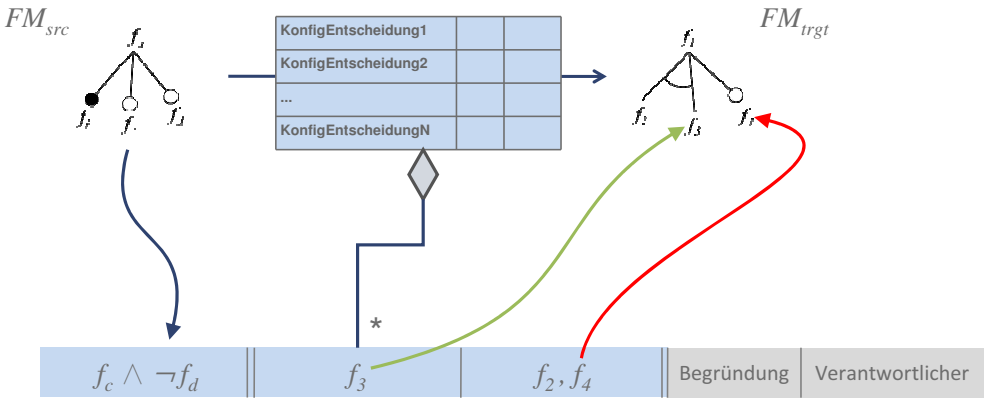
- M. Weber, M.-O. Reiser, PIK'09, Berlin -

16

Lösungsansatz: Configuration Links



Formalisierung – Configuration Links



Konfigurationsentscheidung

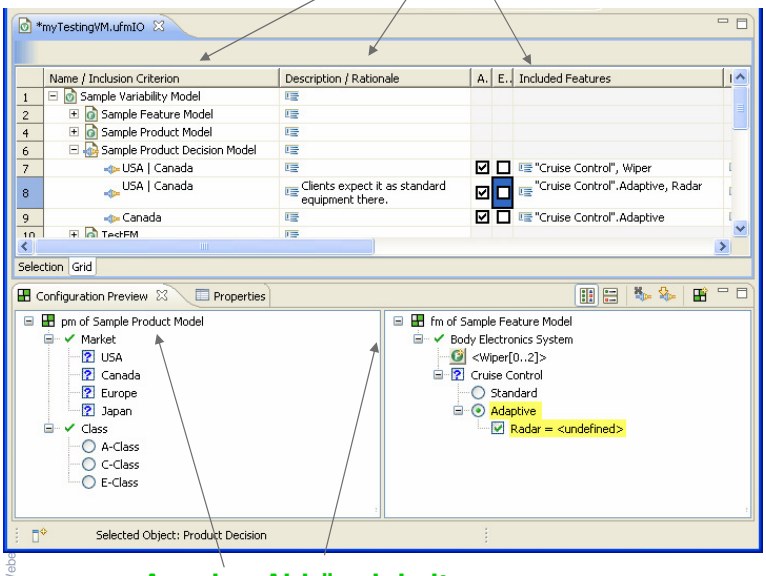
erlaubt:

- autom. Ableitung von Konfigurationen
- Analyse von Konfigurationsentscheidungen (z.B. Widersprüche)

CVM-Framework:
(Compositional Variability Management)

carmeq software & systems

Editieren von Configuration Links



Anzeige Abhängigkeiten

eSafety – Fallstudie
- im EU-Projekt ATESST
- untersucht wurden (u.a.)
  - Configuration Links
  - insbes. in Komponenten-hierarchien
- ABS- und EMS-System

- M. Weber

19

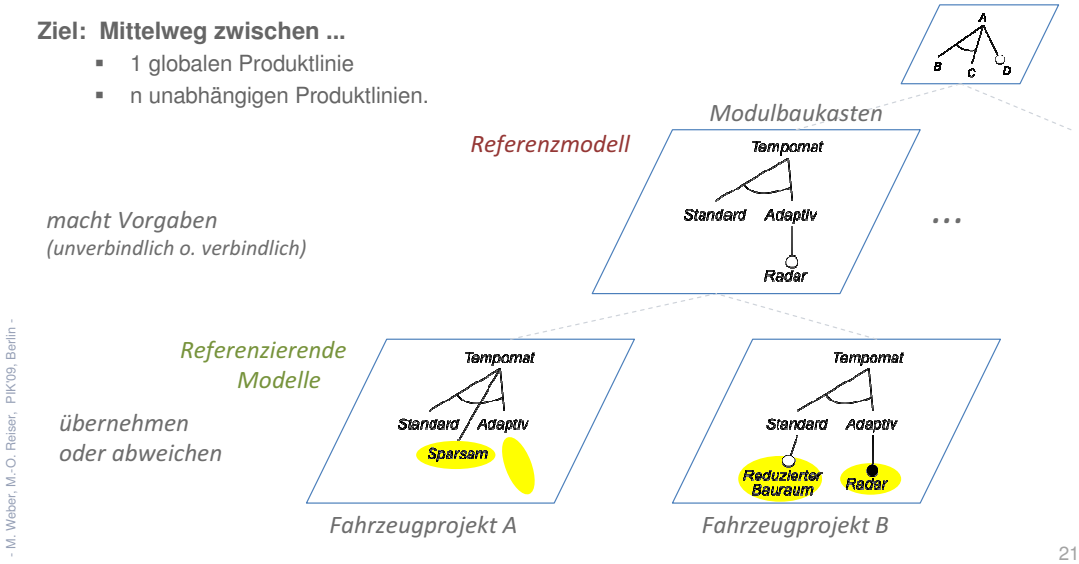
carmeq software & systems

Agenda

- Motivation
- Grundlagen
- Variabilität in Anforderungen
- Strukturierung von Featuremodellen
- **Subscoping von Anforderungen**

Lösungsansatz zur Wiederverwendung: Multi-Level Konzept

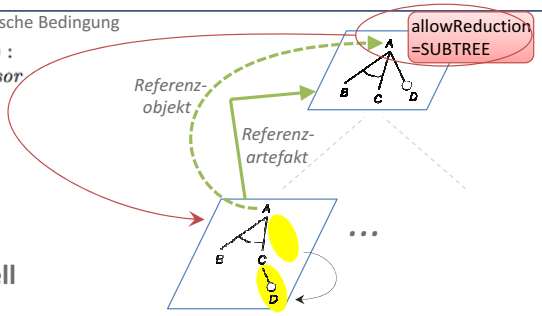
Ziel: Mittelweg zwischen ...
- 1 globalen Produktlinie
- n unabhängigen Produktlinien.



Formalisierung – Multi-Level Beziehung

Logische Bedingung

$$\forall f'_{ref} \in childrenOf(f_{ref}) : \exists f' \in childrenOf(f) : f'_{ref} = RefF(f') \vee (f'_{ref}, RefF(f')) \in Predecessor$$



1. **Referenzbeziehungen**
2. **Abweichungen**
3. **Abweichungsberechtigungen**
4. **Constraints auf referenzierendes Modell**

⇒ Unterscheidung:
- legitime Abweichungen
- illegitime Abweichungen

⇒ Eigenschaft der Konformität:

Identität	
keine Abweichungen	**Konformität**
nur legitime Abweichungen	
illegitime Abweichungen	

erlaubt:
- Statistiken über Abweichungen
- autom. Konformitätsprüfung

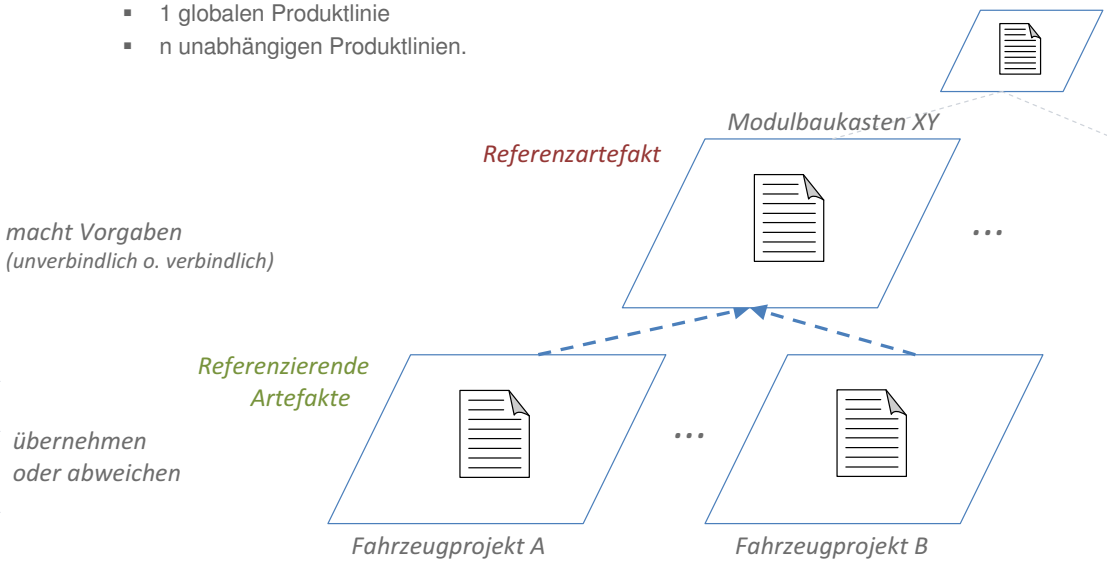
Erweiterbar auf bidirektionale Synchronisation

carmeq software & systems

Verallgemeinerung Multi-Level Konzept

Ziel: Mittelweg zwischen ...

- 1 globalen Produktlinie
- n unabhängigen Produktlinien.



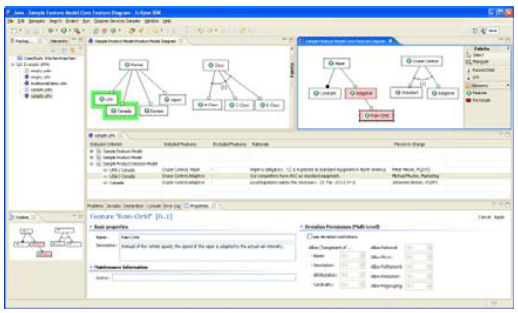
- M. Weber, M.-O. Reiser, PIK'09, Berlin -

23

carmeq software & systems

Werkzeugunterstützung Multi-Level Konzept

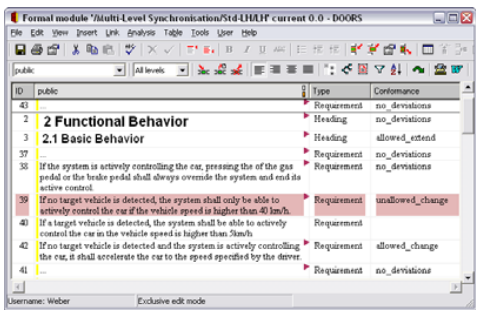
CVM-Framework (Eclipse Plugin)



Multi-Level Featuremodelle:

- Anzeige von Abweichungen
- autom. Konformitätsprüfung

MuLe (Doors-Erweiterung)



Multi-Level Konzept für Anforderungen:

- Anzeige von Abweichungen
- autom. Konformitätsprüfung
- bidirektionale Übernahme von Abweichungen (Top-down und Bottom-up Reuse)

- M. Weber, M.-O. Reiser, PIK'09, Berlin -

24

Plattformübergreifende Referenzbibliothek

carmeq software & systems

MuLe – Grundkonzepte

ID		Type	Deviation Permissions	Attribute Deviation
12	...	Information		
13	...	Requirement		
4	**3 Functional Behavior - General**	Heading	attributes changeable	Object Heading
5	3.1 Basic Behavior	Heading	extendable	
7	If the system is actively controlling the car above 2km/h, pressing of the gas pedal or the brake pedal shall always override the system and end its active control.	Requirement		
8	The system shall only be able to actively control the car if the vehicle spped is higher than 40 km/h.	Requirement		
9	If the system is actively controlling the car, the system shall accelerate the car to the speed specified by the driver.	Requirement	attributes changeable	Object Text
15	The system shall only be active if the driver seat-belt has been fastened.	Requirement	attributes changeable	
18	3.2 Advanced Behavior	Heading		

Erlaubte Abweichungen für referenzierende Specs

Status Konformität zur Referenz

Status Änderungsreview

ID		Conformance	Type	Conformance Detail	Review Status	Last Review Date	New Changes
12	2.1	no reference defined	TBD		not reviewed		
13	...	no reference defined	Requirement		not reviewed		
4	**3 Functional Behavior - P1**	legitimate attribute change conformance	Heading	**Legitimate Deviation:** Object-Heading Reference Value: Functional Behavior - General	reviewed	10.02.2009 21:34:44	
5	3.1 Basic Behavior	illegitimate reduction illegitimate children removal legitimate extension	Heading	**Missing Reference Children**(Standard-Spec-IDs): 15 **Illegitimately Removed Reference Children**: 15, . **Additional Children:** 10	not reviewed		
7	If the system is actively controlling the car, pressing of the gas pedal or the brake pedal shall always override the system and end its active control.	illegitimate attribute change	Requirement	**Illegitimate Deviation:** Object Text Differences (extensions underlined, deletions scored through): ***Text is different:*** If the system is actively controlling the car ~~above 2km/h~~, pressing of the gas pedal or the brake pedal shall always override the system and end its active control.	need to revisit	10.02.2009 21:34:49	**Changes wrt. Reference Module** *"Standard-Spec"* **since Last Review!** Attribute *"Object Text"* changed.
8	The system shall only be able to actively control the car if the vehicle spped is higher than 40 km/h.	conformance	Requirement		reviewed	10.02.2009 21:34:51	
10	If a target vewhicle is detected, the system shall be able to actively control the car if the vehicle speed is higher than 5 km/h.	no reference defined	Requirement		reviewed	09.02.2009 17:59:12	

Plattformspezifische referenzierende Spezifikation

- M. Weber, M.-O. Reiser, PIK'09, Berlin -

MuLe – Status Konformität zu den referenzierenden Spezifikationen

Plattformübergreifende Referenzbibliothek:

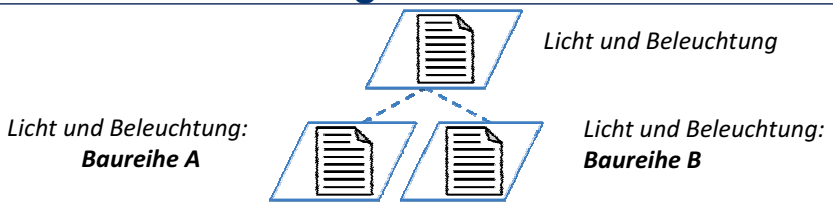
Konformität Spec4

Konformität Spec3

ID	e.g. part of a functional requirements library	Spec 4: Conformance	Spec 4: Review Status	Spec 3: Conformance	Spec 3: Review Status
15	**2 Funktionsbibliothek**	no referencing		conformance legitimate deviations	not reviewed
11	2.1 Wipe-Wash Functions	no referencing		illegitimate deviations	not reviewed
1	2.1.1 Front Wiper	conformance	reviewed	illegitimate deviations	need to revisit
4	2.1.1.1 Description	conformance	reviewed	conformance	reviewed
2	The following Wiping functions are being covered • Tip wiping • Intervall wiping	conformance legitimate deviations	need to revisit	conformance legitimate deviations	reviewed
6	2.1.1.2 Input Signals	conformance legitimate deviations	need to revisit	conformance	need to revisit
7	si_LsWWS_1.Frontwischer_ein_lin	no referencing		conformance legitimate deviations	not reviewed
8	2.1.1.3 Output Signals	conformance legitimate deviations	reviewed	conformance	not reviewed
16	us_signal	no referencing		conformance legitimate deviations	reviewed
9	so_LsMaster_1.Master_BSZ_lin	no referencing		conformance legitimate deviations	reviewed
10	2.1.1.4 Processing	conformance legitimate deviations	not reviewed	conformance	not reviewed
3	The position of the collumn level *(si_mLSM_1.LS1_Tipwischen_can, si_mLSM_1.LS1_Intervall_can, si_mLSM_1.LS1_WischenStufe_1_can, si_mLSM_1.LS1_WischenStufe_2_can)* are beign sent from SMLS to Comfort-CAN and are read by BCM.	no referencing		illegitimate deviations	not reviewed
18	2.1.2 Rear Wiper	no referencing		legitimate deviations	need to revisit

- M. Weber, M.-O. Reiser, PIK'09, Berlin -

Beispiel: Anwendung auf Funktionsfamilie Licht und Beleuchtung



		DOORS Modules			
	Base	Platform A		Platform B	
Objects, thereof ...	1,908	1,351		3,501	
- without reference object		590	43.7%	1,714	49.0%
- with reference object		761	56.3%	1,787	51.0%
Coverage		0.399	39.9%	0.937	93.7%
Innovation		0.437	43.7%	0.490	49.0%
Deviations:					
- Refinement		33	2.4%	107	3.1%
- Reduction		24	1.8%	18	0.5%
- Move		5	0.4%	36	1.0%
- Reorder		–		2	0.1%
- Textual Changes		220	16.3%	241	6.9%
- Merge		–		2	0.1%
- Split		65	4.8%	16	0.5%

Fazit

- **Bei der Einführung von Produktlinien für Requirements Engineering in der Automobilelektronik sind eine Vielzahl von domänenspezifischen Anforderungen zu beachten.**
- **In diesem Zusammenhang wurden von der Carmeq GmbH in Zusammenarbeit mit der TU Berlin zwei werkzeuggestützte Erweiterungen der Variabilitätsmodellierung entwickelt**
  - Durch Configuration Links können komplexe Variabilitätsmodelle nach domänenspezifischen Aspekten strukturiert werden.
  - Durch das Multi-Level Konzept kann die Wiederverwendung zwischen Produktlinien mit begrenztem Scope unterstützt werden.

carmeq
software & systems

Referenzen

Multi-Level Konzept

- **Beschreibung**
  - M.-O. Reiser, M. Weber, Multi-Level Features Trees -- A pragmatic approach to managing highly complex product families, RE Journal, vol. 12, no. 2, pp. 57-75, 2007.
- **Implementierung für Spezifikationen: MuLe**
  - www.mule-re.org
  - Download available in September09
- **Implementierung für Featurebäume: CVM-Editor**
  - s.r.

Configuration Links

- **Beschreibung**
  - Reiser, Mark-Oliver, Managing Complex Variability in Automotive Software Product Lines with Subscoping and Configuration Links, doctoral dissertation,TU Berlin, 2008.
- **CVM-Editor**
  - www.cvm-framework.org
  - Download available in September09

carmeq
software & systems

Vielen Dank für Ihre Aufmerksamkeit

Matthias Weber
Tel. +49-30-39 83 537-230
Email: matthias.weber@carmeq.com

Mark-Oliver Reiser
Tel. +49-30-314 21764
Email: mark-oliver.reiser@tu-berlin.de

A.4 A model-driven product line approach for HMI development

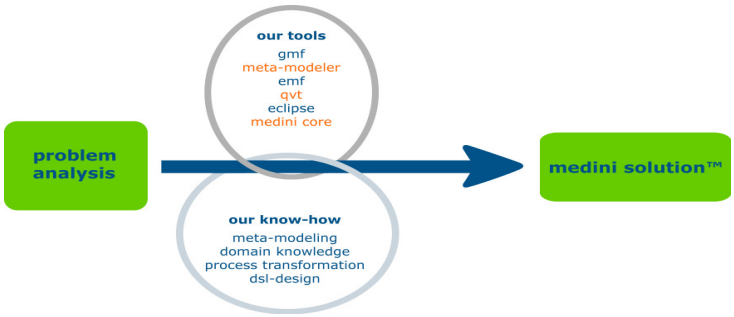
A Model-driven Product Line Approach For HMI Development

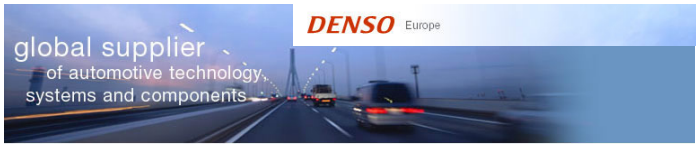
PIK 2009, Berlin
25.05.2009
Marc Born, Omar Ekine

© ikv++ technologies ag 2009

ikv

ikv and DENSO – who we are





confidential

ikv++ technologies ag

roadmap

- motivation
- HMI development process at DENSO
- model-driven HMI development
  - HMI screen flow modeling
  - HMI screen flow simulation
- variant management with software product line approach
  - family models : feature and variation modeling
  - variant models : generated from family models
- tool demonstration

motivation

- car infotainment software belongs to those products with relatively large common behavior but different look and feel
- OEMs (car manufacturers) want a high quality, time and cost saving infotainment system satisfying their individual functional and non-functional requirements
- OEM suppliers like DENSO cannot therefore deliver same looking software to all customer
- DENSO therefore has the challenge of combining the following aspects:
  - mass production to reduce cost
  - high degree of software re-use to increase productivity and quality
  - customize software easily to fulfill OEM‘s individual requirements

navigation system development process at DENSO

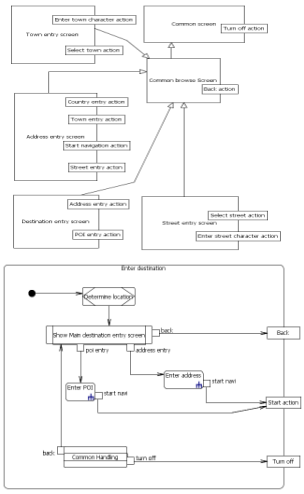
model-driven product line development process

- feature-wise stable core navigation software
- separate screen flow from look and feel (skin) design
  - screen flow and skin models have clear semantics
- consistency checks can be applied to models in order to validate them
- HMI model can be simulated in order to detect errors at an early stage
- system features are modeled and connected to HMI models
  - allows fast feature based configuration and generation of customer specific variants
- HMI code can be generated from the models

confidential
ikv++ technologies ag

model-driven HMI development – screen flow modeling

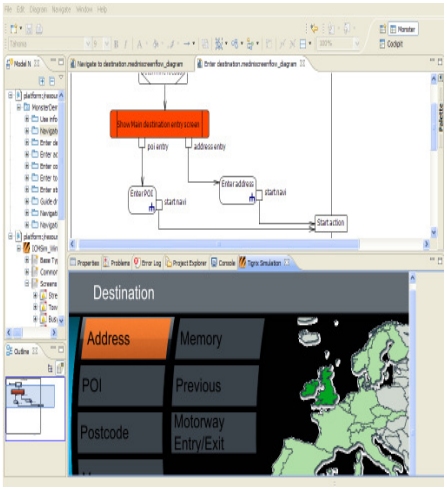
- developed screen flow modeling domain specific language (specialization of UML activities) and graphical editor
- abstract screen model specifies the structure of HMI screens and possible actions that can be triggered from a screen
- a screen flow model specifies the transitions between screens. Transitions are activated by triggering actions
- a task model specifies the tasks expected to be executed by the navigation core



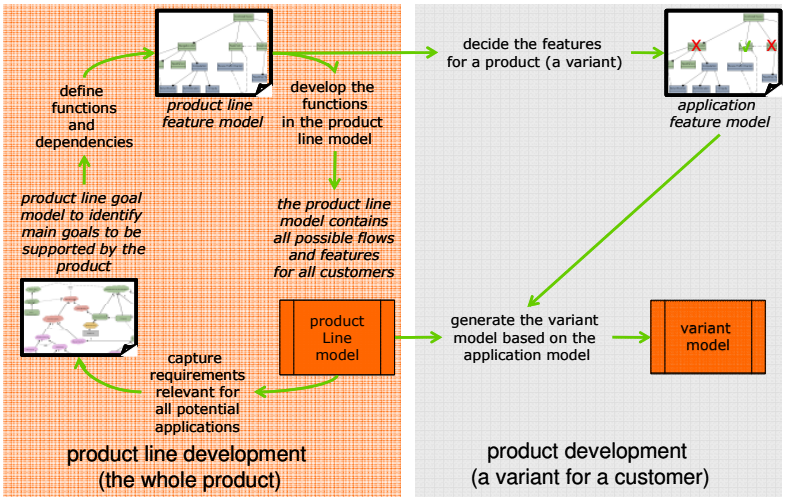
confidential
ikv++ technologies ag

model-driven HMI development – screen flow simulation

- skin is modeled using TiGRiX
- screen flow can be simulated without concrete skin
- connect skin to abstract screens and then simulate flow with skin
- different skins conforming to the same flow can be simulated simultaneously
- skins designed with other technologies like Flash could also be connected and simulated



SPL based HMI development process - overview



SPL based development process

main goals of ***product line development***

- development of reusable artifacts
- identification of commonalities within a family of products
- identification of variations in a family of products

main goals of ***product development***

- development of selected product (variant)
- maximize reuse of existing artifacts
- identification of new common features

➔ **feature models are the central means to organize the relations and dependencies between product line and product artifacts**

confidential
ikv++ technologies ag

feature models

feature

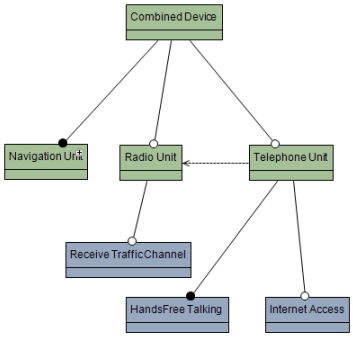
- a characteristic of a system relevant for some stakeholder
- examples:
  - a requirement
  - a technical function or function group
  - a non-functional (quality) characteristic

features are related to each other

- a feature may have optional or mandatory subfeatures
- features may exclude or require each other

features are related to other artifacts of the product line

- a component may implement a feature
- a feature may fulfill a goal or requirement
- a feature may control the flow of activities in a use case



confidential
ikv++ technologies ag

HMI variant management with product line approach

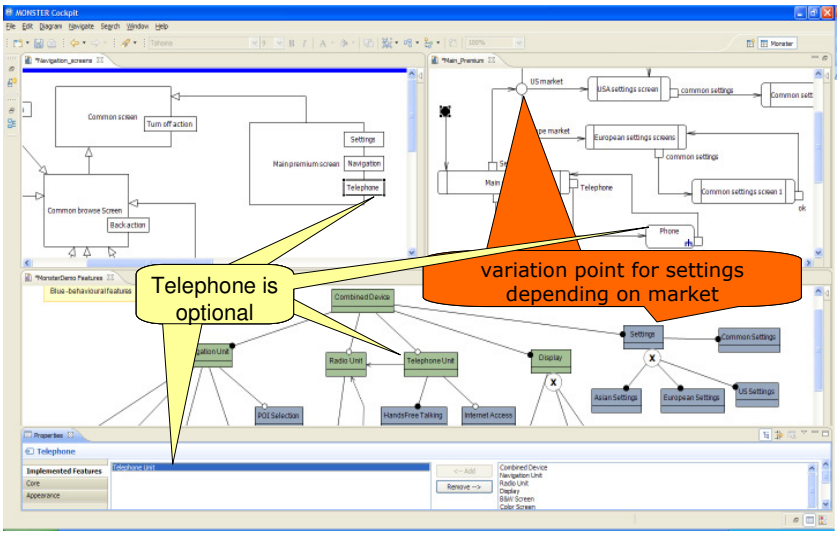
family models

- variation points are explicitly modelled in family screen flow
- family features are applied to screen flow model elements for expressing:
  - **optional flows:** a feature exits or not in a variant
  - **alternative flows:** a feature is realized in different variants differently

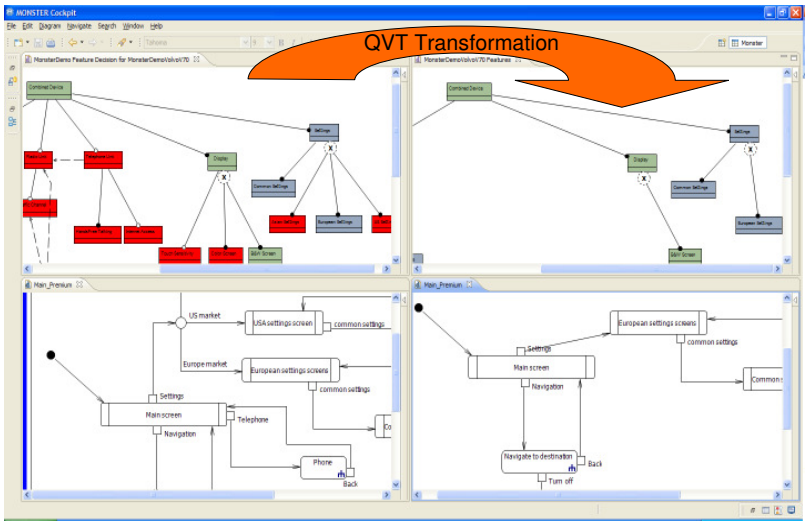
variant models

- feature decision is made in family model for a specific variant and used to derive variant feature model
- variant flow model is derived from family flow based on the variant feature model
- QVT engine is used for the transformation
  - flexible adaptation of generation rules without redeployment
  - bidirectional transformation possible

HMI variant management - family models



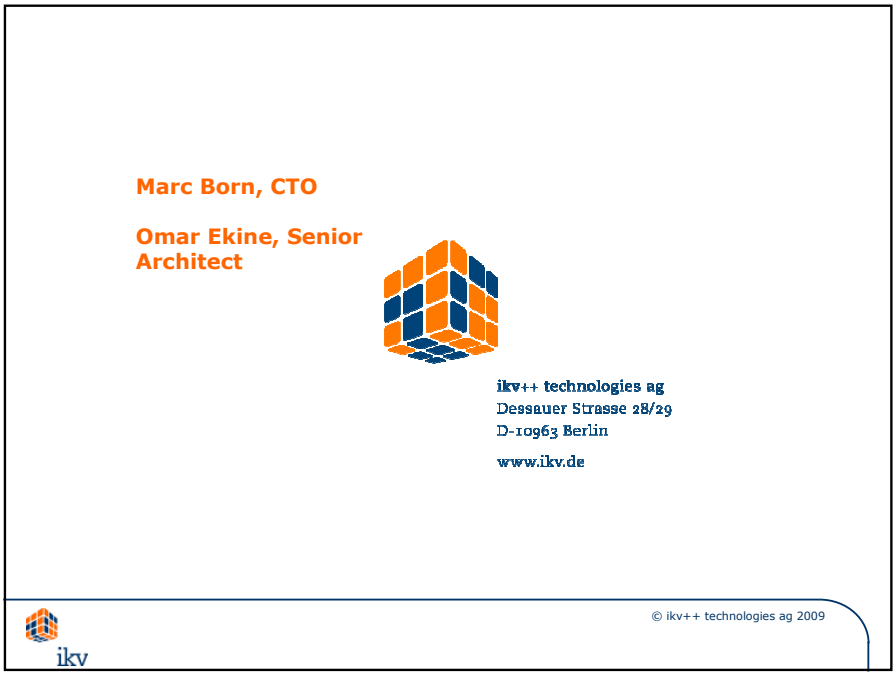
from family to variant models



conclusion

- model driven development provides semantically rich and consistent models which separate flow and skin design but allows connecting them to check conformance and enhance communication
- product line approach increases reuse while allowing easy customization to specific customer requirements
- reduction of the average development time for a new product
- reduced time-to-market and time-to-revenue for a new product
- better product quality and improved company reputation for quality
- increased agility to react on market requirements

➔ **reduction in the average engineering cost per product**


Marc Born, CTO
Omar Ekine, Senior Architect
ikv++ technologies ag
Dessauer Strasse 28/29
D-10963 Berlin
www.ikv.de
© ikv++ technologies ag 2009
ikv

A.5 UML und Softwareproduktlinien – Wie, Weshalb und Wann?

Wie, Weshalb und Wann?

Eine Diskussion

Danilo Beuche, pure-systems GmbH
Enrico Schnepel, b+m Informatik AG

PIK09 - UML und Softwareproduktlinien

Wer sind wir?

Danilo Beuche

- PLE
- pure-systems GmbH
- http://www.pure-systems.com/

Enrico Schnepel

- MDD und DSL
- b+m Informatik AG
- http://engineering.bmiag.de/

Was denken wir?

PLE+MDD = gute Idee

- Gute Kombination des erhöhten Abstraktionslevels
- Erfassung der Produkteigenschaften durch PLE in einem Modell
- MDD ermöglicht einfache Partitionierung und Zusammenfassung von Variabilität

PLE+UML = {gute|schlechte}? Idee

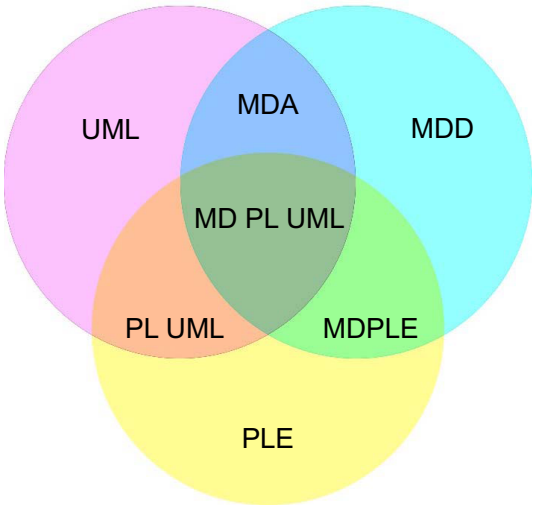
- Was ist der Mehrwert der Nutzung von UML-Modellen?
- UML am Anfang und/oder am Ende des Entwicklungsprozesses?
- Und was ist mit UML eigentlich gemeint?

PIK09 - UML und Softwareproduktlinien

Wofür wird UML verwendet?

- Kommunikation:
  - Beschreibung Architektur (Klass. UML)
  - Beschreibung von Anforderung (SysML)
- Systemgenerierung:
  - Erzeugung von Code-Fragementen (Klassendiagramme)
  - Erzeugung von Anwendungscode (MDA)
  - Prototyping/Simulation (Executable UML)

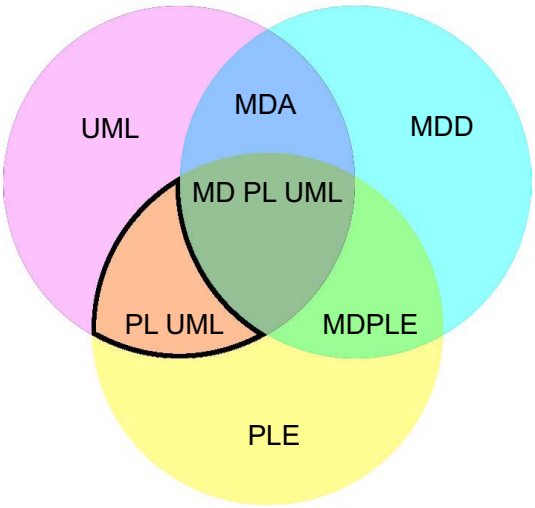
Wie stehen UML, MDD und PLE im Verhältnis?



PIK09 - UML und Softwareproduktlinien

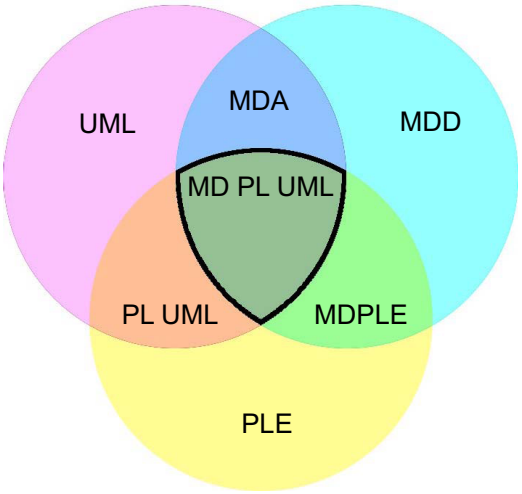
Produktlinien mit UML

- Nutzung von (Standard) UML Modellen als Grundlage für PLE
- Keine automatische Erstellung essentieller Artefakte der Implementierung
- Beispiele:
  - Conditional UML
  - Nutzung von speziellen Variabilitätsprofilen (Gooma, Stahl/Wilkiens)



Model Driven Produktlinien mit UML

- Nutzung von UML Modellen als Grundlage für PLE
- Automatische Erstellung essentieller Artefakte der Implementierung
- Beispiele:
  - Nutzung von domänenbezogenen Profilen mit integriertem Variabilitätskonzept
  - Nutzung von domänenbezogenen Profilen mit Conditional UML



PIK09 - UML und Softwareproduktlinien

Nutzungsszenarien – Matrix

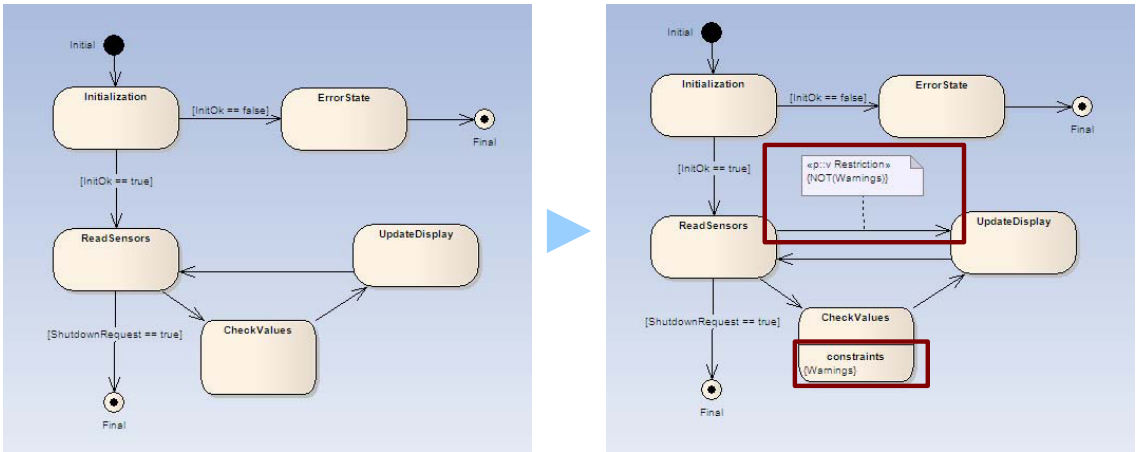
	UML	MDD	PLE	MDA	MDPLE	PLUML	MDPLUML
Dokumentation Architektur	■			■		■	?
Anforderungen System	■		■		■	■	?
Generierung von Code-Fragementen							?
Umfassende Generierung von Code		■		■	■		?

Umsetzung PL UML: Conditional UML

- Übertragung der Idee von „**`#ifdef`**“ in die UML Welt:
  - UML Constraints steuern Vorhandensein von UML Artefakten in den Modellinstanzen der Varianten
  - Constraints referenzieren Feature und andere Konfigurationsinformationen
  - Featuremodelle können separat oder in UML modelliert werden

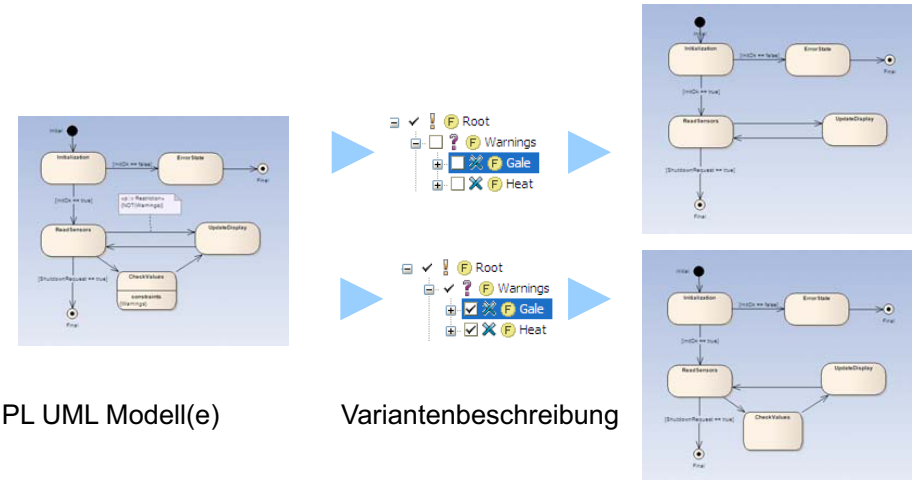
PIK09 - UML und Softwareproduktlinien

Conditional UML: 1. Schritt



Hinzufügen von Variabilität durch optionale UML Elemente und spezielle UML Constraints

Conditional UML: 2. Schritt



PL UML Modell(e) Variantenbeschreibung

Auflösen von Variabilität durch Instanzierung der UML Modelle anhand von Variantenbeschreibungen

PIK09 - UML und Softwareproduktlinien

Umsetzung PL UML: Conditional UML

- Pro:
  - Kann grundsätzlich mit allen UML Modellen angewandt werden
  - Einfaches Konzept
  - Varianten UML Modelle sind „Standard“ UML Modelle
- Con:
  - nur „negative“ Variabilität (Weglassen von Optionen) modellierbar
  - Viele existierende semantische Checks nur auf den Instanzen (Varianten UML Modellen) anwendbar

PLE und MDD

MDD

- Nutzung der PLE-Modelle
- Konfiguration durch Varianten
- Evtl. vorhandene PLE-Tools

PLE

- Einfache Generierung von SW-Artefakten mit der MDD
- Evtl. Modelltransformation für existierende Generatoren

Vorteile auf beiden Seiten

PIK09 - UML und Softwareproduktlinien

Ist UML MDD?

- Profiliertes UML
- MDA (PIM / PSM / Generierung)
- Essentieller Bestandteil des Entwicklungsprozesses
- Generierung wichtiger Teil- oder ganzer Artefakte

✔ DSL-Design Bestandteil von UML-Tools

✗ Keine M2M-Transformationen

✗ UML-Tool basierter Code-Generator

pure·systems b+m engineering

UML und PLE?

- ✔ Einführung von Modellen (DSL mit Profilen)
- ✔ Generierte finale Dokumentation

- ✗ Sprachergonomie
  - Lesbarkeit / Verständlichkeit
  - UML ≠ DSL
- ✗ Abhängigkeit vom UML-Tool
  - PLE ➔ lange Nutzungsdauer
- ✗ Ursprüngliche Intention

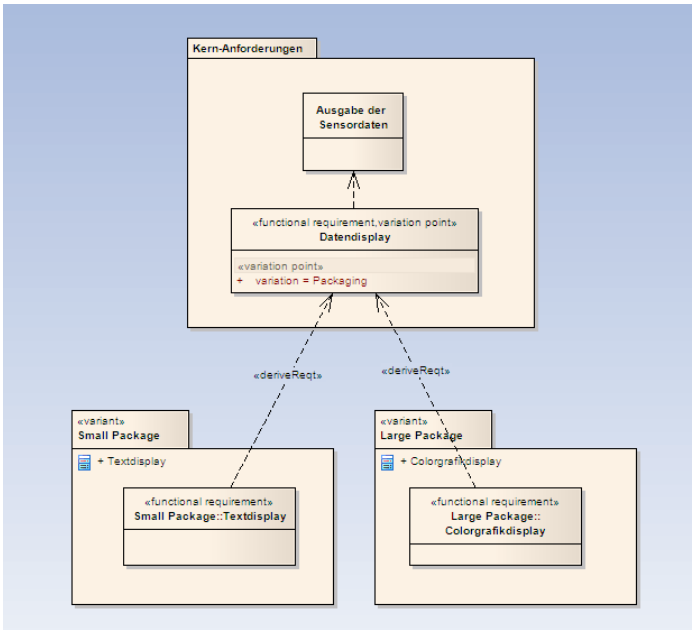
PIK09 - UML und Softwareproduktlinien

pure·systems b+m engineering

Umsetzung MD PL UML: Profiled SysML

- Nutzung eines Profils zur Darstellung von Variabilität
  - Stereotypen und zugehörige Semantik ergänzen UML um Variabilitätsbeschreibungmöglichkeit
  - Beispiel nutzt Profil definiert von Stahl/Weilkiens
    - Markieren von <<variation points>> in Modellen und verbinden mit den Instanzen des Variationspunktes
    - Informationen zu Instanzen von Variantionspunkten werden in <<variant>> Packages in SysML abgelegt

Profiled SysML: Beispieldiagramm



PIK09 - UML und Softwareproduktlinien

Abschlußbetrachtungen

- Für wen ist UML und PLE eine sinnvolle Kombination?
  - „Legacy UML“, insbesondere wenn zur Generierung verwendet
  - Werkzeugkenntnisse/-verfügbarkeit in der Organisation
- Ist MD PL UML eine gute Idee? Oder besser gleich „richtige“ PL-DSL
  - wird bereits UML verwendet: siehe oben
  - bei neuen Projekten: ???

pure·systems

b+m engineering

Wo gibt es mehr?

- Lesen
  - Hassan Gomaa: Designing Software Product Lines with UML, ISBN-10: 0201775956
  - Stahl/Weilkiens: SysML/DSL für Anforderungen mit Varianten
    http://2009.reconf.de/fileadmin/PDF_Dateien/REConf_2009/Vortraege/Weilkiens_Stahl.pdf
  - F. Puhlmann et al: Variability Mechanisms for Process Models.
    http://www.pesoa.de/pages/Publications/Fachberichte062005/PESOA_TR_17-2005.pdf
- Ausprobieren:
  - pure::variants for Enterprise Architect (ab 1.6.2009)
    http://www.pure-systems.com/