SW Produktlinienentwicklung

-Lernskript-

© 2005 René Reiners

Rene.Reiners@post.RWTH-Aachen.de

Hinweis

Bei dem vorliegenden Dokument handelt es sich lediglich um eine *persönliche* Zusammenfassung, wobei Formulierungen teilweise aus dem Skript übernommen, oder aber auch persönlich erstellt wurden. Inhaltliche Gewichtungen oder Interpretationen stimmen nicht unbedingt mit den Inhalten und Aussagen der Vorlesungen überein.
Daher ist das vorliegende Lernskript lediglich als Hilfestellung bei der Strukturierung der Vorlesungsinhalte zu verstehen. ***Es ersetzt keinesfalls die Vorlesung oder die Bearbeitung der Skripte und erhebt in keinster Weise Anspruch auf eine der genannten Eigenschaften sowie Vollständigkeit und Korrektheit!***

Trotz allem hoffe ich, mit diesem Skript ein wenig Unterstützung beim Erarbeiten der Veranstaltungsinhalte oder einer evtl. Prüfungsvorbereitung geben zu können.
Für Verbesserungsvorschläge und Korrekturen bin ich jederzeit dankbar.

28. Januar 2005

Vorlesungsbezug:

- Literatur: „Software-Produktlinien“, Böckle u.a.

Inhaltsverzeichnis

1. Was sind Produktlinien?

Softwareproduktlinien genügen den aktuellen Ansprüchen der kostengünstigen Entwicklung und der Integration höherer Funktionalität. Der Weg führt *weg von Individualsystemen hin zu einer Produktplattform,* die die Gemeinsamkeiten vieler Produkte vereinigt. Ausgehend von dieser Produktplattform können nun Softwareprodukte für die tatsächlichen Anwendungsfälle (Kunden) abgeleitet werden. Dadurch werden die Kosten für die grundlegende Anforderungsspezifikation und Softwarearchitektur nur einmal aufgewendet. Die *Ableitung (Generierung)* von einer möglichen Palette und somit die Konkretisierung eines abstrakten Produkts zu einem konkreten, in das die gewünschte Variabilität einfließt, führt zu dem gewünschten *kundenspezifischen Produkt.*

Vorteile:

- Generierte Indivualsoftware (keine Standardsoftware mit Trade-Offs)
- Kostengünstige Entwicklung
- Kürze Entwicklungszeit (Time-To-Market)

Zu beachten ist bei Produktlinien jedoch, daß die grundlegende Struktur der Koordination erhalten bleibt. So darf die Produktlinie nicht unkontrolliert modifiziert werden, da auch aus Ihr abgeleitete Projekte (vielleicht unbeabsichtigt) dadurch Änderungen unterliegen. Weiterhin ist für jede Modifikation zu entscheiden, ob sie für die gesamte Produktpalette oder nur für die eine Ableitung aus der Palette durchgeführt wird.

Gefahren:

- Lange Wartezeiten auf kleine Änderungen wegen Anpassung der Plattform
- Wiederverwendungspotentiale werden evtl. nicht ausgeschöpft, da Änderungen nur an spezifischen Produkten, nicht aber an der Palette vorgenommen wurden
- Zukünftige Inkompatibilitäten zwischen modifizierten Produktlinenartefakten

2. Variabilität in Produktlinien

Im ***Domain Engineering*** werden gemeinsame (invariante) und variable Teile definiert. Durch Selektion unterschiedlicher Varianten werden dann individuelle Produkte abgeleitet.
Typen von Variabilität:

- Features: unterschiedliche Anzahl Features / unterschiedlicher Funktionsumfang (Beispiel: Geldautomat)
- Abläufen: gleiche Features, allerdings unterschiedliche Abläufe in Funktion / Bedienung (Beispiel: Erst PIN, dann Kontofunktionen und andersherum)
- Datenumfang: Produkte müssen mit unterschiedlichen Datenobjekten und Attributen arbeiten (Beispiel: Zusätzliche Daten bei Entladen der Geldkarte)
- Datenformat: Beispiel: Genauigkeit in der Verarbeitung von Dezimalstellen oder zusätzliche sprachliche Informationen zu reinem Text
- Systemzugang: unterschiedliche Zugangsarten zum gleichen System können abgeleitet werden (bspw.: Zugang eines Bankkontos via Telefon, Internet oder Automat).
- Benutzerschnittstellen: Verwendung verschiedener HMIs (sprachliche, textuelle, multimediale Führung)
- Systemschnittstellen: Realisierung der Produkte mit unterschiedlichen Schnittstellen zu Legacy-Systemen.

- Qualität: Ableitung unterschiedlicher Qualitätsansprüche (Performanz, Wartungszeiten, Stabilität)

Diese Variabilitäten können nicht isoliert betrachtet werden sondern es müssen im Zusammenhang mit anderen Varianten Features oder Variabilitäten verändert, gestrichen oder noch hinzugefügt werden. Es entstehen hierdurch auch Auswirkungen auf verschiedene Entwurfsphasen (Beispiel: Qualität auf Systemarchitektur)

Die Beschreibung der Variabilitäten findet mit Hilfe sogenannter ***Variationspunkte*** und ***Varianten*** statt. Ein Variationspunkt gibt die Stelle an, an der die Auswahl einer oder mehrer Varianten möglich ist. Eine Variante beschreibt konkrete Auswirkungen in Bezug auf einen oder mehrere Variationspunkte. Die Variabilitätsabhängigkeit beschreibt, wie eine Variante einem Variationspunkt zugeordnet ist (z.B. optional, alternativ).

Variabilitätsabhängigkeiten:

- Optionale
- Alternativ
- Pflicht
- Ausschluß

Abhängigkeiten von Varianten oder Variationspunkten untereinander werden durch Constraint - Beziehungen beschrieben:

- Requires: Eine Variante erfordert eine andere Variante oder ein VP benötigt einen anderen VP
- Ggs. Ausschluß zweier Varianten oder VPs.

Bisherige Möglichkeiten der UML erlauben nur eine implizite Repräsentation von Variabilität. Mit der expliziten Repräsentation wird eine Diskussion bzgl. der gegeben Variabilität ermöglicht, die die Wiederverwendung von definierten Varianten erleichtert, Trade-Off-Entscheidungen mit dem Kunden unterstützt und die Redefinition von Varianten und VPs ermöglicht.
Die explizite Repräsentation ist für den Erfolg einer Produktlinie essentiell → Die Produktlinie kann so kommuniziert werden.

Die Modellierung der Variabilität für jede Entwicklungsphase sollte nach einem Vorschlag orthogonal durchgeführt werden; Somit werden die Phasen der Entwicklung auf der horizontalen Achse aufgeführt, die verschiedenen Varianten und VPs werden darüber vertikal angeordnet. So sind die verschiedenen Abhängigkeiten sofort zu ersehen. In diese Darstellung finden Use Cases (Requirements), Strukturbeziehungen (Architektur) und Codefragmente (Implementierung) Eingang.
Somit wird die Variabilität durchgängig verfolgbar. Außerdem bleiben die bestehenden Entwicklungsansätze für jede Phase erhalten (Use Cases für Requirements, usw…).
Weiterhin bildet das orthogonale Modell die Basis für die Variabilitätsmodellierung in der gesamten Produktentwicklung und somit auch die Basis für sämtliche Diskussionen um Variabilität → einheitliches Modell; Alle Beteiligten verfügen über das gleiche Verständnis von Variabilität und reden über das gleiche, wenn sie von Variabilität sprechen.

3. Organisationsstrukturen

Nur wenn die Tätigkeiten und Artefakte, die in der Produktlinienentwicklung anfallen, eindeutige Zuständigkeiten und Verantwortliche haben und wenn die erforderlichen Tätigkeiten und Artefakte (vor allem in der Architektur) sich in der Organisationsstruktur widerspiegeln, kann ein Unternehmen den gewünschten Erfolg erzielen.
Wichtig ist in der Organisation die Trennung der Aufgaben Produktlinienentwicklung (als Domain Engineering) und Anwendungsentwicklung (als ***Application Engineering)***. Diese Trennung muß in der Organisation verankert werden und sollte in vielen Fällen in der Struktur sichtbar sein.
Weiterhin muß die Verantwortung für die Produktlinie als Ganzes, für die wieder-verwendbaren Artefakte, für die Referenzarchitektur, für die einzelnen Produkte etc. eindeutig zugeordnet werden und sollte in der Organisationsstruktur reflektiert werden.

Bei der Produktlinienentwicklung kommen Verantwortungen für Tätigkeiten hinzu, die für alle Produkte verwendet werden, gemeinsame Planung aller Produkte, Tests gemeinsam verwendeter Komponenten etc.

Für eine Reihe von Aktivitäten, die in der PLE ausgeführt werden muß es ***Verantwortliche*** geben. Diese werden bestimmten ***Rollen*** zugeordnet:

- Planung und Produktmanagement der Produktlinie als Ganzes, Erstellung und Verwaltung des Produktportfolios
- Anforderungen der PL als Ganzes
- Referenzarchitektur
- Wiederverwendbare Komponenten (für jede einzelne Komponente)
- Entwicklung und Nutzung der Plattform (Menge der wiederverwendbaren Artefakte als Ganzes)
- Change Management
- Test

Dies führt zu folgenden generellen Anforderungen:

- Zuordnung von Verantwortung soll so geschehen, daß schnelle und einfache Entscheidungsfindung ermöglicht wird (Bei PL Probleme bei Verantwortung für PL als Ganzes und Aufbau und Nutzung der Plattform).
- Zuteilung von Aufgaben und Verantwortung soll sich in der Struktur widerspiegeln
- Zuständigkeit für gemeinschaftliche Aufgaben soll sich in der Struktur widerspiegeln (gilt insbesondere für Aufbau und Nutzung der Plattform)
- Zeit für Overhead, Besprechungen, Abstimmungen und Entwicklungsfindung soll minimiert werden.
- Vergütung und Förderung von Tätigkeiten muß so erfolgen, daß Tätigkeiten im Domain und Application Engineering für die Mitarbeiter attraktiv sind.
- Die sonst bei der projektspezifischen Organisation erreichte Kundenorientierung soll nicht verloren gehen (bei PL-Entw. haben manche Organisationsstrukturen keinen oder nur wenigen Kundenkontakt).
- Erhöhung der Motivation der Mitarbeiter (In machen Organisationen werden Aktivitäten, die nicht zum unmittelbaren Produkterfolg führen, in den Schatten gestellt → Demoralisierung für Mitarbeiter in diesen Bereichen, die gleichwohl wichtig für die Entwicklung sind (fundamentale Arbeit im Hintergrund).

Grundlegende Organisationsstrukturen für die Produktlinienentwicklung

- *verteiltes Domain Engineering* (nicht gut für Zuordnung der Verantwortung, die Struktur spiegelt nicht die Aufgaben wider, gut für Vergütung, Kundenorientierung)
- *zentrales Domain Engineering* – zentrales Element kann stärker oder schwächer ausgeprägt sein, abhängig von der jeweiligen Domäne (neutral für Mitarbeitermotivation)

In größeren Firmen reichen Linienorganisationen nicht aus. Im Allgemeinen sind mehrere Produkte vorhanden, die laufend weiterentwickelt werden. Das Produktwissen der betreffenden Entwickler und die Kooperation verschiedener Fachleute werden für ein Projekt ebenso benötigt wie das technische Fachwissen, das nur in einer Linienorganisation gepflegt werden kann → Matrixorganisation

- mit Domänenaufgabe als Funktion (Entscheidungsgewalt in den Kreuzungspunkten muß geklärt sein!)
- mit Domänenaufgabe als Projekt (Personen aus allen Kompetenzbereichen sind in der Domänengruppe vertreten → dynamische Plattform, gezielte Evolution)
- mit separater Domänenaufgabe (Entscheidungen über die Funktionalität der wiederverwendbaren Artefakte leichter, Probleme allerdings, da Projekte nun unabhängig sind und Artefakte nach Belieben nutzen oder nicht

Funktionen sind die verschiedenen Kompetenzbereiche (PM + Req, Architektur, UI,…).

Eine Festlegung auf eine Organisationsstruktur muß für den jeweiligen Fall abgewogen werden (Heuristiken stehen zu Verfügung).

4. Scoping

Scoping = Produktliniendefinition → Entscheidender Schritt ind der PLE, da Einfluß genommen wird auf die ökonomischen Resultate.

Identifikation der Funktionalität, die eine Produktlinie in wiederverwendbarer Form zur Verfügung stellen soll und der Produkte, die auf Basis der Produktlinie entwickelt werden sollen.

Bei den Komponentengrenzen geht es um die Festlegung der wiederverwendbaren Realisierung, also um den Lösungsraum. Bei der Festlegung der Domänen- und Produktliniengrenzen um Festlegungen im Problemraum (Was ist Gegenstand der Entwicklung?).
Üblicherweise werden diese Techniken zu Beginn des Domain Engineering durchgeführt, z.B. als Teil der Domänenanalyse.

Product Portfolio Scoping: Beschreibung der Produkte, die Bestandteil der Produktlinie werden sollen bzw. die zu den Anforderungen an die Produktlinie beitragen.
Domain Scoping: Ableitung und Abgrenzung von Domänen (Funktionsbereichen), die für die Realisierung der im Produktportfolio enthaltenen Systeme relevant sind (Ermittlung einer Domäne meist featureorientiert).
Asset Scoping: Komponentendefinition. Identifikation der Artefakte, die zur Realisierung der Produktlinienplattform relevant sind. Insbesondere Festlegung des Grades der wiederverwendbaren Unterstützung bestimmter Funktionalitäten (→ Ermittlung von

„Wunschkomponenten“, welche dann als Vorlage für die Referenzarchitekturentwicklung aus Wiederverwendungssicht dienen – Abweichungen aus technischen Gründen möglich → „weiche Anforderungen“).

Praxiserprobter Ansatz: **PuLSE-Eco** (Unterstützung des Domain und Asset Scopings)

Scoping – Ansatz innerhalb des PuLSE™ - Framework (Product Line Software Engineering), Fokussierung vor allem auf die Bereiche Domain Scoping und Asset Scoping. Zielsetzung: Entwicklung einer Produktlinie durch die Identifikation wiederverwendbarer Komponenten unter ökonomischen Gesichtspunkten.

Phasen:
→ *Product Line Mapping* (Ziel: Bestandsaufnahme der Produktlinienentwicklung.
Man erhält eine systematische Beschreibung der verschiedenen Produkte, basierend auf ihren Features. Diese Features sind eindeutig auf ihre Domänen aufgeteilt)

→ *Domain Potential Analysis* (Domain Scoping – Abgrenzung, Auswahlentscheidung. Systematische Bewertung um Bereiche mit dem höchsten Wiederverwendbarkeitspotential aufzuspüren, basiert auf einem Assessmentansatz (Vorbereitung: Bereichsfestlegung, Identifikation von Interviewpartner – Durchführung: Standardisierte Fragebögen – Analyse: Endgültige Bewertung basierend auf Ergebnissen → kommt ein PL-Ansatz überhaupt in Frage und welche Funktionsbereiche für eine Wiederverwendung sind besonders geeignet?)

→ *Reuse Infrastructure Scoping* (Asset Scoping – hoher Aufwand – welche wiederverwendbaren Komponenten sollten entwickelt werden, um einen möglichst hohen ROI zu erhalten? – Nutzenaspekte eines hohen ROI: Geringere Kosten bzw. Entwicklungszeit für das erste und weitere Produkte, geringere Entwicklungsrisiken, geringere Wartungskosten, höhere Qualität, bessere Benutzbarkeit).

Die einzelnen Phasen sollten nochmals nachgelesen werden!!!

5. Bewertungsverfahren für das Produktmanagement

Rolle des Produktmanagers ist bei der strategischen Planung von Produkten und Produktlinien von großer Bedeutung → Schnittstelle zwischen Marketing und Engineering. Assessments werden zur Bewertung der Effizienz und Leistungsfähigkeit des Produktmanagements eingesetzt. Entwickelt wurde der Assessment-Ansatz von der Siemens AG.

Assessments bieten eine quantitative und qualitative Bewertung der gegenwärtigen Situation des Produktmanagements. Beispiele von Anpassungen:

- Anpassung der Organisation (z. B. Einführung von Produktmanagement-Teams)
- Einführung neuer Methoden (z.B. Portfoliotechniken)
- Definition oder Vervollständigung eines Produktmanagementprozesses
- Ausbildung der Mitarbeiter in Methoden des Produktmanagements und die Anwendung von guten Praxisbeispielen
- Überprüfung der Strategie im Hinblick auf Ziele, Ausrichtung, Erfolgsfaktoren und die Überwindung von Barrieren.
- Optimierung des Produktportfolios und klare Strategie zur Einführung und Weiterentwicklung einer Produktlinie

Diese Basis kann zu höherer Effizienz, niedrigeren Kosten, kürzeren Entwicklungszeiten und höherer Qualität führen.

Durchführung des Assessments:
Gruppen- und Einzelinterviews, Präsentation der Ergebnisse vor allen Beteiligten.

Ergebnisse:
Bewertungsdiagramm, detailliertes Stärken / Schwächen-Profil, Maßnahmenkatalog, Handlungsbedarfsmatrix.

6. Requirements Engineering

Das Requirements Engineering in der PLE unterscheidet zwischen Domain und Application Engineering.
Domain Engineering: Festlegung des Scope der Produktlinie (Was soll entwickelt werden?) und Definition gemeinsamer und variabler Artefakte (Assets):
Application Engineering: Kommunikation der Produktlinie zum Kunden, Ermittlung der Produktanforderungen unter Berücksichtigung der PL - Möglichkeiten und Ableitung der zu entwickelnden Produkte.

Mittel der Praxis: Szenarien und Use Cases
→ Möglichkeit zur Repräsentation gemeinsamer und variabler Artefakte einer Produktlinie
→ Kommunikation der Produktlinie für den Kunden und Erhöhung des Anteils wiederverwendbarer Artefakte. Eine grobe Kategorisierung des Realisierungsaufwandes auf der Basis einer Produktlinie unterstützt den Kunden bei Trade-Off – Entscheidungen (Sollen die Anforderungen des Kunden angepaßt werden oder eine Zusatzentwicklung erfolgen?). Der Einsatz von Szenarien unterstützt die Produktableitung
→ Verknüpfung verschiedener Bereiche in einem Kontext.

Zu Use Cases: Gut geeignet, um Interaktionen zwischen Benutzern und Systemen untereinander zu erfassen und daraus Anforderungen abzuleiten. Allerdings muß dieser Ansatz um die Variabilität einer Produktlinie erweitert werden.
Ansätze:

- <<include>> - <<exclude>> - Beziehungen (Notwendigkeit, Alternative), Generalisierung für Varianten → Einführung eines Stereotyps <<Variante>>.
- Kennzeichnung von Varianten durch einen entsprechenden Stereotyp, Variationspunkte durch einen Punkt.
- Von der Maßen und Lichter: Einführung neuer Beziehungen: <<alternative>> und <<optional>>. Weiterhin: Definition eines Modellelements „Variationpoint“. Keine eigene grafische Notation.

Nachteile dieser Ansätze:

- VPs und Varianten sind nicht oder nur schwer erkenntbar
- Darstellung, ob nun optionale oder notwendige Varianten, unzureichend

Vorgeschlagene Erweiterung:
Konzentration auf grafische Darstellung (gut geeignet zur Kommunikation der Produktlinie). Notationen für Variationspunkte und Unterscheidung in notwendige (mandatory) und optionale (optional) Variationspunkte. Einführung des Stereotyps <<Variante>> und Kardinalitäten für Beziehung zwischen Varianten und Variationspunkten.

- VP: Darstellung durch Dreieck, inkludiert von einem Use-Case.
- Variante: Stereotyp <<Variante>>
- Madatory (gefüllter Punkt), optional (leerer Punkt) → müssen Bestandteil der Beziehung sein, nicht der Variante, da nicht allgemein gültig! (Je nach Szeanrio).
- Kardinalitäten definieren, wie viele der Varianten mindestens und höchstens ausgewählt werden können (1..1 – Beziehung bei madatory – Varianten)
- VP mandatory: gefülltes Dreieck → eine Variante muß mindestens gewählt werden. Leeres Dreieck: optional.

Bei der Kommunikation sollen die Varianten ausgeblendet werden, die für den Kunden nicht interessant sind, wie zum Beispiel technische Aspekt. Trade-Off – Entscheidungen können auch nur sinnvoll getroffen werden, wenn sich der Kunde der Möglichkeiten der Produktlinie bewußt ist → grobe Kategorisierung des Aufwands, so daß der Kunde Anhaltspunkte für die Entscheidung erhält.

Szenariobasierte Produktableitung
Use Cases werden in der Produktableitung weiter verwendet.
Produktlinienszenarien: Enthalten die Variabilität eine Produktlinie →Möglichkeiten und mögliche Varianten einer Produktlinie
Produktszenarien: Beschreibung der Anforderungen des zu definierenden Produkts. Ableitung aus Produktlinienszenarien und evtl. Anpassung an Kundenanforderungen. Variabilität noch vorhanden, wenn Kunde bzgl. eines Variationspunktes mehrere Alternativen ausgewählt hat. Wichtig ist die Dokumentation der Ableitungen bzw. Selektionen → Nachvollziehbarkeit (Aus welchen Produktlinienszenarien kommen die Produktszenarien?) →Weitere Entw.-phasen.

Phasen:

1.) Abdeckung (Welche Produktlinienszenarien passen zu den Anforderungen?)
2.) Deltas (Ermittlung der Unterschiede zwischen Kundenwünschen und realisierbarem Produkt → evtl. Trade-Offs oder Neuentwicklungen)
3.) Rückfluß: Interessant für evtl. Umgestaltung der Produktlinie oder verschiedener Variationspunkte (Anpassung und Entwicklung der Produktlinie)

7. Architekturentwicklung

Eine wichtige Voraussetzung für eine erfolgreiche Produktlinie ist die Entwicklung einer gemeinsamen Plattform für alle Produkte.
Systematische Architekturentwicklung bei softwareintensiven Produktlinien mit Hilfe des QUADRAT-Ansatzes (Quality-Driven Architecture Development):
Iteratives und inkrementelles Verfahren, Phasen stehen in Beziehung zueinander. Noch keine Architekturentwürfe: Beginn mit Analyse. Architektur bereits ermittelt → Evaluierung des aktuellen Status.

Drei grundlegende Aktivitäten:

- ***Analyse:*** Identifikation architekturrelevanter Anforderungen (Stichwort: Architekturtreiber: Geschäftsziele, Kundenwünsche/Anforderungen, Randbedingungen) wirken ein auf: Entwurfmechanismen (erprobte Lösungen für Entwurfsprobleme, die sich hinter Qualitätsmerkmalen verbergen), Architektursichten (Darstellungen der wichtigsten Prozesse des Systems und deren Kommunikation untereinander).
- ***Modellierung:*** Erstellung und Dokumentation der Referenzarchitektur (Modellierung der Architektursichten und der Architekturvariabilität (Anzahl der Variationspunkte innerhalb der Produktlinie)).
- ***Evaluierung:*** Identifikation von Risiken im Architekturentwurf → Bewertung der Qualität der Architektur und Absicherung für weitere Entwurfsschritte

Unterstützung bei der Analyse und Evaluieren durch das Softwaretool AET.

(Referenz-)Architektur: Dokumentation der wichtigsten Entwurfskomponenten und Variationspunkte der Produktlinie sowie deren Eigenschaften und Beziehungen untereinander.
→ Sorgfalt wichtig, da sich Fehler bei der Erstellung auf das gesamte Produktspektrum auswirken können.

8. Anforderungsbasierte Selektion von COTS - Komponenten

Wiederverwendung von Softwarekomponenten führt auch in der Produktlinienentwicklung zu Einsparung von Zeit und Entwicklungsaufwand und zur Steigerung der Qualität des Gesamtsystems.
Ansatz: CoVAR

Fokus auf drei eng verknüpfte Faktoren:
Anforderungen, Referenzarchitektur und die darin enthaltene Variabilität
Unterstützung, ob Komponenten nach der Integration in die Produktlinie in der Lage sind, deren Variabilität zu gewährleisten.

Grundschritte:

- Komponentenfilterung
- Detaillierte Komponentenevaluation
- Komponentenauswahl

9. Implementierung von Variabilität

Ansätze, Referenzarchitekturen systematisch zu implementieren und die geforderte Variabilität technologisch umzusetzen.
Die Implementierung muß die Architektur konsistent umsetzen.

Faustregel: Konsequente Komponentenorientierung und entsprechend abgestimmter Einsatz von Konfigurationsmanagementsystemen als Basis einer soliden Produktlinienimplementierung. Verfolgbarkeit ausgehend von variierenden Anforderungen bis zur Implementierung der entsprechenden Varianten.
Kontrolle und Aufsicht der Umsetzung wichtig, da bereits Abweichungen bei der ersten Implementierung festgestellt werden und diese Abweichungen in der laufenden Entwicklung immer größer werden.
Die Herausforderung ist die Umsetzung von Variabilität.

Drei Technologiedimensionen:

- ***Komponenten:*** Teile-und-herrsche-Prinzip und Information-hiding (Auslagerung der Variabilität in immer kleinere Teilkomponenten halten den Wiederverwendungsgrad hoch. Entwickler wählen nur noch aus und können aufgrund der nicht mehr benötigten Detailkenntnis der Architektur effizienter und variabler entwickeln
- ***Konfigurationen:*** zeitliche Änderungen der Implementierung werden zeitlich erfaßt durch Konfigurationsmanagementsysteme. Komplexität entsteht durch Versionsmanagement; Einige Neuerungen (Bugfixing) sollen in alle Bereiche einfließen, andere dienen zur Umsetzung bestimmter Variabilitäten, die nicht in alle Produkte Eingang finden sollen. → Versionshistorie: Probleme: Rückführung in den Hauptzweig nicht mehr möglich, da die Abweichung selbst weiter gewartet werden muß. → Probleme bei der Verwaltung von Alternativen.
- ***Generizität:*** - Unterteilung in vertikale Fragmente: Theoretisch jede Form von Variabilität möglich, jedoch schlecht zu handhaben, genaue Kenntnis der Struktur seitens der Entwickler erforderlich (C-Präprozessor ist ein Beispiel).
  Fragmenttechnologie: Codefragmente werden in eigene Dateien (Frames) ausgelagert und sind somit explizit und leicht identifizierbar. Bereiche der Implementierung von Frames, die von anderen Frames erweitert oder modifiziert werden dürfen (Variationspunkte), werden explizit definiert. Inhärente Abzielung auf Wiederverwenung der Frames; Definition neuer Frames als Überbau auf existierenden Frames.
  → *horizontale Fragmente:* Bekannte Realisierung durch aspektorientierte Programmierung. *Nachteil:* Andere Herangehensweise an Implementierungsaktivitäten erforderlich. Schwierigkeiten bei der Einführung in industrielle Umgebungen (erfordert zusätzliche Kommunikationsstrukturen, orthogonal zu den typischen Organisationsstrukturen).

10. Anforderungsbasiertes Testen

Vorstellung der ScenTED-Methode; Möglichkeit der systematischen Ableitung von Testfallen aus Use Cases, Wiederverwendung von Testfällen durch die Bewahrung von Variabilität wird unterstützt, Verfolgbarkeit zwischen Entwicklungsartefakten und Testartefakten → Grundlage für späteres Änderungsmanagement.
ScenTED kann zwar in aktuell verfügbaren kommerziellen Testwerkzeugen eingesetzt werden, allerdings sind der Modellierung der Variabilität in Testfällen enge Grenzen gesetzt.

Ableitung von Testfällen aus jeder abgeleitete Applikation (sehr hoher Aufwand, nur vertretbar bei sicherheitskritischen Systemen), Wiederverwendung von Testfällen nach der Ableitung der ersten Applikation (Gefahr der falschen Entscheidung bei Auswahl der Testfälle → Effekte der Änderungen in der neuen Applikation könnten nicht hinreichend getestet werden), Ableitung von Testfällen bereits im Domain Engineering (Unterteilung in zwei Teilprozesse: ***Domain Testing*** (gemeinsame Bestandteile der PL werden getestet und wiederverwendbare Testartefakte werden erstellt) und ***Application Testing*** (zu späterem Zeitpunkt werden auf Basis der Domänentestfälle spezifische Applikationstestfälle abgeleitet → Testen der spezifischen Bestandteile
→ → Sicherstellung der systematischen Wiederverwendung
Problem: Vollständigkeit der Domänentestfälle (sehr hohe Anzahl möglicher Testfälle aufgrund der Variabilitäten in den Funktionalitäten).

ScenTED: Testfälle werden bereits im Domain Engineering entwickelt → Systematische Wiederverwendung im Application Engineering. Zur Lösung des Problems der Vollständigkeit wird die Variabilität beibehalten (Die vorgesehenen Varianten sind bereits in den Testfällen modelliert). → Außerdem Reduktion der Gesamtzahl der Domänentestfälle.

Testfall = Testfalldesign (Vorstufe eines ausführbaren Tests ohne konkrete Testdaten)

Grundlegendes Produktmodell: Drei Säulen: Anforderungs-, Architektur- und Testartefakte.

10.1 Vorgehensweise:

Ableitung von Domänen-Testfällen

1.) Ableitung von Domänen-Use-Case-Szenarien (Formalisierung in Aktivitätsdiagrammen, die auch die Variabilität berücksichtigen müssen → Wiederverwendbarkeit der Testfälle. Darstellung mit Hilfe des Stereotyps <<VP>> und Schwarzfärbung, Angabe der Namen in Informationsfeldern → Bezug zu korrespondierenden Use Cases. Entscheidungen an VPs können optional oder mandatory sein. Weiterhin wichig: alternative oder co-existing).
Einsatz von Aktivitätsdiagrammen erlaubt die Anwendung des Zweigüberdeckungskriteriums.

2.) Ableitung von Domänen-Systemtestfällen aus den Domänen-Use-Case-Szenarien
Zweigüberdeckung für jede mögliche Applikation notwendig → jeder Zweig des (Use-Case-)Kontrollflusses muß durch mindestens einen Domänen-Systemtestfall abgedeckt werden.
Zweigüberdeckung muß verschachtelt angewandt werden:
    - Anwendung der Zweigüberdeckung auf Use-Case-Kontrollfluß, Ignorieren von Variationspunkten und Varianten (nur ein spezieller Schritt)

- Anwendung der Zweigüberdeckung auf Varianten und Variationspunkte. Ableitung fragmentarische Testfälle für unterschiedliche Varianten und Ergänzung zuvor erstellter Testfälle.

Beispiel auf Seite 127ff nochmal lesen!

Ableitung von Applikations-Testfällen

aus Domänen-Systemtestfällen

3.) Identifikation von Domänen-Use-Case-Szenarien
4.) Identifikation von Domänen-Systemtestfällen
5.) Ableitung von Applikations-Systemtestfällen

Anforderungsanalyse: Teils bekannte Anforderungen die mit Hilfe der Domänen-Use-Cases und deren Variationspunkte und Varianten mit dem Kunden diskutiert werden können. Teils vollkommen neue Anforderungen, bisher nicht im Domain-Engineering behandelt worden sind. → Jetzt ist die Identifikation der Use-Case-Szenarien, die für das spätere Produkt notwendig sind, möglich.
Möglichkeiten:

- Anforderungen sind im DE berücksichtigt worden → Identifikation eines DUCS entspricht Bindung einer Variante
- Anforderungen sind im DE nur teilweise berücksichtigt worden → DUCS, welches die Anforderungen teilweise abdeckt → Anpassung des Szenarios für die zu erstellende Applikation → Entspricht dem Erstellen einer neuen Variante. Alle identifizierten Testartefakte müssen beim Schritt vom DE zum AE angepaßt werden.
- Anforderungen sind im DE nicht berücksichtigt worden → Es müssen völlig neue Testfälle für die Anforderungen erstellt werden.

10.2 Integrationstest in ScenTED

1.) Erstellung von Domänen-Architekturszenarien (Zusammenlegung der Interaktionen der Domänen-Use-Case-Szenarien und der Komponenten aus der Architekturbeschreibung) aus Domänen-Use-Case-Szenarien und der Architektur
2.) Ableitung von Domänen-Integrationstestfällen aus Domänen-Architekturszenarien
3.) Ableitung von Applikations-Integrationstestfällen aus Domänen-Integrationstestfällen
    a. Identifikation von Use-Case-Szenarien
    b. Identifikation von Domänen-Architekturszenarien
    c. Identifikation von Domänen-Integrationstestfällen
    d. Ableitung von Applikations-Integrationstestfällen