Todo list

1 Einführung

Productlifecyclemanagment, oder kurz PLM, ist der Versuch alle Daten, die im Lebenszyklus eines Produktes anfallen, zu verwalten. Dabei überlappen manche Funktionalitäten mit denen eines Versionskontrollsystems (VCS), welches in der reinen Softwareentwicklung verwendet wird.

PLM kann in die drei folgenden Bereiche eingeteilt werden:

- Product Development
- Product Manufacturing
- Product Ownership

Product Development und Product Manufacturing beschäftigen sich mit der Produktnummer bzw. um das einzelne Produkt, wohingegen Product Ownership eine ganze Produktserie behandelt.

Produkte werden auch noch in Produktfamilien und Produktvarianten eingeteilt. Eine Produktfamilie fasst ähnliche Produktvarianten zusammen, welche sich nur in einzelnen Attributen voneinander unterscheiden.

Produkte sind anhand ihrer Produktdefinitionsdaten nachbaubar, würde in Software also dem Quellcode entsprechen. Im Gegensatz dazu existieren noch die Produktspezifikationsdaten, welche nur das Verhalten des Produktes beschreiben, und somit den Schnittstellendefinitionen einer Software entspricht. Bei der Entwicklung dieser Daten sollten auch die bereits existierenden Normen anderer Firmen oder Organisationen beachtet werden.

1.1 Lebenszyklus

Der Lebenszyklus eines Produktes hängt ganz wesentlich von der Industrie ab. Die Industrien werden in verschiedene Typen eingeteilt.

Process Production Hier läuft ein Prozess durchgehen ab, wie zum Beispiel beim Verarbeiten von Rohöl in einer Pipeline.

Batch Production Bei diesem Typ werden sehr große Mengen, wie zum Beispiel ganze Paletten, von einem Produkt produziert.

Embedded Systems Hier ist die Verwendung einer reinen Versionskontroll etwas schwierig, da die Software Teil eines physikalischen Produktes ist. Wenn man diese Produktdaten zusammen verwalten will, muss man ein PLM-System einsetzen.

Construction Das Bauwesen stellt einen Sonderfall dar, da das Produkt, typischerweise ein Gebäude, vor Ort erstellt werden muss.

2 Spezifizierungstechniken

2.1 Objektorientierte Grundlagen

Alles (z.B. ein Fenster, eine Person) kann ein Objekt sein. Alle Objekte verfügen zudem über verschiedene Charakteristiken (z.B. Gewicht, Größe) und Verhalten (z.B. offen, geschlossen). Objekte die die selben Arten von Charakteristiken, Verhalten und Constraints haben, können in Klassen zusammengefasst werden.

2.1.1 Klassen

Klassen kann man sich im Informatikbereich als eine Art Bauplan für Objekte vorstellen. Im Prinzip sind Klassen selbst auch wieder Objekte und können reale Dinge oder computer-interne Representationen sein. Ein Objekt kann zu mehr als einer Klasse gehören (Mehrfachvererbung).

Spezialisierung und Generalisierung Die Beziehung zwischen Klassen kann so gesehen werden, dass jedes Objekt einer spezielleren Klasse auch ein Objekt einer genereleren Klasse ist.

- Multiple Klassifizierung
- Überlappende Klassifizierung
- Dynamische Klassifizierung

2.2 UML Klassendiagramm

UML ermöglicht verschiedene Sichtweisen auf ein und dasselbe Model. Je nach Programmiersprache gibt es angepasste Varianten von UML die nur das als UML zulassen was sich auch in der jeweiligen Programmiersprache wirklich umsetzen lässt.

2.2.1 Klassen

Eine Klasse ist in UML eine Repräsentation von einer Gruppe von Dingen mit denselben Charakteristiken und Verhalten. Operationen von Klassen sind die Definition eines Verhaltes einer Klasse.

- Die UML-Spezifikation sagt nichts darüber aus, welche Art von Klassifizierung verwendet werden darf / kann.
- UML sagt auch nichts darüber aus welche Art der Konfliktlösung bei Mehrfachverwendung von selben Operationsnamen oder Attributnamen verwendet wird (z.B. bei Diamantenproblem). Darum ist die Forderung nach einer expliziten Neudefinierung eines Attributs / einer Operation möglich.

2.2.2 Assoziation

Eine Assoziation ist eine Verbindung zwischen 2 oder mehreren Objekten und wird als Linie zwischen den dazugehörigen Klassen dargestellt. Verbindungen können auch auf die selbe Klasse verweisen um mehrere Instanzen einer Klasse miteinander zu verbinden. Assoziationen sind an sich auch wieder Klassen und bieten somit die Möglichkeit bei den Verbindungen selbst weitere Attribute / Operationen zu definieren.

Nie mehr als 2 Klassen miteinander Verbinden da es zu Probleme mit den Kardinalitäten bzw. deren logischer Auflösung führen kann.

Navigation In welche Richtung eine Assoziation führt wird mit Pfeilen ausgedrückt - ohne Pfeile bedeutet es, dass die Assoziation in beide Richtungen führt.

Aggregation Eine Verbindung die Darstellt welche Klasse ein Teil / im Besitz einer anderen Klasse ist(leere Raute).

Composition Eine Komposition ist auch eine Verbindungsvariante zwischen Objekten, wobei ein Objekt nur so lange exisitieren kann, solange es von einem anderen Objekt besessen wird. Außerdem kann das Objekt nicht von mehr als einem anderen Objekt besessen werden (ausgefüllte Raute).

Dependency Dient der Dokumentierung irgend einer Verbindung ohne weitere Aussage darüber, welcher Art diese Verbindung ist (strichlierte Linie).

2.3 Weitere Diagramm-Arten

Neben herkömmlichem UML gibts es auch weitere Diagrammarten:

SysML Anforderungsdiagram Diese Diagramme gibt es in graphischer und tabellarischer Form sowie als Metamodel.

UML Objekt Diagramm Das Objektdiagramm ist ein Strukturdiagramm und umfasst ausprägungen von Klassen und Assiciationen.

2.4 UML Communication Diagramm

Das Kommunikationsdiagramm ist ein Verhaltensdiagramm. Es zeigt eine bestimmte Sicht auf die dynamischen Aspekte des modellierten Systems und stellt Interaktionen grafisch dar, wobei der Austausch von Nachrichten zwischen Ausprägungen mittels Lebenslinien dargestellt wird.

2.4.1 Lifeline

Eine Lifeline ist in UML eine Art instance eines Objektes, allerdings enthält es keinen Status der Instance. Also besser gesagt ist es ein Platzhalter für ein konkretes Objekt.

2.4.2 Konkreter Nachrichten Fluss

Informationen in diesem Diagramtyp:

- Abfolge der Nachrichten
- Richtung der Nachrichten
- Typen der Nachricht (synchron, asynchron)

Eine andere Darstellungsform für dieses Diagramm ist das Sequenz Diagramm. Es enthält die selben Informationen, stellt diese aber anders dar.

2.4.3 Abstrakter Nachrichten Fluss

Informationen in diesem Diagramtyp:

- Nicht technische Darstellung
- Richtung der Nachrichten
- Typen der kommunizierten Nachrichten

Beispiel Crawler für eine Suchmaschiene

Beispiel aus den Folien

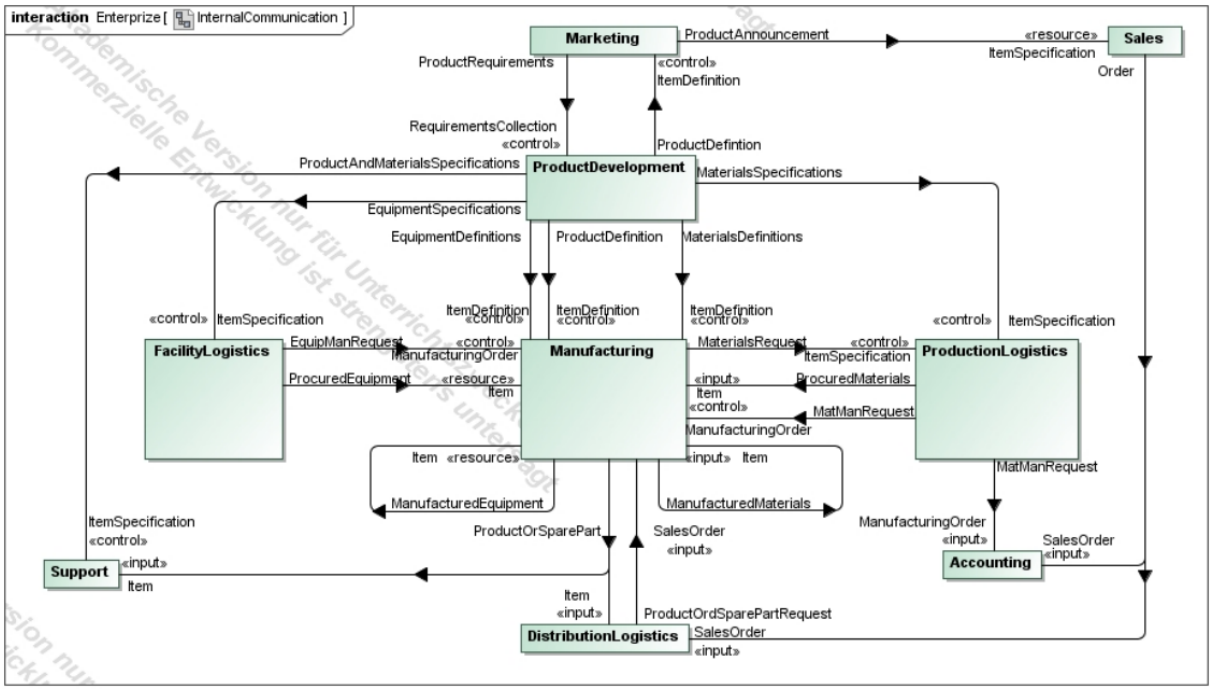

Abbildung 2.1: Abstrakter Informations Fluss

Notizen:

- Marketing schickt anforderungen an ein Produkt an Product Development (PD)
- PD entwickelt das Produkt => Ergebniss Product Definition
  - Materials = Material um das Auto zu bauen (Stückliste)

  - Equipment = Zubehör (Ablauf, Werkzeuge, Prozesse ...) um das Auto zu bauen
- Product Definition geht an Manfacturing
- PD -> MaterialSpecification wird an Products Logistics geleitet wen das Teil eingekauft wird
- Manufacturing bestellt über MaterialsRequest
- Manufactoring verbaut den Motor (input item)
- Fals gewisse Teile doch selber (oder andere Firma) Produziert werden ManufactoringOrder
- FacillityLogistics bekommt ManufactoringOrder und liefert eine Resource item (unterstützt den Prozess)
- PD liefert ProductDefinition an Marketing für neues Produkt
- Marketing liefert ProductSpecification an Sales um Order zu erstellen
- Sales liefert Order an DistributionLogistics (DL)
- DL liefert (input) an Manufactoring
- Stereotypen:
  - Input wird verbracht (nach der verarbeitung nicht mehr da)
  - Output
  - Resource ... Hilfsmittel um die Aufgabe zu erledigen
  - Control ... Steuert den Ablauf im Block

Beispiel: Kompilierung

- Input ist der Source code (weg für den Kompiler)
- Resource ist der Kompiler, BS
- Control ist z.B. Compiler direktiven

2.4.4 UML Aktivitäts Diagramm

Beschreibt das Verhalten (Fluss), nicht die statischen Strukturen.

Anwendungen

- Geschäftsprozesse
- Systemprozesse
- ...

Petri Netze

Mathematische Modelierung von Verteilten Systemen. Entwickelt 1962 con Carl Adam Petri. Grundlagen für Petri Netze sind gerichteten (bipartiten) Graphen:

- Nodes (Knoten):
  - Places: tragen Tokens in sich
  - Transition: Forked oder Joined Kanten
    - Input Place: Verbunden über Kante am Eingang
    - Output Place: Verbunden über Kante am Ausgang
  - Die Transition feuert, wenn jeder Input Place mindestens soviel Tokens wie das Gewicht der Kante besitzt.
  - Die Output Places erhalten nach feuern der Transition soviele Tokens, wie das Gewicht der verbindenden Kante

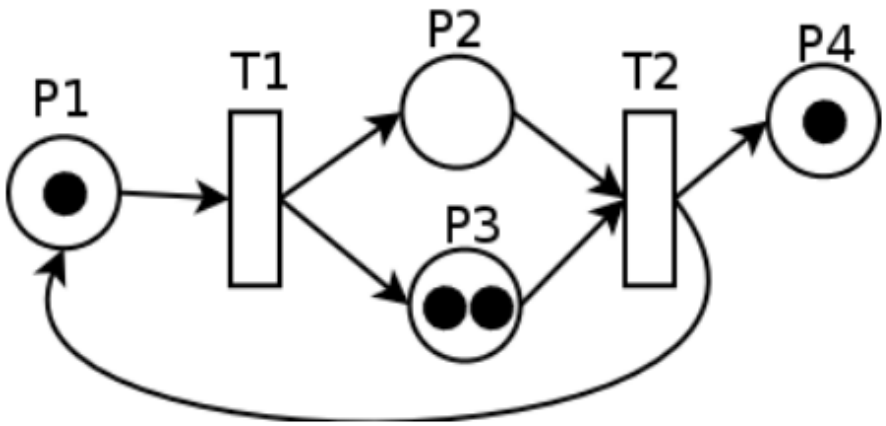

Abbildung 2.2: Activity Diagram

Activity Entspricht einem Verhalten im System und besteht aus mehereren Actions.

Abbildung 2.3: Activity

Action Eine Aktion ist ein einzelner Schritt in einer Activity. Muss aber nicht Atomar sein (kann also aus weiteren Activities und Actions bestehen)

Abbildung 2.4: Action

Start und Endknoten Start enthält genug Token für die Eingaben, Feuert am start der Aktivität. Wen ein Token den Ende Knoten erreicht ist die Activity zu ende.

Kanten Die Kanten des Diagramm besitzt einen Namen und weitere Inforamtionen wie den Fluss (Richtung, Bedinungen, weiteres Verhalten). Zusätzlich spezifiert wird die Kante über Stereotypes.

Weitere Informationen zu Kanten:

- Gewicht {Geschwungenen Klammern}
- Guard (Filter oder Evaluationen) [Eckige Klammern]
- Zuerst Guard dann Gewicht überprüfen

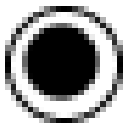

Abbildung 2.5: ActivityFinalNode

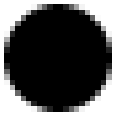

Abbildung 2.6: InitialNode

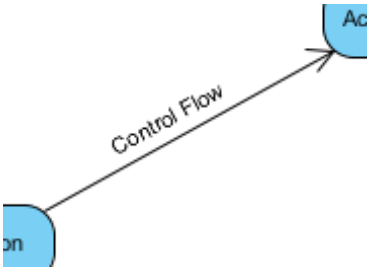

Abbildung 2.7: Knoten

Ein und Ausgänge (In/Output Pins) Entspricht Parameter in der Softwareentwicklung. Damit kann bestimmt werden welche Eingaben welche Rollen spielen. Damit können verschiedenen Eingabeparameter verschiedene Verhalten zugewiesen werden.

Objekt Knoten Repräsentiert Objekte die in Aktionen verwendet werden.

Parameter Knoten Erweiterte Version von Ein und Ausgängen

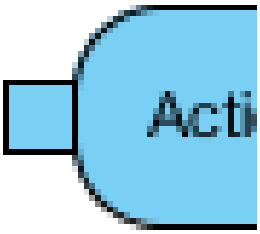

Abbildung 2.8: InputPin

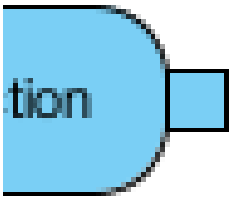

Abbildung 2.9: OutputPin

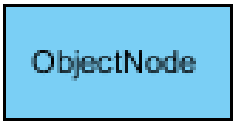

Abbildung 2.10: ObjectNode

Entscheidungsknoten Nur ein Ausgang wird gefeuert. Bedinung für jeden Ausgang an der jeweiligen Kante.

Merge Es reicht wenn ein Eingang gefeuert wurde, damit der Ausgang gefeuert wird.

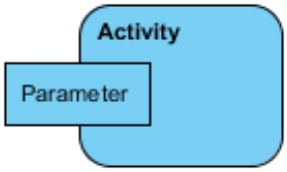

Abbildung 2.11: ParameterNode

Abbildung 2.12: DecisionNode

Abbildung 2.13: MergeNode

Fork / Join Erzeugt / Synchronisiert Parallele Flüsse.

Abbildung 2.14: Fork / Join Node

Advanced Activity Modelling

Einführung von Activity Partitions, also mehrere Swimlanes. Swimlanes können später noch weiter unterteilt werden (FHV kann im Laufe des Prozesses z.B. noch in einzelne Räume unterteilt werden).

Der external Stereotype beschreibt einen Bereich, der nicht von uns konstruiert wird, sondern von außen vorgegeben ist.

Wenn auf eine große Menge von Daten die gleiche Aktion angewendet werden soll, dann können sogenannte Expansion Regions verwendet werden. Dazu gibt es optionale Stereotypen für die Verarbeitung in einem iterativen, parallelem oder einem Streaming-Prozess. Die Verwendung dieser Stereotypen ist zum Teil allerdings an gewisse Hardware gebunden.

Darüberhinaus gibt es auch noch für die Darstellung von Loops die Standard Loops und Loop Nodes. Bei den Loop Nodes werden drei Bereiche angegeben: Setup, Test und einen ausführenden Body.

Streaming Actions können Output generieren, während dem man einen Input erhaltet.

Central Buffer stellen Puffer als Knoten dar, was z.B. bei einer Produktionskette ganz nützlich sein kann. Eine Erweiterung dazu stellt der Data Store Node dar, auf welchen auch noch Daten gespeichert werden. Diese Daten können auch später noch einmal abgerufen werden.

Statecharts

In vielen Fällen hängt das Verhalten eines Objekts von seinem Zustand ab, und die Zusammenhänge zwischen diesen Zuständen werden in einem Statechart mit Zustandsübergangen gezeigt. Wenn ein Übergang zwischen zwei States nicht definiert ist, könnte man das als nicht erlaubter Übergang interpretieren.

Status Ein Status haltet immer eine invariante Bedingung, die sich nicht ändert, solange man den Status nicht wechselt. Diese Bedingung ist im Normalfall implizit definiert.

Es gibt verschiedene Verhalten bei einem Status. Am Anfang gibt es ein Eintrittsverhalten, welches ausgeführt wird, sobald man den Zustand betritt. Während dem Verharren in einem Zustand gibt es ein Verhalten, welches gültig ist, solange der Status aktiv ist. Zuletzt gibt es noch ein Austrittsverhalten, welches beim Verlassen des Status ausgeführt wird.

Status können auch verschachtelt, und in dieser Verschachtelung auch noch in Swimlanes angeordnet werden. Man kann auch Submaschinenstatus verwenden, um die mehrfache Verwendung des gleichen Status in verschiedenen Diagrammen zu verwenden.

Zustandsübergang An einem Zustandsübergang hängt ein Trigger, welcher beschreibt bei welchem Event der Übergang eintritt. Dazu gibt es auch noch einen Guard, also ein boolscher Ausdruck, welcher wahr sein muss um den Übergang zu erlauben. Zuletzt gibt es einen Effekt, der beim Zustandsübergang ausgeführt wird.

Pseudostatus Es gibt auch sogenannte Pseudostatus, allen voran der Startzustand. Allerdings ist der Endzustand kein Pseudozustand, da an diesem wirklich etwas ankommt.

Darüber hinaus gibt es auch noch einige andere Pseudostatus, wie z.B. der Entscheidungsknoten, das Forken und Joinen, das Zusammenführen usw.

Signal Dieser Teil ist vor allem für Embedded Software wichtig, damit kann man das Empfangen und Versenden von Signalen modellieren.

3 Produktmanagment

Jedes Produkt hat verschiedene Lebenszyklen:

- product business lifecycle
- product engineering lifecycle
- manufacturing engineereing lifecycle
- product support lifecycle
- operator lifecycle

Je nachdem in welchem Zyklus sich ein Produkt befindet hat es mehr oder weniger Auswirkungen auf Entscheidungen von Produktmanagern –> z.B. wegen Rentabilität | Break Even Point.

Wichtig im Bereich des Produktlebenszyklusmanagement sind die Bereiche Herstellungs- und Produktionsprozesse. Hier kann unterschieden werden zwischen:

- Kontinuierlicher Produktion
- Stapelproduktion
- Diskrete Herstellung
    1. Einzigartiges Produkt
    2. Produkt in geringem Umfang mit und ohne Varianten

Der Produktmanager ist das Bindeglied zwischen Produktion, Marketing, Vertrieb usw. und muss vermitteln. Außerdem sollte er deb vielen verschiedenen Breichen (Technik, Wirtschaftlich, Marketing) ein wenig Ahnung und auch Verständniss dafür haben.

Neben dem Wissen über die ganzen Teilgebieten (Produktion, Marketing, Verkauf) sollte ein guter Produktmanager auch wissen wieso Kunden ein Produkt kaufen oder auch nicht. (Beispiel v. Prof.)

	Product Definition Lifecycle	Process Definition Lifecycle	Physical Product Lifecycle	Prod. Equipment Lifecycle
Continuous Process Production				
Batch Production				
Discrete Manufacturing				
Unique Product				
Low Volume Product				
no Variants				
with Variants				
Mass Product				
no Variants				
with Variants				

Abbildung 3.1: Klassifizierung von Industrien (Rot ist das Wichtigste)

Abbildung 3.2: Produktmanagerverantwortungen

3.1 Anforderungsmanagementprozess

Hier wird festgestellt ob Anforderungen gültig sowie konsisten untereinander (keine Wiedersprüche) sind. Außerdem wird überprüft ob es keine Duplikate gibt und die gesamte Sammlung der Anforderungen valide ist (nichts vergessen und nichts was entfernt werden sollte).

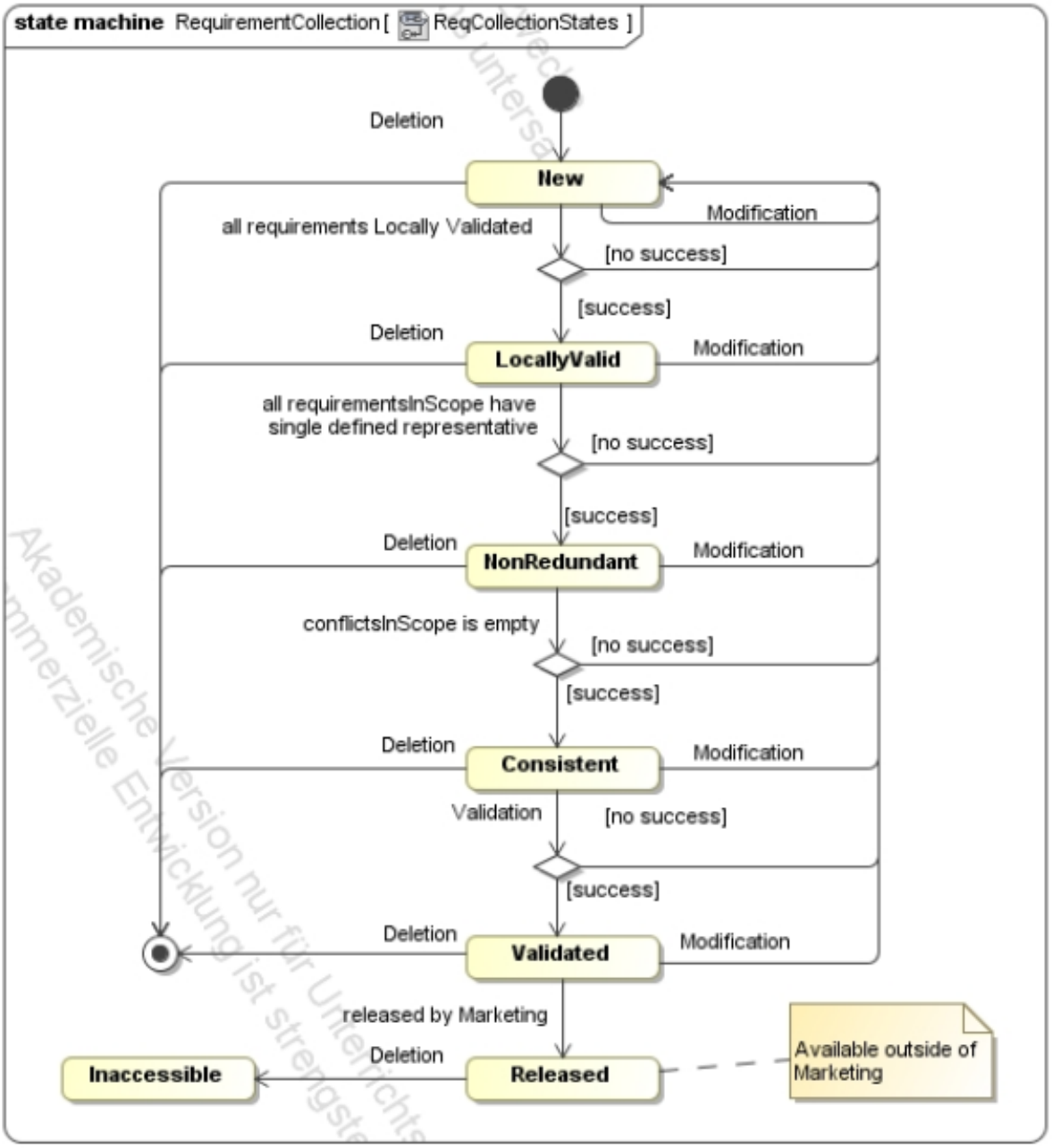

Abbildung 3.3: Anforderungsmanagement-Prozess