In diesem Kapitel:
- *Schwierigkeiten der Anforderungsphase*
- *Ablauf der*
- *Übersicht*
- *Bedeutung der Anforderungen im Software Entwicklungsprozess*
- *Mit dem Domänen Modell arbeiten*
- *Mit den Business Modell arbeiten*
- *Zusätzliche Anforderungen*
- *Zusammenfassung*

12 ROP Anforderungsanalyse (requirement capture)

Nachdem wir die grundlegenden Konzepte verstehen oder kennen, geht es darum, die grundlegenden Abläufe kennen zu lernen. Darum geht es ab diesem Kapitel.

Wir werden im Folgenden die grundlegenden Abläufe der organisatorischen Phasen "Anforderungen, Analyse, Design, Implementation, Test" kennen lernen und formal beschreiben und mit einfachen Beispielen illustrieren. Begleitend werden wir unser komplexeres Anwendungsbeispiel weiterführen und implementieren.

Auf den ersten Blick irritiert es Sie eventuell, dass wir vorher immer über Iterationen und Spiralmodelle sprachen und nun ein lineares Modell besprechen. Linear ist das Modell nur, weil wir im Folgenden keine Iterationen besprechen und uns stattdessen auf die Inhalte der einzelnen Phasen konzentrieren. Die einzelnen Phasen werden in realen Projekten mehrfach durchlaufen; dadurch entsteht das Spiralmodell.

Wann immer Software Entwickler mit einem Entwicklungsauftrag konfrontiert werden, stehen sie vor vielen noch unbekannten Problemen.
- In der Regel wird der Software Entwickler die entwickelte Software nicht selber einsetzen. Der Software Entwickler bekommt keinen direkten Feedback bezüglich der Nützlichkeit, Brauchbarkeit, ... der entwickelten Software
- Die Systemanforderungen sind nicht wie bei anderen Produkten seit Jahrhunderten bekannt. Es besteht noch viel Unsicherheit beim Entwickeln neuer Systeme.

Beim "*Requirement Capture*" geht es darum, möglichst ein brauchbares Bild betreffend der Benutzeranforderungen zu erhalten. Oft wird auch mit der Programmierung begonnen, bevor die Anforderungen im Detail bekannt sind.

12.1. Warum ist das Bestimmen der Benutzeranforderungen so schwierig?

Da die Software in der Regel für jemanden entwickelt wird, also nicht für den Selbstgebrauch, muss man auf irgend eine Art und Weise feststellen und festhalten können, wer welche Anforderungen an das System hat oder haben wird.

Die Annahme, dass der Benutzer genau weiss, was er braucht, ist sehr riskant, aus verschiedenen Gründen:
- der Benutzer kennt die Möglichkeit nur ungenügend
- der Benutzer ist vielleicht betriebsblind

Falls der Benutzer nun hingeht und eventuell mit fremder Hilfe ein Anforderungsdokument von einigen hundert Seiten schreibt, ist das zwar eine nette Beschäftigung, und führt in der Regel auch zu unerwarteten Ergebnissen, Ablaufverbesserungen, die man schon lange hatte durchführen sollen. Aber es ist absurd anzunehmen, dass ein solches Dokument konsistent und vollständig sein könnte. Viele Anforderungen werden in der Regel überhaupt nicht berücksichtigt.
Auch mit Hilfe von Business Process Managern sieht die Situation nicht wesentlich rosiger aus. In der Regel besteht dann ein Kommunikationsproblem zwischen dem Benutzer und dem Analysten. Der Benutzer erahnt erst fast zum Zeitpunkt der Fertigstellung, was das neue System wirklich können wird. Viele Änderungen, die der Benutzer gewünscht hat, bleiben unberücksichtigt oder folgen später.
In den letzten Jahren hat man sich in der Regel eingeredet, dass der Benutzer weiss, was er braucht. Es liegt also nur an uns, den Benutzer zu fragen, zu interviewen.
Die Realität sieht anders aus:
es stimmt zwar, dass man den Benutzer befragen muss. Aber noch wichtiger ist es, dass die Software die *Ziele* unterstützt, die man mit einem neuen Informationssystem erreichen will. Probleme gibt es, falls die Entwicklung sehr lange dauert und sich in der Zwischenzeit die Zielsetzungen ändern.

12.2. Ziele des Requirements Workflows

Das wesentliche Ziel der Anforderungsphase (Requirements Workflow) besteht darin, das richtige System zu entwickeln. Dies erreicht man, indem man die Systemanforderungen erfasst und festhält (Bedingungen, Fähigkeiten, die das System erfüllen, haben muss). Diese Systembeschreibung muss so präzise sein, dass ihr der zukünftige Kunde und der Entwickler zustimmen können.
Wichtig ist, dass der Benutzer des zukünftigen Systems diese Spezifikation versteht, obschon er kein Informatikspezialist ist. Damit wir dies erreichen, muss die Spezifikation in der *Sprache der Anwender* geschrieben sein. Wir müssen entsprechend vorsichtig sein bei der Modellierung und der Wahl der Modellierungskonzepte.
Die Ergebnisse der Anforderungsphase helfen auch dem Projektmanager, die Iterationen zu planen und zu entscheiden, welche Funktionalität wann realisiert wird und verfügbar wird.

12.3. Übersicht über den Ablauf

Jedes Softwareprojekt ist einzigartig, leider immer noch. Ursache dafür sind die unterschiedlichen Systemanforderungen auf Grund unterschiedlicher Voraussetzungen beim Anwender. Es gibt auch unterschiedliche Ausgangspunkte für jedes Projekt.

In einigen Fällen startet man mit der Entwicklung eines Business Modells, oder man verwendet ein Business Modell aus einem andern Projekt oder ein Referenzmodell.

In andern Fällen handelt es sich beim System um ein eingebettetes System, welches nichts direkt mit dem Kerngeschäft zu tun hat.

Wieder in andern Fällen kann es vorkommen, dass der Kunde bereits ein detailliertes Objektmodell besitzt, zum Beispiel aus andern Projekten, so dass man dieses als Ausgangspunkt verwenden kann und allfällige Anpassungen daran verdeutlichen kann.

Im andern Extremfall besitzt der Kunde nur eine vage Vorstellung dessen, was er eigentlich möchte. Vielleicht hat er diese Anforderungen aus seiner Vision hergeleitet, bleibt also zwangsweise eher im Abstrakten.

Beispiel **Das Interbanken Konsortium beschliesst eine neues Computersystem zu installieren**

Das Interbanken Konsortium, eine hypothetische Finanzinstitution, muss sich auf Grund dramatischer Veränderungen, wie Deregulation, neuen Wettbewerbern und neuen Geschäftsmöglichkeiten auf Grund des World Wide Webs, den neuen Herausforderungen anpassen. Das Konsortium beschliesst, dass neue Applikationssoftware beschafft werden soll, mit deren Hilfe diese schnellen Veränderungen positiv genutzt werden können. Die Tochterfirma Interbank Software soll die Entwicklung übernehmen und als erstes das Gesamtprojekt initialisieren..
Interbank Software beschliesst, ein Zahluns- und Rechnungswesen System zusammen mit den Hauptkunden zu entwerfen. Das System wird über das Internet Bestellungen, Rechnungen und Zahlungen zwischen Verkäufern und Käufern abwickeln können. Die Motivation für die Entwicklung einer neuen, Internet-basierten Software kommt von daher, dass die Bank kostengünstige Transaktionen für ihre Geschäfte anbieten möchte. Dies impliziert fast automatisch den Einbezug des Internets, da eine elektronische Transaktion einige Rappen, eine manuelle aber einige Franken kostet.
Die Motivation der Käufer und Verkäufer an der Spezifikation des Systems teilzunehmen ist auch im Wunsch begründet, Kosten zu reduzieren, auf das Papier weitestgehend zu verzichten und die Transaktionen in kürzerer Zeit abwickeln zu können. Zum Beispiel müssen keine schriftlichen Bestellungen auf Papier versandt werden. Der Kundenrechner kommuniziert direkt mit dem Anbieterrechner. Bestellungen und Rechnungen werden elektronisch weiter geleitet. Käufer und Verkäufer haben damit auch eine wesentlich bessere Übersicht über getätigte Transaktionen.

Die Möglichkeit so vieler unterschiedlicher Startpunkte, von der Vision bis zum detaillierten Konzept, impliziert, dass die Analyse der Benutzeranforderungen flexibel sein muss und leicht an die Gegebenheiten angepasst werden kann. Wesentlich für solche Anpassungen ist das Ziel, Systeme ohne zu grosses Risiko entwickeln zu können. Wir kommen noch auf die Risikoabschätzung zurück.

Trotz der unterschiedlichen Ausgangspunkte werden in jedem Projekt bestimmte Schritte durchlaufen, sozusagen archetypische Arbeitsschritte.. Diese Arbeitsschritte und Arbeitsabläufe werden wir im Folgenden für den UML basierten Rational Objektory / Unified Process aber auch allgemeiner kennen lernen.:

- Auflisten möglicher Anforderungen
- Verstehen des Systemkontextes
- Festhalten funktionaler Anforderungen
- Festhalten genereller Anforderungen

Fangen wir einfach mit dem ersten Punkt an:

Auflisten möglicher Anforderungen

Während des Lebenslaufes eines Systems werden Kunden, Entwickler und Benutzer viele gute Ideen haben, die lediglich zum Teil in ein Produkt einfliessen können. Wir sollten aber eine Liste mit all diesen Ideen anlegen, da einige der Ideen in späteren Versionen unseres System berücksichtigt werden können. Diese „*Featuresliste*“ oder „Wunschliste“ wächst und lebt, weil dauernd neue Ideen dazu kommen können. Einige Ideen können auch gestrichen werden, entweder weil sie nicht realisierbar oder aber bereits realisiert sind. Diese *Anforderungsliste* bildet die Basis für die Projektplanung.

Jede gewünschte Funktion / jedes gewünschte Feature erhält einen kurzen Namen und eine Erklärung oder Definition, einfach soviel Informationen, dass wir auch wissen worüber wir sprechen. Jede Funktion besitzt bestimmte Planungsattribute:

- Status (zum Beispiel : bewilligt, eingebaut oder validiert)
- Geschätzte Kosten für die Implementation (Resourcen und Aufwand)
- Priorität (zum Beispiel : kritisch, wichtig, zusätzlich)
- Risikolevel, mit dem bei der Implementation gerechnet werden muss (kritisch, signifikant, nicht aussergewöhnlich)

Diese Planungswerte werden zusammen mit andern Aspekten verwendet, um die Projektgrösse abzuschätzen und zum Beispiel die Anzahl Iterationen festzulegen.

Prioritäten und Risiko Level (pro Anforderung) werden benötigt, um festzulegen, in welcher Iteration welche Anforderung implementiert werden soll. Zusätzlich muss ein Zusammenhang zwischen dem Implementationsplan und den Anwendungsfällen erkennbar sein.

Mit diesen Werten müssen wir versuchen, die Grösse der Projekte abzuschätzen, so gut es geht. Auch die Anzahl Iterationen und der Inhalt der Iterationen muss geplant und festgelegt werden.

Prioritäten und Risikolevel helfen beim Entscheiden, in welcher Iteration welche Funktionen implementiert werden sollten, und wann. Alle externeb Features müssen auf Use Cases, also Anforderungen des Benutzers zurückführbar sein.

Den Systemkontext verstehen

Viele Personen, die am Software Entwicklungsprozess beteiligt sind, sind Softwarespezialisten, in der Regel Spezialisten für Teilgebiete der Software.

Damit das System korrekt, speziell das richtige System realisiert wird, muss sehr früh ein Architekt und Schlüsselentwickler beigezogen werden. Diese benötigen ein vertieftes Verständnis der Geschäftsprozesse.

Es gibt mindestens zwei Arten, die Abläufe und das gesamte System aufzubereiten und dem Software Entwickler gegenüber zu spezifizieren: domain modelling und business modelling, also Modellierung des Anwendungsgebietes oder Geschäftsmpdellierung.

Ein Domain Modell beschreibt die wichtigsten Konzepte als Domainobjekte und verbindet diese Objekte untereinander / miteinander. Die Benennung und die Identifikation dieser Objekte hilft, ein Verständnis zu erreichen, über Begriffe, die in diesem Anwendungsbereich üblich sind. Dies hilft Kommunikationsprobleme zu reduzieren. Die Anwendungsobjekte helfen uns später die Klassen und Objekte des zu implementierenden Systems zu identifizieren. Das Analyse und Design-Modell unterscheidet sich in der Regel wesentlich vom Domainmodell, baut aber auf diesem auf. Ein wichtiger Schritt beim Übergang ist die

Abstraktion. Implementationsmodelle sind in der Regel abstrakter als das konkrete Domainmodell.

Das Business Modell ist oft ein Obermodell des Domainmodells. Es umfasst oft mehr Objekte, als das Domainmodell, ist also umfassender.
Das Ziel des Businessmodells istes, Geschäftsabläufe zu beschreiben - existierende und geplante - um diese besser verstehen zu können.

Business Modelling ist Teil des Business Engineerings, welches unter anderem zum Ziel hat, Prozesse zu verbessern.

Beim Modellieren der Geschäftsabläufe lernt der Analyst sehr viel über den Kontext, in dem die Software eingesetzt werden wird. Das Business Modell beschreibt, welche der Geschäftsprozesse durch das System unterstützt werden. Zudem lassen sich aus den Geschäftsprozessen die Resourcen bestimmen, die man benötigt, um sie zu implementieren, also die Fachspezialisten.

Diese Ergebnisse sind fundamental für den Erfolg eines Projektes. Business Engineering ist die systematischste Art und Weise zur Erfassung der Anforderungen an ein neues Informationssystem.

Festhalten funktionaler Anforderungen
Der direkteste Weg zu einer Anforderungsliste sind die Use Cases. Use Cases, Anwendungsfälle, beschreiben sowohl funktionale als auch nicht-funktionale Anforderungen.

Fassen wir noch einmal kurz zusammen, wie Anwendungsfälle uns helfen, die korrekten Anforderungen zu erfassen:
jeder Benutzer möchte, dass das System bestimmte Aufgaben erfüllt. Wenn es uns gelingt, diese Anforderungen in Form von Anwendungsfällen zu erfassen, sind wir ziemlich sicher, dass wir ein brauchbares System bauen.

Jeder Anwendungsfall zeigt eine Sicht, wie das System benutzt werden kann. Jeder Benutzer wird in der Regel mehr als einen Anwendungsfall einsetzen. Wenn wir die Anwendungsfälle festhalten, die das Geschäft wesentlich unterstützen, werden wir ein System erhalten, welches einen echten Businessvalue darstellt. Daher müssen wir den Systemkontext kennen und in Interviews, Reports und Workshops erfragen.

Als ein Nebeneffekt dieser Arbeiten müssen die Analysten auch die Benutzerschnittstelle spezifizieren und dem Benutzer zu erläutern, wie er seine Anforderungen später im neuen System wieder finden wird.

Dieses Vorgehen entspricht einem Prototyping. Das Modell ist also nicht immer anwendbar. Wie weit dieser interaktive Prozess geht, hängt auch von der Komplexität des Projektes ab. Bei komplexeren Projekten sind detailliertere Abklärungen nötig, eventuell eine Zerlegung des Projektes in mehrere Teilprojekte. Falls der Benutzer mit einem neuartigen Benutzerinterface noch nicht vertraut ist, muss es entsprechend geschult werden, es sind also mehrere Iterationen mit dem Benutzer nötig.

Festhalten nichtfunktionaler Anforderungen
Nichtfunktionale Anforderungen spezifizieren Anforderungen an das System, welche zum Beispiel die Performance, die Wartbarkeit, die Zuverlässigkeit betreffen.

Die Zuverlässigkeit kann man die Anzahl Fehler pro Tausend Zeilen Programmcode.

Die Leistung lässt sich mit Hilfe der Anzahl Transaktionen pro Sekunde oder einer andern Zeiteinheit messen.

Beispiel **Spezielle Anforderungen für den Anwendungsfall "Rechnung bezahlen"**

Leistungsanforderungen
Wenn ein Käufer eine Rechnung Online anzeigen lassen möchte, muss diese Transaktion innerhalb einer Sekunde machbar sein, zumindest in 90% aller Fälle. Die Verifikation der Überweisung darf nie länger als 10 Sekunden dauern, ausser die Netzwerkverbindung ist unterbrochen.

Einige nichtfunktionale Anforderungen hängen mit Anforderungen der beteiligten, wie zum Beispiel der Bank, zusammen. Andere lassen sich eindeutig einem Anwendungsfall zuordnen.

Zusätzliche Anforderungen, die nicht einem Anwendungsfall zugeordnet werden können, werden in einem separaten Dokument "Zusätzliche Anforderungen" zusammengefasst.

Zusammenfassung
Um die Anforderungen an ein System erfassen zu können, müssen unterschiedliche Techniken eingesetzt werden. Die resultierenden Artifakte beschreiben zusammen die Anforderungen an das System. Die traditionelle Anforderungsanalyse aus den alten Prozessmodellen, werden ersetzt durch diese Artifakten, Anwendungsfälle, zusätzliche Anforderungen, Business Modelle u.ä.

Da sich die Anforderungen dauernd ändern, müssen wir diese bei jeder Iteration noch einmal kurz validieren, nicht neu aufnehmen. Bei jeder Iteration werden die Anforderungen präziser fassbar und bei der Implementation werden wir die Anforderungen definitiv fixiert haben.

Pendenzenliste	**resultierende Artifakte**
Liste der Anforderungskandidaten aufstellen	Funktionsliste
Systemkontext verstehen	Business oder Domain Modell
funktionale Anforderungen festhalten	Use Case Modell
nichtfunktionale Anforderungen festhalten	zusätzliche Anforderungen oder zusätzliche Anforderungen pro Anwendungsfall

12.4. Die Rolle der Anforderungen im Software Lebenszyklus

Ein Use Case Modell entwickelt man iterativ, also schrittweise. Jede Iteration fügt weitere Use Cases hinzu und / oder weitere Detaillierungen eines Anwendungsfalles.

Die folgende Abbildung zeigt, wie der Anforderungsprozess sich im Laufe des Projektablaufs ändert

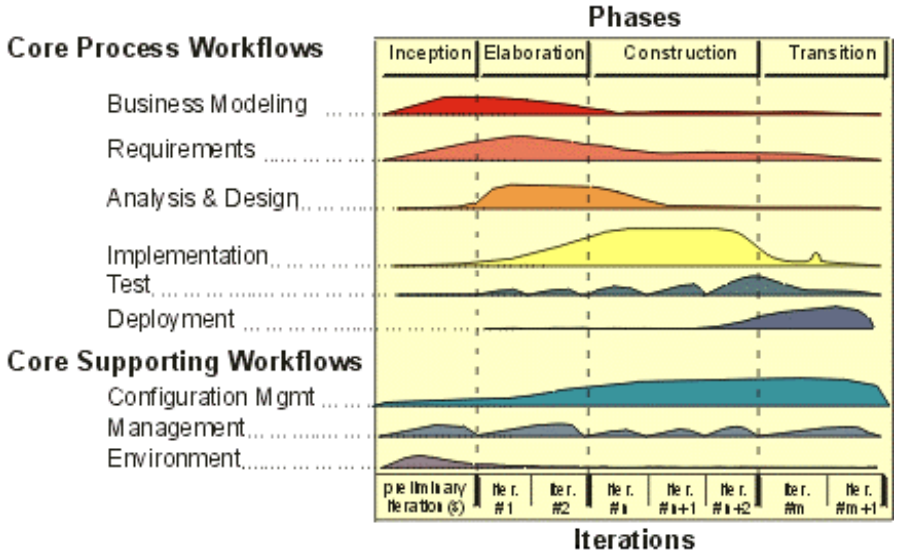

Abbildung 0-1 Anforderungen werden in der Startphase und der Entwurfsphase erfasst

Die obige Abbildung illustriert, dass Anforderungen in mehreren Schritten erfasst werden:

- in der Startphase ("Inception") werden die meisten Anwendungsfälle durch die Analysten erfasst, um die Systemgrenzen festlegen zu können und die kritischen Anforderungen eventuell weiter zu verfeinern (maximal 10% der Anforderungen).
- in der Entwurfsphase ("Elaboration") halten die Analysten die meisten der restlichen Anforderungen fest. Dies ist notwendig, um eine Aufwandschätzung für die nächsten Aktivitäten abgeben zu können. Das Ziel ist es, 80% der Anforderungen erfasst zu haben und die meisten Anwendungsfälle am Ende der Entwurfsphase beschrieben zu haben. 5-10% dieser Anforderungen haben einen direkten Einfluss auf Architekturentscheide und müssen als "provisorische Architektur" festgehalten werden.
- die restlichen Anforderungen werden in der Konstruktionsphase ("construction") erfasst und implementiert.
- in der Übergangsphase ("transition") werden in der Regel keine neuen Anforderungen mehr akzeptiert, ausser die Anforderungen ändern sich und können noch berücksichtigt werden.

12.5. Den Systemkontext mit Hilfe eines Domain Modells verstehen

12.5.1. Was ist ein Domain Modell

Ein Domain Modell hält die wichtigsten Objekttypen im Systemkontext fest. Die Domain Objekte stellen "Dinge" dar, die echt existieren oder Ereignisse, welche im Systemumfeld geschehen.

Viele der Domain Objekte und Klassen können aus der Anforderungsanalyse hergeleitet oder erfragt werden. Es gibt drei Typen von Domain Klassen:

- Business Objekte:
  diese stellen Dinge dar, welche im Business manipuliert werden, wie zum Beispiel Bestellungen, Konten oder Kontrakte
- Objekte der realen Welt und Konzepte, welche das System zeitlich gesehen verfolgen muss, wie zum Beispiel die Bahn einer Rakete, eines Roboters, ...
- Ereignisse:
  diese geschehen während der Lebensdauer des Systems, wie zum Beispiel Landen eines Flugzeuges, Mittagessen, ...

Das Domain Modell wird mit Hilfe von UML beschrieben, speziell mit Hilfe von Klassendiagrammen. Diese Diagramme illustrieren, oder sollten dies wenigstens, dem Benutzer, dem Reviewer, dem Kunden oder andern Beteiligten wie die einzelnen Klassen zusammenhängen.

Beispiel **Die Domain Klassen**
Bestellung, Rechnung und Konto

Das System verwendet das Internet, um Rechnungen, Bestellungen und Zahlungen zwischen dem Käufer und dem Anbieter hin- bzw. herzusenden. Das System hilft den Beteiligten auch, diese Dokumente vorzubereiten und Zahlungen zu validieren.
Eine Bestellung ist eine Anforderung eines Käufers an einen Verkäufer, für eine bestimmte Anzahl Artikel. Jeder Artikel "besetzt eine Zeile" in der Bestellung. Jede Bestellung besitzt Attribute, wie zum Beispiel Datum der Rechnungstellung, Datum der Bestellung, Datum des Zahlungseinganges....
Eine Rechnung kann mehrere Lieferungen umfassen. Eine Lieferung kann mehrere Bestellungen umfassen.
Eine Rechnung wird bezahlt, indem Geld vom Kundenkonto zum Lieferantenkonto transferiert wird. Ein Konto besitzt Attribute, wie zum Beispiel einen Kontostand, einen Kontoinhaber, ...

Fassen wir diese Erkenntnisse in Form eines Klassendiagramms zusammen, ergibt sich folgendes Diagramm:

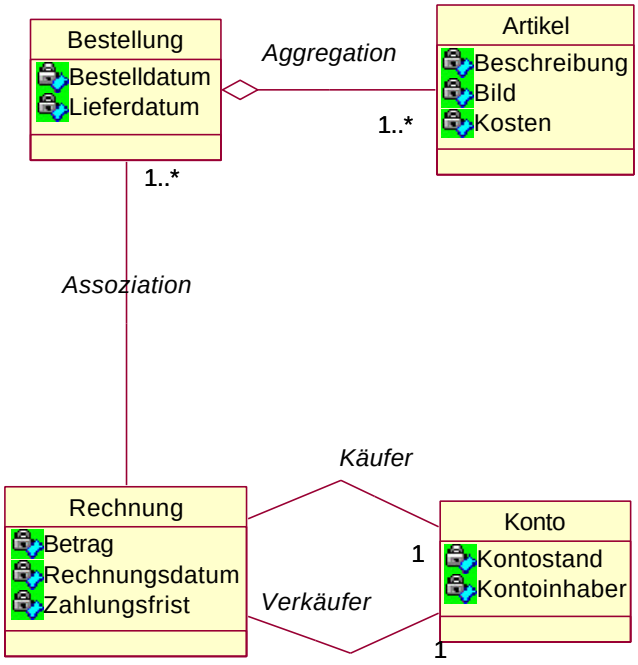

Abbildung 0-2 Klassendiagramm im Domain Modell : nur die wichtigsten Klassen werden festgehalten

UML Notation

Klassen (Rechtecke), Attribute (Text) und Assoziationen (Linien, Verbindungen zwischen den Klassen). Der Text am Ende der Verbindung einer Assoziation sagt aus, welche Rolle eine Klasse in Relation zu einer andern Klasse hat. Die Multiplizität - die Zahlen und Sterne am Ende eines Assiziationplans, sagt aus, wieviele Objekte der Klasse an diesem Ende einen Beziehung zu einem Objekt am andern Ende der Assoziation besitzt.
Beispiel:
die Assoziation zwischen Bestellung und Rechnung besagt, dass eine Rechnung, ein Rechnungsobjekt, eine Aufforderung zur Bezahlung einer oder mehrerer Bestellungen ist: 1..*

12.5.2. Entwicklung von Domain Modellen

Domain Modelle wird in der Regel in der Form von Workshops erarbeitet. Die Diagramme werden vom Analysten erstellt, in Form von UML oder ähnlichen Diagrammen. Damit diese Workshops Ergebnisse liefern können, muss in der Regel das Team optimal zusammen gesetzt werden. Es werden Domain Experten genau so benötigt, wie Analysten und Personen, die in der Lage sind Modelle zu erstellen.

Ziel des Domain Modells ist es, die wichtigsten Klassen des Anwendungskontextes zu verstehen und zu beschreiben. Mittlere Projekte führen zu 10 - 50 solcher Klassen. Grössere Projekte führen zu entsprechend mehr Klassen.

Die restlichen, eventuell mehrere hundert Kandidaten für Klassen, werden in eine Liste

eingetragen und erläutert (als Glossar). Falls man alle Kandidaten ins Diagramm eintragen würde, wäre das Diagramm viel zu unübersichtlich und würde auch viel zu viel Aufwand zur Folge haben.

Falls das Projekt klein ist, kann es auch sinnvoll sein, das Domain Modell ganz wegzulassen und statt dessen einfach ein Glossar zu erstellen.

Glossar und Domain Modell helfen dem Benutzer, Entwickler und andern Stakeholdern / Beteiligten ein gemeinsames Vokabular aufzubauen. Gemeinsame Sprache ist notwendig, um Wissen austauschen zu können. Konfusion oder Mehrdeutigkeit der Begriffe führt zu Problemen und Missverständnissen.

Domain Modelle können sehr effizient sein, wenn sie richtig eingesetzt werden:
- Ziel darf nicht sein, ein perfektes Modell zu erstellen;
- Ziel ist es, ein besseres Verständnis zu gewinnen, über das zu entwickelnde System.

Man sollte also mit einem Domain Modell nicht unnötig noch mehr Probleme schaffen als Verständnis zu gewinnen.

12.5.3. Einsatz von Domain Modellen

Die Domain Modell Klassen und das Glossar werden bei der Entwicklung der Anwendungsfälle sowie des Analyse Modells eingesetzt. Die Klassen werden benutzt:
- beim Beschreiben der Anwendungsfälle und beim Design der Benutzerschnittstelle
- um Klassen vorzuschlagen, die innerhalb des zu entwickelnden Systems in der Analyse benutzt werden, um Analyseklassen zu definieren.

Eine systematischere Art und Weise die Anwendungsfälle zu definieren ist das Business Modell, auf das wir noch zu sprechen kommen. Domain Modelle sind auf eine Art ein Spezialfall eines kompletten Business Modells. Die Entwicklung eines Business Modells ist somit eine mächtige Alternative zur Entwicklung eines Domain Modells.

12.6. Den Systemkontext mit Hilfe eines Business Modells verstehen

Business Modelling ist eine Technik, ein besseres Verständnis über die Geschäftsprozesse einer Organisation zu entwickeln.

Wie können wir vorgehen, wenn das Business nichts mit dem zu tun hat, was wir unter Business verstehen?

Beispiel:
wie sollen wir vorgehen, wenn wir einen Kameracontroller entwickeln müssen?

In diesem Fall ist die Kamera als Ganzes das Business und wir modellieren ein Teilsystem dieses Businesses.

Ein Anwendungsfall ist in diesem Sinne aus Sicht des umgebenden Systems zu definieren, nicht unbedingt aus Sicht eines Benutzers, der mit dem zu entwickelnden System nicht direkt in Verbindung steht.

Ziel ist es, Anwendungsfälle der Software und relevanten Business Entitäten zu identifizieren. Das Ergebnis dieser Aktivitäten ist ein Domain Modell, welches aus einem Verständnis des Funktionierens des Business Systems hergeleitet ist, also aus einem umfassenderen Systemverständnis.

Business Modelling wird durch zwei UML Modelle unterstützt:
- Use Case Modelle (Anwendungsfälle)
- Objektmodelle

12.6.1. Was ist ein Business Modell?

Ein Business Use Case Modell beschreibt einen Geschäftsprozess einer Firma, ausgedrückt in Form von Geschäftsfällen und Geschäftsaktoren, entsprechend den Geschäftsprozessen und den Geschäftspartnern.

Wie die Use Cases eines Software Systems stellen die Business Use Cases ein Modell eines Systems dar (hier des Business), aus Sicht des Benutzers; der Business Use Case zeigt, welchen Wert ein bestimmter Geschäftsfall für den Benutzer hat, welchen Wertzuwachs der Prozess bringt.

Beispie Business Use Cases

Das Interbanken Konsortium Beispiel offeriert einen Business Case, der das Senden von Rechnungen, das Überweisen von Zahlungen zwischen Käufer und Verkäufer umfasst - Verkauf : von der Bestellung , Lieferung bis zur Rechnungsbegleichung.
In diesem Business Use Case weiss der Käufer, was er bestellen will und von wem.

Die folgende Sequenz verwendet die Interbank als Broker in diesem Business Use Case, indem der Käufer und Verkäufer miteinander verbunden werden, und eine sichere Verbindung für Zahlungsüberweisungen besteht:
1. der Käufer bestellt die Ware oder Dienstleistungen
2. der Verkäufer liefert die Ware oder Dienstleistungen
3. der Verkäufer stellt dem Käufer eine Rechnung
4. der Käufer bezahlt die Rechnung

In diesem Kontext sind der Käufer und der Verkäufer die Business Actors der Interbank, und Sie verwenden Business Anwendungsfälle / Geschäftsprozesse, welche die Interbank anbietet.

Ein Geschäft stellt normalerweise viele Geschäftsfälle zur Verfügung; Interbank ist keine Ausnahme. Betrachten wir zwei Geschäftsfälle dieses Interbanksystems:
- der Bankkunde ist ein generischer Kunde der Bank. Käufer und Verkäufer sind spezifischere Formen des Kunden
- Transfer, Abheben oder Einzahlung sind Business Cases / Geschäftsfälle der Bank

---

Das Business Use Case Modell kennen wir bereits aus einem früheren Kapitel! Es beschreibt, wie jeder Use Case von Aktoren, Business Workers eingesetzt wird. Jeder Geschäftsfall kann als Sequenzdiagramm oder Kollaborationsdiagramm dargestellt werden.

Eine Business *Entität* stellt etwas dar, was in einem Business Use Case eingesetzt wird, also zum Beispiel eine Rechnung, eine Überweisung.

Eine *Work Unit* ist ein Set solcher Entitäten, die zusammen eine sinnvolle Grösse für den Endbenutzer des Systems darstellt.

Business Entitäten und Workunits werden eingesetzt, um Domain Klassen aus Business Sicht zu modellieren, also zum Beispiel Rechnungen, Bestellungen, Artikel, Konto. Wir könnten also ein Business Modell analog darzustellen, wie wir das für das Domain Modell gemacht haben, mit dem selben Resultat.

Zusätzlich könnte man die Interaktion der Benutzer / Aktoren als Diagramm darstellen und aufzeigen, wie die einzelnen Aktoren und Geschäftsfälle miteinander in Beziehung stehen.

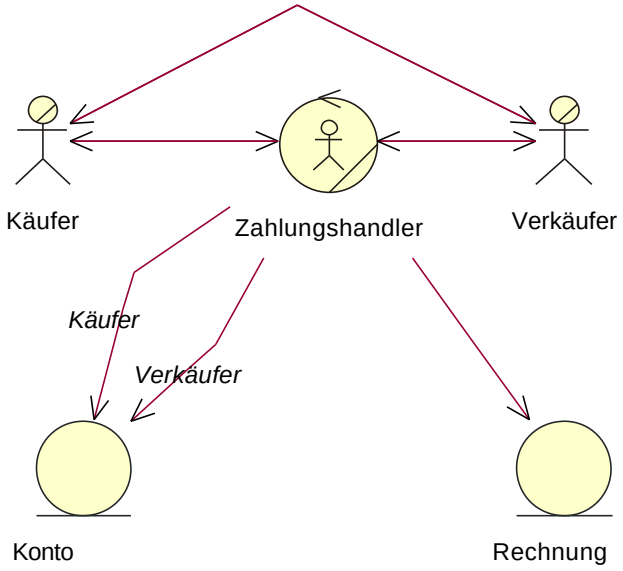

Abbildung 0-3 Verkauf: Käufer, Verkäufer, Payment Handler (transferiert Geld von einem Konto auf das andere)

Jeder Business Worker, jede Business Entität und Work Unit kann an mehr als einer Business Use Case Realisierung beteiligt sein.

Als Beispiel : die Kontoklasse ist vermutlich an mehreren Business Use Cases beteiligt

- Geldleihe :
  das geliehene Geld wird auf das Konto überwiesen
- Transfer, Abheben und Einzahlung von Geld :
  Geld wird zwischen den Konten transferiert oder auf ein Konto transferiert
- Verkauf:
  von der Bestellung bis zur Auslieferung der Ware und dem Bezahlen der Rechnung
  Das Konto ist an diesem Geschäftsprozess mehrfach beteiligt: Geld wird von einem Konto auf das andere überwiesen (der Zahlungshandler ist für diese Überweisungen zuständig)

Beispie Verkauf : von der Bestellung bis zur Lieferung (Business Use Case)

Geschäftsprozessbeteiligte haben die folgenden Aktivitäten zu erledigen:

1. der Käufer bestellt Ware oder Dienstleistungen, indem er den Verkäufer kontaktiert.
2. der Verkäufer sendet eine Rechnung an den Käufer, mit Hilfe des Zahlungs Handlers
3. der Verkäufer liefert die Ware oder die Dienstleistung an den Käufer
4. der Käufer bezahlt mit Hilfe des Zahlungs Handlers. Dieser Schritt beinhaltet den Geldtransfer vom Konto des Käufers auf das Konto des Verkäufers.

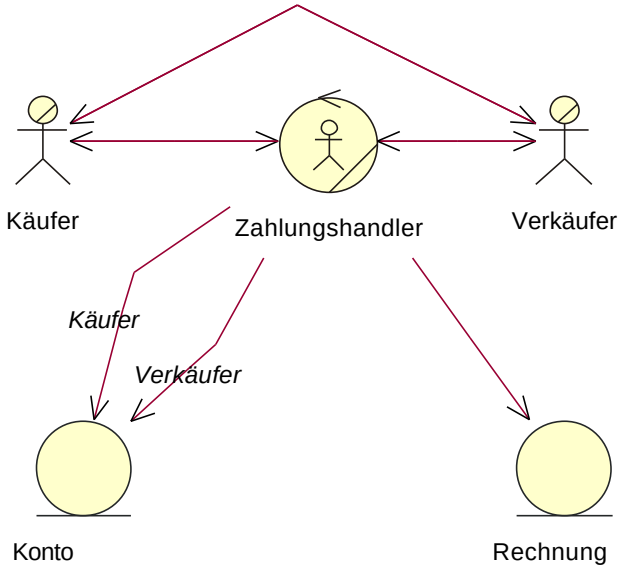

Abbildung 0-4 Verkaufsablauf : Business Use Case

Der Zahlungs-Handler kann ein Bankangestellter sein, der die Kontotransfers veranlasst.

Der Zahlungs-Handler trägt zur Wertschöpfung bei, sowohl auf Käufer-, als auch auf Verkäuferseite:

- auf Käuferseite:
  durch Vereinfachung der Zahlungsabwicklung und besseren Übersicht über offene Zahlungen, Zahlungstermine ...
- auf Verkäuferseite:
  durch Senden einer Rechnung an den Käufer und Verfolgung der offenen Rechnungen.

12.6.2. Wie wird ein Business-Modell entwickelt?

Ein Business-Modell wird in der Regel in zwei Schritten entwickelt:

1. der Business Modeller sollte einen Business Use Case beschreiben, der die Aktoren des Business beschreibt;
   zusätzlich sollte er eine Use Case beschreiben, der zur Wertschöpfung für die Aktoren

beiträgt.
Beide Geschäftsfälle können zu einem kombiniert werden.

2. der Business Modeller sollte ein Business Objekt Modell entwickeln, welches aus Workers, Business Entitäten und Work Units besteht, welche zusammen den Geschäftsfall beschreiben.
   Geschäftsregeln und Vorschriften werden den Objekten zugeordnet.
   Ziel der Modellierung ist es, ein Geschäftsmodell zu finden, welches effizient, kostengünstig und korrekt ist.

Business Modelling und Domain Modelling sind gewissermassen ähnlich:
das Domain Modell ist eine vereinfachte Form des Business Modells, wobei die Vereinfachung darin besteht, dass man sich beim Domain Modell auf "Dinge" beschränkt, die am Geschäftsfall direkt beteiligt sind. Domain Klassen und Business Entitäten sind sehr ähnlich; man kann sie gegeneinander austauschen.

Es gibt aber auch Unterschiede zwischen Business Modellen und Domain Modellen:

- Domain Klassen werden mit Hilfe der Domain Experten gefunden. Für viele Anwendungsgebiete bestehen "Referenzmodelle",also Modelle, die das Anwendungsgebiet bereits recht gut beschreiben.
  Business Entitäten werden gefunden, indem man die Geschäftsfälle analysiert. Man geht also von den Kundenanforderungen aus. Alle Entitäten müssen eindeutig einem Anwendungsfall zugeordnet werden können.
  Die beiden resultierenden Modelle unterscheiden sich in der Regel voneinander, und zwar auf allen Ebenen: Entitäten, Assoziationen und Operationen (Methoden). Im Domain Modell kann man die einzelnen Grössen zurück verfolgen bis zu dem Erfahrungsschatz des Domain Experten; im Falles des Business Modells bildet der Benutzer und seine Anforderungen die Basis für das Modell.
- Domain Klassen besitzen in der Regel Attribute aber wenige Methoden.
  Bei Business Entitäten startet man oft mit den einzelnen Methoden, zum Beispiel bei der Modellierung mit Hilfe eines Sequenzdiagramms.
- die Mitarbeiter / Worker eines Geschäftfalls werden zu den Aktoren im Business Modell; im Domain Modell wird der Domain Experte in der Regel die Aktoren vorschlagen.

Der Business Modelling Approach zur Beschreibung des Unified Processes

Der Unified Process kann selber als "Geschäftsprozess" aufgefasst werden und daher auch mit Hilfe von UML modelliert werden. Dadurch sieht man auch gleich die Stärken und Schwächen der Modellierung.
Der Unified Process wird also zum Business Use Case, zum Geschäftsfall für das Software Geschäft. Der Prozess selber besteht aus einer Sequenz von Workflows: Requirements, Analyse, Design, Implementation und Testen. Jeder Workflow realisiert einen Teil des Unified Processes:

- Workers : System Analyst, Use Case Spezialist, Architekt, ...
- Business Entitäten (Artifakten) : Use Cases und Testfälle, ...
- Work Units (Artifakten): Use Case Modell, Architekturbeschreibung, ...

Diese Grössen kann man auch mit Hilfe von UML darstellen, also den RUP als UML Diagramm visualisieren.

12.6.3. Finden von Use Cases auf Grund des Business Modells

Falls ein Business Modell vorliegt, kann man recht systematisch ein vorläufiges Use Case Modell herleiten:

1. der Analyst identifiziert einen (System) Aktor für jeden Business Aktor und Business Worker. Diese werden die Benutzer des geplanten / zu planenden Informationsystems.

Beispie Der Käufer Aktor

Der Käufer benutzt das Rechnungswesen und das Zahlungssystem, um Güter und Services, Dienstleistungen zu kaufen und Rechnungen zu bezahlen. Der Käufer ist also gleichzeitig ein Kunde und ein Aktor, da er das System benutzt, um zu bestellen und zu bezahlen (Use Cases: Bestellweses::Bestellen von Ware oder Dienstleitungen und Kreditorenbuchhaltung::gekaufte Waren und Dienstleistungen bezahlen.

---

2. Dabei müssen wir alle Situationen aus dem Business Modell heraus holen, an denen der Busines Actor beteiligt ist.
   Damit gelangen wir zu einem Use Case des Aktors. Alle Use Cases zusammen ergeben unser Use Case Modell

---

Beispie Identifizieren von Anwendungsfällen auf Grund des Business Modells

Fortfahrend mit dem obigen Beispiel könnte man als möglichen Use Case "Kaufen von Ware und Services" vorschlagen. Dieser würde den Käufer Aktor als Business Aktor im Business Use Case "Verkauf" unterstützen: von der Bestellung zur Auslieferung.
Bei genauerer Analyse kann es sich zeigen, dass es besser wäre, den Anwendungsfall "Kaufen von Ware und Services" besser in mehrere Use Cases unterteilen würde: eine Möglichkeit wäre die Unterteilung in zwei Use Cases

a) Einkaufen
b) Rechnungen begleichen

Der Grund für diese Unterteilung ist, dass der Käufer lieber nicht einen Geschäftsprozess hat, sondern zwei: es ist bequemer den Einkauf abzuschliessen und später den Geschäftsprozess "Rechnung begleichen" zu aktivieren. Bei vielen Organisationen sind die zwei Prozesse auch organisatorisch völlig getrennt.

Bisher haben wir gesehen, dass wir den Kontext des Systems entweder mit einem Domain oder oder einem Business Modell beschreiben können. Oben haben wir gesehen, wie man aus Geschäftsfällen (Business Cases) des Business Modells ein Use Case Modell herleiten kann. ein Modell, welches die funktionalen und viele nicht funktionale Anforderungen erfassen und beschreiben kann. Die so noch nicht erfassten Anforderungen werden in den "Zusätzlichen Anforderungen" (*supplementary requirements*) zusammen gefasst.

12.7. Zusätzliche Anforderungen

Die zusätzlichen Anforderungen sind primär nichtfunktionale Anforderungen, die keinem bestimmten Use Case zugeordnet werden können. Diese Anforderungen betreffen in der Regel mehrere oder keinen Use Case.
Beispiele:

- Performance
- Interfaces
- physischer Design
- Architektur
- Design und Implementations- Rahmenbedingungen (kleiner als 8K gross, echtzeitfähig)

Diese zusätzlichen Anforderungen werden wie früher üblich in einem Dokument erfasst, einer Liste. Während der Analyse und Design Phase wird dieses Dokument als zusätzliche Spezifikation eingesetzt.

Eine *Interface Anforderung* spezifiziert das Interface zu einem externen System, mit dem das betrachtete System interagieren muss. Es kann sich auch um Formate, Timing Anforderungen (Echtzeitsysteme) oder andere Faktoren handeln.

Eine *physische Anforderung* spezifiziert eine physikalische Charakteristik, die das System haben muss, wie zum Beispiel Grösse, Gewicht, ... Diese Art Anforderungen kann, zum Beispiel, verwendet werden, um sicher zu sein, dass Hardwaresysteme, wie etwa Netzwerkwerkkomponenten, Stromversorgungen in die dafür vorgesehenen Räume passen.

Beispie Hardware Plattform Anforderungen

Server
Sun Sparc oder PC Pentium

Clients
PCs (mindestens Pentium 500 MHz mit 128 MB RAM) oder Sun Sparc

---

Eine *Design Einschränkung (Design Constraint*) beschränkt das System in irgend einer Art und Weise, zum Beispiel in Bezug auf Erweiterbarkeit oder in Bezug auf Legacy Systems.

Eine *Implementations-Einschränkung (Implementation Constraint*) spezifiziert oder beschränkt das System in Bezug auf Codierung und / oder Implementierung. Beispiele könnten sein: Standards, Implementations-Guidelines, Datenbanken-Vorschriften, Resourcenlimiten, operative Umgebung.

Beispie Dateiformate (Constraint)

Die Version des Release 1.2 des Rechnungswesens unterstützt keine Namen, die länger als 25 Zeichen lang sind.

Beispie Software Plattform Einschränkungen

System Software
Client Betriebsystem : Windows NT 4.0, Windows 98 oder Solaris
Server Betriebsystem: Windows NT 4.0, Solaris oder Linux

Internet Software
Netscape Communicator 4.x oder Internet Explorer

Oft werden auch *andere Anforderungen* aus gesetzlichen Gegebenheiten hergeleitet.

Beispie Andere Anforderungen

Security
die Datenübermittlung muss sicher sein, es dürfen nur autorisierte Personen die Information benutzen, senden und empfangen. Die einzigen autorisierten Personen sind: der Bankkunde, der Kontoinhaber und der Systemadministrator.

Verfügbarkeit
Das Finanz- und Rechnungswesen dürfen pro Monat höchstens zwei Stunden nicht verfügbar sein.

Ease of Learning (Einarbeitungszeit)
Die Zeit, sich mit dem neuen System vertraut zu machen, darf für den durchschnittlichen Benutzer maximal 10 Minuten betragen.

12.8. Zusammenfassung

Bis hier haben wir die Anforderungen an unser System spezifiziert. Wir haben gesehen, wie man Business Modelle und Domain Modelle benutzen kann, um die Anforderungen an das System herzuleiten. Mit Hilfe von Use Cases sind wir in der Lage, Anforderungen zu visualisieren und formal zu fassen (Textversion).

In den nächsten Kapiteln werden wir die Anforderungen verfeinern und schliesslich sehen, wie man aus Use Cases Text Cases machen kann, um sicherzustellen, dass unser System das macht, was der Benutzer will, was er verlangt hat.