In diesem Kapitel:
- *Beschreibung von Use Cases mit Hilfe von Diagrammen*
- *Bezug zum Projektplan*
- *Reviews*
- *Implementierung, System Architektur und System Betrieb*
- *Zusammenfassung: Phasen und Ergebnisse*

11
UML - Use Cases

11.1. Beschreibung von Use Cases mit Hilfe von Diagrammen

In diesem Kapitel betrachten wir die Anwendungsfälle über den gesamten Projektverlauf, inklusive der daraus hergeleiteten Modelle. Auf die Modelle werden wir aber im Rahmen der Besprechung der einzelnen Phasen nochmals zurück kommen.

11.1.1. Einleitung

Wir haben uns sehr ausführlich mit der Aufgabe befasst, die Use Cases mit Hilfe von Text zu beschreiben.

Jetzt möchten wir versuchen, die Prozesse, die global mit Hilfe eines Use Cases beschrieben wurden, im Detail zu erfassen und zu visualisieren!

Dabei werden wir auf unterschiedlichen Ebenen verschiedene Beschreibungen kennen lernen. Jede Ebene hat spezielle Schwerpunkte und jede Visualisierung hat spezielle Vorteile und Nachteile

Die Visualisierung mit Hilfe der UML Notation schafft eine einfache Übersicht über das noch zu schaffende System.

Dabei geht es weniger darum, die einzelnen Anwendungsfälle im Detail korrekt festzuhalten: das würde zuviel Zeit in Anspruch nehmen und unnötig genau sein; unnötig, weil in der Regel in späteren Phasen und Modellen viel mehr auf Details geachtet wird.

Falls Sie bereits mit Rational Rose arbeiten, dann sollten Sie darauf achten, dass die Activity Diagramme erst relativ spät als Zusatz direkt vom Rational Web Site erhältlich waren.

Aktivitätendiagramme erscheinen auch nicht als möglicher Diagrammtyp sondern innerhalb der Klasse als Modellierungsoption.

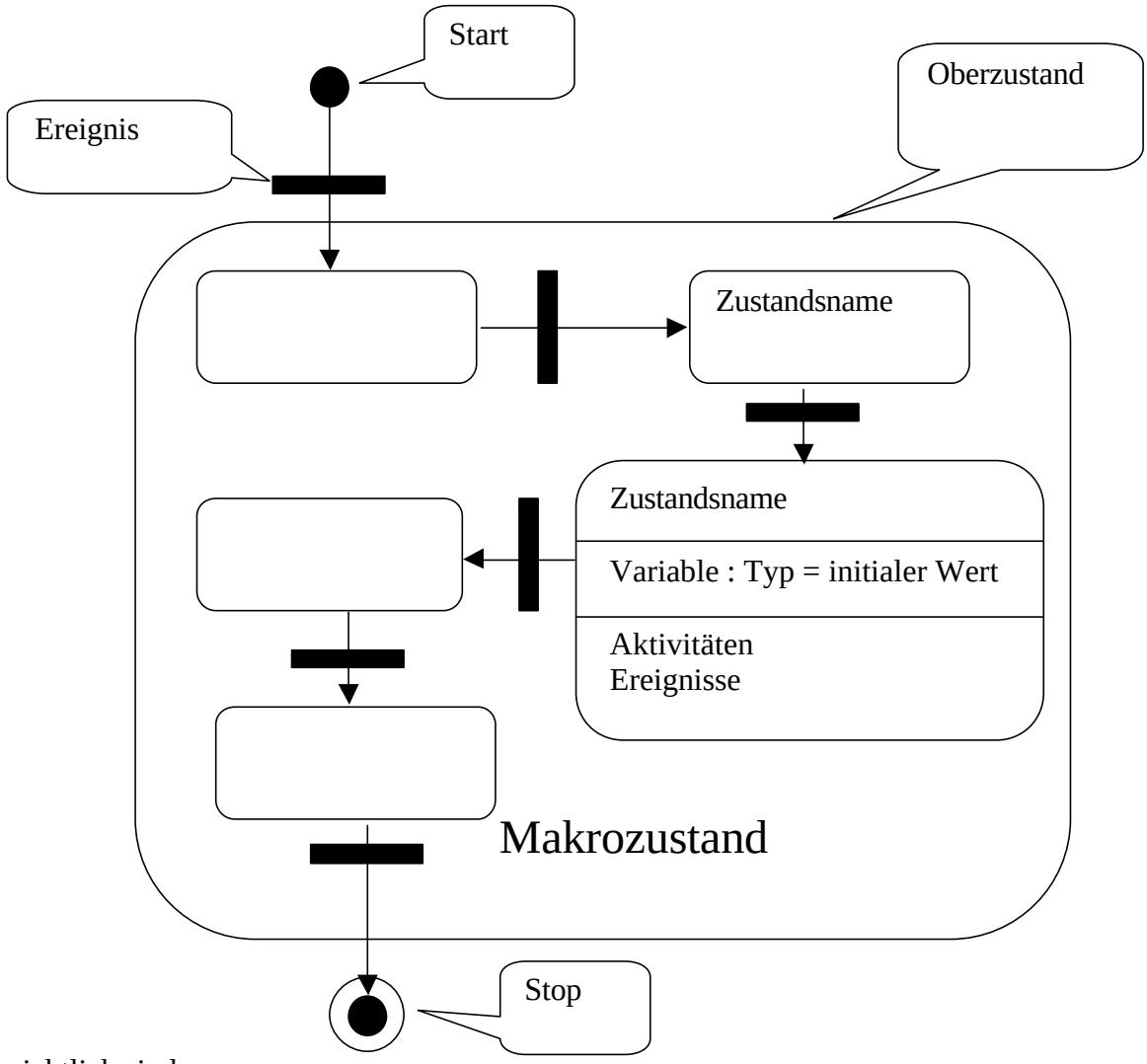
Das unterstreicht die Tatsache, dass Prozessmodellierungen nicht im Vordergrund einer Objektmodellierung stehen dürfen. Primäres Anliegen muss sein, die Verantwortlichkeiten festzulegen und dadurch zu einer besseren Modellierung zu gelangen.

11.1.2. Aktivitäten Diagramme

Die Beschreibung der Dynamik einzelner Anwendungsfälle werden oft mit Hilfe der eher informellen Aktivitätendiagramme modelliert.

11.1.2.1. Ein Beispiel

Betrachten wir zuerst ein kleines Beispiel, aus dem bereits die wichtigsten Definitionen



ersichtlich sind.

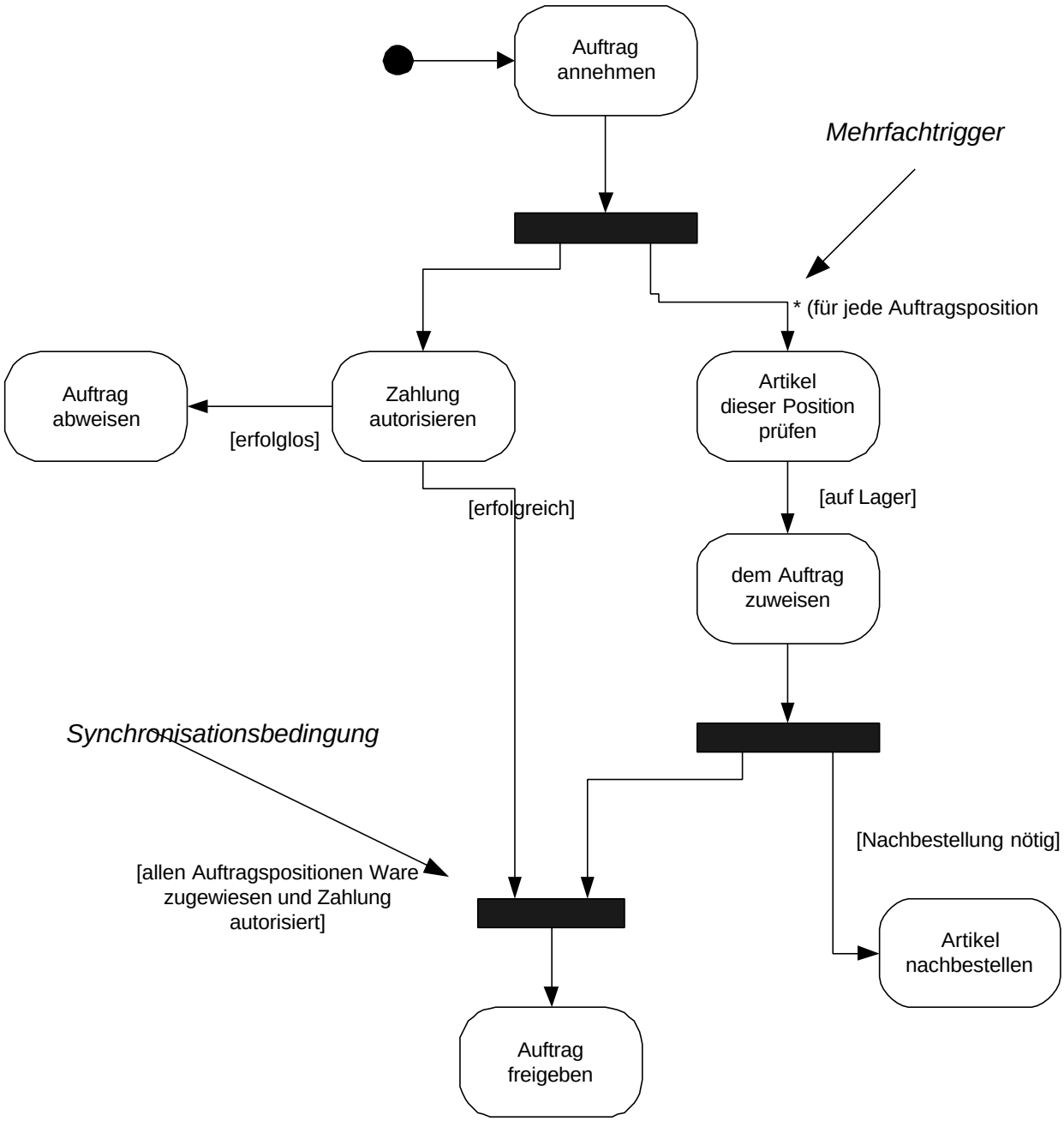
Aktivitäten Diagramme wurden schon vor geraumer Zeit in der Informatik zur Modellierung von Abläufen eingeführt. Wie fast immer bei Methoden und Verfahren, gibt es verschiedene Ausprägungen, verschiedene "Dialekte" der Diagramme.

In UML wurden die Aktivitäten Diagramme als Subset der Zustandsdiagramme definiert.

Gemäss UML sind Aktivitäten Diagramme Zustandsdiagramme, bei denen alle Zustände Aktivitäten beschreiben und die Übergänge automatisch geschehen.

11.1.2.2. Beispiel : Auftragsabwicklung

Um die Begriffe verständlicher zu machen, betrachten wir gleich noch ein weiteres Beispiel.



11.1.2.3. Selbsttestaufgaben

Beschreiben Sie mit Hilfe eines Aktivitäten Diagramms den Anwendungsfall (Use Case) **Bestellung eingeben.**

a) Der Einfachheit halber reduzieren Sie den Use Case auf folgende Aktivitäten.

1 Der Anwendungsfall beginnt, sobald der Kunde "Bestellung eingeben" anwählt
2 Der Kunde gibt seinen / ihren Namen ein
3 Der Kunde gibt Produktbestellnummern ein
4 Das System ermittelt die Produkt(Kurz)Beschreibung und den Preis
5 Das System ermittelt das Gesamttotal des Einkaufs
6 Der Kunde gibt die Kreditkarteninformation ein
7 Der Kunde bestätigt die Bestellung
8 Das System verifiziert die Angaben in der Bestellung, speichert diese zuerst als pendent ab und verlangt vom Buchhaltungssystem eine Bestätigung der Kreditfähigkeit
9 Sofern das Kreditinformation eine Bestätigung liefert, dann wird die Bestellung aktiviert, eine Bestellnummer dem Kunden mitgeteilt und der Anwendungsfall abgeschlossen.

b) Modifizieren Sie den Anwendungsfall so, dass er mehrere Artikeleingaben behandeln kann. Es entsteht also eine Schleife "3-4-5-3-4-5..."

11.1.3. Gleichzeitige Abläufe

In der Selbsttestaufgabe im obigen Abschnitt , haben wir rein sequentiell eine Aktivität an eine andere angehängt.

Im Graphen davor erkennen wir aber bereits eine mögliche Parallelität in den Abläufen:

- Den Auftrag freigeben UND den Artikel nach bestellen
  sind zwei Aktivitäten, die unabhängig voneinander "gleichzeitig" ablaufen.

Stellen wir uns nun den realen Ablauf eines Dialoges aus der Sicht des Innendienstmitarbeiters, des Kundenbetreuers, genauer an.

Für ihn ist es sehr wichtig, dass er alle Informationen über einen Kunden möglichst sofort und gleichzeitig auf dem Bildschirm angezeigt bekommt.

Da in der Regel *eine* Bildschirmdarstellung dafür nicht ausreicht, wird im Falle dieses Benutzers das System *mehrere* Bildschirme gleichzeitig starten müssen.

Unser System benötigt also folgende Möglichkeiten:

- Darstellung von Entscheidungspunkten:
  ist der Benutzer ein "normaler" Kunde oder ein Innendienstmitarbeiter

- Darstellung einer Verzweigung im Sinne "MEHRERE THREADS", mehrere Ausführungspfade
  der Standardbildschirm des Kunden wird angezeigt
  zudem wird auch der Bildschirm "Kundeninformationen" angezeigt

Betrachten wir zuerst den Fall des "Decision Points"

11.1.4. Entscheidungspunkte : Decisions

Dieses Konstrukt kennen Sie aus jeder Programmiersprache (if ... then... else).

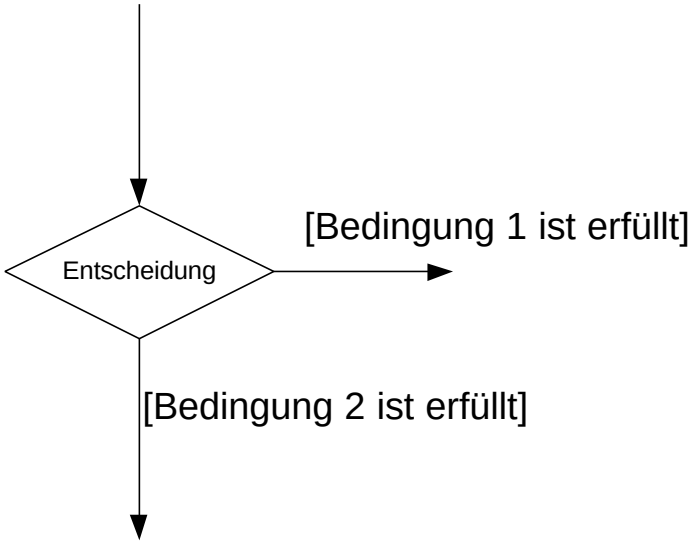
11.1.4.1. Semantik

Ein Zustandsdiagramm und das darauf beruhende Aktivitäten Diagramm zeigt einen ***Entscheidungspunkt*** , falls es Bedingungen ("Guard Conditions") gibt, welche den Ausführungspfad bestimmen, abhängig von Boole'schen Variablen im vorgängigen Objekt.

11.1.4.2. Notation

Eine Entscheidung wird dargestellt, in dem die Ausgangs-Transitionen (die Ausgangs-Übergänge) beschriftet. Man schreibt also einfach die Bedingung an den Übergangspfeil.

Als Symbol verwendet man den "Diamanten"



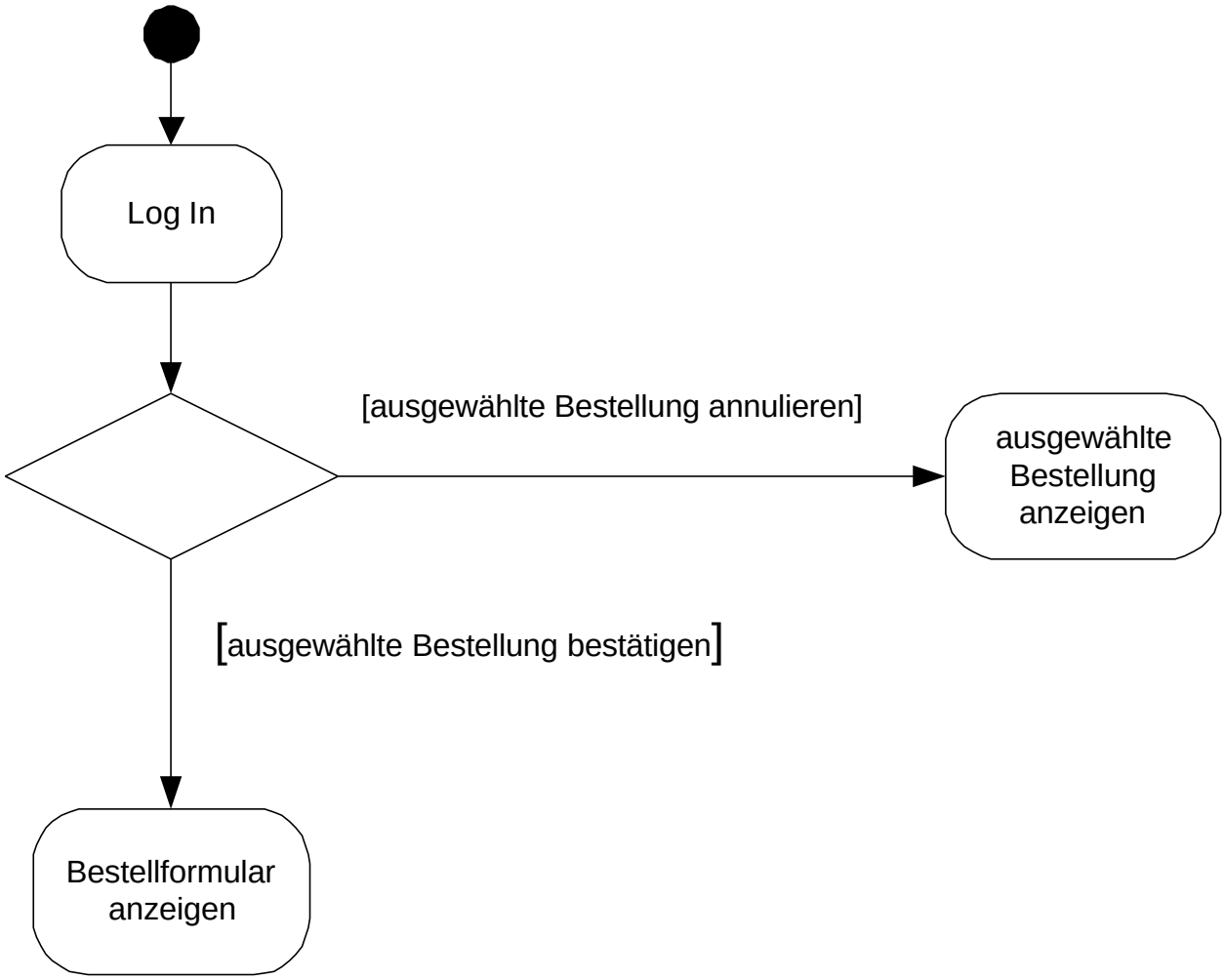
11.1.5. Selbsttestaufgabe

Lesen Sie das Kapitel 10 "Activity Diagram" in der UML Notationsbeschreibung "UML Notation Guide" .

Sie können diesen Guide von WWW.Rational.COM oder der OMG WWW.OMG.ORG oder einem SunSite runter laden; falls genug Platz auf dem Server ist, dann finden Sie eine Kopie bei den Unterlagen, als PDF.

Achten Sie darauf, dass Sie die neuste Version lesen:
die aktuellste Version finden Sie bei der OMG (www. omg.org).

11.1.5.1. Beispiel : Decision Point

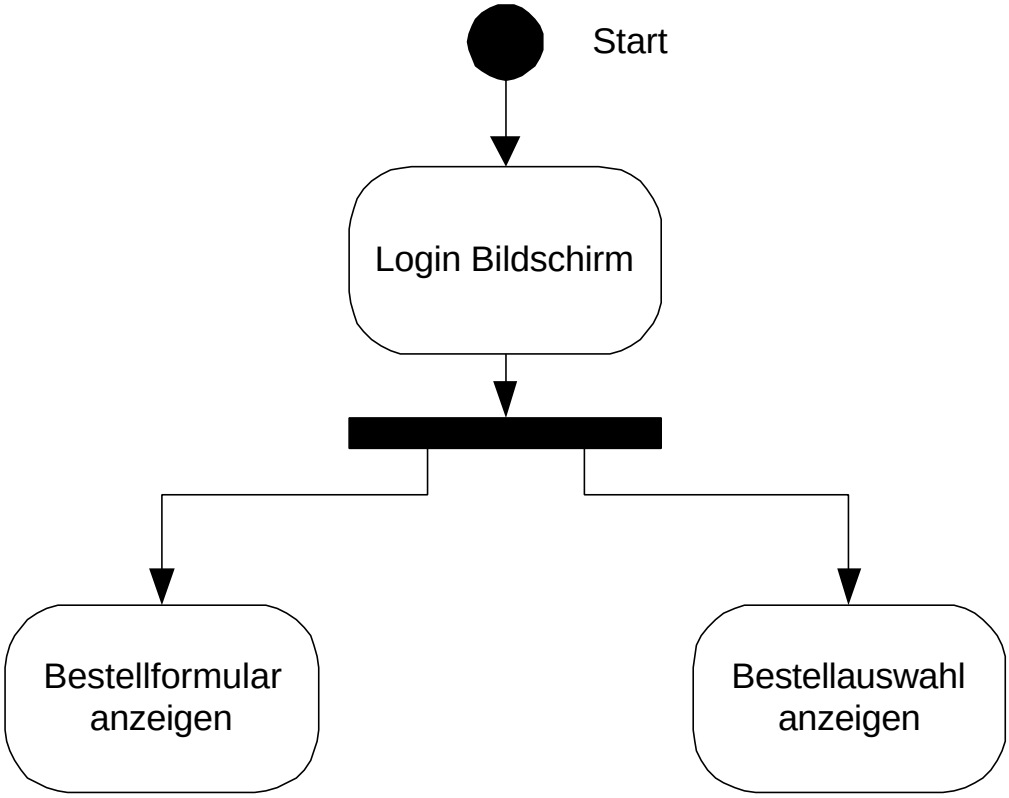


Die Semantik , die Bedeutung dieser Diagramme ergibt sich, glaube ich, sehr leicht aus den entsprechenden Konstrukten in den klassischen Flussdiagrammen.

Vergessen Sie aber nicht, dass Sie primär Objektsysteme modellieren wollen und müssen, keine klassischen Abläufe.

Im Mikrokosmos sind aber auch Objekte immer noch prozedural. Also lassen sich die klassischen Konzepte neu, verändert und mit der richtigen Perspektive einsetzen.

11.1.6. Verzweigung : FORK



In diesem Falle wird gleich nach dem einloggen "quasi-parallel" ein Thread das Bestellformular anzeigen, ein anderer Thread lädt den Bestellauswahl-Bildschirm.

Der Innendienstmitarbeiter kann somit, bei genügend Hardware und passender Software, fast gleichzeitig mehrere Bildschirme anschauen.

Dass diese Aufteilung "FORK" heisst ergibt sich aus der Ähnlichkeit zu einer (etwas gar einfachen) Gabel!

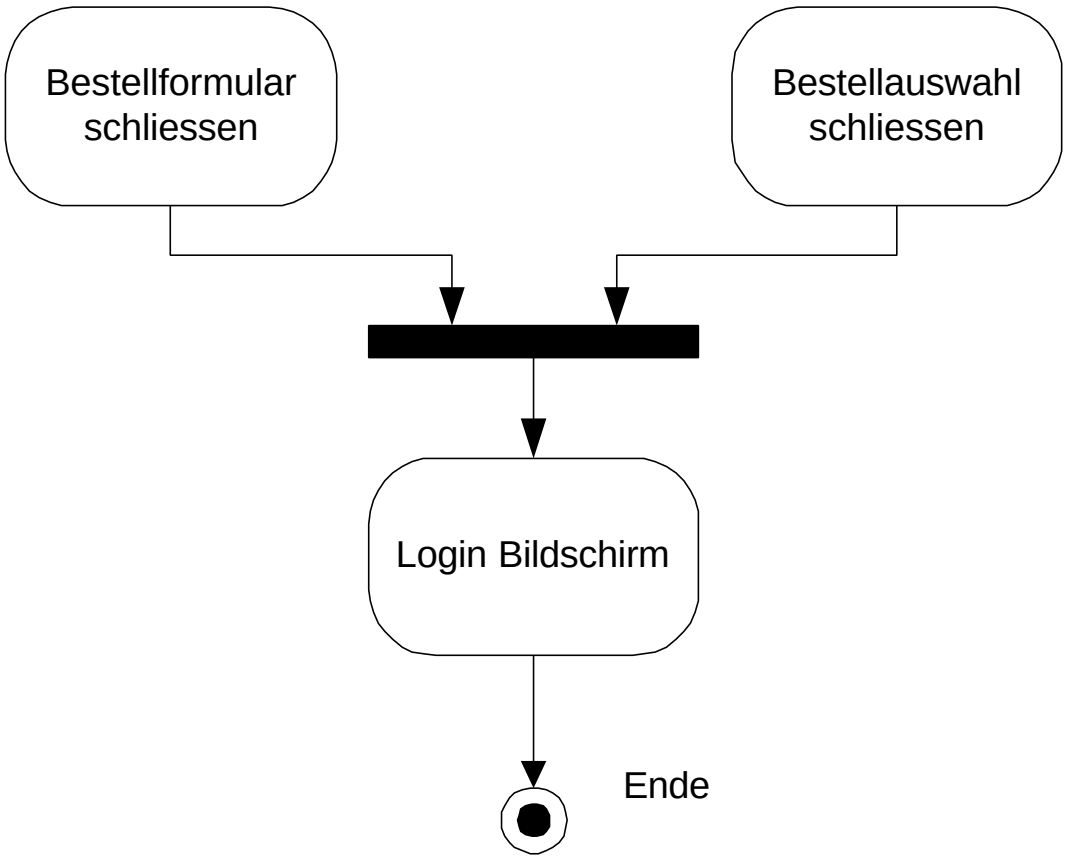
11.1.7. Join

Das Gegenteil von Fork ist "JOIN". Bei einem JOIN werden mehrere Threads beendet und die Ausführung des Prozesses geschieht ab diesem Zeitpunkt wieder "vereint".

11.1.7.1. Beispiel

Unser Innendienst-Mitarbeiter möchte sich vom System verabschieden, also ausloggen.

Das System muss daher den folgenden Ablauf durchlaufen:



Gemäss diesem Diagramm kann man NICHT ausloggen BEVOR beide Bildschirme geschlossen sind.

11.1.8. Makro-Zustände

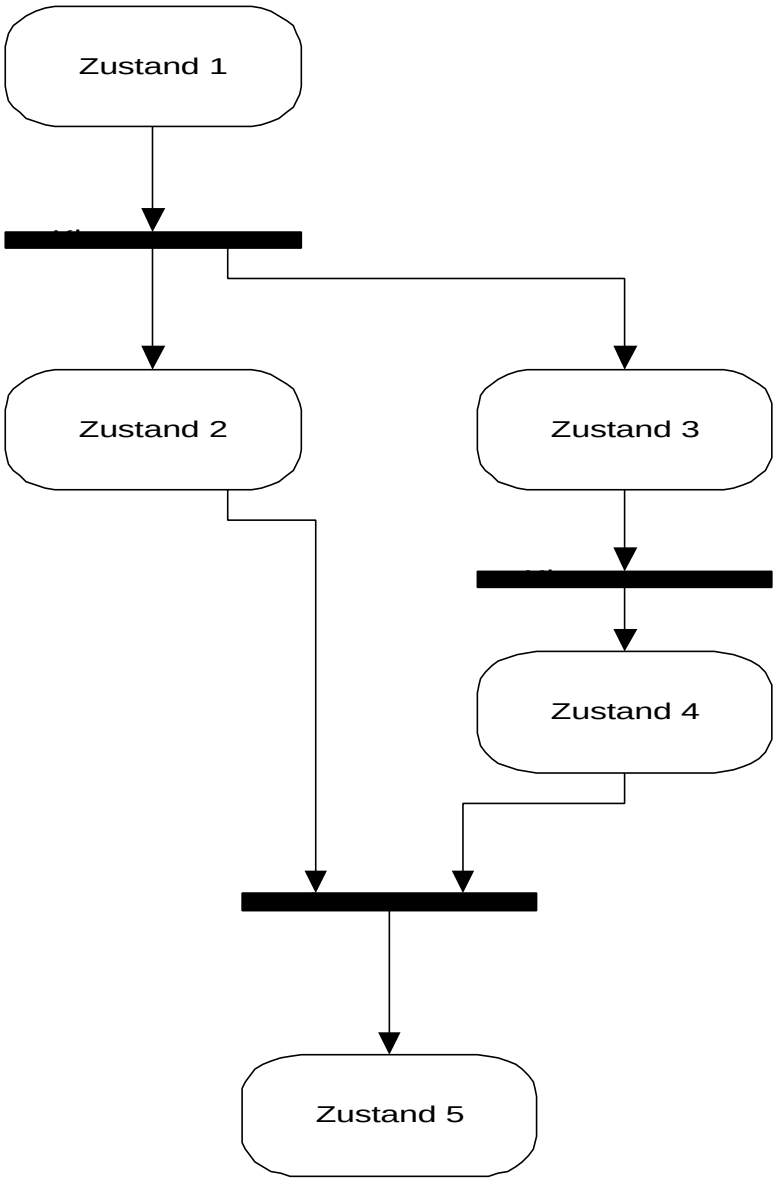
Wir haben in unserem Beispielen lediglich einfache Fälle besprochen. In der Praxis erweist es sich in der Regel als sinnvoll und notwendig, die Modellierung schrittweise zu entwickeln und die Darstellung zu verdichten. Dazu fasst man verschiedene Zustände zu einem Makro-Zustand zusammen.

Betrachten wir ein Beispiel

11.1.8.1. Beispiel für Makro-Zustände

Jeder Zustand in unseren Diagrammen kann sich eventuell weiter aus einem ganzen Subsystem zusammen setzen.

Wichtig ist bei einer solchen Modellierung, dass die detailliertere Modellierung genau so viele Schnittstellen hat wie der Makro-Zustand.



Zustand 4 wird weiter verfeinert. Wichtig ist offensichtlich, dass dieses Detaillierungs-Diagramm genau einen "Eingang" und einen "Ausgang" hat.

Wenn der obige Ablauf rekursiv eingesetzt werden soll, dann können wir einfach das gesamte Diagramm nochmals zeichnen:
Die Anzahl "Eingänge" ist 1
Die Anzahl "Ausgänge" ist auch 1

11.1.9. Verantwortlichkeiten

Aktivitäten Diagramme beschreiben, was passiert, nicht jedoch, wer was tut. Für die Programmierung heisst dies, dass das Diagramm nicht beschreibt, welche Klasse für welche Aktivität verantwortlich ist.

Für die Problembereichsmodellierung heisst dies, dass das Diagramm nicht beschreibt, welche Personen oder Abteilungen für jede Aktivität verantwortlich zeichnet. Ein Ausweg wäre, jede Aktivität mit der verantwortlichen Klasse oder Person zu beschriften. Dies funktioniert, bietet aber nicht soviel Klarheit wie sie Interaktionsdiagramme (die wir noch im Detail behandeln werden) bei der Visualisierung der Kommunikation zwischen Objekten bieten.

Der Ausweg sind hier **Verantwortlichkeitsgrenzen (***swimlanes)* .

Zur Verwendung von Verantwortlichkeitsgrenzen müssen die Aktivitätsdiagramme in durch Linien abgetrennte vertikale Bereiche gegliedert werden. Jeder Bereich repräsentiert die Verantwortlichkeit einer bestimmten Klasse oder einer bestimmten Abteilung.

In unserem Beispiel könnten wir somit die Kreditprüfung ganz links, die eigentliche Auftragsabwicklung in der Mitte und die Lagerverwaltung und Logistik ganz rechts aufzeichnen. Damit wäre die Organisation und die Abläufe innerhalb der Organisation ersichtlich.

11.1.10. Wann benutzt man Aktivitätsdiagramme?

Die grosse Stärke der Aktivitätendiagramme liegt in deren Unterstützung paralleler oder nebenläufiger Abläufe, in den Programmen also Threads, Sub-Prozessen und Prozessen.

In folgenden Situationen bieten sich Aktivitätsdiagramme an:

- Bei der Analyse von Anwendungsfällen
  Auf dieser Ebene ist die Zuweisung der Aktionen (Methoden) zu den Objekten noch nicht interessant. Man muss lediglich verstehen, welche Aktionen stattfinden müssen und wie die Verhaltensabhängigkeiten sind. Später muss man die Methoden den Objekten zuordnen. Aber dies geschieht leichter mit den Interaktionsdiagrammen
- Geschäftsvorgang (workflow) über viele Anwendungsfälle hinweg verstehen.
  In diesem Falle sind Aktivitätendiagramme ein gutes Werkzeug zur Repräsentation und zum Verständnis dieses Verhaltens.
- Beim Umgang mit Anwendungen mit mehreren Threads
  wobei in diesem Falle eher die formal besser beschreibbaren Petri oder Condition / Event Systeme und deren Verallgemeinerungen benutzt werden.

11.1.11. Wann sind Aktivitätendiagramme eher ungeeignet?

In den folgenden Situationen sind Aktivitätendiagramme zwar einsetzbar, aber es gibt bessere Modellierwerkzeuge:

- Zusammenarbeit zwischen Objekten verstehen
  in diesem Falle benutzt man besser Interaktionsdiagramme
- Objektverhalten über seinen Lebenszyklus verstehen
  Dafür benutzt man besser ein Zustandsdiagramm.

11.1.11.1.1. Sebsttestaufgabe

Beschreiben Sie die Funktionsweise eines Einkaufszentrums bestehend aus mehreren Warenregalen ,mehreren Einkäufern und mehreren Kassen.

Die Kapazität der Regale, die Anzahl Kassen und die Anzahl gleichzeitig einkaufender Personen ist jeweils beschränkt. Falls in irgend einem Fall die kritische Schwelle erreicht wird, wird ein Ausnahmefall ausgelöst.

11.1.12. Darstellung der Benutzerschnittstelle

Die Diagramm-Technik "Aktivitätendiagramme" zeigt uns relativ wenig oder schlecht, wie der Benutzer sein zukünftiges System bewerten wird. Obschon viele Benutzer Aktivitätendiagramme nach einiger Anlaufzeit verstehen werden, fehlt doch noch einiges, um sich ein klares Bild vom Layout und der Bedienung des Systems zu machen.

Üblicherweise führt man dazu einen einfachen Prototypen ein, das heisst. man entwickelt die Bildschirme, hinterlegt aber kaum Ablauflogik und schon gar nicht Verifikations-Logik.

Für das Prototyping verwendet man in der Regel ein Screen Design Tool beziehungsweise "leere" Screens, bei denen keine Verifikation der Eingabe erfolgt.

11.1.13. Zerlegen von grossen Systemen

Oft müssen zerlegt werden, damit man ins Detail gehen kann. Die Systembeschreibung als Ganzes wäre zu umfangreich, wir müssen das System aufteilen und in den Subsystemen die Details darstellen.

Am Besten fängt man damit an, eine Architektur des zu realisierenden Systems zu skizzieren. Die Architektur soll das Gesamtsystem und die Prinzipien aufzeigen, jedoch Details weglassen.

11.1.13.1. Architektur - Pattern

Bei der Untersuchung vieler Software Projekte und Software Designs stiessen die Mitarbeiter der Zentralen Forschung bei Siemens München auf sich immer wieder wiederholende "Muster", Patterns. Sie fassten diese in einem Buch zusammen:

Buschmann, F, .R. Meunier, H. Rohnert, P. Sommerlat, M. Stal : "Pattern-Oriented Software Architecture : A System of Patterns" (1996)

Bereits vorher wurden für die Kodierung verschiedene Patterns veröffentlicht:
Gamma, Erich, R. Helm, R. Johnson, J. Vlissides : "Design Pattern : Elements of Object Oriented Architecture" (1995)

Es hat sich gezeigt, dass in der Regel (auf allen Gebieten) nicht von Grund auf Neues erfunden und kreiert wird, sondern, dass in der Regel bekanntes neu kombiniert wird. Das Bekannte bildet die Basis für die Patterns.

Sie können sich das durchaus so vorstellen wie beim Hausbau : wie ein Haus in etwa aussehen muss, ist klar. Wie es konkret aussieht, hängt von vielerlei Faktoren ab: von meinem Budget, der Lage, ...

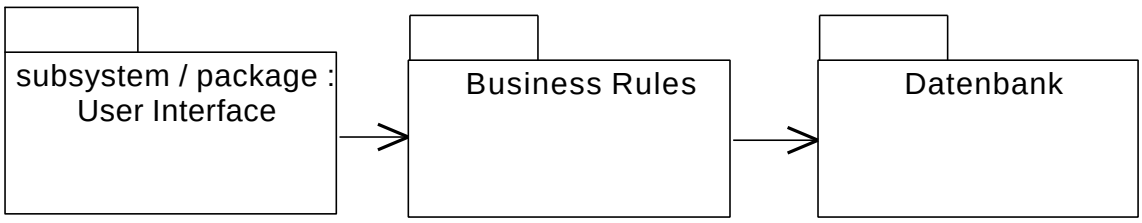
Im Folgenden möchten wir einige der Patterns kennen lernen und anwenden (PowerPoint Unterlagen über Architektur und Design Patterns).

11.1.13.2. Three-Tier Pattern

Ein Grundmuster bei Geschäftsapplikationen ist das "dreistufige Modell":

Jede Anwendung besteht aus
1. Dem Benutzerinterface - Subsystem
2. Den Geschäftsregeln
3. Der Datenbank

In UML stellt sich das wie folgt dar:



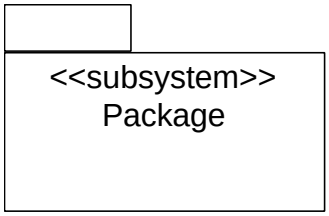
Wie sieht dies in unserem Beispiel der MailOrder Firma aus?

Betrachten wir als Erstes die Bestellungen:
- Für die Bestelleingabe benutzen wir ein Formular : User Interface Tier
- Die Daten werden in einer Datenbank gespeichert : Datenbank Tier
- Damit wir die Bestellung bearbeiten können, nutzen wir : Business Rule Tier

In unserem Beispiel zeigt es sich fast natürlich, dass das System in diese drei grossen Teile zerlegbar sein sollte:

- Die Benutzerschnittstelle entwickeln wir evtl. mit HTML
- Die Datenbank kaufen wir dazu
- Die Business Logik besteht zum Beispiel aus der Verifikation der Angaben zur Kreditkarte

Als neues Symbol wurde das "Package" eingeführt.



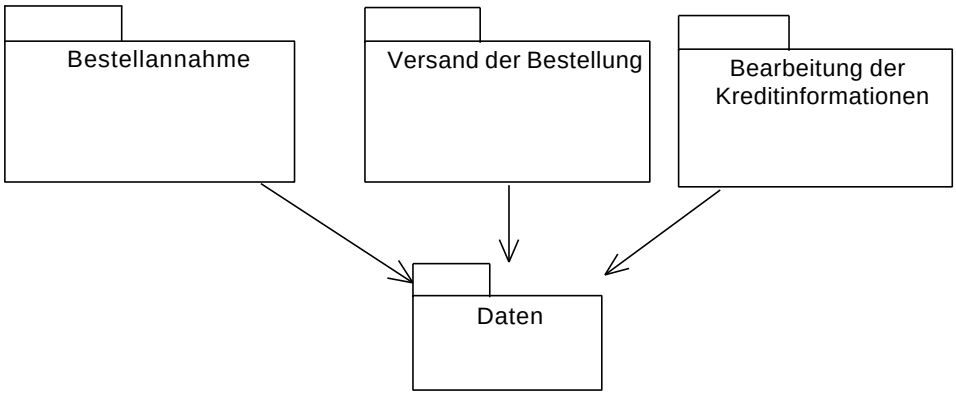
11.1.13.3. Pipe und Filter Architektur Pattern

Pipe und Filter ist ein völlig anders gelagertes Architektur-Pattern.

Die Grundstruktur einer Pipe ist folgende:

- Daten werden erfasst
- Daten werden transformiert, bearbeitet
- Daten werden ausgegeben
- Die Daten werden weitergereicht, also wieder erfasst
- ...

In UML sieht das Pattern vereinfacht wie folgt aus:



Jedes der Subsysteme kann seine Aufgabe unabhängig vom andern erfüllen. In unserem Beispiel liefert jedes Subsystem Daten, nämlich Informationen über erledigte oder offene Bestellungen.

11.1.13.4. Liste der aktuell veröffentlichten Architektur Pattern

Architektur Patterns halten grundsätzliche Strukturen und Organisationsprinzipien auf dem Systemlevel fest (siehe auch PowerPoint Unterlagen).

Die bekannten Architektur Pattern (Buschmann & all) sind

- Layers
- Pipes & Filters
- Blackboard
- Model-Viewer-Controller
- Presentation-Abstraction-Control
- Microkernel
- Reflection

Neben Architektur-Patterns kennt man

- Design Patterns (Gamma & al)
- Idioms (Gamma & al)

Man kann sich die Patterns selber auf unterschiedlichen Ebenen vorstellen :
Zuoberst die Architektur-Patterns, dann die Design-Patterns und schliesslich die Idiome.

11.1.14. Testen der Architektur mit Hilfe von Use Cases

Nachdem wir die wichtigsten Komponenten unseres Systems erfasst haben, geht es darum zu prüfen, ob die Architektur auch verhebt, ob wir eventuell weitere Architektur Patterns einsetzen könnten und ob unsere Architektur erweiterbar, veränderbar, wartbar,... ist.

Jedes Subsystem muss folgende Eigenschaften besitzen:

- Es muss sich auf EINE Grundfunktion beschränken
  wir wollen nicht beliebig komplexe Subsysteme, die funktional nicht zusammen hängen
- Sie müssen (interne) Kohäsion zeigen
  die einzelnen Teile des Subsystems müssen eng miteinander gekoppelt sein
- Sie müssen (extern) lose Kopplung zeigen
  die Subsysteme dürfen für die Erfüllung Ihrer Aufgaben nicht wesentlich von andern Subsystemen abhängen
- Die Kommunikation der Subsysteme untereinander muss minimal sein
  die Subsysteme müssen für die Erfüllung Ihrer Aufgaben nicht auf wesentliche Informationen aus andern Subsystemen angewiesen sein

Zum Testen der Architektur testet man das geplante System mit Hilfe der Use Cases.

Betrachten wir ein konkretes Beispiel:
Bestellen / Bestellung erfassen

1. **Der Benutzer wählt die Bestelloption an**
   die Login Informationen werden bearbeitet
   <<subsystem>> System Access
   Login
   Logout
   Zugriffsberechtigungsprüfung

2. **Der Kunde gibt seinen Namen und seine Adresse ein**
   gleiches Subsystem wie oben

3. **Der Kunde gibt die Artikelnummer / Bestellnummer ein**
   die Artikelstammdaten werden in aufbereiteter Form zur Verfügung gestellt
   <<subsystem>> Auftragswesen
   Bestelleingang
   Retouren
   Status einer Bestellung, einer Lieferung
   Bestellstornierung

4. **Das System ergänzt die Bestellnummer mit der Artikelbezeichnung**
   <<subsystem>> Produkte Informationen
   Informationen über Produkte
   Schnittstelle zum Lager

5. **das System kalkuliert das Gesamttotal**
   <<Auftragswesen>>

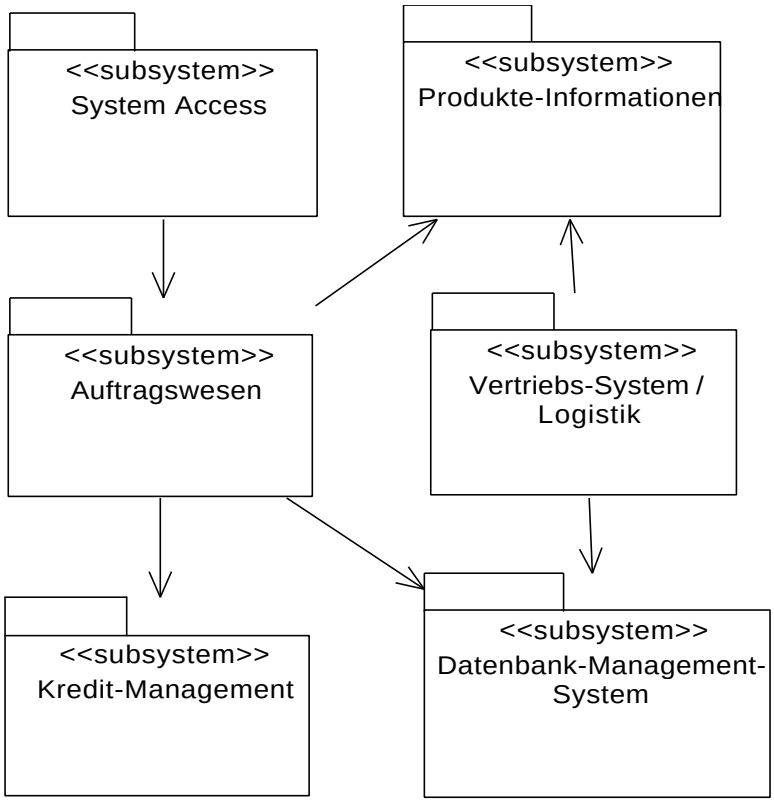
6. **Kunde gibt Kreditkarteninformationen ein**
   <<subsystem>> Kredit-Management
   Schnittstelle zum Finanz-und Rechnungswesen
   Spesen
   behandelt Kreditkarten, Checks, Überweisungen

7. **Der Kunde schickt die Information weg (submit / bestätigen der Transaktion)**
   <<Auftragswesen **>>**

8. **Das System überprüft die eingegebenen Informationen**
   Kreditinformationen müssen weiter geleitet werden
   Bestellung muss als provisorisch abgespeichert werden
   <<Auftragswesen>>, <<Kredit-Management>>

9. **Kreditinformation wird bestätigt**
   Kunde wird informiert
   Informationen müssen abgespeichert werden
   <<Datenbank-Managementsystem>>

Wir erhalten somit ein leicht modifiziertes Architektur-Modell (Package Level)!



11.1.15. Definition von Schnittstellen zwischen Subsystemen

Es gibt oft spezielle Subsysteme, die keine eigene Funktionalität aufweisen (keine eigenen Methoden).

Beispiel:
Auftragswesen

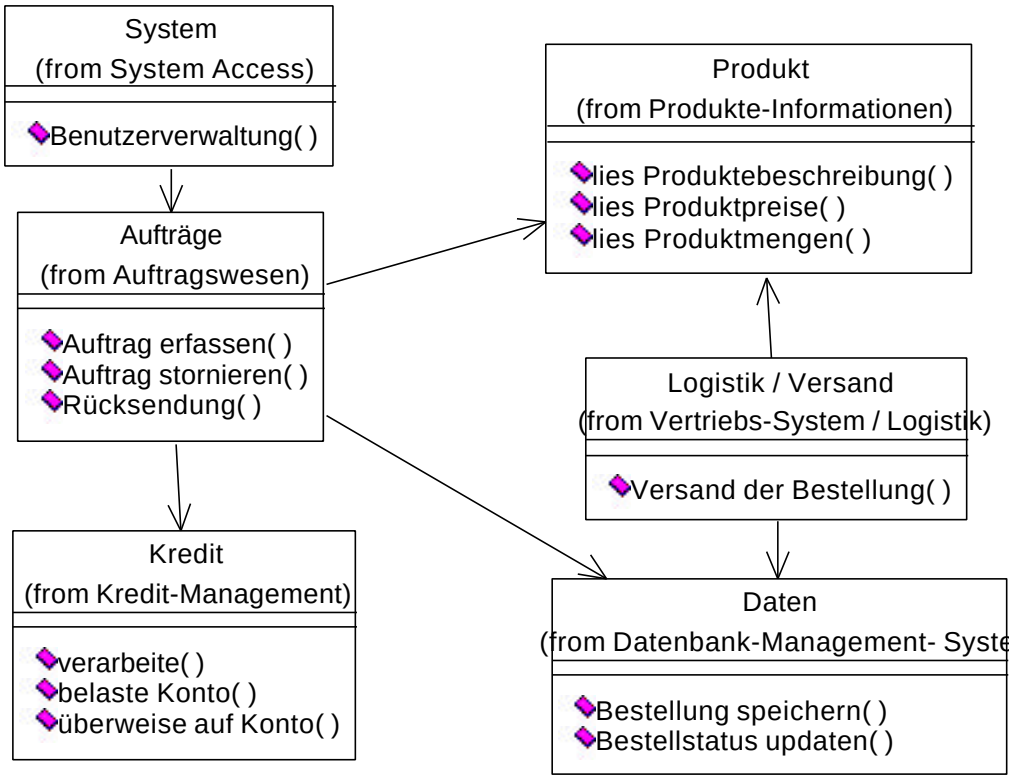
- Auftrag eingeben : Aktivität
- Auftrag stornieren : Aktivität
- Bestellung retournieren : Aktivität

Auf der andern Seite müssen wir wissen, welche Informationen an die Datenbank senden.

Um unsere Architektur besser zu verstehen, zeichnen wir sie nochmals, diesmal aber mit den "Methoden" ("Aufgaben" aus Sicht des Benutzers) : hier als Klassendiagramm.

Praktische Hinweise

1. Zuerst wurde auf der Package Ebene (in Rose : Logical View) gezeichnet
2. Durch Anklicken auf die Packages "zoomed" man in das Package und "verfeinert" es indem Klassen erfasst werden, die zu diesem Package gehören
3. In einem neuen Diagramm (Logical View Class Diagram) werden die Klassen einfach zusammen gezogen (drag & drop aus dem Fenster links oben in Rose)
   Da die Klassen in den Packages definiert wurden, wird diese Zusatzinformation oben in der Klassendefinition angezeigt.



11.1.16. Zuordnen von Use Cases zu Subsystemen

Analog zum Beispiel weiter vorne (Validation der Architektur) müssen wir nun JEDEN Use Case einem oder mehreren Subsystemen zuordnen.

Wir verzichten hier auf dieses Aufteilen!

11.1.17. Dokumentation der Subsysteme

Wir sind nun mit unserer Architektur-Definition so gut wie fertig und müssen nur noch die Dokumentation erstellen.

Als nächstes folgt der Projektplan!

11.2. UML - Use Cases : Bezug zum Projektplan

Bevor wir unser System auch tatsächlich bauen, müssen wir den Projektplan erstellen.

Der Projektplan ist in der Regel eine Richtlinie, an die man sich mehr oder weniger halten möchte. Bei komplexeren Systemen wird der Projektplan kaum eingehalten. In der Regel ist daher sinnvoll, das Projekt in Phasen zu zerlegen und jeweils nach einer Phase einen Review und eine weitere (detailliertere) Planung anzuhängen.

11.2.1. Planung des Projektes

Gemäss dem Objectory oder Unified Process Modell entstehen Produkte iterativ. Die Projektabwicklung muss dem Rechnung tragen und ein iteratives Vorgehensmodell (Spirale) zu Grunde legen. Die Phasen, die durchlaufen werden, sind nach wie vor

- Analyse
- Design
- Codierung
- Integration
- Test

einfach etwas anders benannt!

Sobald die Anforderungen alle abgedeckt sind, kann das Projekt abgeschlossen werden und das Produkt ausgeliefert werden. Für diese Produktfertigstellung kann selber wieder ein ganzer Durchlauf in Anspruch genommen werden.

Wie findet man nun zu Beginn des Projektes heraus, wie viele Iterationen benötigt werden? Wie lange sollte ein Zyklus sinnvollerweise dauern?

Diese Fragen lassen sich nicht generell beantworten, da sie zu sehr von der Umgebung und dem Projekt abhängen.

Grundsätzlich sagt man, dass die Planung auf einer Vorphase basieren sollte. Die Vorphase ist in Wahrheit ein Durchlauf. Dabei wird ermittelt wo man etwa welche Probleme haben wird und wie lange man für deren Lösung benötigen wird.

Was geschieht nun INNERHALB der Iterationen?
Wir benötigen eine RISKOLISTE mit Prioritäten! Diese muss auch immer nach geführt werden, inklusive Angaben zu den Gründen und der Begründung, warum das Risiko so eingeschätzt wurde.

Die grössten Risikoteile schauen wir uns als erstes genauer an, da sie zum Absturz des Projektes führen könnten. Jetzt schauen wir uns die Liste der Use Cases an und bestimmen, welche einen Zusammenhang mit unseren Risikos haben.

Betrachten wir ein Beispiel : unsere Auftragsabwicklung

11.2.1.1. Iteration 1

Ziele:
Wir möchten die grundsätzlichen Funktionen der Auftragsabwicklung implementieren, inklusive Schnittstellen zum Lagersystem und zur Buchhaltung.
Dadurch sollten wir in der Lage sein, Benutzerschnittstellen und Durchgängigkeit des Konzeptes überprüfen zu können.

Wir beschränken uns allerdings in folgenden Punkten:

- Wir implementieren jeweils nur das Primär-Szenario fur
  Auftragserfassung
  Produktinformation
  Konten nachführen
  Auftragsstatus abfragen

Warum nur Primär-Szenarios? Wir wollen dir grundsätzliche Durchgängigkeit testen, nicht jeden Sonderfall.

11.2.1.2. Iteration 2 (drei Monate später)

Ziele :
Schnittstellen zum Lagersystem und Schnittstellen zum externen Carrier (DHL, ...)
Im Speziellen:

- Packlisten : Primäres-Szenario
- Produktemengen nachführen : Primares-Szenario
- Nachbestellte Ware einlagern : Primäres Szenario

11.2.1.3. Iteration 3 (ein Monat später)

Ziele:
Alle primären Szenarios sollten abgeschlossen sein. Wir könnten also produktiv starten, allerdings ohne Ausnahmefälle:
Primäres Szenarios:

- Login
- Rücksendungen
- Bestellung stornieren
- Bestellung suchen
- Katalog versenden
- Verkaufs-Statistik generieren und drucken

11.2.1.4. Iteration 4 (ein Monat später)

Ziele:
Vollständiges System, voll funktionsfähig

- Alle sekundären Szenarios
- Alle Fehlerbehandlungsroutinen

11.2.2. Bauen oder Kaufen?

Im Rahmen der Planung müssen wir auch die Frage beantworten, ob man nicht das ganze System kaufen kann oder wenigstens Teile davon.

Dadurch sparen wir uns Entwicklungszeit, verlagern Kosten und begeben uns in eine Abhängigkeit.

Teile müssen wir immer dazu kaufen, da wir unsere Entwicklungsumgebung nicht selber von Grund auf bauen werden.

Risiken beim Kauf bezw. Fragen (einen vollständigeren Fragekatalog ist als Anhang des Kapitels Pflichtenheft vorhanden)

- Wie zuverlässig ist der Anbieter
- Wie gut ist das System
- Wie lange ist die Firma schon im Geschäft
- Wer sind die Kunden
- Wie schwer würde uns en Eingehen der Firma treffen
- Gibt es einen Industriestandard
- Gibt es bestimmte Standards, die berücksichtigt werden sollten
- Wie können wir das System in unsere Umgebung integrieren
- Wie lange brauchen wir, um das neue System kennen zu lernen

- Wie gut sind unsere Entwicklungsleute
- Sind sie auf dem Stand der Technik
- Wie schnell lernen sie Neues
- Sind sie in der Lage, zeitgerecht zu liefern
- Haben wir genügend finanzielle Resourcen, um das Projekt überhaupt zu starten

11.2.3. Prototyping

Unter Umständen bietet es sich an, in der Entwurfsphase mit Prototypen zu arbeiten.

Dadurch können bestimmte Wünsche genauer untersucht werden und eventuell realistischere Anforderungen resultieren.

11.2.4. Aufwandschätzungen mit Hilfe von Use Cases

Das folgende Vorgehensmodell basiert auf der Methode der Funktionspunkte und hat sich im Softwarebereich als recht zuverlässige Methode erwiesen, den Aufwand abzuschätzen.

11.2.4.1. Gewichtsfaktoren

Schauen wir uns als erstes die Akteure in unserem System an. Jeder Aktor wird einer der folgenden Kategorien zugeordnet:

- Leicht
- Mittel
- Schwierig

Beispiel: ein leichter Actor

- Ein anderes System mit einer klar definierten Schnittstelle

Beispiel : mittlerer Actor

- Ein anderes System, welches über ein klar definiertes Protokoll (z.B. TCP/IP) mit unserem System kommuniziert.

Beispiel: komplexer / schwieriger Actor

- Ein System, welches zum Beispiel mit Hilfe einer graphischen Schnittstelle kommuniziert

Wie viele Actors jeder Gruppe gibt es?
Dies liefert eine einfache Analyse und ein Summentotal.

Gewichtung der Actors:

- Leicht : Gewicht = 1
- Mittel : Gewicht = 2
- Schwer: Gewicht = 3

Das Ergibt ein Gesamttotal (Summe).

11.2.4.2. Auftragsabwicklung

Actors und Schwierigkeitsgrad

- Kunde schwierig
- Lagerverwaltungssystem einfach
- Buchhaltung einfach
- Kundeninnendienstmanager mittel
- Kundeninnendienst komplex
- Hilfsarbeiter komplex
- Carrier mittel

Gewichtung gemäss der Tabelle:

Actor Typus	Beschreibung	Faktor
Einfach	Programm Interface	1
Mittel	interaktiv, Protokoll	2
Schwer	GUI	3

Also erhalten wir für unser Beispiel:
2 einfache = 2
2 mittlere = 4
3 schwierige = 9
TOTAL 15

11.2.4.3. Gewichtung der Use Cases

Jetzt versuchen wir das Analoge für die Use Cases.Wir definieren wieder drei Kategorien.

Transaktionsbasierte Gewichtung

Leicht: 3 oder weniger Transaktionen	Gewicht	5
Mittel: 4-7 Transaktionen		10
Schwer: mehr als 7 Transaktionen		15

Klassenbasierte Gewichtung (Klasse = Analyse Klasse : diese werden evtl. später verfeinert)

Leicht	weniger als 5 Klassen	5
Mittel	5 - 10 Klassen	10
Komplex	mehr als 10	15

Dabei kann man etwa von folgender Abschätzung ausgehen:
- Ein einfacher Use Case liefert etwa 3-5 Analyse-Klassen
- Ein mittlerer Use Case liefert 5-10 Analyse Klassen
- Ein komplexer Use Case liefert mehr als 10 Analyse Klassen

11.2.4.4. Auftragsabwicklung

Bestellung eingeben	mittel
Rücksendungen	mittel
Bestellung stornieren	einfach
Bestellstatus abfragen	einfach
Katalogversand	einfach
Verkaufs-Statistik	einfach
Beschwerde erfassen	einfach
Bestellung rüsten	mittel
Nachlieferungen einbuchen	mittel

Also:
5 * einfach = 25
4 * mittel = 40
0 * komplex = 0

TOTAL = 65

Addiert man die Punktzahlen aus der Use Case Analyse und der Actor Analyse, dann bekommt man die sogenannte UNADJUSTED USE CASE POINTS (UUCP)

Im Falle der Auftragsabwicklung : UUCP = 15 + 65 = 80

11.2.5. Technische Faktoren

Als erstes müssen wir die technische Komplexität abschätzen. Dies liefert den TCF den TECHNICAL COMPLEXITY FACTOR

Faktor	**Faktor Beschreibung**	**Gewicht**
T1	Verteiltes System	2
T2	Antwortszeit kritisch	1
T3	Enduser effizient	1
T4	komplexe Programmlogik	1
T5	wiederverwendbar	1
T6	leicht installierbar	0.5
T7	leicht zu benutzen	0.5
T8	portabel	2
T9	leicht anpassbar	1
T10	Mehrbenutzerbetrieb	1
T11	Sicherheitsvorschriften	1
T12	durch Dritte einsetzbar	1
T13	Spezialtraining erforderlich	1

Vorgehen:
Gewichten Sie alle obigen Faktoren mit einem Gewicht von o bis 5. 5 heisst, dass der Faktor für das Projekt wichtig ist, 0 heisst, dass er irrelevant ist.

TFactor = Summe (TLevels * Gewichtung)
TCF = 0.6 + (0.01 * TFactor) : Technical Complexity Factor

Faktor	**Faktor Beschreibung**	**Gewicht**	Beispiel	berechnet	Begründung
T1	Verteiltes System	2	0	0	Nicht geplant
T2	Antwortszeit kritisch	1	3	3	Eingabezeit ist kritischer
T3	Enduser effizient	1	5	5	Wichtig
T4	komplexe Programmlogik	1	1	1	Einfach
T5	Wiederverwendbar	1	0	0	Später vielleicht
T6	leicht installierbar	0.5	5	2.5	Einfach zu bedienen
T7	leicht zu benutzen	0.5	5	2.5	Einfach zu bedienen
T8	Portabel	2	0	0	Noch nicht
T9	leicht anpassbar	1	3	3	Sicher
T10	Mehrbenutzerbetrieb	1	5	5	in etwa
T11	Sicherheitsvorschriften	1	3	3	ja, aber einfach
T12	durch Dritte einsetzbar	1	5	5	Kunden
T13	Spezialtraining erforderlich	1	0	0	Keine Schulung nötig

Als Total erhalten wir 30 Punkte

Jetzt müssen wir noch den technischen Erfahrungslevel abschätzen:

Dies liefert den ENVIRONMENTAL FACTOR EF

EFactor = SUMME(Flevel * Gewichtung)
EF = 1.4 + (-0.03 * EFactor) Environmental Factor

Die Gewichtung ist analog zu vorhin :

- 0 Keine Erfahrung
- 3 durchschnittlich
- 5 hohe Projektmotivation

Faktortabelle:

Faktor Nr	Beschreibung	Gewicht
F1	vertaut mit der Methode	1.5
F2	Anwendungserfahrung	0.5
F3	Objekt Erfahrung	1
F4	Analyse Erfahrung	0.5
F5	Motivation	1
F6	stabiles Umfeld	2
F7	Teilzeit-Mitarbeiter	-1
F8	Schwierige Programmiersprache	-1

In unserem Beispiel:

Faktor Nr	Beschreibung	Gewicht	Wert	Berechnet	Grund
F1	vertraut mit der Methode	1.5	1	1.5	neue Methode
F2	Anwendungserfahrung	0.5	1	0.5	keine Programmierer
F3	Objekt Erfahrung	1	1	1	wie oben
F4	Analyse Erfahrung	0.5	5	2.5	alle machen mit
F5	Motivation	1	5	5	alle wollen etwas zeigen
F6	stabiles Umfeld	2	5	10	keine grossen Änderungen
F7	Teilzeit-Mitarbeiter	-1	0	0	keine Teilzeit-Mitarbeiter
F8	Schwierige Programmiersprache	-1	2	-2	einfach

Als TOTAL erhalten wir 18.5 Punkte!

In unserem Beispiel erhalten wir also folgende Kennzahlen:

TFactor = 30 **EFactor = 18.5**
TCF = 0.9 **EF = 0.845**

11.2.6. Use Case Points

Als grosse Krönung berechnen wir schliesslich noch die Kennzahl

Use Case Points UCP = UUCP * TCF * EF

In unserem Beispiel: UCP = 80 * 0.9 * 0.845 = 60.84

Stellen sich bloss die Fragen:
Was haben wir damit?
Was machen wir damit?
Wozu das Ganze?

Es gilt nun folgender Erfahrungswert:
Je UCP Punkt muss man ungefähr mit 20 Manntagen rechnen. Als obere Grenze setzt man 28 Tage an als untere Grenze 18-20 Tage.

Die Erfahrung zeigt:
In den Projekten muss man versuchen diese Zahlen zu ermitteln und zu verifizieren. Beim zweiten Durchlauf kennt man bereits bessere Richtzahlen, kann die Aufwandschätzung also verfeinern.

Ohne Erfahrungswerte sind diese Schätzungsverfahren SINNLOS!

11.2.7. Beispiel eines Projekt-Proposals

PROJEKT RUDERWELTMEISTERSCHAFTEN LUZERN 2001

AN: BERHARD RAST

VON: JOSEF M. JOLLER

BETREFF: S. OBEN

DATUM: 11.03.02

KOPIEN AN: HTA

11.2.7.1. Ausgangslage

Für die Ruderweltmeisterschaften 2001 auf dem Rootsee ist ein Informationssystem zu schaffen welches alle relevanten Informationen umfasst und einem breiten Nutzerkreis zugänglich macht. Das System soll ab der Ausschreibung der Weltmeisterschaft produktiv im Einsatz sein und diese informationstechnisch unterstützen. Anmeldungen und allfällige Mutationen der Teilnehmer werden laufend nach geführt und stehen damit andern Systemen aktuell zur Verfügung. Das eigentliche Rennprogramm wird Online angeboten, mit entsprechender visueller Unterstützung und Verbindungen zu andern relevanten Informationsquellen. Kern des Informationssystems ist jedoch die eigentliche Auswertung mit Vorläufen, Hoffnungsläufen und Halbfinal bezw. Final.

Als Vorbereitung und Test für das System ist dessen Einsatz in der Saison 1999 und 2000 geplant. Denkbar ist auch, das System so mobil zu gestalten, dass es auch an anderen Wettbewerben, nach entsprechender Anpassung, nutzbar wird.

11.2.7.2. Vorgehensweise

Das Projekt soll in Phasen abgewickelt werden. Das Vorgehensmodell sieht wie folgt aus:

11.2.7.3. Startphase

Die Startphase soll ein besseres Bild über das Gesamtprojekt liefern. Die einzelnen Informationen liegen aber erst in einer Form vor, dass entschieden werden kann wie das Projekt weitergeführt bezw. was alles in das Projekt "gepackt" werden soll.

Falls sich einzelne Anforderungen als zu ehrgeizig oder zu innovativ erweisen, kann in dieser Phase noch eine Korrektur der Projektziele vorgenommen werden.

Das setzt voraus, dass speziell die risikoreichen Projektteile genauer untersucht werden.

11.2.7.4. Ergebnisse am Ende der Startphase:

Grobes Modell aller zu bearbeitenden Problembereiche (domains)

- Generelle Übersicht über alle wichtigen Nutzerforderungen innerhalb der verschiedenen Nutzungsbereiche
- Grober Phasen- und Projektplan (Zeit, Meilensteine, personelle und finanzielle Resourcen), beteiligte Firmen, Schulen Mitarbeiter; Finanzierungsplan;
- Überblick über die zu erwartenden Projektkosten und die Erfolgskriterien für das Projekt
- Erste Identifizierung der Risikofaktoren
- Allgemeine Übersicht über das Use Case Modell (10-20% fertig) mit Use Case Beschreibung
- Ein Glossar der wichtigsten Begriffe zur Vervollständigung zwischen Benutzer und der Informatik.

es kann sich als notwendig erweisen, in der Startphase bereits einen Prototypen zu erstellen. Dieser dient dazu, bestimmte Systemeigenschaften (Performance, Benutzerschnittstelle) zu testen und besser planbar zu machen.

Der erste Prototyp ist in der Regel ein Wegwerfprodukt, wird also nie produktiv.

Ein weiterer Prototyp kann dann evolutionär zum eigentlichen Produkt weiter entwickelt werden.

Die Startphase kann sehr kurz sein, einige Tage oder wenige Wochen. Geplant ist, dass diese Phase Ende Februar abgeschlossen wird.

11.2.8. Die Entwurfsphase

In der Entwurfsphase ist der Anwendungsbereich zu analysieren. Wesentlich für diese Phase ist die Entwicklung eines tragfähigen Architekturmodelles, sowie einem detaillierten Projektplan.

Die Risikoanalyse, die in der Startphase bereits gestartet wurde, wird fortgesetzt und vertieft und Lösungsmöglichkeiten beziehungsweise worst case Szenarien ausgearbeitet.

Die wesentlichen Ergebnisse der Entwurfsphase:

- Use Case Modell (ca. 80% vollständig), für alle Bereiche, speziell die riskoreichen
- Softwarearchitektur, die tragfähig für das Projekt ist
- Verfeinerter Projektplan, der gegebene *Rahmenbedingungen* berücksichtigt (Hardware, Systemsoftware, Netzwerke, Datenbanken) der allfällig planbare *Iterationen* berücksichtigt

11.2.8.1. Welche Risiken können typischerweise auftreten:

Anforderungsrisiken : bauen wir das richtige System

Technologische Risiken : setzen wir die richtige Technologie ein?

Politische Risiken : wer hat welche Interessen

Die Entwurfsphase dauert wenige Wochen. Geplant ist, dass diese Phase Ende März / Mitte April abgeschlossen wird.

11.2.9. Die Konstruktionsphase

In der Konstruktionsphase sind alle Software-Produkte iterativ fertig zu stellen, inklusive Tests.

Ausgangspunkt der Softwareerstellung ist eine detaillierte Softwarearchitektur, die in der Entwurfsphase erstellt wurde, sowie deren Abbildung auf eine vorhandene bezw. konzipierte Hardwareumgebung.

Die restlichen Use Cases müssen vervollständigt werden.

11.2.9.1. Die wesentlichen Ergebnisse der Konstruktionsphase

- Erstellte und getestete Software
- Benutzermanual
- Beschreibung des aktuellen Releases

Die Konstruktionssphase dauert zwei bis drei Monate. Geplant ist, dass diese Phase Mitte Juni abgeschlossen wird.

Phase 1 : Startphase

Beteiligte:

Herren B. Rast, J.Joller

In dieser Phase wird das Realisierungsteam, allfällige Sponsoren ... zusammen gestellt.

Geplant ist eine Zusammenarbeit mit: Schule für Gestaltung Designs), Swisscom / Blue Window (Web Hosting und High Speed Kommunikation); gesucht wird ein Sponsor für Hardware und Systemsoftware (lokaler Anbieter, global Player)

Es ist unser Ziel, in dieser Phase bereits erste Skizzen eines möglichen Web Auftrittes, sowie eine Skizze der Gesamtarchitektur zu erarbeiten.

Phase 2 : Entwurfsphase

Beteiligte:

Wie oben; zusätzlich abhängig von der Phase 1 weitere Mitarbeiter (Schule für Gestaltung, Swisscom, BlueWindow)

Phase 3 : Konstruktionsphase

Beteiligte:

Wie Phase 2; zusätzlich Studenten (und je nach Ergebnis der Phase 2 : Swisscom, BlueWindow, Schule für Gestaltung).

11.2.10. Dokumentation der Use Cases (Dokumentations-Template)

Es sind verschiedene Raster im Umlauf. Eines wird mit der Objectory Methode mitgeliefert.

11.2.10.1. Systembeschreinungs-Template

System Name
Kurze Beschreibung, bei komplexen Systemen eventuell mehrere Seiten. Keine detaillierte Anforderungsbeschreibung, eher eine kurze Gesamtübersicht!

Risikofaktoren
Liste der Risiken und deren Priorität.

Use Cases auf Systemebene
Liste mit Aktoren und eine Liste der Use Cases

Architektur Diagramm
Beschreibung der Packages und der Schnittstellen, als Diagramm oder als Liste.

Subsystem-Beschreibung
Kurze Beschreibung der einzelnen Subsysteme.

11.2.10.2. Use Case / Anwendungsfall Beschreibungs- Template

Use Case Namen
Beschreibung des Use Cases, grob ein Paragraph.

Actors
Eine Liste der Actors, die am Use Case beteiligt sind.

Priorität
Wie wichtig ist der Use Case für das Projekt?

PreConditions / Prä-Konditionen / Vorbedingungen
Welche Bedingungen müssen erfüllt sein, damit der Use Case überhaupt gestartet werden kann?

Post Conditions / Nachbedingungen
Eine Liste der Bedingungen, die nach dem Use Case zutreffen (müssen), unabhängig vom durchlaufenen Szenario.

Extension Points / Erweiterungen
Falls der Use Case Erweiterungen hat, dann müssen diese aufgelistet werden.

"Benutzte" Use Cases
Welche Use Cases benutzt der aktuelle?

Flow of Events /Ablaufbeschreibung
Primäres Szenario evtl. mit Ergänzungen.

Aktivitätendiagramm
Ein Aktivitätendiagramm für all jene Abläufe, die zu komplex sind um selbstredend verständlich zu sein.

User Interface / Benutzerschnittstelle
Ein Storyboard oder ein sonstwie gemachter Prototyp der Benutzerschnittstelle.

Secondary / Sekundäre Szenarios
Beschreibung der alternativen Abläufe und der Ausnahmen (Exceptions), zusammen mit einer kurzen Beschreibung (als Basis für die Implementierung).

Sequence Diagramme
Falls Sie den Ablauf als Sequenz-Diagramm dargestellt haben, dann gehört diese Beschreibung zur Use Case Dokumentation.

Untergeordnete Use Cases
Falls der Use Case auf untergeordnete / subordinate Use Cases aufbaut, dann müssen diese kurz erwähnt (evt. Hinweis auf entsprechende Dokumentation) werden.

Partizipierende Klassen
Falls ein Klassendiagramm erstellt wurde, die Packages also weiter verfeinert wurden, gehört diese Dokumentation zum Use Case .

Weitere Fakten
Referenzen auf weitere Systeme und Subsysteme, weiteren Unterlagen, eventuell bereits erstellten Design-Dokumenten.

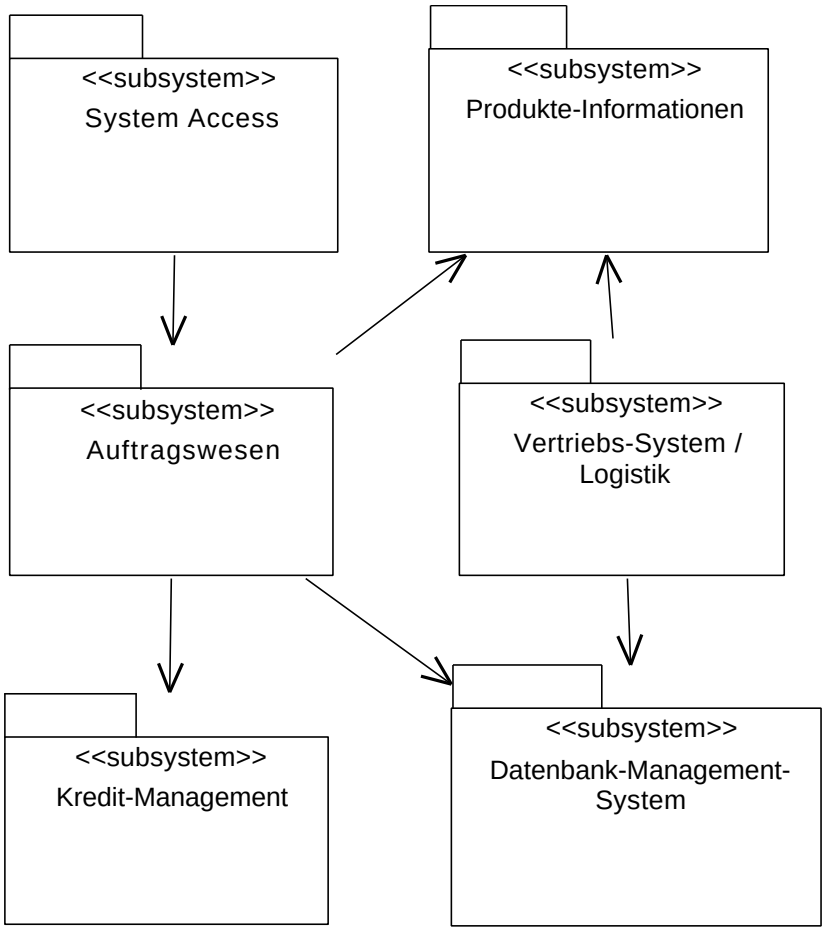
Weitere Anforderungen
Führen Sie weitere Anforderungen auf, die Sie bisher "unterschlagen" haben zum Beispiel : spezielle Anforderungen wegen dem Einsatz des Equipments in einer speziellen Umgebung (Schlachterei : Spritzwasser; Hochofen : Hitzebeständigkeit; Gebirge : generelle Wetterfestigkeit).

11.2.11. Ein Beispiel

11.2.11.1. Systemlevel-Beschreibung

Es soll ein Online Auftragsabwicklungssystem geschaffen werden. Dieses System soll eine Schnittstelle zum Finanz- und Rechnungswesen haben und zur Vertriebslogistik.

11.2.11.2. Architektur



System Subsystem
Dieses Subsystem überprüft den Systemzugriff (login, logout) und prüft die Zugriffsrechte.

Produkte-Information
Dieses Subsystem ist in der Lage, alle Informationen auf Artikelbasis zu liefern (Artikelstamm).

Auftragswesen
Aufträge, Rücksendungen, Status der Bestellung und Bestell-Stornierungen.

Logistik
Packlisten, Labels, Vertriebskosten / Frachtkosten werden von diesem System berechnet und weiter belastet.

Kredit-Management
Schnittstelle zum Finanz- und Rechnungswesen, verantwortlich für das Verbuchen von Zahlungen und Gutschriften. Bearbeitet Kreditkarten, Checks, ...

Datenbank-Management-System
Applikatorische Daten werden im Datenbank-Subsystem gespeichert.

Use Case - Produktinformation
Dieser Use Case liefert Produkteinformationen aus dem Lagersystem

Actors
Lagerverwaltungssystem

"Benutze" Use Cases
keine

Ereignisfluss
Basis Pfad
1. Dieser Use Case beginnt sobald der Produktecode eigegeben wurde
2. Das System sendet eine Anfrage an das Lagerverwaltungssystem und verlangt bestimmte Produkteinformationen, basierend auf dem Produktecode.
3. Das Lagerverwaltungssystem liefert die Produkteinformation
   Beschreibung
   Preise
   Ware an Lager

Aktivitätsdiagramm
Keines

User Interface
Keines

Sekundäres Szenario
1. Produkt existiert nicht
2. Produkt wird nicht mehr geliefert
3. Das Lagerverwaltungssystem ist nicht aktiviert

Untergeordnete Use Cases
Produktinformation - liefer Produktinformation

11.3. UML - Use Cases : Reviews

In jedem Projekt müssen wir periodisch Review durchführen.
Reviews werden aus verschiedenen Gründen und mit verschiedenen Zielen durch geführt.

11.3.1. Vollständigkeits-Review

Hier eine Liste möglicher Punkte die untersucht werden sollten:

- Stimmt die Projektbeschreibung noch
- Kennen wir die erfolgskritischen Faktoren
- Haben wir neue Zusatzerkenntnisse gewonnen, die für das Projekt wichtig sind
- Stimmt unsere Risiko-Einschätzung noch
  Sind Risiken verschwunden
  Sind neue Risiken sichtbar geworden
  Hat sich die Priorität geändert
- Wurden aus einzelnen Annahmen in der Zwischenzeit Tatsachen
  Welche neuen Annahmen gibt es

Architekturfragen:

- Passen die Use Cases zu unserer Architektur
- Stimmen die Subsysteme
  Hat sich deren Beschreibung geändert
- Stimmen die Schnittstellen noch
  Treten neue Schnittstellen auf
- Stimmen die Schnittstellen zu den Actots
- Wurde die Architektur mit Hilfe von Szenarios getestet
  Funktioniert die Architektur
  Ist sie stabil

Use Case Fragen:

- Gibt es neue Actors
  gibt es Actors im System, die wir nicht mehr benötigen
- Hat sich die Rolle der Actors geändert
  Sind Actors Teil des Systems geworden
  Sind Teile des Systems ausgelagert worden und wurden dadurch zu Actors
- Stimmt die Bezeichnung der Actors und deren Funktion noch
- Gibt es neue Use Cases
  Sind bestimmte Use Cases unnötig
  Stimmt die Beschreibung der Use Cases noch
- Haben wir pro Use Case mindestens ein Szenario
- Stimmt das Interface Diagramm it den Use Cases überein

11.3.2. Review des Problempotentials

Typische Fragen:

- Haben wir die Ausnahmefälle berücksichtigt
- Stimmen unsere Annahmen betreffend Exceptions noch
- Haben wir ein Worst Case Szenario definiert
  für das Projekt
  für die betrieblichen Abläufe
  für die Einführungsphase

11.3.3. Review mit dem Endbenutzer

Typische Fragen:

- Erfüllt das System die Benutzeranforderungen
- Fehlen teile des Systems
- Liefert das System zuviel
- Verstehen Sie wie das System funktioniert
  als Ganzes
  in Ihrem Bereich
- Ist das System aus Ihrer Sicht angenehm zu bedienen
- Verhält sich das System so, wie Sie es erwarten

11.3.4. Review mit Kunden

Typische Fragen:

- Stimmen unsere Annahmen
- Was sollte Ihrer Ansicht nach geändert werden
- Verstehen Sie was das System Ihnen liefert
- Verstehen WIR was der Kunde möchte
- Verstehen Sie wie das System funktioniert
  bei Ihnen
  insgesamt

11.3.5. Review mit den Entwicklern

Der Entwickler weiss sowieso alles besser!

- Macht das Ganze Sinn
- Kann man basierend auf den Use Cases ein System bauen
- Was brauchen wir sonst noch bevor wir mit der Entwicklung beginnen können

11.4. UML - Use Cases : Implementierung, Architektur und Betrieb

Wie gelangt man nun von der Beschreibung der Use Cases zu einer Implementation des Systems, und zwar so, dass das System möglichst lange genutzt werden kann. Wesentlicher Faktor für eine lange Nutzung ist eine gute Architektur. Die Architektur ergibt sich aus der Abstraktion der Applikations-Domäne. Diese hat auch viel mit der Suche nach den Entitäten beim Datenbank-Design zu tun. Das Vorgehen ist entsprechend analog.

11.4.1. Abstraktion der Applikations-Domäne

Schlüssel zu einer guten Abstraktion ist die Identifikation der für das System wesentlichen Funktionsblöcke. Inwiefern tragen nun die Use Cases zu einer guten Abstraktion bei?

11.4.1.1. Identifikation der Schlüsselabstrationen in Use Cases

Einige der Schlüsselabstraktionen ergeben sich aus der Aufgabenstellung. In einem Auftragsabwicklungssystem werden wir sicher primär mit Aufträgen zu tun haben.

In der Regel kann man die Abstraktionen des Anwendungsgebietes in drei Kategorien einteilen:

1. Entitäten
2. Schnittstellen
3. "Controller"

Eine Entität ist etwas, worüber wir Informationen sammeln und speichern möchten. Entitäten beschreiben den Zustand unseres Systems, also die Daten.

Schnittstellen beschreiben in der Regel die Kommunikation zwischen dem Use Case und dem Actor. In der Praxis handelt es sich also um ein Benutzerinterface, ein Programmaufruf oder etwas ähnliches.

Ein "Controller" ist etwas, das die Logik des Systems kennt und eine Aktivität steuert. Ein Controller kann beispielsweise den Zugriff auf eine Resource überwachen. Resourcen sind Drucker, Software, ..., Datenbanken.

11.4.1.2. Beispiel : die Schlüsselabstraktionen einer Auftragsabwicklung

Auftrag	-	Liefer-Adresse, Produkteliste mit Preisen, Zahlungsbedingungen
Produkt	-	Produkteinformationen : Produktcode, Beschreibung, Preis, Lagerbestand
Kunde	-	Informationen über den Kunden, seine Adresse und evtl weitere Angaben
Bildschirm	-	die Eingabemasken (als Schnittstellen)
Datenbank-Manager		Controller für die Datenbankzugriffe

11.4.2. Darstellung der Szenarios mit Hilfe von Diagrammen

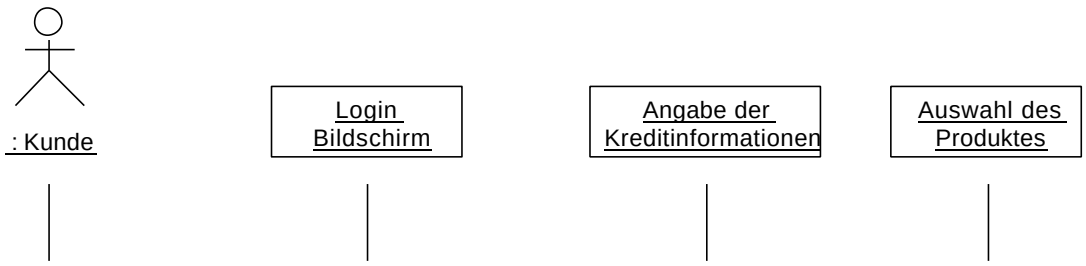
In UML verwendet man zur Darstellung von Abläufen und dem Zusammenspiel der einzelnen Abstraktionen in der Regel das *Sequence-Diagramm*.

Diese werden wir im nächsten Themenblock (Analyse) noch genauer kennen lernen.

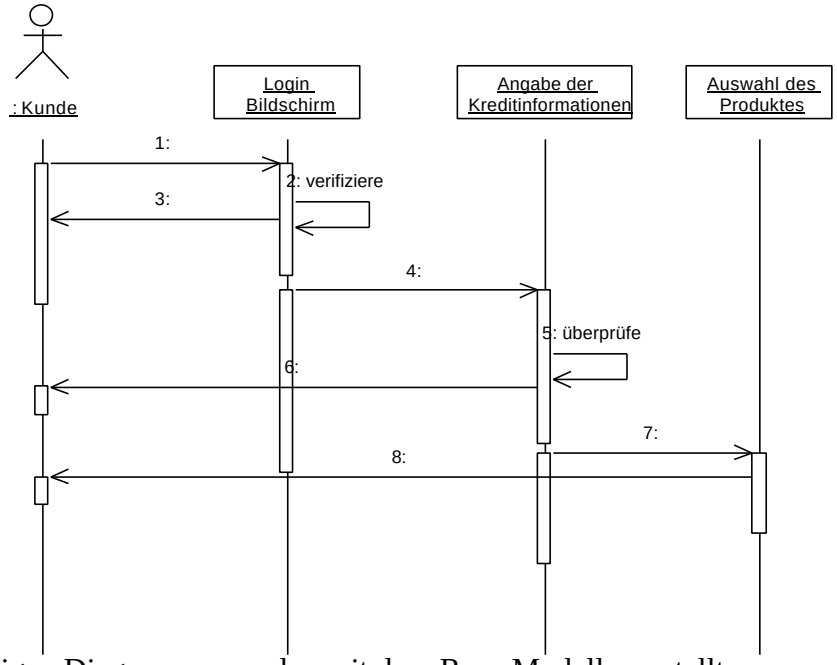
Wie gelangt man zu einem Sequence-Diagramm?

- Wählen Sie einen Use Case aus
- Starten Sie mit dem Primärszenario
- Identifizieren Sie die Schlüsselabstrationen, die im Primär-Szenario verwendet werden.
- Zeichnen Sie diese (entweder mit Rose oder von Hand) oben auf einer Seite als Kasten und beschriften Sie diese Kasten
- Ergänzen Sie die Liste durch die Actors : links und evtl. rechts auf der Seite

Ihre Seite sieht nun beispielsweise so aus:



- Tragen Sie jetzt das Verhalten, das eigentliche Szenario, im Diagramm ein:
  Wie sieht das Zusammenspiel zwischen diesen Schlüssel-Abstraktionen aus?
  Ein Beispiel :



Die obigen Diagramme wurden mit dem Rose Modeller erstellt.

Die Diagramme sind sehr anschaulich aber auch sehr hilfreich. Wenn wir die Verbindungen mit den "Methoden" der "Objekte" beschriften, dann haben wir ein mächtiges Hilfsmittel zum Finden und Beschreiben von Klassen, Objekten und deren Methoden:

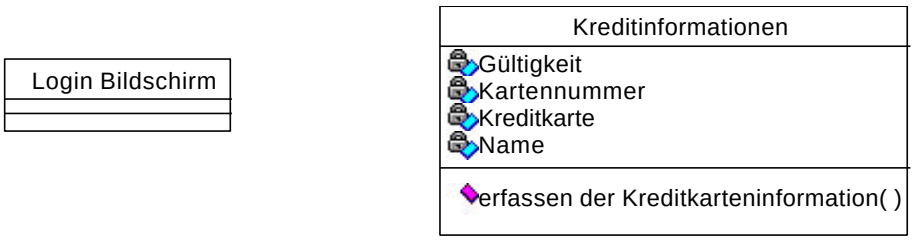
- oben stehen die Objekte
- die Verbindungen stellen Methodenaufrufe (des Zielobjektes!) dar
- senkrecht sehen wir das Zeitverhalten

Der Benutzer ist häufig überfordert mit diesen Diagrammen : sie sind (schon) zu abstrakt.

11.4.3. Diagramme für Schlüsselabstraktionen

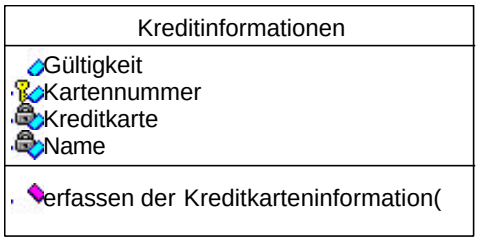
Die Schlüsselabstraktionen sind Kandidaten für die in unserem System vorhandenen Klassen. Diese werden in UML als Rechtecke dargestellt. Wir nehmen also die Abstraktionen aus dem Szenario-Diagramm und stellen diese in einen anderen Kontext: das *Klassendiagramm* des Use Cases.

Aus den Sequence Diagrammen sehen wir aber wesentlich mehr: die Methoden der Klassen können wir ebenfalls identifizieren. Diese werden im Klassendiagramm in einem separaten "Kasten" des Klassendiagramms. Betrachten wir ein Beispiel:



In der Mitte befinden sich die Datenfelder, also die Beschreibung des Zustandes des "Objektes" und unten finden wir die "Methoden", jeweils qualifiziert mit Zugriffsrechten.

Da die Kreditkartennummer vermutlich der Schlüssel sein wird, müssten wir in unserem Diagramm das entsprechende Datenfeld entsprechend kennzeichnen:



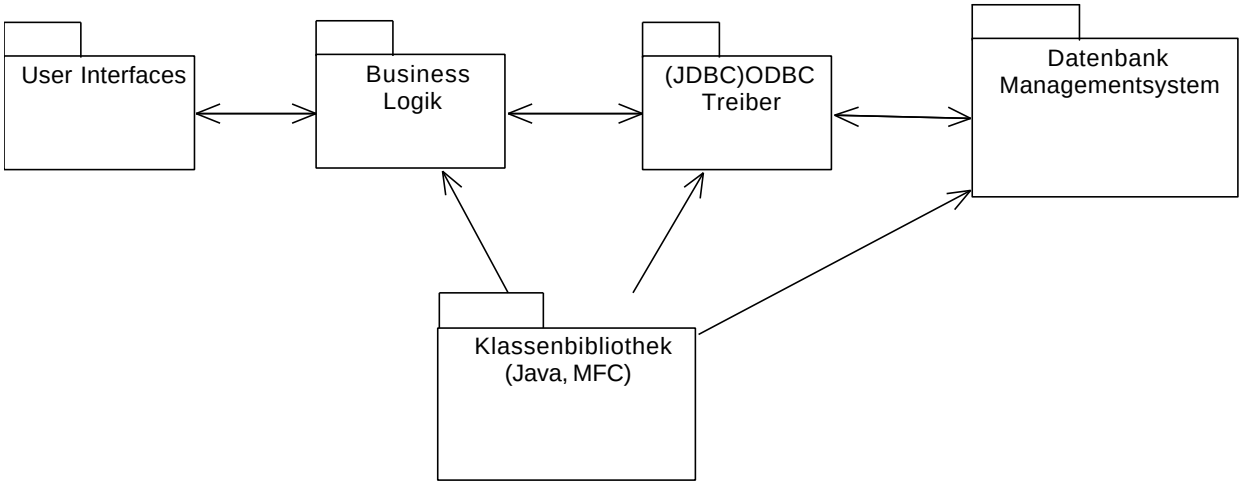
Eine Variante des "Restes" kennen Sie bereits aus der Datenbank-Theorie : die Diagramme lassen sich in die dritte Normalform bringen. Das hat auch bei Klassen und Objekten seine Bedeutung (*has-a, is-a* Relationship).

11.4.4. Use Case und Architektursicht

Für unsere Systembeschreibung haben wir zwei unterschiedliche Beschreibungsformen gewählt:

- Die Use Case Sicht, die uns hilft das System aus der Sicht des Benutzers zu beschreiben
- Die Architektur Sicht, die uns wesentlich hilft, das System als Ganzes in sinnvolle Subsysteme zu zerlegen und die Subsysteme zu entkopplen. Die Architektursicht kann sehr hilfreich sein, wieder verwendbare Subsysteme zu finden. Zudem können wir damit auch besser entscheiden, ob und welche Systemteile wir selber bauen, bezw. kaufen wollen.

UML verwendet auf der obersten Ebene die Packages als Strukturierungsmittel. Wir haben bereits mehrere Beispiele gesehen.



11.4.5. Iterationsplanung

Nachdem wir uns so lange und intensiv mit Konzepten und Abläufen befasst haben, ist die Planung des Gesamtablaufes des Projektes wesentlich einfacher. Oft ergibt es sich recht natürlich, Teile des Systems abzuspalten und in einer "zweiten Runde" zu realisieren.

Die Implementierung des Systems erfolgt wie bereits in der einführenden Übersicht über verschiedene Phasenmodelle in den "Schritten":

- Analyse
- Design
- Kodierung
- Testen

Zum Testen können wir wieder auf die Use Cases zurück greifen: die Use Cases beschreiben bereits im Detail was der Benutzer erwartet!

Wie immer müssen wir nach Abschluss der Use Case Beschreibung einen Blick zurück machen und versuchen zu verstehen, was wir alles gut oder schlecht gemacht haben und was wir demnach in Zukunft besser machen könnten (Riskofaktoren ,....).

11.4.6. Ablieferung des Produktes

Der Kunde erwartet in Regel folgende Dokumentationen und Ergebnisse:

- Benutzerhandbuch
- Trainingsunterlagen
- Demos

Der Verkauf erwartet seinerseits zusätzlich (und vorallem):

- Verkaufsunterstützende Unterlagen
- Marketing Pläne
- Werbekampagnen
- Verkaufskits

Auch diese Unterlagen basieren wesentlich auf den Use Cases!

Was passiert *nach* der Ablieferung des Produktes mit den Use Cases?

- Use Cases helfen ein Fachgebiets-Know How aufzubauen (Requirement Patterns, Referenzmodell, ...)
- Use Cases helfen neuen Mitarbeitern sich in ein Gebiet einzuarbeiten
- Das Know How der Projektmitarbeiter wird gespeichert und wird dadurch wiederverwendbar

11.5. UML - Use Cases : Phasen und Ergebnisse

Als Abschluss der Use Case Beschreibungen, hier noch eine Zusammenstellung der Ergebnisse der einzelnen (zeitlichen) Phasen der Projektabwicklung gemäss ROP.

Inception Phase / Startphase
- Ergebnisse
  - Projektbeschreibung
  - Risikoanalyse
  - Use Case Diagramme
  - Beschreibung der Actors und der Use Cases
  - Projektvorschlag

Elaboration Phase / Entwurfsphase
- Ergebnisse
  - Detaillierte Beschreibung der Primärszenarios
  - Sekundär-Szenarios
  - Aktivitäts-Diagramme
  - Optional : Benutzerinterface
  - Architektur
  - Projektplan

Construction Phase / Konstruktionsphase
- Ergebnisse
  - Iterationsplan
  - Programmcode
  - Testplan
  - Testergebnisse
  - Review der Iteration
  - Benutzerdokumentation
  - Trainings-Unterlagen
  - Demos
  - Evtl. Verkaufs- und Marketing-Unterlagen