Anforderungsmanagement

Hierbei geht es, dass man als Manager seinen Kunden „kennen“ muss um zu verstehen, welche Wünsche der Kunde an das zu entwickelnde Produkt hat. Man muss den Kontext dahinter verstehen, da sonst keine Verständigung auf einer gleichen Ebene stattfinden kann. Vielleicht verdeutliche ich Ihnen das mit einem kleinen Beispiel, anhand eines komplett auf dem Kontext gezogen Satzes:

> Mary had a little lamb.

(englischsprachiger Kinderreim, aus dem 19. Jahrhundert, Quelle: Wikipedia)

Ganz Frei aus dem Kontext gezogen ist die Bedeutung des Satzes vielseitig. Und kann demnach kenn eigentliche Aussage nicht eindeutig bestimmt werden. Unterteilen wir den Satz in seine einzelne Bestandteile:

Mary : Das Subjekt, des Aussagesatzes, das eine Person mit dem Namen Maria ist.

had : Das indikative Präteritum von „to have“ für die 3. Person ist und dessen Bedeutung folgende sein kann:

- to hold in possession as a property, also etwas besitzen
- to trick or fool someone (been had by a partner), jemanden veräppeln
- to beget or bear (have a baby), etwas in sich tragen, z.B. ein Baby
- to partake (have as a dinner), etwas praktizieren, ein Essen haben

Ich habe keine Linguistik studiert, mit dem Spezialgebiet der englischen Sprachwissenschaft, aber ich bin mir sicher, dass es weit aus mehr Möglichkeiten gibt „to have“ zu übersetzen.

a lamb : Die Bestimmung des eigentlichen Objektes des Aussagesatzes ist dann schon wieder sehr fragwürdig und schwierig, da es wieder viele Übersetzungsmöglichkeiten gibt. Weitere Möglichkeiten.

- a young sheep less than one year, ein junges Schäfchen, Lamm
- the young of various other animals (antelope etc.), Jungtiere anderer Tiere
- a person as gentle or weak as a lamb, eine schwache Person
- a person easily cheated or deceived, eine betrogene Person
- the flesh of lamb used as food, frisches Lammfleisch, oder auch Lammkotelett
- kann auch für etliche Synonyme stehen

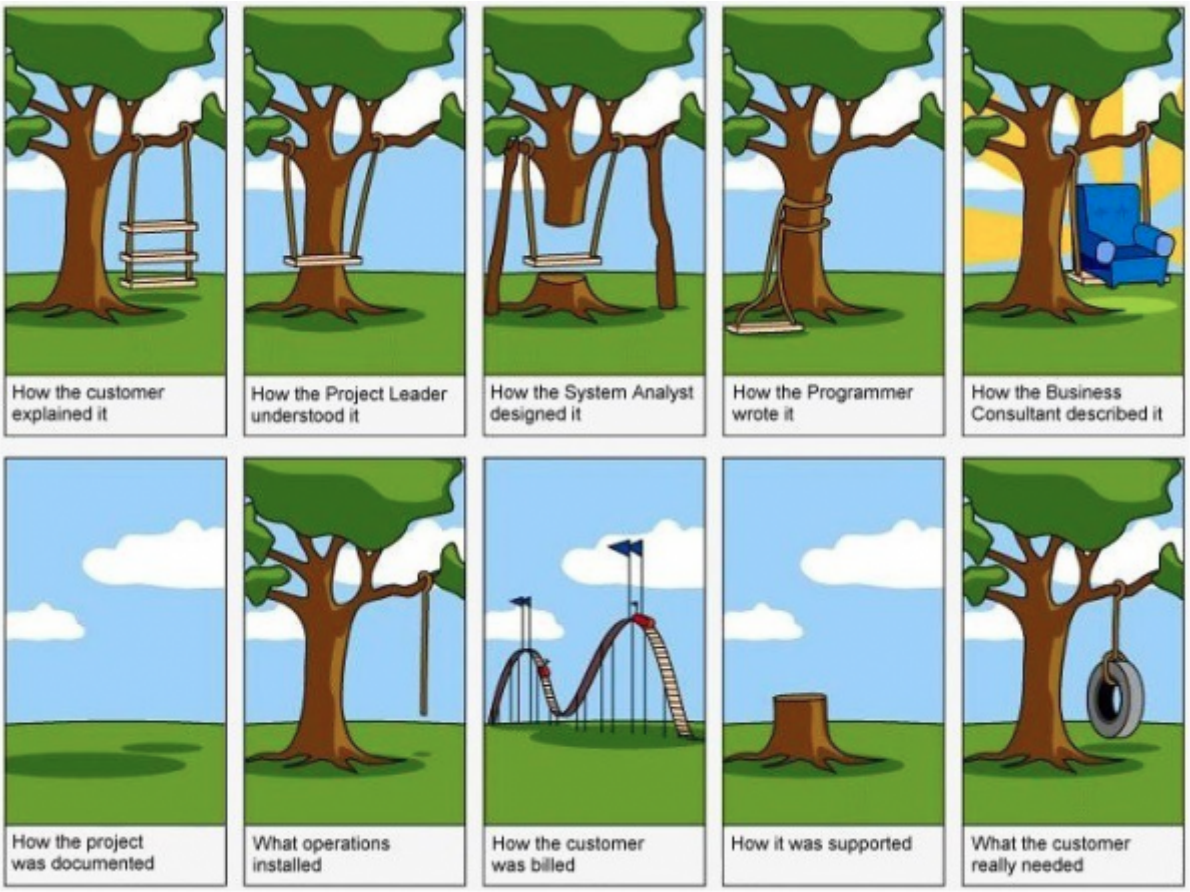
Somit besteht also viel Interpretationsfreiheit, denn Mary kann:

- Mary owned a little sheep under one year
- Mary gave birth to an antelope
- Mary ate the flesh of lamb
- Mary had a gentle or weak person to care of
- …

Klar können wir einige Interpretationen ausschliessen, da diese logischerweise nicht möglich sind. Aber da es sich ja um eine Sprichwort handelt, muss es nicht unbedingt realistisch sein.

Ein anderes Beispiel ist die Aussage: „Shoes must be worn.“ Steht hier „must be worn“ für „müssen getragen werden“ und wenn ja, wie? Oder steht es für „müssen abgenutzt werden“? Für eine

Aussenstehende Person, welche keinen Bezug zum Thema / Problem hat, ist es oft sehr wage eine genaue Beschreibung oder Aussage zu treffen. Ebenso ist es bei Anforderungen bei Projekten.



Quelle: medium.com

Requirements

Requirements Anforderungen oder auch Voraussetzungen sind die Beschreibungen, für die vom System bereitgestellten Dienste und gleichzeitig die betrieblichen Einschränkungen. Hierbei muss klar unterscheiden werden zwischen den Anforderungen an oder vom Benutzer und den Anforderungen an das System, auf dem Ihre Software laufen soll.

Benutzeranforderungen

Sie geben in schriftlicher, natürlicher Sprache unter Verwendung von Diagrammen an, welche Dienstleistungen / Eigenschaften Ihr Produkt besitzen muss. Es wird erwartet, dass das System die (Mindest-) Vorraussetzungen liefert die es zum Betreiben der Software haben muss. In der Regel werden diese Anforderungen schriftlich vom Kunden angegeben und bilden die Grundlage für das „Lastenheft“.

Systemanforderungen

Legt im Detail die Funktionen, Dienste und Betriebsabläufe des Systems fest eben so wie Einschränkungen. Oft werden diese Anforderungen im System als Anforderungsdokument (auch Funktionsspezifikation genannt) festgehalten. Diese Systemanforderungen sind dann in der Regel auch Bestandteil des Vertrags, auf die sich der Kunde wie auch Sie jederzeit beziehen können. Daher sollten die Anforderungen so genau wie nur möglich spezifiziert sein und nur wenig Freiräume geben. Die Systemanforderungen werden zusammen mit Softwareentwickler, Auftragnehmer und Kunde erfasst und bilden die Grundlage für das „Pflichtenheft".

Beispiel für eine Definitionen der Benutzeranforderung:

> Das Bibliothekssystem muss alle vom Urheberrecht geforderten Daten, der Lizenzagenturen in Großbritannien und anderswo überwachen.

Beispiele für eine Definitionen der Systemanforderung:

> Bei der Anforderung eines Dokuments aus dem Bibliothekssystem wird dem Antragsteller ein Formular vorgelegt, in dem der Benutzer Angaben über sich und seine Anforderung machen muss.
>
> Das Anforderungsformulare wird dann im Bibliothekssystems für fünf Jahre ab dem Datum der Anfrage gespeichert.
>
> Alle Anforderungsformulare des Bibliothekssystems müssen anhand des Namens nach Benutzer, des angeforderten Materials und vom Lieferanten der Anfrage identifizierbar sein.
>
> Das Bibliothekssystem muss Protokoll über alle Anfragen führen, die bereits ausgeführt wurden.

Sichtweisen

Aus der Sicht eines Entwicklers können wir zwischen funktionalen und nichtfunktionalen Anforderungen unterscheiden.

Funktionale Anforderungen Die Funktionale Anforderungen geben an, wie das System auf bestimmte Eingaben reagieren oder nicht reagieren soll. Und wie das System dann mit speziell dieser Eingabe in bestimmten Situationen verhalten oder nicht verhalten soll.

Zusammenfassung

Um diese Anforderungen zu erfüllen, ist es notwendig, „Code schreiben".

Nicht-funktionale Anforderungen

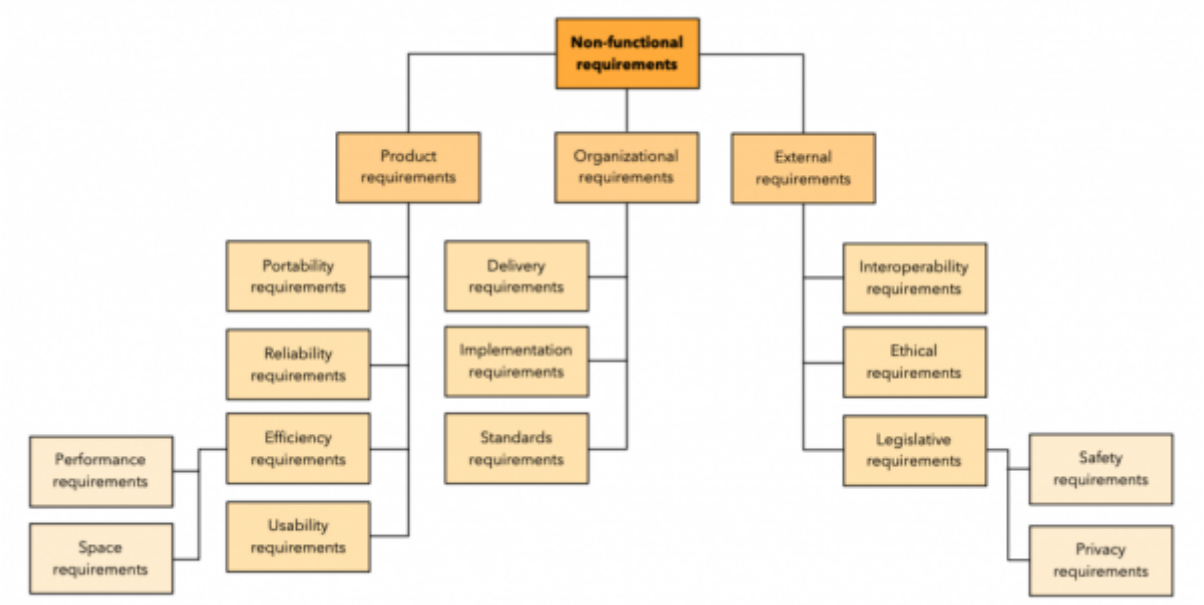
Einschränkungen für die vom System angebotenen Dienste oder Funktionen, welche beispielsweise zeitlich oder technische bei der Entwicklung vorliegen. Diese Einschränkungen gelten oft für das gesamte System. Darum is es normalerweise nicht möglich, einen genau definierten Code zu schreiben, der diese Anforderungen erfüllt. Es ist jedoch manchmal möglich Tests für diese

Einschränkungen zu schreiben, die diese Anforderungen erfüllen!

Die Grenzen zwischen funktional und nicht funktional

Die Anforderungen an Ihr Produkt sind nicht immer eindeutig. So ist die Nichtfunktionale Anforderung an Ihr Produkt: „Das Produkt muss sicher sein!“ eine auf ersten Blick eine nicht funktionale Anforderung, doch auf einen zweiten detaillierteren Blick können einzelne funktionalen Anforderungen daraus gemacht werden. In dem man die Sicherheit genauer beschreibt. Beispielsweise das System muss sicher gegen falsche Eingaben sein usw.

Quelle: Johner Institut



Quelle: Software Technology Group TU Darmstadt VL. WS18/19 EISE

Organisatorische Anforderungen

- Lieferanforderungen, z. wie das Produkt geliefert wird (online, verpackt,…) und wie es geliefert wird
- Art der Verwendung: „nur“ online oder off- und online
- Anforderungen an die Implementierung, z.B. die zu verwendende Programmiersprache oder die Tools bzw. Plattformen, die verwendet werden sollen.
- Normanforderungen, z.B. ISO-Normen oder auch zu verwendenden Entwicklungsprozesse

Externe Anforderungen

Anforderungen, die außerhalb des Systems liegen, z.B. gesetzliche Anforderungen:

- die Verwendung bestimmter Bilder ist auf Grund unterschiedlicher Altersbeschränkungen in verschiedenen Regionen nicht möglich
- Vorgaben wie Daten gespeichert und archiviert werden sollen
- welche Informationen als sensibel gelten und welche Schutzvorkehrung dafür getroffen werden muss

Product Requirements oder Produkt Anforderungen

- Portabilität, muss unter Windows, Linux, Mac,… laufen
- muss mit allen Android-Telefonen kompatibel sein, die eine Auflösung von mindestens XY Pixel haben.
- Zuverlässigkeit, die Sicherheit, dass die Software ordnungsgemäß in einer spezifizierten Umgebung läuft.
- Anforderung an die Effizienz, die Software darf nicht mehr als x MB RAM verwenden oder die Berechnung muss in x Sekunden abgeschlossen sein

Oft sind nicht-funktionale Anforderungen kritischer als einzelne schwierige funktionale Anforderungen. So ist beispielsweise ein Bankensystem, das den Export eines Kontoauszuges im PDF Format nicht unterstützt eher noch verwendbar, als ein Banksystem das nicht sicher ist.

Wie kann man beurteilt, ob eine nicht-funktionale Anforderung erfüllt ist? Folgende nicht funktionalen Anforderung an einen eCommerce Web-Shop: „Die Benutzeroberfläche sollte einfach zu bedienen sein.“ Was ist eine einfache, intuitive Benutzeroberfläche? Diese hängt doch eng mit der Empfindung und der Fähigkeit des einzelnen Benutzer zusammen.

- Die Anzahl der Fehler aufgrund von Eingabefehlern durch Benutzer, soll bei der serverseitige Überprüfung der Formulare, unter X Prozent sein.
- Ein durchschnittlicher Benutzer soll eine Bestellung in unter X Minuten durchführen können.
- Die Anzahl der nicht abgeschlossenen Transaktionen sollte Z nicht überschreiten.

Diese Anforderungen können ein indirektes Maß für die Qualität einer Benutzeroberfläche sein. Aber können nicht oder nicht vollständig in Code umgesetzt werden.

Domainanforderungen

Domainanforderungen werden von der Anwendung abgeleitet, dabei ist das zu verwende Systems ausschlaggebend und nicht die Bedürfnissen der Nutzer. In der Regel sind domänenspezifische Anforderungen nur schwer zu erklären und werden aus Sichtweise der Software-Ingenieuren nicht verstanden, da hier meist keine explizite Angabe gemacht werden kann. Darum kann auch nicht immer entschieden werden, ob diese Anforderung funktional oder nicht funktional ist.

Software Requirements Document oder Pflichtenheft

Das „Software Requirements Document“ gibt an, was die Entwickler implementieren sollen. Das Dokument hat jedoch verschiedene Benutzer beziehungsweise Stakeholders. Zum einen die Systemadministratoren, die Manager, die System Ingenieure, System Test-Ingenieure und zu letzte auch die System Maintenance-Ingenieure.

Der Grad der detaillierten Beschreibung ist unterschiedlich und hängt oft davon ab um welche Art von System es sich handelt, des verwendeten Entwicklungsprozesses und für wen das System gebaut wird: externer Auftragnehmer oder Inhouse.

IEEE / ANSI 830/1998 Standard

Der IEEE / ANSI 830/1998 Standard zur Strukturierung eines Anforderungsdokuments oder auch Pflichtenheftes:

1. Einleitung
    1. Zweck des Anforderungsdokuments
    2. Geltungsbereich des Produkts
    3. Definitionen, Akronyme und Abkürzungen
    4. Verweise
    5. Überblick
2. Allgemeine Beschreibung
    1. Produktperspektive
    2. Produktfunktionen
    3. Benutzereigenschaften
    4. Allgemeine Einschränkungen
    5. Annahmen und Abhängigkeiten
3. Spezifische Anforderung
4. Anhänge
5. Index

Um wieder auf die Anfangsproblematik der Aussagen: „Mary had a little lamb.“ und deren Interpretationsfreiheit zu kommen, sollten Spezifikation an die Systemanforderungen ein Glossar haben. Da jede Person unter Angaben wie beispielsweise „nicht viel“ oder „nicht lange“ unterschiedliche Auffassungen haben.

Ganz klar, die Anforderungen für die bereitzustellenden Dienste durch das System und die

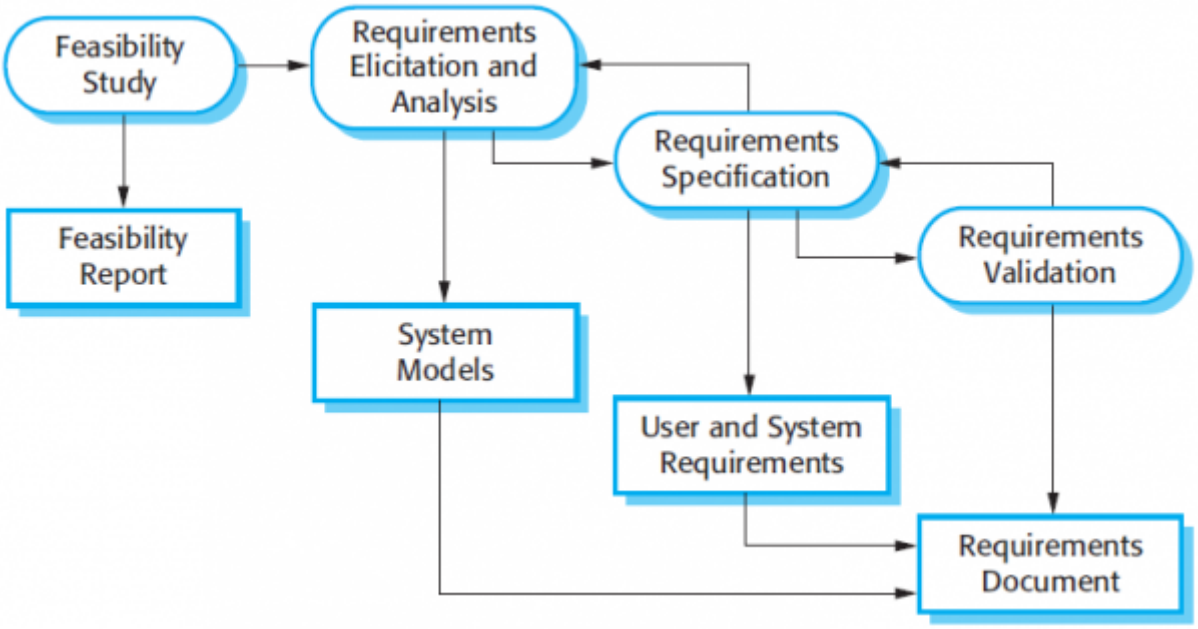
betrieblichen Einschränkungen so wie und Anforderungen müssen klar und deutlich in den Systemanforderungen beschrieben werden.

Dabei haben Sie als Anforderungsmanager folgende Aufgaben:

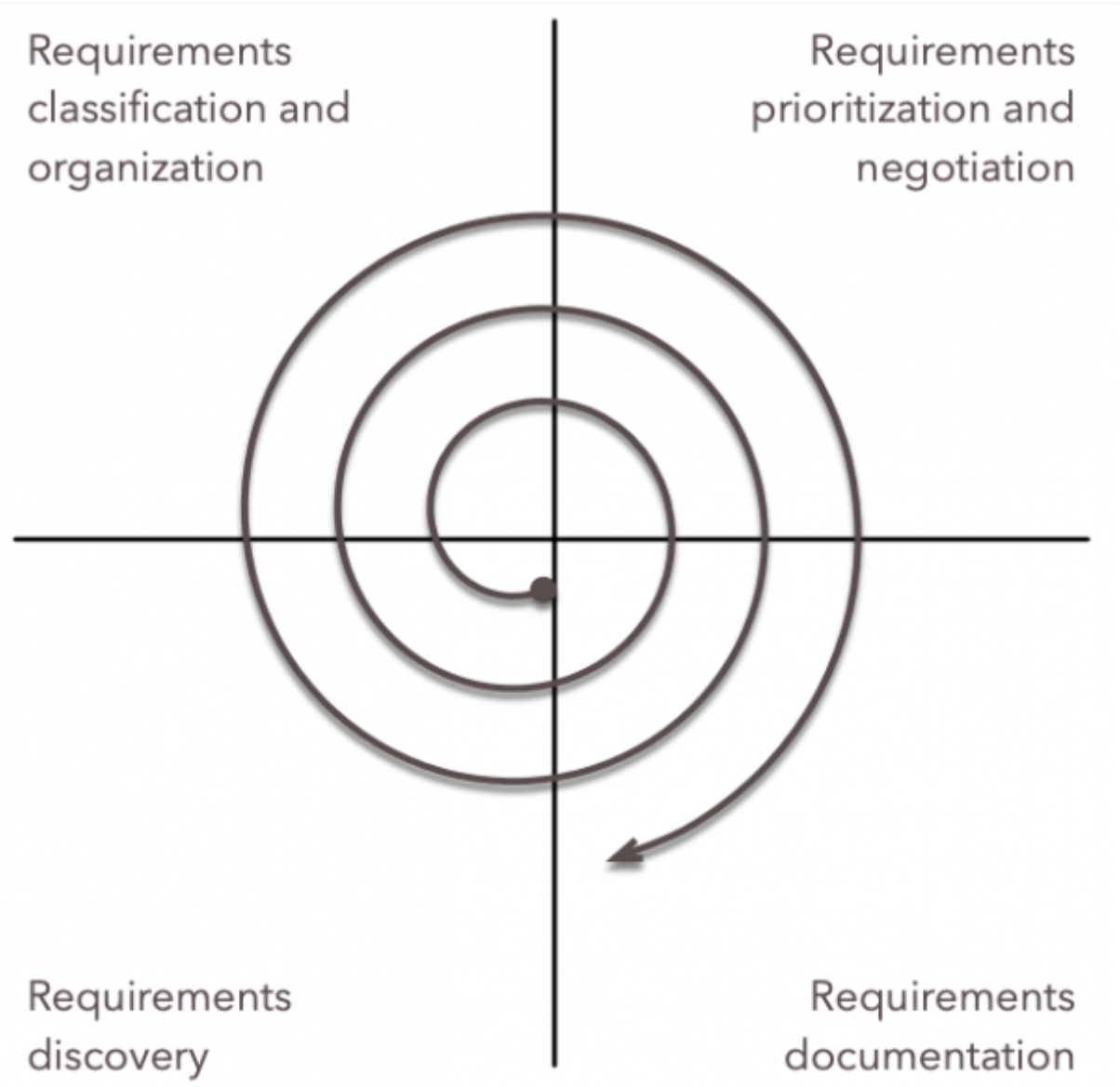
- Anforderungen herausfinden,
- Anforderungen Analysieren,
- Anforderungen dokumentieren und
- Überprüfung

Das Systemanforderungsdokument oder dt. Pflichtenheftes wird durch Sie erstellt und über den gesamten Entwicklungsprozess des Projektes auch von Ihnen regelmässig gepflegt.

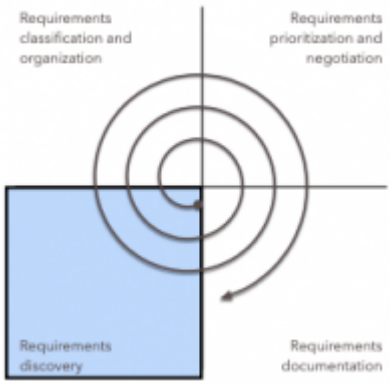
Der Entwicklungsprozess oder Requirements Engineering-Prozess



Quelle: medium.com



Der Anforderungserhebungs- und Analyseprozess



Anforderungsermittlung ist der Prozess der Interaktion mit den Stakeholdern Sie sammeln dabei alle Anforderungen an das von Ihnen zu entwickelnde Produkt ein. Dabei liegt das Hauptproblem bei der systematischen Erfassung und Erkennung der Anforderungen. Ein Ansatz ist, dass Sie sich in die Position des Anforderers begeben, und aus seiner Sichtweise Ansätze für die Anforderungen finden.

Blickpunktorientierte Ansätze

- Interaktionsanssicht: Personen, die mit dem System interagieren müssen
- Indirekter Gesichtspunkt: Stakeholder verwenden das System nur indirekt, jedoch beeinflussen

diese die Anforderungen an das System
- Domain Ansicht: Domainmerkmale und deren Einschränkungen an die System Anforderungen

Begeben Sie ich in das Umfeld der einzelnen Personen und versuchen Sie möglichst genauere Angaben aus derer Sicht zu machen. Schauen Sie sich den Arbeitsablauf der Zielgruppe an. Nachdem Sie die wichtigstenArbeitsschritte erkannt haben, beginnen Sie diese genau zu einer Anforderungen zu formulieren.

Interviews

Interviews sollten nur im Beisein von Entwicklern gehalten werden, denn nur diese können den Aufwand und vor allem über die Plausibilität gültige Aussagen treffen.

Da die Befragten meist nur über ihr Domänenwissen verfügen und von der dahinter befindlichen, tatsächlichen Struktur oft kaum oder keinerlei Kenntnisse haben.

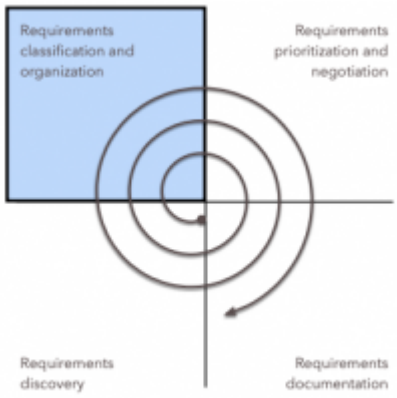
- Achtung: Arbeiter mit Domänenwissen arbeiten sogar sehr oft gegen das von Ihnen geplante Projekt, sobald diese der Auffassung sind, dass ihre Arbeit auf dem Spiel stehen könnte.

Arten von Interviews

- abgeschlossene Interviews: bei denen der Stakeholder vordefinierte Fragen gestellt bekommt.
- offene Interviews: ohne vordefinierte Agenda, darum oft Langwierig.
- Szenarien basierte Interviews: decken Sie sich mögliche Szenarien aus, bei denen Sie Interaktionen mit dem System abfragen. Dabei sind die Interaktionen anfangs grob umrissen und werden immer detaillierter.
- Die meisten Menschen können ein Szenario verstehen und kritisieren, diese mit dem System interagieren, dabei können Sie sehr nützlich Informationen für beispielsweise das Hinzufügen neuer Features herausfinden.
- Beschreiben Sie diese Anforderungen sehr detailliert, da Sie diese Informationen zu einem späteren Zeitpunkt benötigen werden.
- Use Cases: auf diese werde ich später noch genauer eingehen.

Eine Mischung aus fest vordefinierten Fragen, welche aber noch genügend Freiräume für individualisierte Fragen haben, aber dennoch zielführend sind, eigenen sich am besten. Zögern Sie nicht Zwischenfragen zu stellen. Und Abläufe anhand ausgedachter Szenarien Durchspielen. Achten Sie auf Handlungsabläufe, versuchen Sie Verbesserungsmöglichkeiten aus der Sicht des Benutzer zu erlangen, denn der Benutzer der täglich mit einem System arbeitet, kennt die „nervigsten Griffe“ und weiss wo das System seine „Tücken“ hat.

Anforderungsklassifizierung & Organisation oder Requirements classification and organization



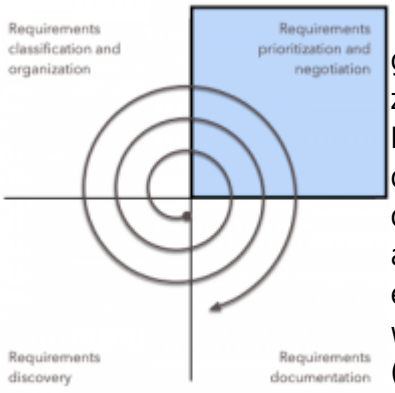
Angesichts der unstrukturierten Sammlung von Anforderungen werden die Anforderungen nun gruppiert und in kohärente (schlüssige, zusammenhängende) Cluster organisiert. Ein mögliches Modell zur Kategorisierung der Anforderungen ist das **FURPS + Model**. Hierbei stehen folgende Punkt im Mittelpunkt der Entwicklung.

- **F**unctional : Funktional
- **U**sability : Benutzerfreundlich
- **R**eliability : Zuverlässig
- **P**erformance: Leistungsstark
- **S**upportability: Wartbar / Erweiterbar
- **+** (PLUS)

Das PLUS ist dann wieder etwas Individueller zu betrachten, beispielsweise in Hinsicht auf die Implementierung, Art der Schnittstelle, durchzuführenden Operationen, Umfang des Produktes oder auch der Rechtliche Aspekt im Kontext, in dem der Kunde steht.

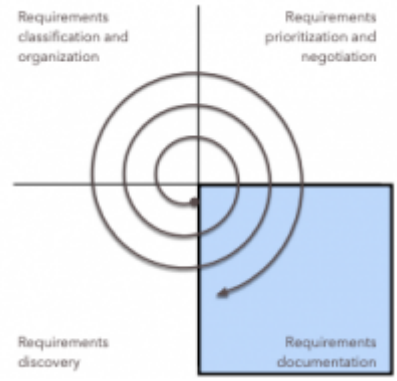
Kleines Beispiel für ein Plus wäre: Die Implementierung der Schnittstellenerweiterung für ein gegebenes Programm benötigt eine BSI zertifizierte Verschlüsselung zur Kommunikation zwischen verschiedener Behörden.

Anforderungspriorisierung und Verhandlung



Die Anforderungen welche Sie nun zusammen mit den Benutzern gefunden und dann ausarbeitet haben, werden dann mit dem Kunden zusammen priorisiert. Hierbei kommt es erfahrungsgemäss oft zu Konflikten, da der Kunde eine andere Auffassung von der Verwendung des Produktes hat, als die Benutzer. Diese Konflikte müssen Sie nun durch zielorientierte Verhandlungen lösen. Der Nutzer möchte meist ein ansprechendes, leicht verständliches Produkt, während dem Kunden eine gewinnbringende Lösung für möglichst kleines Geld haben möchte, wobei dieser keinerlei Interesse an der Benutzbarkeit oder dem UX (Bediendungsdesign) hat. Achten Sie darauf und ergreifen Sie zu keiner Zeit keine Position in dieser Diskussion.

Anforderungsdokumentation



Die Anforderungen werden dokumentiert und als Input für die nächste Abschnitt in der Spirale verwendet. Dabei können die vorgelegten Dokumente formell oder auch informell sein. Achten Sie aber darauf, dass alle relevanten Informationen für den Kunden wie auch die Entwickler dabei enthalten sind und keine Freiräume in der Interpretationen geben.

Die Anforderungsvalidierung befasst sich bei der Anforderungsdokumentation mit dem Nachweis, dass die definierete Anforderungen tatsächlich das vom Kunden gewünschte System / Produkt ist.

Anforderungsvalidierung versucht dagegen, die gefundenen Probleme in Hinsicht auf die Anforderungen in der Anforderungsdokumentation zu beschreiben.

Prüfungen, die während der Anforderungsvalidierung und vor der Dokumentation durchgeführt werden müssen:

- Gültigkeitsprüfung: Erfassen die Anforderungen die richtigen Funktionen; brauchen wir zusätzliche oder andere Funktionalitäten?
- Konsistenzprüfung: Stellen Sie sicher, dass die Anforderungen nicht miteinander in Konflikt stehen.
- Vollständigkeitsprüfung: Definieren Sie die Anforderungen, alle Funktionen und alle Einschränkungen. Sind diese
- Einschränkungen vom Systembenutzer beabsichtigt? (z.B. Lese-/Schreib-Rechte)
- Realismusprüfung: Können die Anforderungen sinnvoll umgesetzt werden?
- Überprüfbarkeit: Ist es möglich, einen Test zu entwickeln, der überprüft, ob eine Anforderung vorliegt, bzw. diese erfüllt wird?
- Rückverfolgbarkeit: Ist jede Anforderung auf ihre Quelle rückverfolgbar, also woher kommt diese Anforderung? Da sonst zum Schluss niemand für diese Anforderung zahlen möchte.

Zusammenfassung

Die verschiedene Arten der Anforderungen:

- funktionale und nicht funktionale Anforderungen
- Benutzer- und Systemanforderungen
- Requirements Engineering-Prozess

Setzen Sie (große) kommerzielle Projekte oder Open-Source-Projekte durch systematisches Aufteilen in kleine Pakete leichter um.

Use Cases oder Anwendungsfälle

Ein Anwendungsfall oder Use Case besteht aus einer Reihe von Szenarien, die mit einander

verbunden sind und meist ein gemeinsames Benutzerziel haben. Ein Beispiel dafür ist z.B. „Banküberweisung durchführen“.

Anwendungsfälle / Use Cases sind kleine Textgeschichten, die zum Erkennen und Erfassen von speziellen Anforderungen verwendet werden. Dabei kapselt ein Anwendungsfall / Use Case oft eine Reihe von Aktionen, die in einer klar definierten Reihenfolge ausgeführt werden müssen. Dabei können Anwendungsfälle / Use Cases viele andere Artefakte, z. Analyse, Design, Implementierung und Testartefakte beeinflussen.

Aber Achtung: Anwendungsfälle / Use Cases können nicht für alle Arten von Anforderungen verwendet werden!

Beispiel

Ein POS-System (Point of Sale) oder auf deutsch Kassensystem:

- POS-Systeme werden normalerweise in Einzelhandelsgeschäften verwendet, um den Umsatz zu erfassen Zahlungen abzuwickeln.
- POS-Systeme interagieren mit verschiedenen Dienstanwendungen um beispielsweise Steuern oder Bestandsdaten zu berechnen.
- POS-Systeme umfassen Computer, Barcode-Scanner und Software. Die Anzahl der Clients kann variieren. Umsetzung durch Thin-Client-Webbrowser Terminals oder Rich-Client-Anwendungen.

Point of Sale Systems Domain Terminology

- Verkauf: Tausch Ware für Geld oder Ware für Gutschrift.
- Beleg als Quittungsnummer / Belegnummer (fortlaufend)
- Einzelposten mit Anzahl oder als Belegposition
- Zahlung oder Vergütung, ob in bar oder anderer Zahlungsart
- Kunde oft Kundennummer
- KassiererIn als belegende Person

Anwendungsfälle sind Kurztexte, die häufig zum Entdecken von Anwendungen und dann zum festhalten der Anforderung verwendet werden.

Anwendungsbeispiel Verkaufsprozess:

> Ein Kunde kommt an die Kasse mit einer Anzahl von Artikeln.
> Der Kassierer verwendet das POS-System zur Erfassung eines jeden Artikels.
> Das POS-System gibt Details zur laufenden Summe und zu jedem Einzelposten.
>
> Danach gibt der Kunde seine gewünschte Bezahlungsweise an.
> Dabei muss das POS-System dieses validieren und aufzeichnen.
> nach der Bezahlung muss dann das System das Inventar es speziellen Ladens aktualisieren.
> Der Kunde erhält eine Quittung über seinen Kauf.

Use Cases Definitionen

- Actor: Eine Person oder ein System mit einem gewissen Verhalten. Meistens eine Person, kann aber auch eine automatisierte Handlung durch ein Computersystem sein.
- Use case oder Anwendungsfall: Ist eine Abhandlung von Erfolgs- und Misserfolgsszenarien, die den Actor dabei beschreibt, wie dieser eine systemunterstützte Handlung durchführt, eine Ziel zu erreichen. Der Kassierer berechnet den Gesamtbetrag.
- Szenario oder Anwendungsfallinstanz: Ist nur eine bestimmte Abfolge von Aktionen und Interaktionen zwischen Akteuren und dem System. Gibt nur eine Möglichkeit an, wie ein System verwendet, oder eine Abhandlung eines Anwendungsfall angewendet werden kann. Der Kassierer Scannt jeden Artikel mit dem Barcode-Scanner um den Gesamtbetrag zu erfassen. ODER Der Kassierer gibt den Bacode per Hand in das System ein, wenn der Barcode für den Scanner nicht lesbar ist. ODER der Kassierer gibt die ID des Produktes und einen Betrag per Hand in das System ein.

Use Case-Formate

- Brief oder kurz: Umfassende Zusammenfassung, in der Regel aus einem Absatz, der Hauptproblem oder Haupt-Szenario beschreibt. (Wie beispielsweise oben)
- Casual oder ungezwungen, zwanglos: Informelles Erfassen, ein paar Absätze, die verschiedene Szenarien abdecken.
- Fully dressed oder vollständig bearbeitet: Alle Schritte und Variationen werden detailliert beschrieben. Dabei gibt es auch erklärende Abschnitte in denen Voraussetzungen und oder auch wie der genaue Erfolg garantiert werden kann steht.

Konzentrieren Sie sich auf die Korrektheit der Anwendungsfälle, bevor Sie sich auf die (Detail-) Genauigkeit konzentrieren!

identifizieren Sie alle aktuell relevanten Anwendungsfälle auf sehr hohem Niveau (geringe Detailgenauigkeit, hohe Korrektheit). Danach arbeiten Sie die Details aus und erhöhen damit die Detailgenauigkeit.

Fully Dressed Use Case Format

Use Case Name	WikiName, start with verb
Scope	System boundaries (corp, prog)
Level	Summary, subfunction, etc
Primary Actor	Primary system user
Stakeholders	Who cares and what they want
Preconditions	Must be true to start
Postconditions	What is guaranteed by success
Main Success Scenario	Typical, unconditional path scenario
Extensions	Alternative success or failure scenarios
Special Requirements	Related non-functional requirements (RAM)
Technology & Data Variations List	Varying IO methods and data formats
Frequency of Occurrence	Is this system used often
Miscellaneous	Open issues; eg unmanageable failure scenarios

Quelle: royaleducation.info

Beispiel eines fully dressed Use Case:

Name:	**Kryptowährungen online (über das Web) kaufen**
Haupt-Aktor:	Käufer / Kunde
Scope:	die spezielle Kryptowährung
Level:	Benutzerziel
Stakeholders and Interests:	

Kryptobörsen Betreiber – möchte vollständige Kaufinformationen. |

Precondition:	Kunde ist im System authentifiziert / angemeldet und ausreichend finanzielle Mittel
Minimal guarantee:	Es sind auf dem System ausreichend Protokollinformationen vorhanden, damit rekonstruiert werden kann, was beim Kaufablauf schief gelaufen ist.
Success guarantee:	Die Website hat den Kauf bestätigt, die Protokolle und das Portfolio des Benutzers werden aktualisiert.

Main Success Scenario:	1. Der Käufer entscheidet sich für eine Kryptowährung, die er kaufen möchte. 2. Plattform prüft den zum verkaufstehenden Bestand dieser Kryptowährung 3. Käufer kauft nun die gewünschte Menge 4. Plattform generiert Protokolle, passt Bestände an und überträgt die Anzahl der gekauften Coins in dass Portfolio des Kunden 5. Plattform zeigt dem Benutzer sein neues Portfolio an
Extensions:	2) 2a) Der Käufer möchte eine Währung die von der Plattform nicht unterstützt wird. 2a1) Das System erhält vom Käufer einen neuen Vorschlag mit Option zum Abbrechen 4) 4a) Die Plattform kann den Kauf nicht durchführen und legtKauf mit den Status „zur Klärung ausstehend“ an. 4a1) Plattform protokolliert die Verzögerung und stellt einen Timer, um eine Antwort vom Käufer über weitere Massnahmen bei der Durchführung zu erfragen
Special Requirements	…
Technology and Data Variation List	…
Frequency of Occurrence	…
Miscellaneous	…
…	…

Richtlinien für Anwendungsfälle Use Cases

- Halten Sie bei Anforderungen die Benutzeroberfläche ausser Acht. Konzentrieren Sie sich nur auf auf die Absicht der Anforderungen.
- schreiben Sie knappe Anwendungsfälle
- schreiben Sie Black-Box-Anwendungsfälle, also nur was die Anforderungen machen muss und nicht wie die Anforderungen es tun soll. Zum Beispiel: „Das System zeichnet den Verkauf auf aber nicht: „Das System schreibt den Verkauf in eine SQL-Datenbank.“
- Versetzen Sie sich die Perspektive eines Anwenders und Konzentrieren Sie sich nur auf die Benutzung des Systems und fragen Sie nach deren Zielen und typischen Situationen. Versuchen Sie die Anwender genau zu verstehen was diese durch die Verwendung erreichen wollen.

Suchen Sie schon während der anfänglichen Analysephase nach Anwendungsfällen.

Anwendungsfälle können verschiedenen Perspektiven unterschiedlich sein, z.B. sind folgende Anwendungsfälle nicht in Software umsetzbar: Aushandeln eines Lieferantenvertrag oder Retourenbehandlung, gehören aber mit in das gesamte System. Oder das Bewegen einer Spielfigur auf dem Spielfeld?

Überlegen Sie darum genau, ob das Ergebnis einen messbaren Wert (VAL) für das Geschäft erzielt?

Fügt eine Aktion, die von einer Person, an einem Ort, an einer Stelle, zu einer gewissen Zeit, als Reaktion auf ein Ereignis ausführt wird, eine messbaren Geschäftswert hinzu und belässt dabei die Daten in einem konsistenter Zustand?

Also das Aushandeln eines Lieferantenvertrag ist aus meiner Sicht viel Weitläufiger und unüberschaubar.

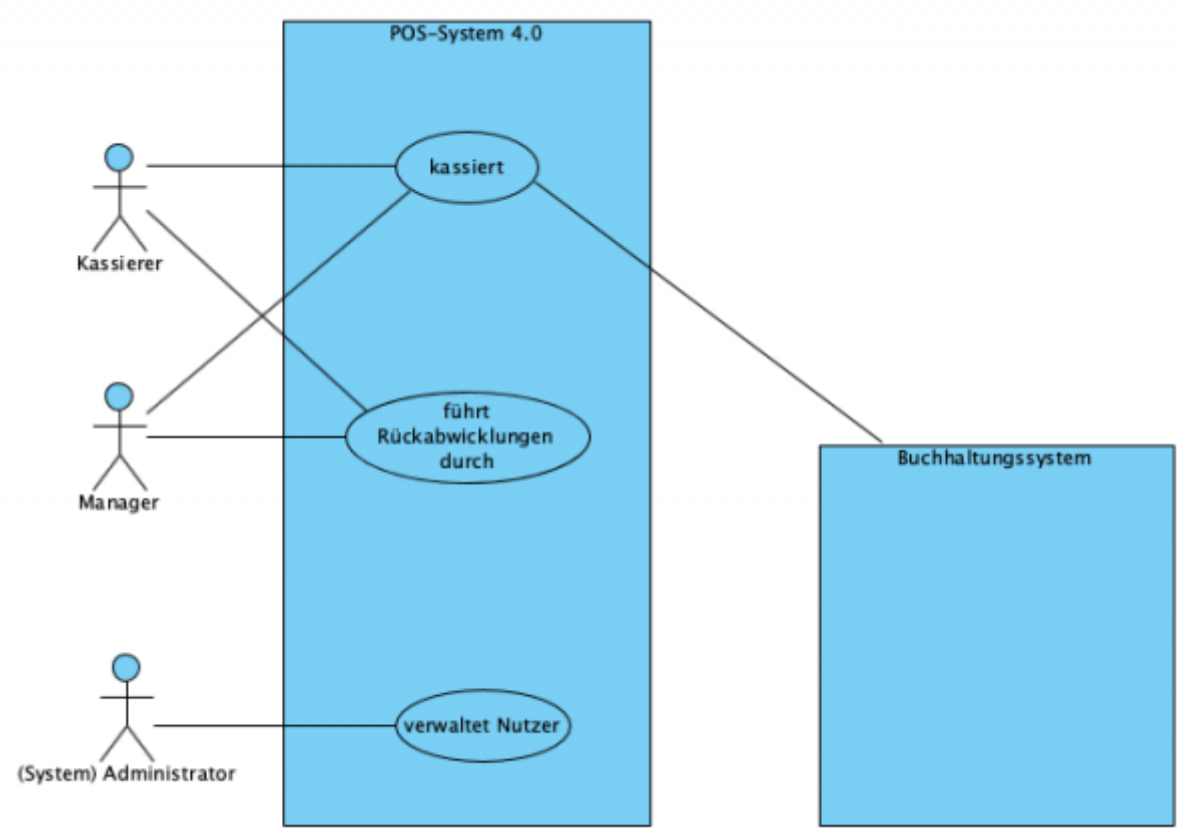
Die Behandlung von Retouren sind dabei messbarer im Geschäftswert und von der Größenordnung viel überschaubarer.

Beim Bewegen einer Spielfigur auf dem Spielfeld ist dies nur eine einziger Schritt und somit „kein Teil vom großen und Ganzen“.

Use Cases-Diagramm

UML-Anwendungsfalldiagramme bieten eine Notation zur Veranschaulichung von Namen der Anwendungsfälle, der Akteure (Rollen) und deren zugehörige Beziehung der Use Cases. Zur Erstellung dieser Diagramme verwende ich das Programm Visual Paradigm.

Als Beispiel dient hier wieder das obig genannte POS-System:



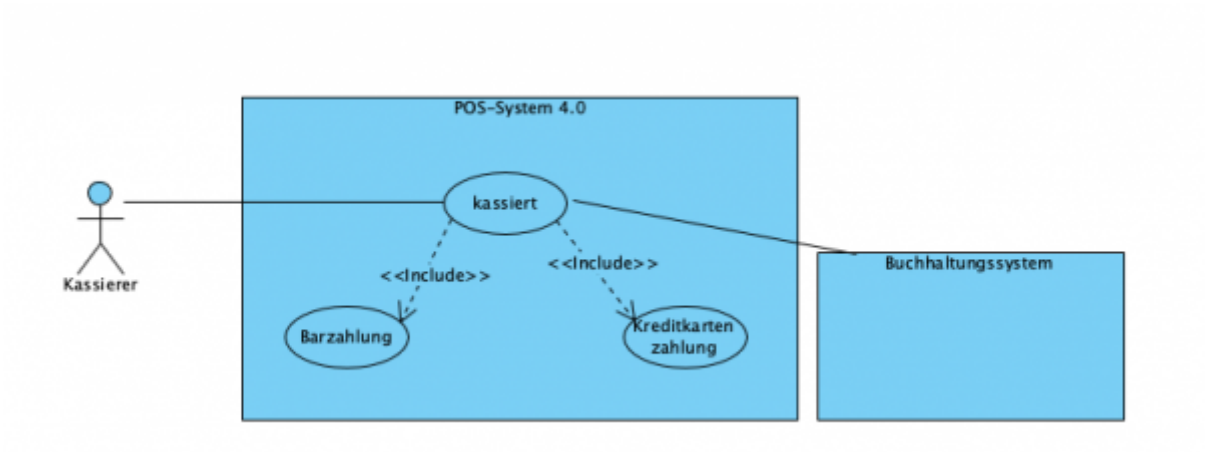
Ergänzung des Diagramms: Die „Männchen“ stellen hier die Rollen des Kassierers, des Manager und des System-Administrators da zusammen mit dem Buchhaltungs-System sind sie die Akteure in diesem Use Cases-Diagramm. Das Buchhaltungssystem als weiteres System kann regelmässig Abfragen stellen, um zum Beispiel Verkaufszahlen eines bestimmten Artikels, Tages, Woche oder

Monat auszugeben. Beide Systeme, also POS-System und das Buchhaltungssystem sind dabei eigenständig anzusehen und haben ihren eigens abgegrenzten Raum (System-Boundaries).

Die UML-Anwendungsfalldiagramme sind einfach darzustellen und eben so einfach zu verstehen. Damit sollen UML-Anwendungsfalldiagramme zur verbessernden Organisation von Kommunikation und Verständnis der Anwendungsfälle beitragen. Auch reduziert diese Darstellung doppelte Textbeschreibung. Gut ausgearbeitete UML-Diagramme ersparen Ihnen seitenweise Text. Dennoch bieten die UML-Anwendungsfalldiagramm nur eine Black-Box-Ansicht auf einem System, weshalb sie nur in der frühen Projektphasen nützlich sind.

Die Beziehung in UML-Anwendungsfalldiagramme

Für partielles Verhalten, das in mehreren Anwendungsfällen üblich ist, wie zum Beispiel bei der speziellen Form des Zahlungsprozesses durch „Bezahlung mit Kreditkarte“, ist es wünschenswert, einen eigenen Anwendungsfall als Unterfunktion darzustellen und diesen dann mit einzubeziehen.



Der Hauptzweck besteht darin, Wiederholungen zu vermeiden oder einen extrem langen Verwendungtext damit zu zersetzen um dadurch das Diagramm verständlicher zu machen.

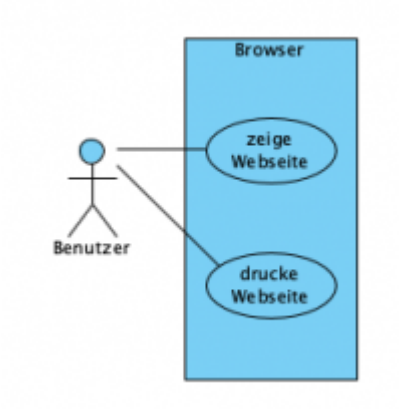
Die Erweiterungsbeziehung kann immer dann verwendet werden, wenn man genau zeigen möchte, wo und unter welchen Bedingungen eine Erweiterterung oder ein zusätzlicher Anwendungsfall das Verhalten einiger Basis-Anwendungsfälle verändert. Konzentrieren Sie sich dabei mehr auf die Textbeschreibung, da sich Beziehungen selten in der Praxis erweitern.

Eine Erweiterungen in unserem POS-System wäre dann unter anderem beim Zahlungsvorgang, wenn der Kunde eine Kundenkarte besitzt über diese er Rabatte bekommt oder auch die Verwendung einer Guthabenkarte in Form von einer Geschenkkarte oder Ähnlichem.

Vererben von Anwendungsfällen

Auch wenn es nicht sehr häufig auftritt, gibt es auch Vererbungen in Anwendungsfällen. Der Vererben von Anwendungsfällen ersetzt eine oder mehrere Vorgehensweise. Der erbende Anwendungsfall überschreibt das Verhalten des vererbten Anwendungsfalls.

Agilität



Denken Sie immer agil! Bleiben Sie nicht starr auf einer Ansicht sitzen. Zum Beispiel wenn Ihr Kunde einen Web-Shop möchte, reicht es nicht, dass dieser über den Browser der verschiedenen Endgeräte aufrufbar und darstellbar ist. Wie sieht es aus, wenn ein Kunde eine Seite ausdrucken möchte? Die Ansicht im Browser ist keinesfalls gleich der ausgedruckten Ansicht.

Craig Larman; 2005:

> UML ist nur eine Diagrammnotation und es ist nutzlos, UML und Use Case-Tool zu verwenden, wenn Sie nicht wissen, wie >man Objektorientiert arbeitet.

Domain Modeling

Domain Model:

- Warum: Die Domänenmodellierung hilft uns, Relevantes wie Konzepte und Ideen einer Domäne zu identifizieren.
- Wann: Immer wenn wir weiter Konzepte in einer Domäne verstehen müssen.
- Leitlinie: Erstellen Sie nur ein Domänenmodell für anstehende Aufgaben.

Curtis'law (sinngemäß) aus Albert Endres and Dieter Rombach: A Handbook of Software and Systems Engineering; Addison Wesley 2003:

> "Ein guter (Software-)Entwurf benötigt ein tiefgreifendes Verständnis des Einsatzgebiets der zu erstellenden Software."

Das Domain Model oder Analysemodell (Konzeptmodell) veranschaulicht bemerkenswerte Konzepte in einer Domain oder Domäne.

Das Domänenmodell wird erstellt, um die realen Welt in einzelne Domänen, also Konzepte oder Objekte zu zerlegen. Das Domänenmodell wird iterativ abgeschlossen und ist die Basis für das Design der Software. Das Modell sollte die Menge der konzeptuellen Klassen identifizieren. Das Domänenmodell wird auch als konzeptionelles Modell, Objektmodell oder Analysemodell bezeichnet.

Begriffsklassen sind Ideen, Dinge oder Objekte in der Domäne. Eine konzeptuelle Klasse hat ein Symbol, das die Klasse darstellt, eine Bedeutung und eine Ausprägung hat.

Domänenkonzepte oder konzeptuelle Klassen sind beispielsweise in Java, C#, Ruby, nicht unbedingt erforderlich, da diese die Eigenschaft schon von sich auch mitbringen.

Visualisierung von Domänenmodelle

In Domänenmodellen sind keine Vorgänge definiert sondern nur:

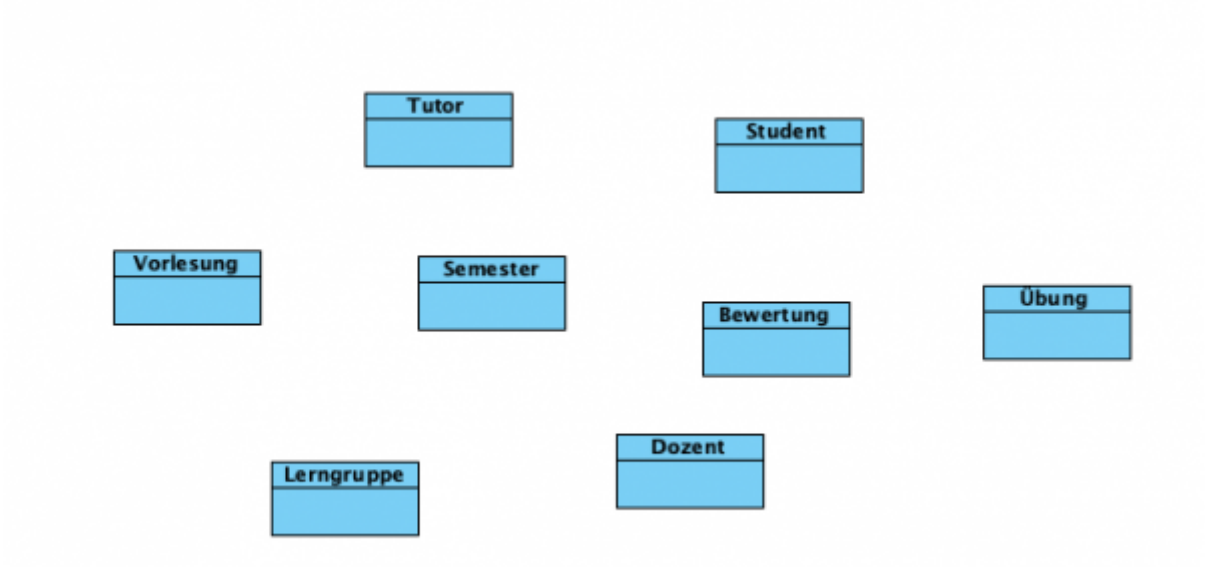
- Domänenobjekte und konzeptuelle Klassen
- Assoziationen zwischen ihnen
- Attribute konzeptueller Klassen

Daraus ergibt sich ein konzeptionelles Perspektivenmodell.

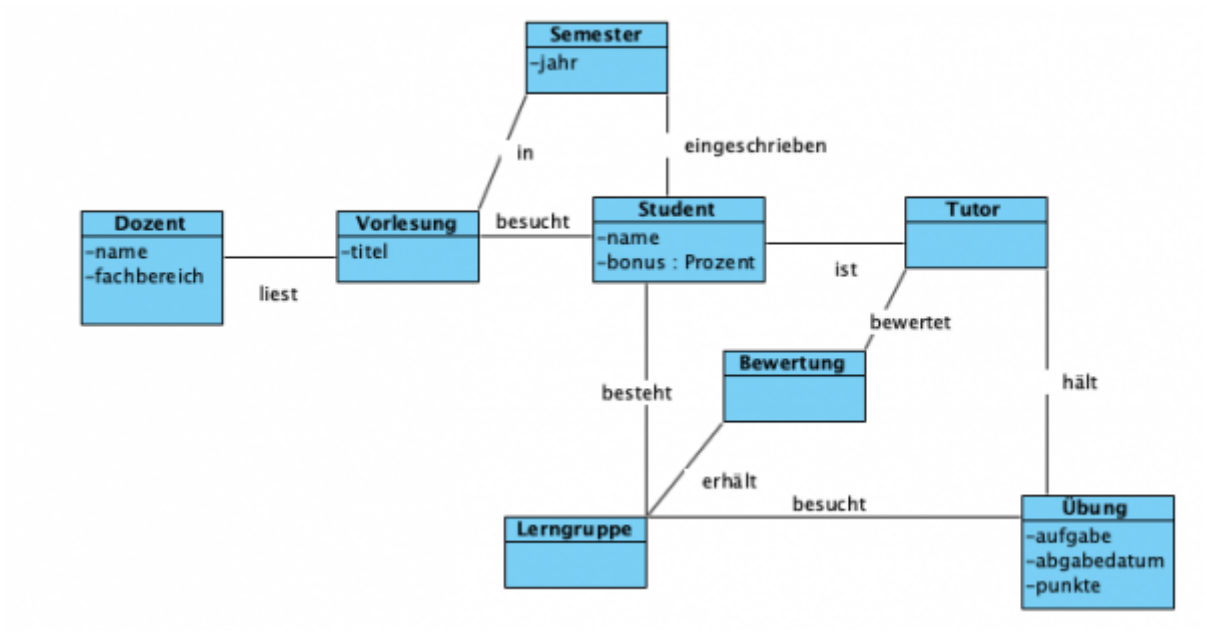
Exemplarisch dargestellt an einem grundlegenden Kursverwaltungssystems an einer Universität. Folgende Aussagen haben wir für ein Kursverwaltungssystems:

- Während eines Semesters liest ein Dozent eine oder mehrere Vorlesungen.
- Manchmal muss ein Dozent vertreten werden, dann hält dieser keine Vorlesung.
- Ein Student besucht normalerweise eine oder mehrere Vorlesungen.
- Während des Semesters werden verschiedene Übungen für die Student angeboten.
- Jeder Student ist einer bestimmten Lerngruppe über das ganze Semester zugeordnet.
- Eine Lerngruppe besteht aus zwei bis drei Studenten
- Nach Einreichung einer Lösung einer Lerngruppe wird diese durch einen Tutor bewertet.

Eine Klasse beschreibt eine Menge von Objekten mit derselben Semantik, Eigenschaften und Verhalten. Bei der Domänenmodellierung wird eine realistische Welt als visualisiertes Konzept dargestellt.

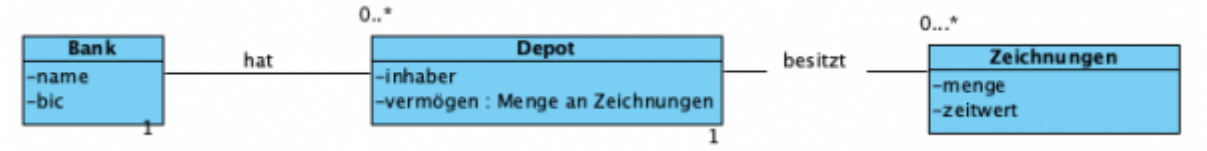


Attribute sind logische Datenwerte eines Objekts. Es ist darum nützlich, die Attribute der konzeptionellen Klassen zu identifizieren, um die Informationsanforderungen Szenarien in der Entwicklung zu erfüllen.



Erstellung eines Domain-Modell

Beispiel: Bankkonten und zugehörige Buchungen. {Vermögen / Wert = Sum(Zeichnungen.Zeitwert)}



Verwenden Sie eine Kategorieliste, um konzeptuellen Klassen zu finden.

konzeptionelle Klassenkategorie	**Klassen (Hinblick auf das POS)**
geschäftliche Transaktionen	Verkauf und Bezahlung
Einzeltransaktionen	Einzeltranaktion
Produkt oder Dienstleistung im Zusammenhang mit einer Einzeltransaktionen Transaktion	Artikel
Wo wird die Transaktion erfasst?	Kasse
Rollen von Personen oder Organisationen im Zusammenhang mit der Transaktion oder Akteure im Anwendungsfall.	Kassierer, Kunde, Geschäft
Ort der Transaktionen.	Im Geschäft
Bemerkenswerte Ereignisse, zum Beispiel Zusammenhang mit Zeit oder einem Ort	Verkauf, Bezahlung
…	…

Identifizieren Sie dabei die Nominalphrasen, um die konzeptuellen Klassen durch Sprachanalyse:

Identifizieren Sie die Substantive und Nominalphrasen in der Textbeschreibung der Domäne und betrachten Sie diese als mögliche konzeptionelle Klassen oder Attribute.

Eine mechanische Zuordnung von Nomen zu Klassen ist nicht möglich. Wörter der natürlichen

Sprachen sind mehrdeutig, das heisst, dass das gleiche Nomen mehrere Dinge bedeuten kann und multiple Substantive können gelegentlich auch dasselbe bedeuten.

„Use Cases“ sind auch eine ausgezeichnete Quelle zur Identifizierung von konzeptionellen Klassen.

Beispiel: Verkaufsprozess: Ein Kunde kommt mit Artikel an die Kasse. Der Kassierer benutz den Barcode-Scanner des POS-System, um jeden Artikel zu erfassen. Das System zeigt dabei eine Gesamtsumme und den Einzelposten an. Der Kunde gibt seine bevorzugte Zahlungsart ein, welche durch das System validiert und aufgezeichnet wird. Das System aktualisiert das Inventar. Der Kunde erhält eine Quittung über seinen Einkauf vom System und verlässt dann den Laden mit den Artikeln.

Kasse = Kassierer, da dies zusammen mit dem Kassensystem und dem Barcode-Scanner eine Einheit bilden. Eine Kasse ohne Kassierer funktioniert genauso wenig wie ein Kassierer ohne Kassensystem.

Gesamtsumme besteht aus der Artikelanzahl der einzelnen Artikel = Einzelposten.

Quittung ist die Auflistung der Gesamtsumme mit zusätzlichen Angeben, wie Kundennummer, Datum, usw. Somit ist eine Quittung ist nur ein Bericht über einen Verkauf und eine Zahlungsart.

Im Allgemeinen ist es nicht nützlich, alle Informationen oder abgeleitete Werte aus der Quellen zu kopieren. Wenn eine bestimmtes Objekt geschäftliche Relevanz hat muss es aufgenommen werden.

Identifizierte Kandidaten für konzeptionelle Klassen am Beispiel der Quittung. Eine Quittung ist nur ein Bericht über einen Verkauf und eine Zahlung. Sollte sich die Quittung also im Domänenmodell befinden? Hängt davon ab, ob die Quittung im aktuellen Szenario berücksichtigt werden sollte. Eine Quittung ist nur ein Bericht aber wenn wir auch Retouren in Betracht ziehen, dann benötigen wir eine Quittung. Oder aber wie sind die gesetzlichen Einschränkungen bezüglich einer Quittung?

Sie sehen, es kommt auf das gesamt System an. Während wir eine Quittung als reiner Beleg firmenintern nicht benötigt wird, so ist die Quittung bei der Behandlung von Retouren essentiell. Oder eine gesetzliche Einschränkung verlangt einen Beleg über einen Kauf.

Wann sollte ich etwas als Attribut oder als Klasse modellieren? Faustregel: Wenn wir uns eine konzeptuelle Klasse X nicht als Zahl, Datum oder Text in der realen Welt vorstellen können, ist X wahrscheinlich eine konzeptuelle Klasse und kein Attribut.

Wir arbeiten an dem konzeptionellen Modell. Wir modellieren keine Assoziationen auf Softwareebene.

Wann sollte ich eine Assoziationen zum Domänenmodell hinzufügen?

- Wenn eine Assoziation zur Liste der allgemeinen Assoziationen gehört:
- A ist eine Transaktion, die sich auf eine andere Transaktion B bezieht
- A ist eine Position einer Transaktion B
- A ist ein Produkt oder eine Dienstleistung für eine Transaktion B
- A ist eine mit einer Transaktion B verbundene Rolle
- A ist ein physischer oder logischer Teil von B
- z.B. Die Beziehung zwischen einem Verkauf und einem Einzelposten
- Es ist jedoch nicht erforderlich, die Beziehung zwischen einem Kassierer und einer Produktbeschreibung zu speichern.

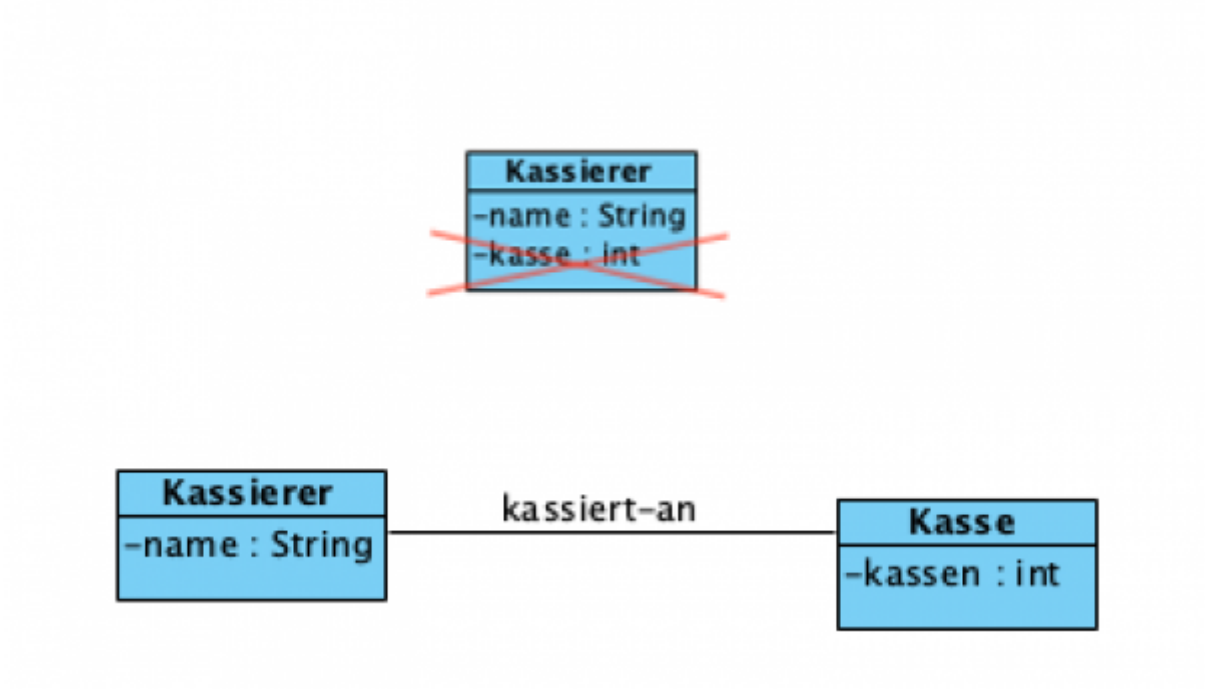
Benennen Sie eine Zuordnung, nach „ClassNameVerbPhrase“-Format. Die Verbphrase erzeugt eine Sequenz, die lesbar und sinnvoll ist. Gute Beispiele: Spieler „befindet-sich-auf“ Platz, Kauf „bezahlt-durch“ Barzahlung. Schlechte Beispiele sind dagegen: Kauf „mit“ Barzahlung „mit“ ist in der Regel

generisch und sagt nichts aus, ebenso Spieler „hat“ Platz, auch hier ist „hat“ generisch und sagt nichts aus.

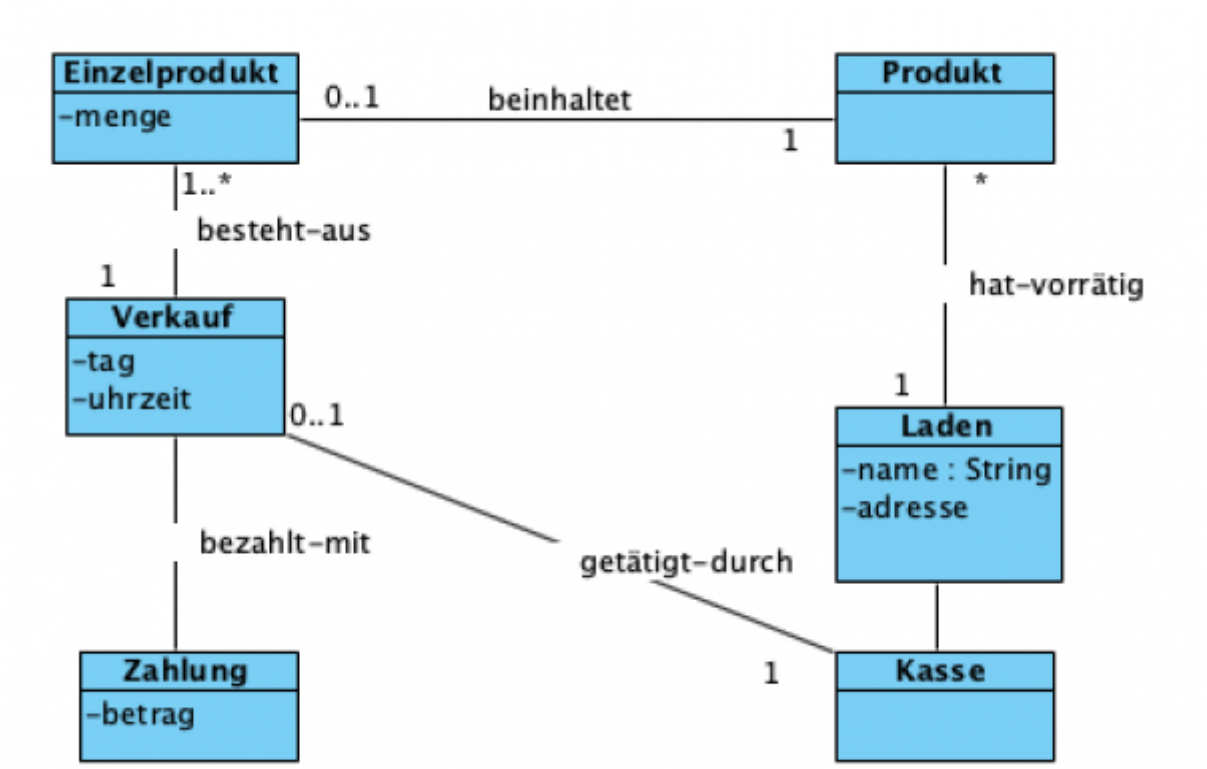
Die Attribute in einem Domänenmodell sollten vorzugsweise „primitive“ Datentypen sein. Sehr häufige Datentypen für Attribute sind: bool’sche Werte, Datum, Zahl, Buchstaben, Zeichenfolge, Adressen, Farben, Telefonnummern.

Betrachten Sie Modellierungsgrößen als Klassen, um die assoziierte Einheiten. Zum Beispiel Datentyp eines Betragsattributs einer Zahlung sollte einen numerischen Wert haben.

Verwenden Sie Assoziationen, um Abhängigkeiten zwischen konzeptuellen Klassen zu modellieren. Verwenden Sie keine Attribute.

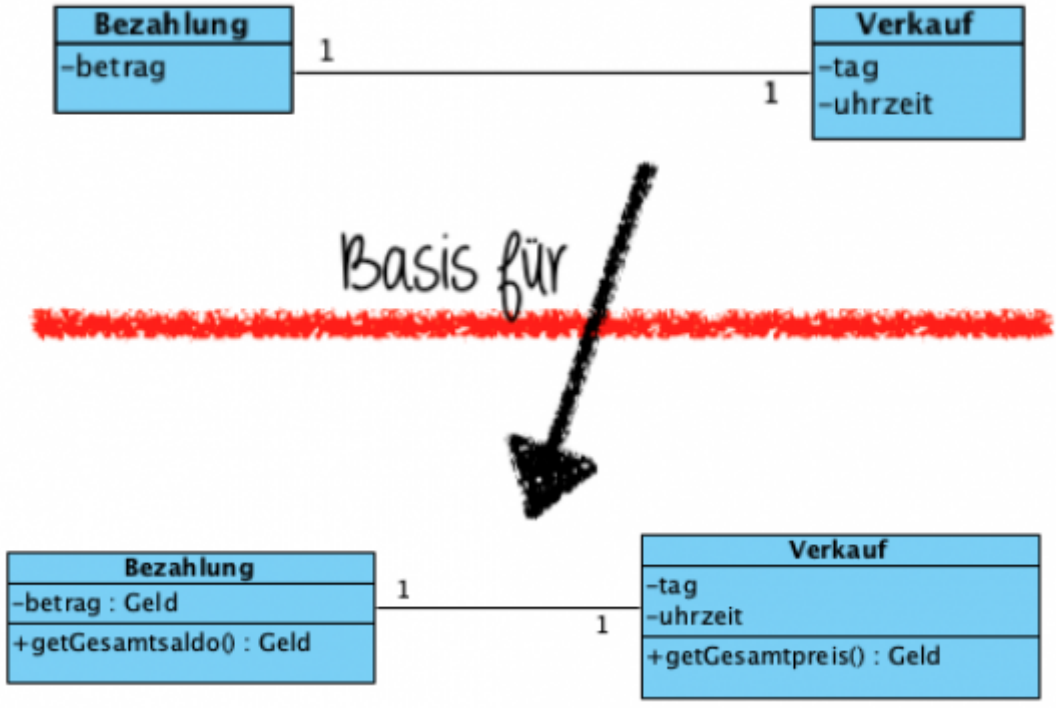


Definieren Sie eine neue Datentypklasse für etwas, das anfänglich als Zeichenfolge betrachtet wird. Zum Beispiel wenn die Zeichenfolge aus separaten Abschnitten besteht, wie Telefonnummer, Name der Person, usw. Oder wenn der Zeichenfolge unterschiedliche Operationen zugeordnet sind, wie bei der Sozialversicherungsnummer. Auch wenn der String andere Attribute hat, also die Zeichenfolge eine Menge mit einer Einheit besitzt, wie Beispielsweise Geld. Geld hat eine Währungseinheit welche unterschiedlich sein kann.



Domainkonzepte / Domainobjekte

Durch die Verwendung eines Domänenmodells als Basis für Software-Klassen ist die Repräsentationslücke zwischen den Domänenkonzepten und dem Programm um einiges geringer.



Das Domain-Modell dient als Inspirationsquelle für das Designmodell, welches ich später noch

besprechen werde. Das Ziel ist es, durch die systematische Bearbeitung kleinerer Software-Pakete gut konzipierte Software produzieren zu können.

- Domänenmodellierung ist nützlich, um die Ideen und Konzepte der Domäne und ihre Zusammenhänge besser verstehen zu können
- Ein Domänenmodell wird normalerweise zu Beginn eines Projekts erstellt und dient dann als Grundlage für das Entwurfsmodell
- Ein Domänenmodell wird durch kleine Teilmengen der UML-Klassendiagrammnotation erstellt