KIT

Karlsruhe Institute of Technology

Softwaretechnik II

Oliver Hummel, IPD

Topic 5

Use Cases

SOFTWARE DESIGN AND QUALITY GROUP
INSTITUTE FOR PROGRAM STRUCTURES AND DATA ORGANIZATION, FACULTY OF INFORMATICS

sdq.ipd.kit.edu

KIT – University of the State of Baden-Wuerttemberg and
National Research Center of the Helmholtz Association

www.kit.edu

Course Schedule

Date	Tentative Content
Mo. 21.10.	Warm-Up
Di. 22.10.	Software Processes
Mo. 28.10.	cont.
Di. 29.10.	Agile Development
Mo. 04.11.	*Guest Lecture by Andrena Objects*
Di. 05.11.	Agile Development
Mo. 11.11.	Requirements Elicitation
Di. 12.11.	Use Cases
Mo. 18.11.	Requirements Analysis
Di. 19.11.	cont.
Mo. 25.11.	Software Architecture
Di. 26.11.	cont. + Component-Based Architectures
Mo. 02.12.	cont.
Di. 03.12.	Persistence Patterns

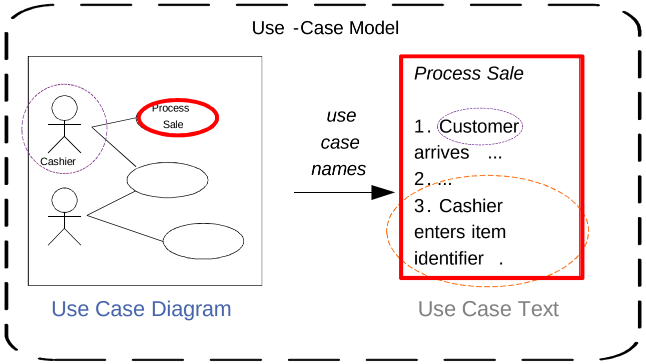
Overview on Today's Lecture

- Content
  - Model-Based Requirements Elicitation
    - Use Cases and Goal Levels
  - Requirements Specification and Management

- Learning Goals
  - Understand the principles of model-based requirements capturing approaches
  - Be able to recognize different goal levels of requirements and to arrange requirements together on the way towards a software specification

Model-based Requirements Elicitation

- … describes the requirements for a software system with various models
- … may contain various views
  1. Use Case Texts
  2. Use Case Diagram
  3. *(Activity Diagrams/Petri Nets)*



[Larman]

➔ requirements are usually captured as text first

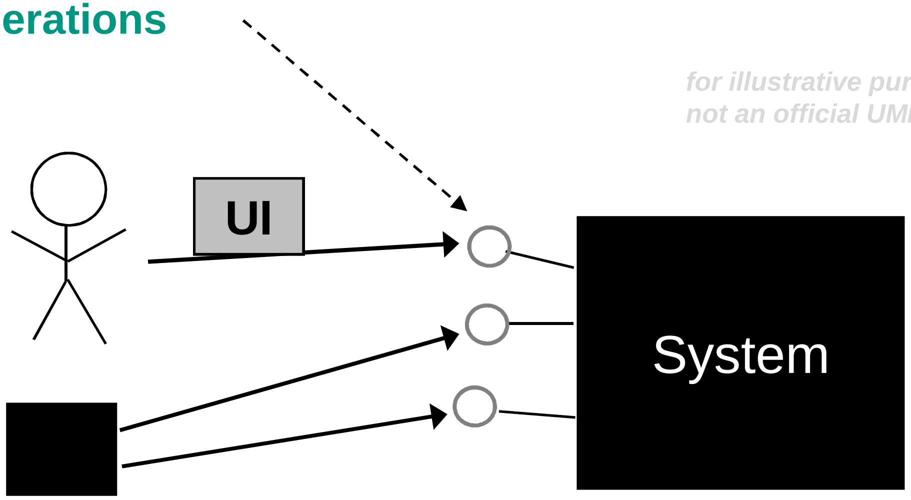
➔ *diagrams are only used for illustration*

➔ Use Case Texts are just a notation

- can be used for arbitrary other flows
  - such as cake recipes ;-)
- hence it is important to clarify their goal levels and their scope

Basic Use Case Modeling Idea

- The System is perceived as a **black box**
  - with which the users (humans or other systems) interact
    - via well-defined interfaces
      - called **system operations**

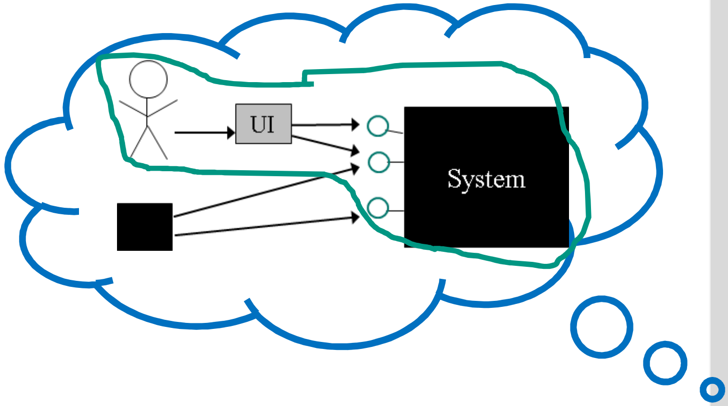


for illustrative purposes only, not an official UML diagram

  - a small system operation might have a huge impact in the backend of the system
    - the goal is to derive and model that systematically
      - by assigning **responsibilities** to software objects

Use Cases

- In general, **use cases are a notation** to textually capture an interaction of (typically) two parties
  - in our context a use case typically captures a story of how a system is used by a user
    - ➔ i.e. use cases therefore focus on functional requirements



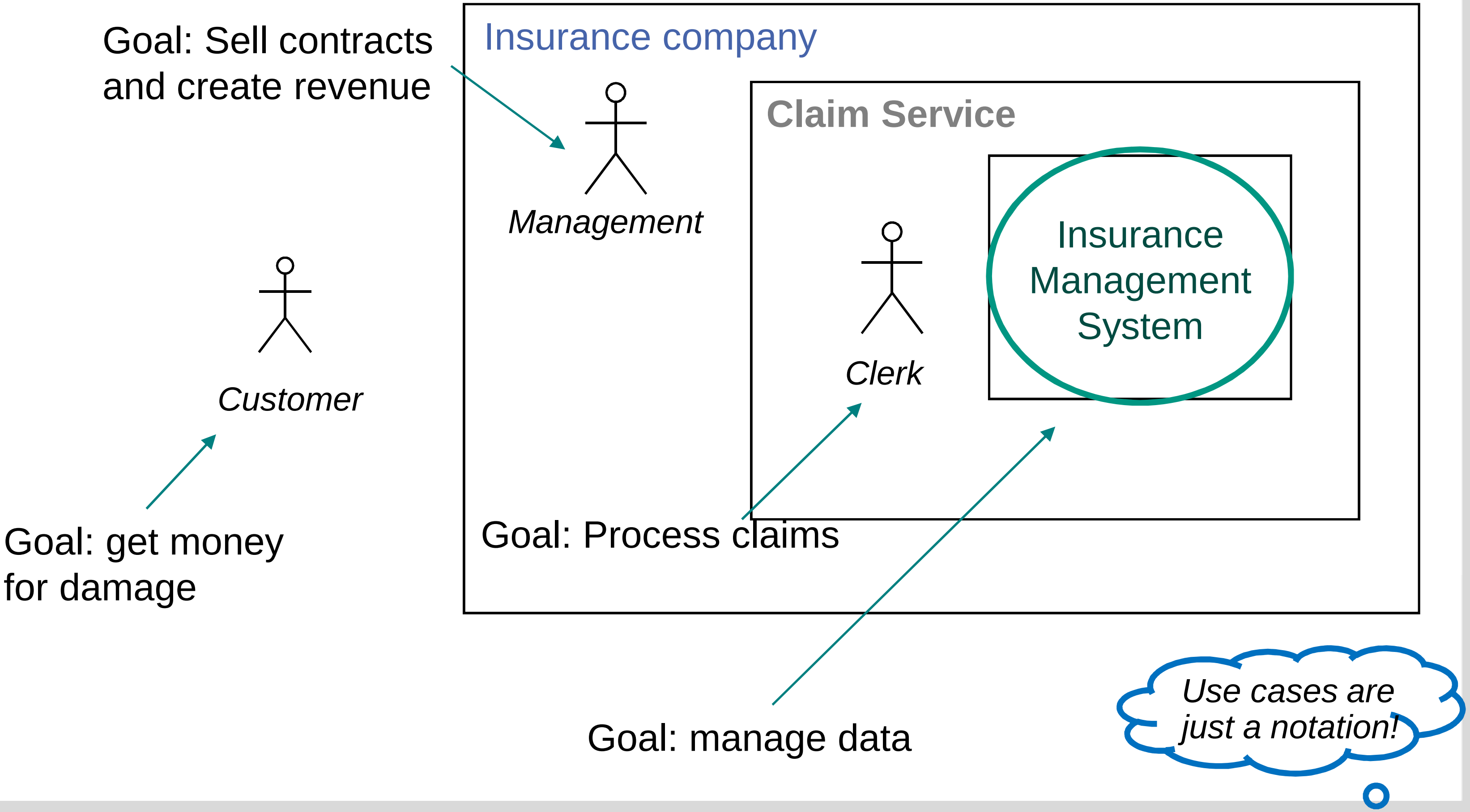
- In the context of software development
  - the term usually denotes a **“black-box user goal use case”**

- ***Other goal levels and modeling scopes*** are also feasible
  - however, in practice people often ignore this fact
    - which can cause a lot of confusion

Goals, Actors and Scopes

- It is important to understand what should be modeled as a use case
  - identify the boundary of the "system" under consideration

[Larman], [Cockburn]


Goal: Sell contracts and create revenue
Insurance company
Management
Claim Service
Clerk
Insurance Management System
Customer
Goal: get money for damage
Goal: Process claims
Goal: manage data
Use cases are just a notation!

System Context and Scope

- System Context
  - part of the environment of a system relevant for definition and understanding of requirements
- System Boundary (Scope)
  - border between editable system and non-editable environment
  - interface of the system to the environment
    - information sources and sinks
    - hardware and software
- Context Boundary
  - separates system context from irrelevant environment

➔ *Most boundaries are usually unsharp in beginning*

  - and only sharpened during requirements elicitation

What makes a good Use Case?

- It is common to be unsure if something is a valid (or useful) use case

- Activities can be grouped at many levels of granularity
  - from individual small steps (operations) to enterprise level activities
  - which of the following is a valid use case?
    - *negotiate a supplier contract*
    - *handle returns*
    - *withdraw money*

  ➔ arguably, these are all uses cases at different levels, depending on the system boundary, actors and goals

➔ For the requirements analysis of a computer application a *good rule of thumb* is to focus on elementary business processes (EBPs)

- that usually form good user goal use cases

Elementary Business Process (EBP)

- In business process engineering an EBP is defined as –
  - *a task*
  - *performed by one person*
  - *in one place at one time,*
  - *in response to a business event,*
  - *which adds **measurable business** value*
  - *and leaves data in a consistent state*

- Approving a *credit request* might be a simple example

- A common mistake is defining many use cases at too low a level
  - that should be a step, subfunction or subtask within an EBP
  - but it is often useful to create “sub” use cases
    - similar to subfunctions
    - sub use cases represent frequently performed or common (i.e. shared) subtasks

Shaping Heuristics

- Two helpful rules of thumb for identifying valuable user goal use cases are –

1. the boss test [Larman]
    - imagine your boss asks you what you have been doing all day?
    - how pleased would he be if you answer –
        - *I have been login in …*
        - *I have been searching customer data …*
        - *I have been creating customer accounts …*

2. the coffee break test [Cockburn]
    - finish a use case when would you intuitively make a coffee break, e.g. –
        - *would you leave your computer for a coffee after you have searched the customer data you need to edit?*
        - *or rather after you have edited and saved the data?*

Possible Use Case Goal Levels

- [Cockburn] identifies a number of practical goal levels for use cases
  1. (High-Level) Summary *(often equals a business process)*
     - clues together a number of user goal and subfunction use cases
       - e.g. …
  2. **User Goal** *(= EBP)*
     - describes how a user's goal is reached by interacting with the system
       - e.g. …
  3. Sub(function)
     - used to factor out "subgoals" required to achieve a goal
       - typically without "direct" business value
       - e.g. …

---

  4. Too low *(= feature or system operation)*
     - everything that is "smaller" than a subfunction is considered being "too low" for its own use case
       - usually just a "call-return scenario"
       - e.g. …

Let‘s practice a bit …

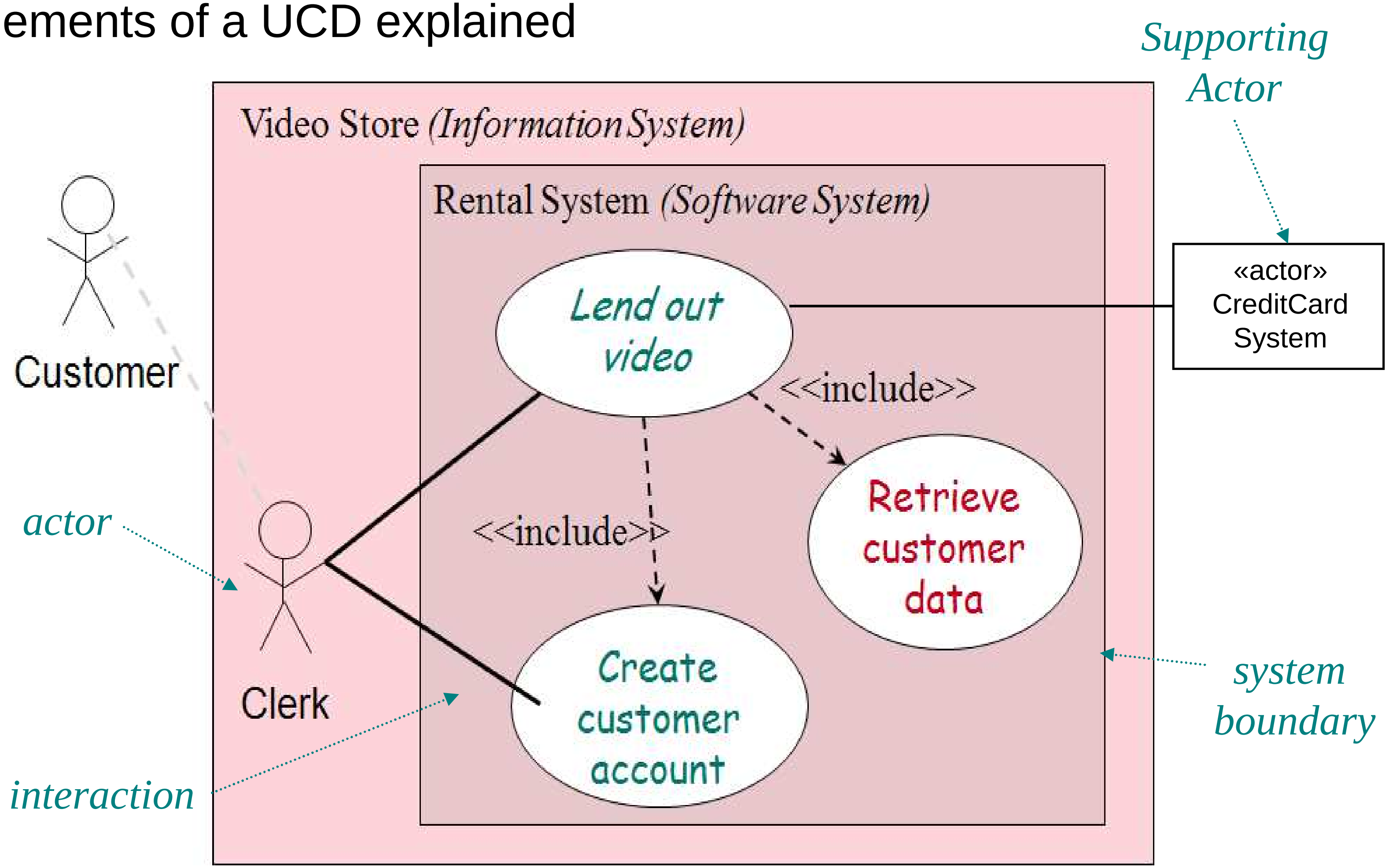
- *Identity the correct goal level for each activity*

1. process a claim in an insurance company
2. create customer account in a video store
3. book a flight ticket on an online platform
4. login to your online banking account
5. retrieve a list of customer accounts
6. check credit card data in an ATM
7. withdraw money from ATM

Summary Use Case   User Goal   Sub Use Case   System Operation

Use Case Diagrams

- Elements of a UCD explained



<<include>> ~ subfunction call;   <<extend>> ~ callback of an interrupt

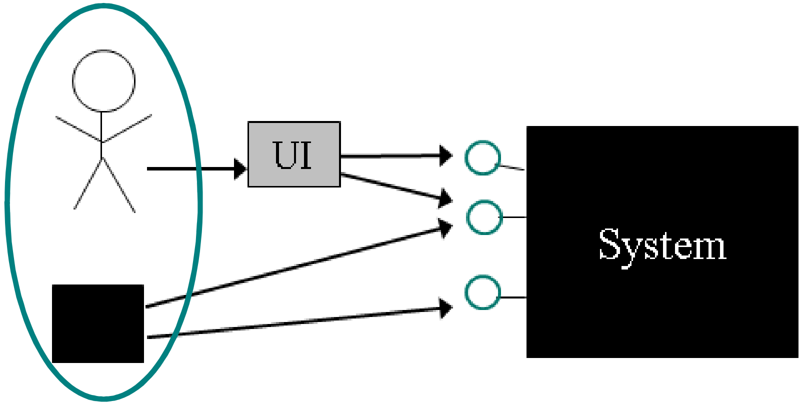
Use Case Diagramming Recommendations

- ***Diagrams are of secondary importance to writing text at this point***
  - ➔ don't get bogged down into too much diagramming work

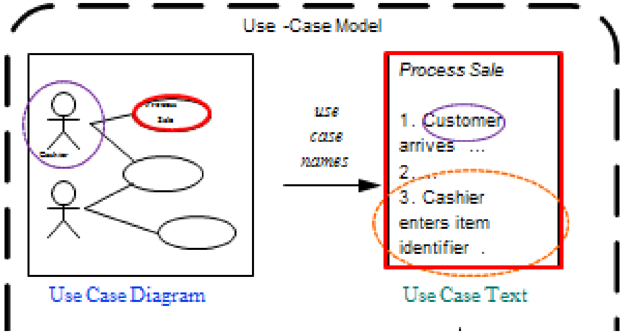
- Recommendations
  - for a use case diagram, limit the use cases to user-goal level use cases and subfunction use cases
  - show computer system actors with an alternative notation to human actors
  - place the primary actors on the left and the supporting actors on the right
  - do not spend too much time worrying about use case relationships
    - but use include and extend correctly

Key Use Case Terminology I

- An **actor** is an entity with behavior outside the system under discussion (SuD)
  - e.g. a person (i.e. role), computer system, organization
  - i.e. typically a user of the system

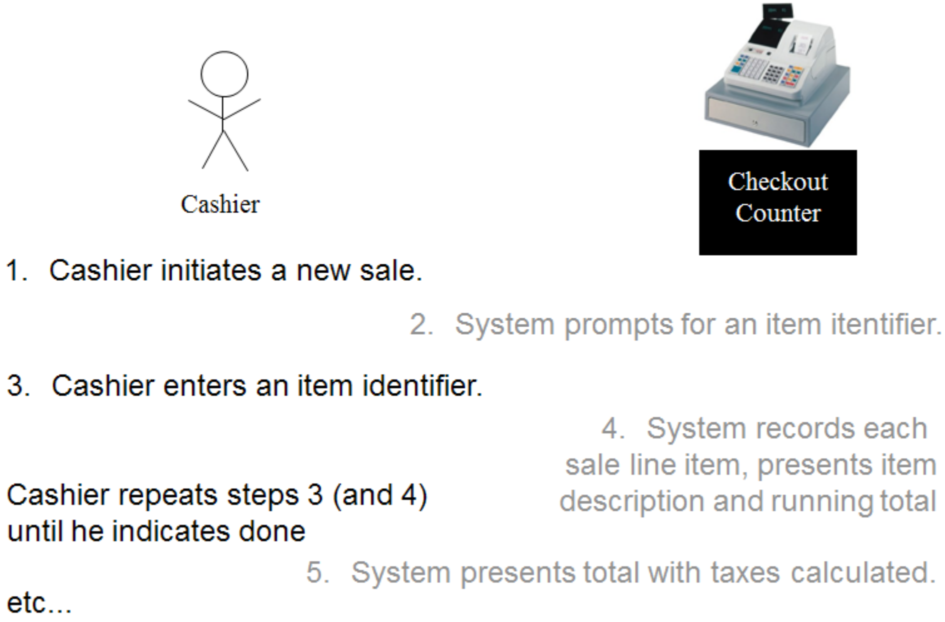


- The **use case model** is the set of all use cases and related diagrams
  - *may also include activity diagrams to illustrate flows*



Key Use Case Terminology II

- A **scenario** is a specific sequence of actions and interactions between actors and the SuD
  - a.k.a. a use case instance
  - one particular story of using the system
    - *e.g. successfully withdrawing money from an ATM*



1. Cashier initiates a new sale.

2. System prompts for an item itentifier.

3. Cashier enters an item identifier.

4. System records each sale line item, presents item description and running total

Cashier repeats steps 3 (and 4) until he indicates done

5. System presents total with taxes calculated.

etc...

- A **use case** is a collection of related success and failure scenarios that describe actors using a system to support a goal
  - a use case is a set of use case instances, i.e. scenarios
    - *e.g. all scenarios for withdrawing money from an ATM*

Use Case Elaboration

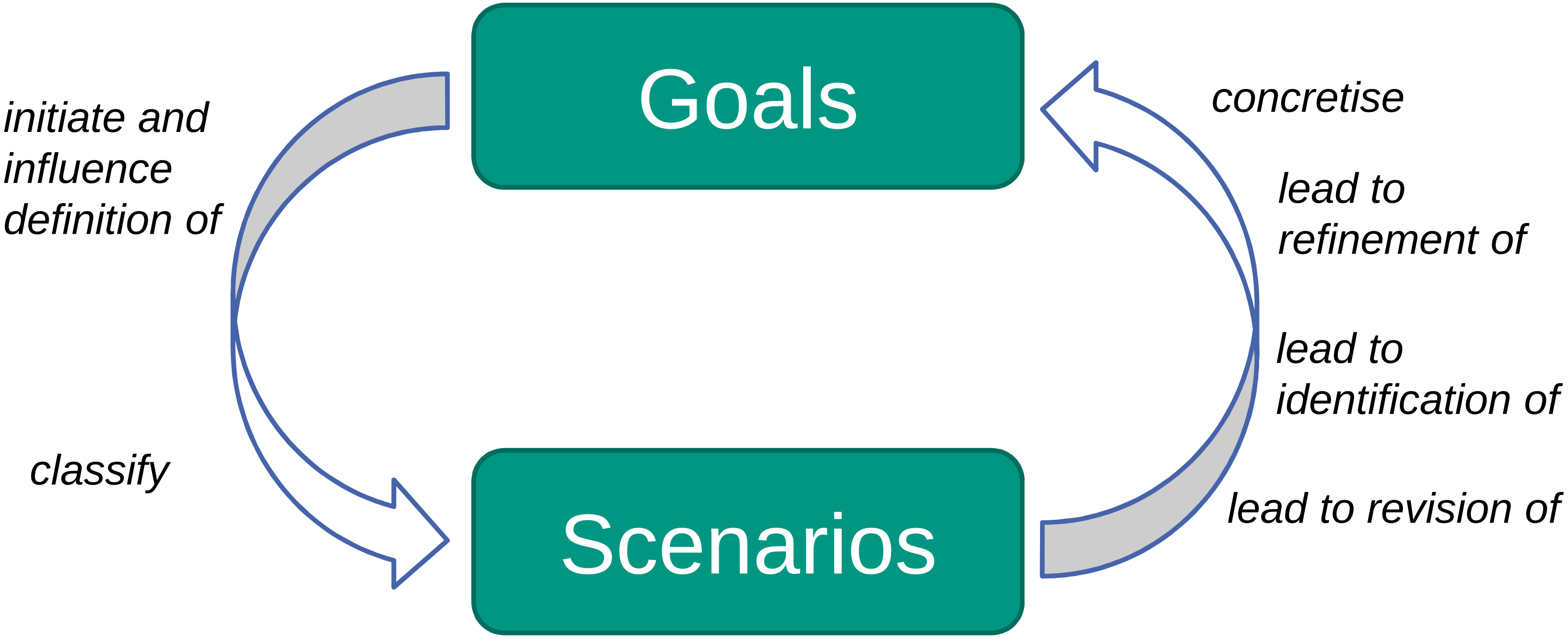
[Larman]

1. Choose the system boundary
   - e.g. the *Point of Sale* system
2. Identify the primary actors, i.e. those that have user goals fulfilled through using services of the system
   - the *cashier*, the *administrator*, the *store owner …*
3. For each actor, identify its user goals (~ user stories)
   - raise them to the highest user goal level that satisfies the EBP guideline
   - *process sale*, *print report* …
4. Define use cases that satisfy user goals
   - name them according to their goal in an imperative style
   - usually, user goal-level use cases will be one-to-one with user goals
     - one common exception are so-called CRUD (CREDO) use cases
     - *CRUD customer accounts*

Create/Retrieve/Update/Delete

Create/Retrieve/Edit/ Delete/Overview

Relation between Goals and Scenarios



Iterative Use Case Elaboration

[Cockburn]

- Fully elaborated uses cases are complex
  - hence it is common to refine them iteratively
  - [Cockburn] recommends to work breadth-first first
    - i.e. to brainstorm for use cases first and to refine them later

- Feasible precision levels are –
  1. primary actor's name and goal *(or user stories)*
  2. use case briefs *(name & brief main success scenario)*
  3. add extension conditions *(casual use cases)*
  4. add extension handling steps + more *(fully dressed use cases)*
     - usually written according to a standard template
       - as e.g. [http://alistair.cockburn.us/Basic+use+case+template]

➔ You may need to extract or merge sub use cases afterwards

- use cases are „living“ until they are approved by the customer
  - formal change requests may be necessary even later

Examples

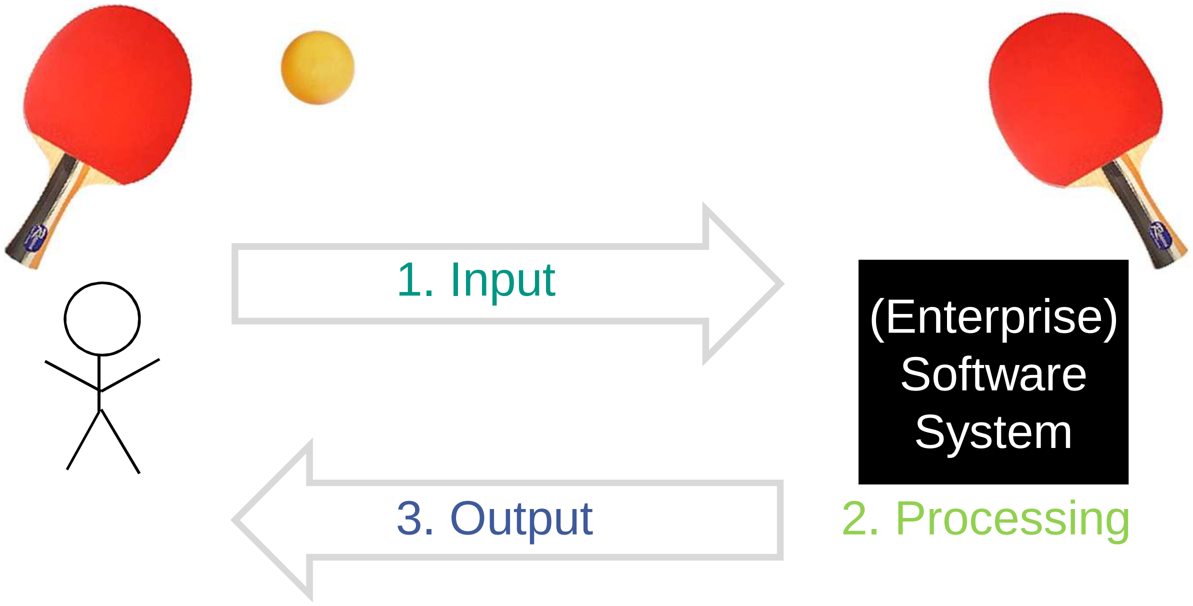
- User story
  - As a cashier I would like to process sales in order to earn money for the store
- Use Case Brief
  - The cashier enters the item identifiers into the system. The system shows the item description and calculates the running total. When the cashier indicates finished, the system calculates the total and the taxes. When the cashier enters cash payment the system stores the data of the sale, updates the inventory and prints the receipt.
- Casual Use Case *(additionally)*
  + an item identifier may be incorrectly recognized
  + the cashier may cancel the sale process
  + the cashier might need to remove an item from the sale
  + the customer may chose to pay by credit card …
- Fully Dressed Use Case
  - cf. Appendix 1

Fully Dressed

[Mad Magazine]

How to write Fully Dressed Use Cases

- User plays “ping pong” with the system



1. Input

(Enterprise) Software System

2. Processing

3. Output

- Focus on responsibilities!
  - ➔ an exemplary use case transaction might look as follows

1. *The Customer arrives at the ATM and inserts his debit card*
2. *The ATM validates the debit card and asks the customer to enter his PIN*

1. Input

2. Processing

3. Output

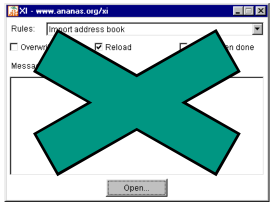
Further Writing Guidelines [Cockburn]

- It is important that each step –
  - shows a goal succeeding
  - captures the actor's intention
  - has an actor –
    - passing information **OR**
    - validating a condition **OR**
    - updating system state

  ➔ the first step of a use case may not fit this classification
    - it is often merely an event that triggers the use case

- It is common practice to start actor names with an uppercase letter

- Do **not** have UI details in use cases
  - no clicking buttons or entering text into textfields etc.
    - ➔ *this is the job of UI designers*
  - however, need to define the data exchanged
    - *such PIN, account number etc.*



XI - www.ananas.org/xi

Rules: Import address book

☐ Overwri ☑ Reload ☐ en done

Messa

Even More Guidelines

- *Data descriptions*
  1. *Precision Level 1: Data nickname*
     - *item identifier*
  2. *Precision Level 2: Data fields associated with the nickname*
     - *item identifier : String (e.g. in the domain model)*
  3. *Precision Level 3: Field types, lengths, and validations*
     - *item identifier : String, a 13 character EAN barcode, validations is specified in Standard XYZ…*

- Conditions and extensions
  - keep the main scenario free from if statements
  - rather they should be defined in the extensions section
    - *3a. Invalid identifier:*

    1. System signals error and rejects entry.

- Mark <<include>> through underlining
  - as well as linking (of course depending on the used tool)
  - *… the Cashier performs Find Product Help to obtain true item ID…*

Fully Dressed Use Case Sections (1/4)

[Larman] and [Cockburn]

1. **Preface elements**
   - many optional preface elements are possible
     - only important elements should be placed here (such as scope and goal level)
   - also common is the primary actor element
     - identifies the principal actor which calls upon system services to fulfill a goal

2. **Stakeholders and Interest List**
   - suggests and bounds what the system must do
   - provides the originating source for each responsibility

3. **Preconditions**
   - state what must always be true before beginning a scenario in the use case
   - are not tested by the use case but are assumed to be true
   - typical implies that a scenario of another use case has successfully completed
     - *e.g. logging in, cashier identified and authenticated*
   - only noteworthy assumptions should be identified
     - not every possible precondition, like the system has power etc.

Fully Dressed Use Case Sections (2/4)

4. **Post conditions** (Success Guarantees)
    - what must be true on successful completion of the use case
        - either the main success scenario or some alternative path
    - guarantee should meet the needs of all stakeholder
    - *it may be helpful to add separate post conditions for each scenario*
        - *or at least guarantees for what should not be changed in case of a failure*

5. **Main Success Scenario**
    - a.k.a basic flow or “happy path” scenario
    - describes the typical success path that satisfies the interests of the stakeholders
    - does not (normally) include conditions or branching
        - these should be deferred to the extensions sections

Fully Dressed Use Case Sections (3/4)

6. **Extensions** (a.k.a alternative flows)
   - describe all other scenarios or branches both success or fail
   - should ideally cover all functional interests of the stakeholders together with the main flow
   - are branches from the main success scenario and so can refer to it
   - alternative extensions are labeled with letters and have two parts
     - Condition
       - the condition should be written as **something that can be detected by the system**
     - Handling
       - the handling part can be written as a sequence of sub steps
   - by default, an alternative scenario merges back with the main scenario
     - unless explicitly halted
   - if an extension point is complex, it might be better to express the extension as a separate use case
     - that is usually included
   - an extension condition that could happen at any point (like an exception) is labeled with a *

Fully Dressed Use Case Sections (4/4)

7. Special requirements

- record that a non-functional requirement, quality attribute or constraint relates specifically to a use case
  - e.g. performance, reliability, usability design constraints
- non-functional requirements can later be collected within a Supplementary Specification

8. Technology and Data variations

- describes technical variations in how things must be done rather than what
  - e.g. a constraint imposed by a stakeholder concerning for example an I/O device
  - such design constraints should in general be avoided, but if they are unavoidable they should be included
- also helpful to record variations in data schemes
  - e.g. using UPCs or EAN’s for item identifiers
  - you may add your data descriptions here

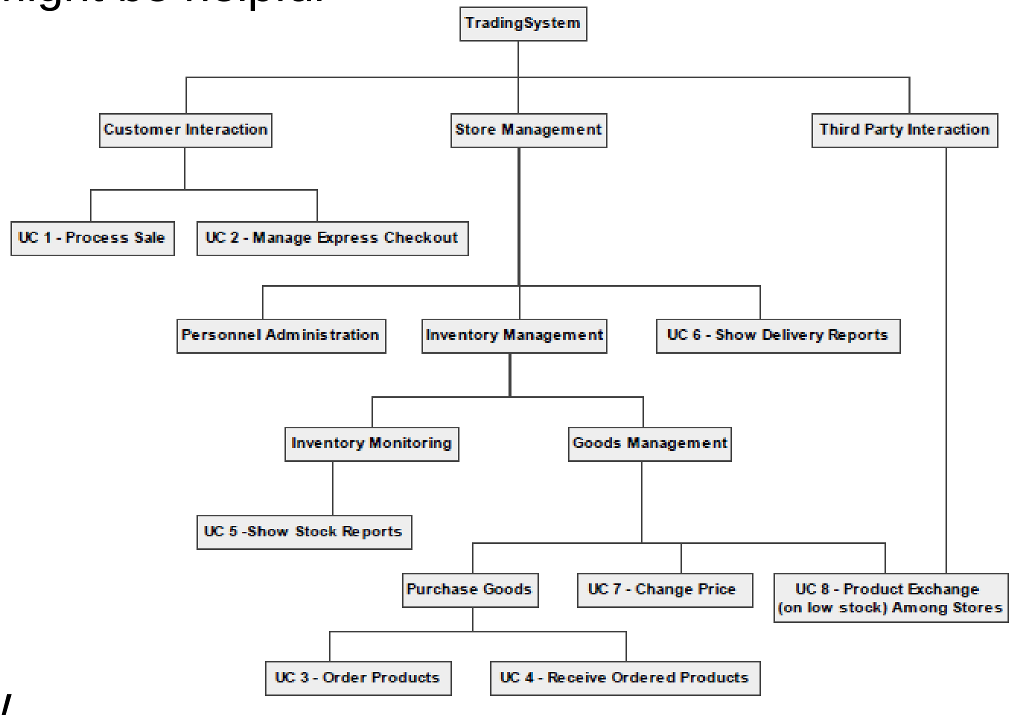
Helpful Stuff for Real Life I

- Checklists can become handy for verifying requirements
  - e.g. the one from Cockburn listed in Appendix 2
- Special tools to manage use cases are available
  - such as –
    - Borland Caliber
    - Serena Dimensions RM
    - …
- However, use case texts can be written with every word processor
  - they may even be managed with a simple spreadsheet application

ID	Lev.	Area	Name	Prior.	Freq. of Use	Impor.	State
UC1	UG	CRM	Cr. Cust. Account	VH	M	Must	Appr.
UC2	SF	CRM	Retr. Cust. Acc.	H	VH	Should	Written
UC7	UG	CRM	Send Advert. Mail	M	L	Nice	Open

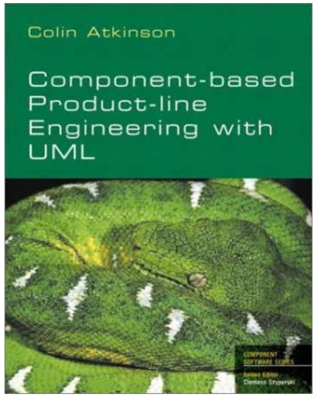
Helpful Stuff for Real Life II

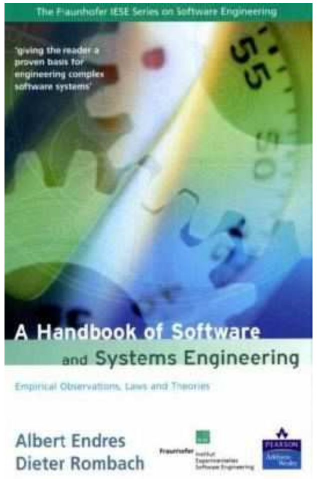
- Software systems in practice may have dozens of use cases
  - a simple Enterprise Process Diagram as proposed by KobrA might be helpful



- TradingSystem
  - Customer Interaction
    - UC 1 - Process Sale
    - UC 2 - Manage Express Checkout
  - Store Management
    - Personnel Administration
    - Inventory Management
      - Inventory Monitoring
        - UC 5 -Show Stock Reports
      - Goods Management
        - Purchase Goods
          - UC 3 - Order Products
          - UC 4 - Receive Ordered Products
        - UC 7 - Change Price
        - UC 8 - Product Exchange (on low stock) Among Stores
    - UC 6 - Show Delivery Reports
  - Third Party Interaction
    - UC 8 - Product Exchange (on low stock) Among Stores

- Conway‘s law
  - *A system reflects the organizational structure that built it.*



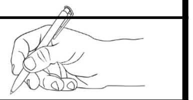

The Fraunhofer IESE Series on Software Engineering
'giving the reader a proven basis for engineering complex software systems'
A Handbook of Software and Systems Engineering
Empirical Observations, Laws and Theories
Albert Endres
Dieter Rombach

Software Requirements Specification

- Complete description of the external behavior of the software
  - usually based on prosaic text captured in a word processor
    - ➔ can lead to ambiguities
  - can include references to more formal models
    - e.g. include UML diagrams as figures
  - use a standarised use case and document templates
    - ➔ stand on the shoulders of giants

- Documentation of all interfaces between software and its environment
  - describes what and not how
    - system remains a **black-box**
    - BUT: design decisions taken by constraints can be included (e.g., platform)

IN or OUT? (of a SW spec.)

Element	IN	OUT
Functional requirements	X	
Interfaces to other software		
Schedules and cost estimates		
Requirements verification and validation plan		
UI & Usability Requirements		
Milestones		
Quality assurance plan / test plans		
Design constraints		
Non-functional requirements		
Performance requirements		
Costs and cost limitations		
Reliability requirements		
Software designs		

Requirements Document Structure

- … usually contains chapters like –
  - Introduction
    - Purpose, System Scope, Stakeholders, Definitions, Acronyms, Abbrevitions, References, Overview
  - Overall Description
    - System environment, architectural description, system functionality, users and audience, constraints, assumptions
  - Requirements
    - Functional, quality, …
  - Appendix
  - Index
- Examples: *Rational Unified Process (RUP), IEEE 830, V-Modell* (Pflichten- und Lastenheft), etc…, -> Appendix
  - templates usually need project specific adaptations

Requirements Management

- Maintain requirements in a central repository
  - usage of spreadsheets or databases
  - special requirements software

- Link requirements
  - with documents and dependent requirements

- Use extended attributes
  - ID, Name, Description, Version, Author, Source
  - Stability, Criticality, Priority
  - Status, Type, Links, Release, Infos, Effort

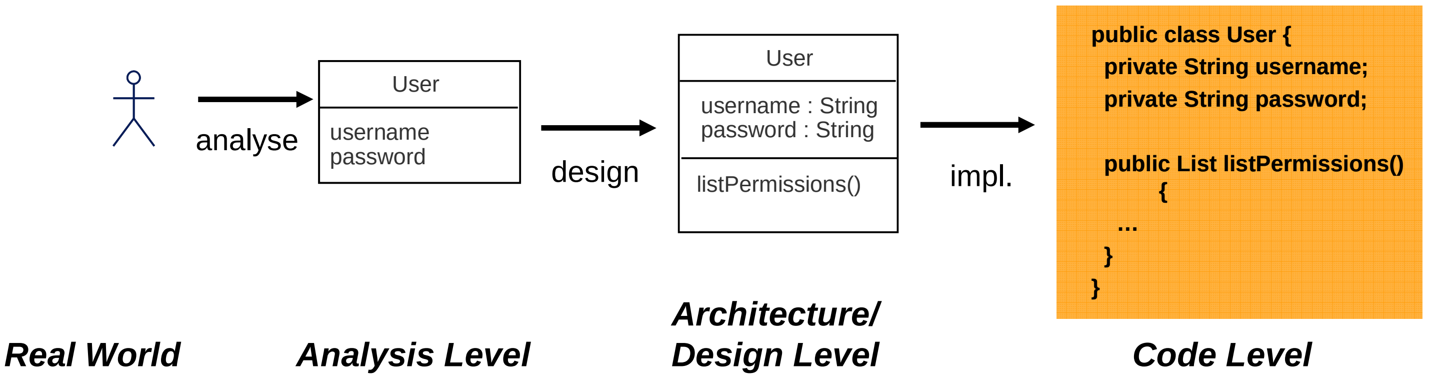
  ⇒ Traceability

Tooling

- Specialised software helps managing requirements
- Tree-like organization of use cases/features
  - Support automatic generation of SRSs
  - Support the RE process
    - Guide the requirements engineer
  - Automatic validity and conflict checks
  - Versioning
- Usually central repository on dedicated server
  - Multi-user access
  - Web-Frontend for customer access
    - High client license fees

Divide and Conquer in Domain Analysis

- Since software problems can be complex, **decomposition (divide and conquer)** is a commonly applied strategy
- In traditional (structured analysis) techniques the dimension of decomposition is the function
- In object-oriented analysis the dimension of decomposition is things or entities in the domain (i.e. objects)



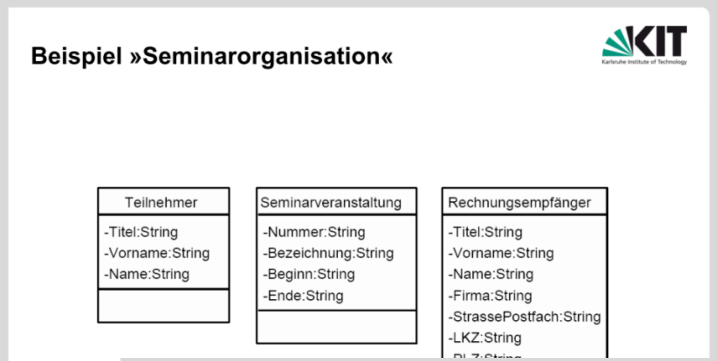
```
public class User {
  private String username;
  private String password;

  public List listPermissions()
       {
    ...
  }
}
```

➔ *Thus, a central distinction between object-oriented and structured analysis is decomposition by conceptual classes (objects) rather than by functions*

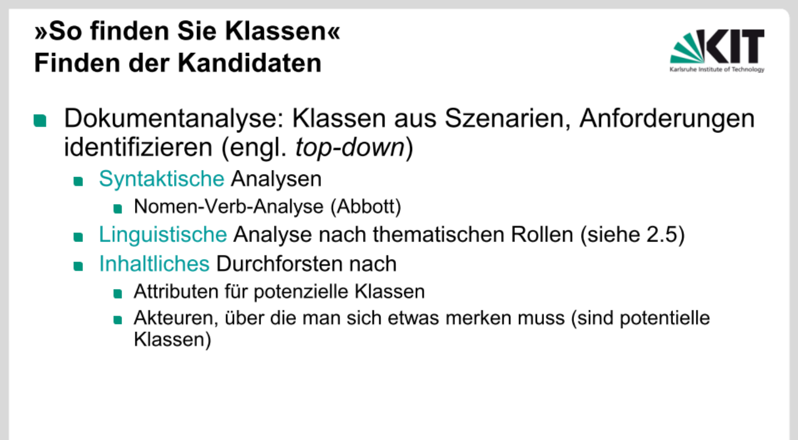
And then..?

- Domain Analysis [SWT I]



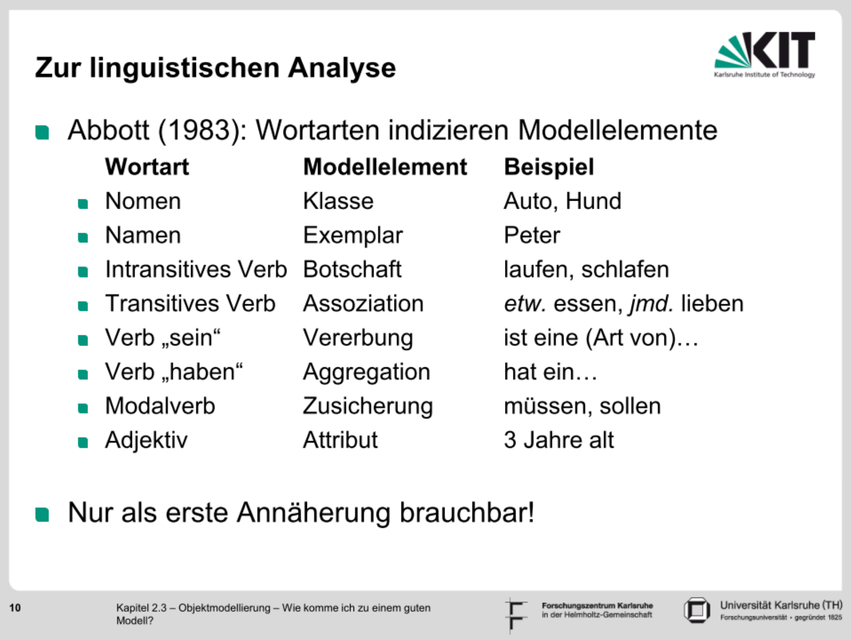
Beispiel »Seminarorganisation«

7 Kapit… Mode…



»So finden Sie Klassen«
Finden der Kandidaten

- Dokumentanalyse: Klassen aus Szenarien, Anforderungen identifizieren (engl. *top-down*)
  - Syntaktische Analysen
    - Nomen-Verb-Analyse (Abbott)
  - Linguistische Analyse nach thematischen Rollen (siehe 2.5)
  - Inhaltliches Durchforsten nach
    - Attributen für potenzielle Klassen
    - Akteuren, über die man sich etwas merken muss (sind potentielle Klassen)



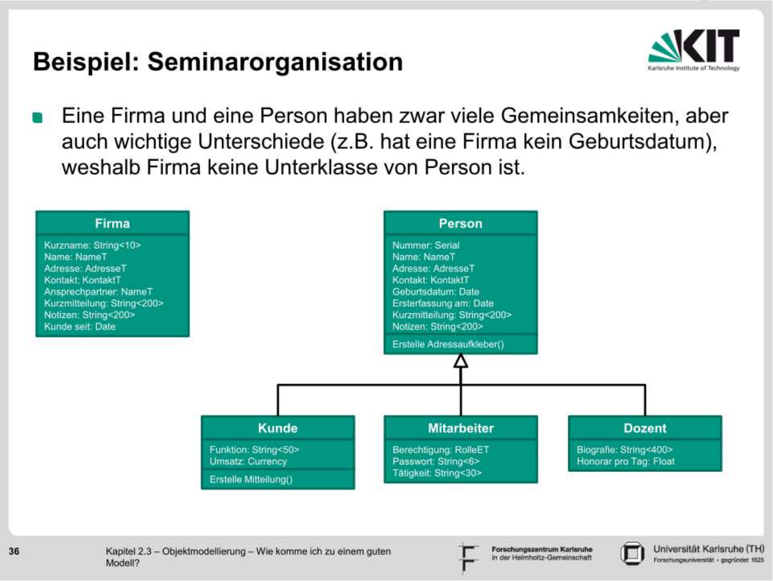
Zur linguistischen Analyse

- Abbott (1983): Wortarten indizieren Modellelemente

Wortart	Modellelement	Beispiel
Nomen	Klasse	Auto, Hund
Namen	Exemplar	Peter
Intransitives Verb	Botschaft	laufen, schlafen
Transitives Verb	Assoziation	*etw.* essen, *jmd.* lieben
Verb „sein“	Vererbung	ist eine (Art von)…
Verb „haben“	Aggregation	hat ein…
Modalverb	Zusicherung	müssen, sollen
Adjektiv	Attribut	3 Jahre alt

- Nur als erste Annäherung brauchbar!

10 Kapitel 2.3 – Objektmodellierung – Wie komme ich zu einem guten Modell? Forschungszentrum Karlsruhe in der Helmholtz-Gemeinschaft Universität Karlsruhe (TH) Forschungsuniversität • gegründet 1825



Beispiel: Seminarorganisation

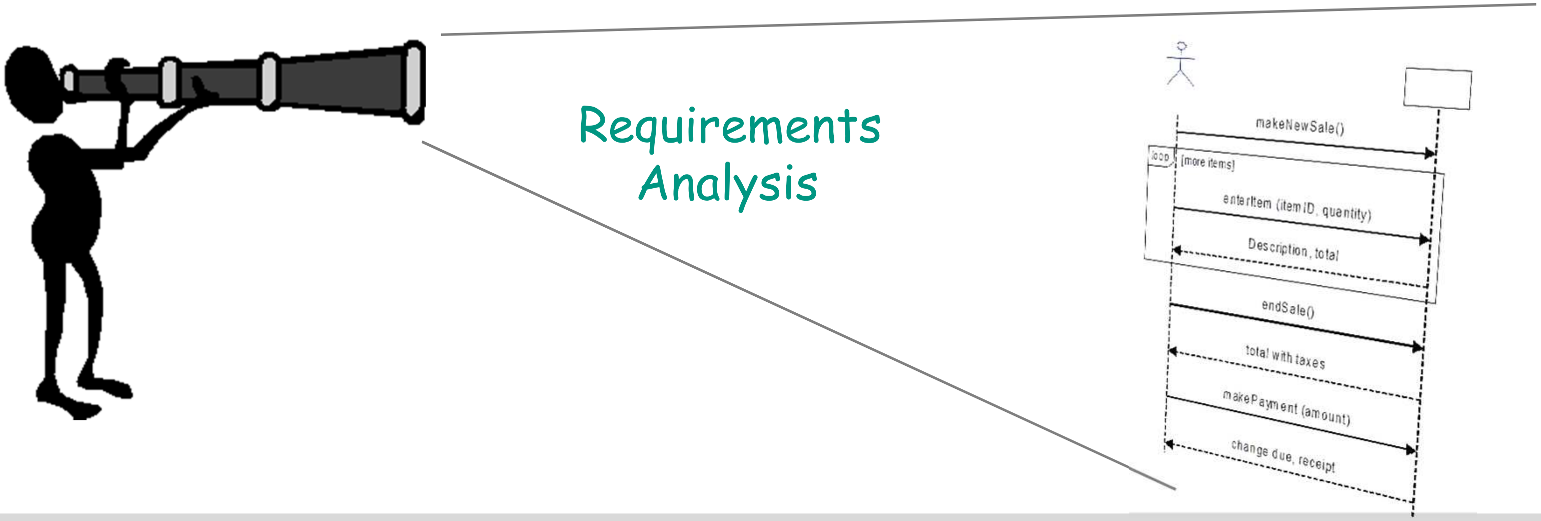
- Eine Firma und eine Person haben zwar viele Gemeinsamkeiten, aber auch wichtige Unterschiede (z.B. hat eine Firma kein Geburtsdatum), weshalb Firma keine Unterklasse von Person ist.

36 Kapitel 2.3 – Objektmodellierung – Wie komme ich zu einem guten Modell? Forschungszentrum Karlsruhe in der Helmholtz-Gemeinschaft Universität Karlsruhe (TH) Forschungsuniversität • gegründet 1825

Conclusion

- Use cases are an established best practice for requirments elicitation
  - though simple at a first glance their are many subtle details to consider –
    - goal level
    - clear assignment of responsiblities
    - etc.

- Thank you for your attention!



References

- [Ambler] S. Ambler
  The Object Primer: Agile Model-Driven Development with UML 2.0
  Cambridge University Press, 2004

- [Ambler05] S. Ambler
  A Manager's Introduction to the Rational Unified Process
  Online, 2005
  http://www.ambysoft.com/downloads/managersIntroToRUP.pdf

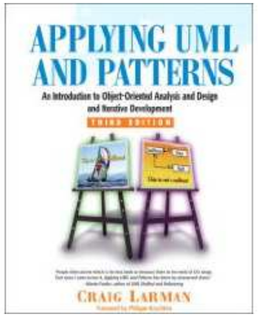
- [Cockburn] A. Cockburn
  Writing Effective Use Cases
  Addison-Wesley, 2000

- [Endres/Rombach03] A. Endres, D. Rombach
  A Handbook of Software and Systems Engineering
  Addison-Wesley, 2003
- Lamsweerde, A. V. (2001), Goal-oriented requirements engineering: A guided tour

- [Larman] C. Larman
  Applying UML and Patterns (3rd ed.)
  Prentice Hall, 2004

- Pohl, K. (2007), Requirements Engineering: Grundlagen, Prinzipien, Techniken, dpunkt, Heidelberg

Larman-Buch


APPLYING UML AND PATTERNS
An Introduction to Object-Oriented Analysis and Design and Iterative Development
THIRD EDITION
CRAIG LARMAN

- Auskunft der Info-Bib 2012

„(…) von der deutschen Ausgabe werde ich für die Informatikbibliothek 1 Präsenz- und 3 Ausleihexemplare kaufen. Die KIT-Bibliothek besitzt bereits 3 Exemplare und stockt den Bestand um weitere 30 Exemplare auf.

Von der englischen Ausgabe stehen in der Informatikbibliothek 1 Präsenz- und 2 Ausleihexemplare. Hier werde ich ein weiteres Ausleihexemplar kaufen, so dass von der deutschen und von der englischen Ausgabe jeweils 1 Präsenz- und 3 Ausleihexemplare zur Verfügung stehen. Die KIT-Bibliothek wird die englische E-Book-Ausgabe kaufen. Hier erhielt ich folgende Nachricht "Bei der englischen Ausgabe bitten wir um etwas Geduld. Eine E-Book-Ausgabe ist nachgewiesen aber eine Zeitangabe über die feste Zusage kann ich leider nicht machen. "

Eine kleine Bitte noch: könnten Sie die Studierenden bitte darauf hinweisen, dass in der Informatikbibliothek auch Ausleihexemplare zur Verfügung stehen. Es kommt bei den Informatikbüchern immer wieder vor, dass alle Exemplare der KIT-Bibliothek ausgeliehen und bereits sogar mehrfach vorgemerkt sind, obwohl es in der Informatikbibliothek noch ausleihbare Exemplare gibt.“

Appendices

- Examplary Requirements Templates
  - RUP (FURPS+)
  - IEEE
- Requirements Tools
  - DOORS
  - Requisite Pro
  - Requirements Composer
  - Caliber
  - SoftWiki
- Fully Dressed Use Case Example
- Use Case Checklist

RUP SRS Document Structure

- 1. Introduction
  - 1.1 Purpose
  - 1.2 Scope
  - 1.3 Definitions, Acronyms and Abbreviations
  - 1.4 References
  - 1.5 Overview
- 2. Overall Description
- 3. Specific Requirements
  - 3.1 Functionality
    - 3.1.1 <Functional Requirement One>
  - 3.2 Usability
    - 3.2.1 <Usability Requirement One>
  - 3.3 Reliability
    - 3.3.1 <Reliability Requirement One>
  - 3.4 Performance
    - 3.4.1 <Performance Requirement One>
  - 3.5 Supportability
    - 3.5.1 <Supportability Requirement One>
  - 3.6 Design Constraints
    - 3.6.1 <Design Constraint One>
  - 3.7 Online User Documentation and Help System Requirements
  - 3.8 Purchased Components
  - 3.9 Interfaces
    - 3.9.1 User Interfaces
    - 3.9.2 Hardware Interfaces
    - 3.9.3 Software Interfaces
    - 3.9.4 Communications Interfaces
  - 3.10 Licensing Requirements
  - 3.11 Legal, Copyright and Other Notices
  - 3.12 Applicable Standards
- 4. Supporting Information

IEEE SRS Document Structure [IEEE 830-1998]

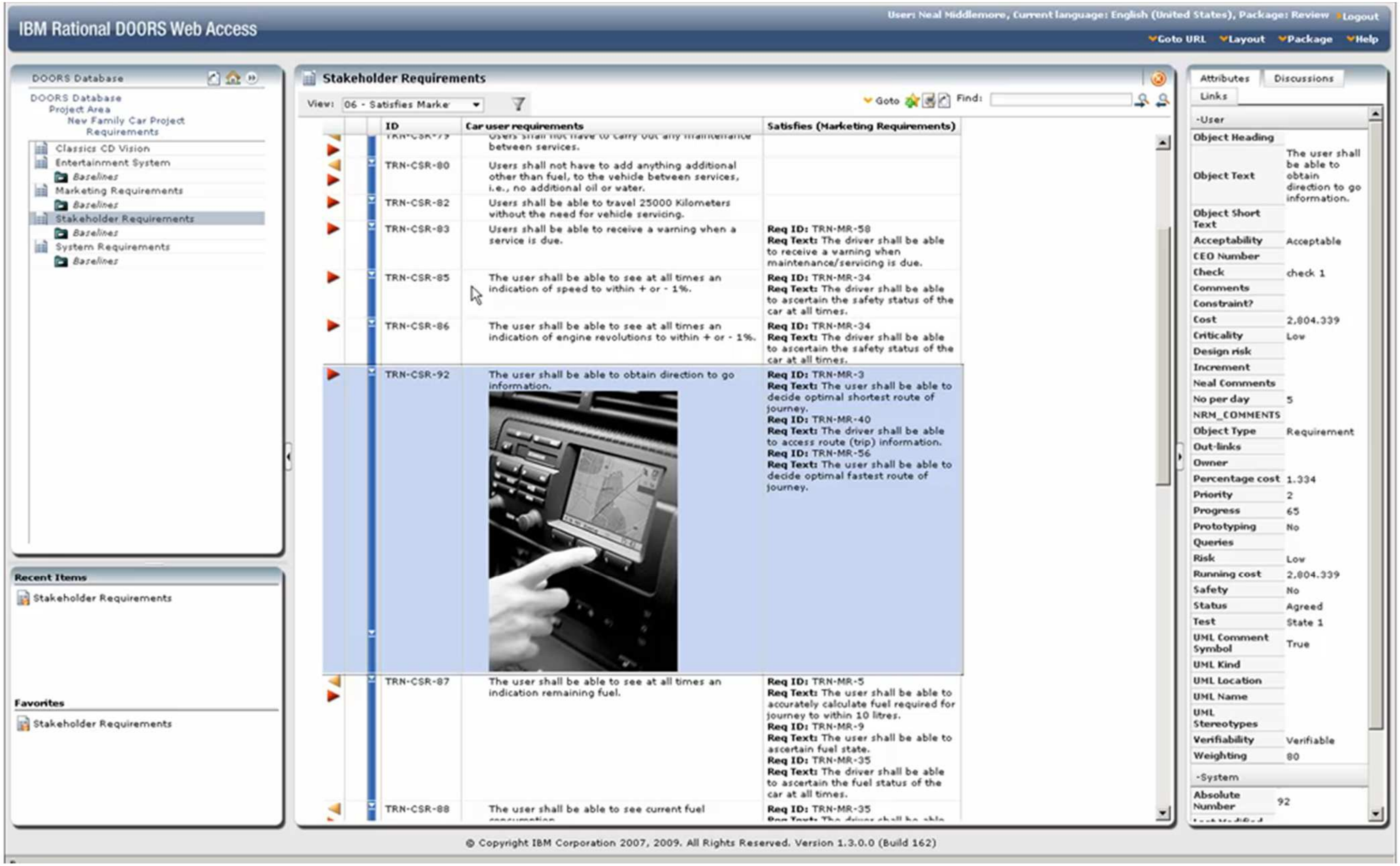
- Table of Contents
- Introduction
  - Purpose
  - Scope
  - Definitions
  - References
  - Overview
- General Description
  - Product Perspective
  - Product Function
  - User Characteristics
  - General Constraints
  - Assumptions and Dependencies
- Specific Requirements
  - Functional Requirements
  - External Interface Requirements
  - Performance Requirements
  - Design Constraints
  - Attributes
  - Other Requirements
- Appendix
- Index

Tool: IBM Rational DOORS

- Formely know as Telelogic DOORS
- Dedicated to requirements change management
  - Supports custom requirements processes / workflows
- Server based
- Reviewer web client
- Integrated with many other development tools
  - Rational Requirements Composer etc.

- Can generate requirements documents

DOORS Web Access



ID	Car user requirements	Satisfies (Marketing Requirements)
TRN-CSR-79	Users shall not have to carry out any maintenance between services.	
TRN-CSR-80	Users shall not have to add anything additional other than fuel, to the vehide between services, i.e., no additional oil or water.	
TRN-CSR-82	Users shall be able to travel 25000 Kilometers vithout the need for vehide servicing.	
TRN-CSR-83	Users shall be able to receive a warning when a service is due.	**Req ID:** TRN-MR-58 **Req Text:** The driver shall be able to receive a warning when maintenance/servicing is due.
TRN-CSR-85	The user shall be able to see at all times an indication of speed to within + or - 1%.	**Req ID:** TRN-MR-34 **Req Text:** The driver shall be able to ascertain the safety status of the car at all times.
TRN-CSR-86	The user shall be able to see at all times an indication of engine revolutions to within + or - 1%.	**Req ID:** TRN-MR-34 **Req Text:** The driver shall be able to ascertain the safety status of the car at all times.
TRN-CSR-92	The user shall be able to obtain direction to go information.	**Req ID:** TRN-MR-3 **Req Text:** The user shall be able to decide optimal shortest route of journey. **Req ID:** TRN-MR-40 **Req Text:** The driver shall be able to access route (trip) information. **Req ID:** TRN-MR-56 **Req Text:** The user shall be able to decide optimal fastest route of journey.
TRN-CSR-87	The user shall be able to see at all times an indication remaining fuel.	**Req ID:** TRN-MR-5 **Req Text:** The user shall be able to accurately calculate fuel required for journey to within 10 litres. **Req ID:** TRN-MR-9 **Req Text:** The user shall be able to ascertain fuel state. **Req ID:** TRN-MR-35 **Req Text:** The driver shall be able to ascertain the fuel status of the car at all times.
TRN-CSR-88	The user shall be able to see current fuel	**Req ID:** TRN-MR-35

-User

Object Heading	
Object Text	The user shall be able to obtain direction to go information.
Object Short Text	
Acceptability	Acceptable
CEO Number	
Check	check 1
Comments	
Constraint?	
Cost	2,804.339
Criticality	Low
Design risk	
Increment	
Neal Comments	
No per day	5
NRM_COMMENTS	
Object Type	Requirement
Out-links	
Owner	
Percentage cost	1.334
Priority	2
Progress	65
Prototyping	No
Queries	
Risk	Low
Running cost	2,804.339
Safety	No
Status	Agreed
Test	State 1
UML Comment Symbol	True
UML Kind	
UML Location	
UML Name	
UML Stereotypes	
Verifiability	Verifiable
Weighting	80

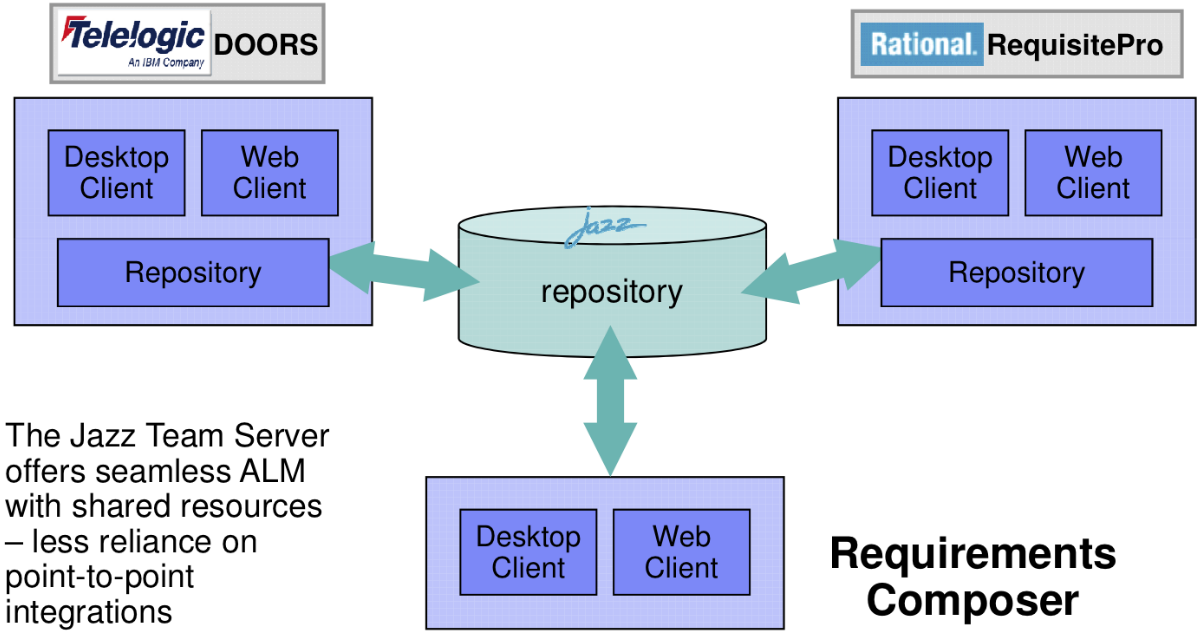
-System

Absolute Number	92

© Copyright IBM Corporation 2007, 2009. All Rights Reserved. Version 1.3.0.0 (Build 162)

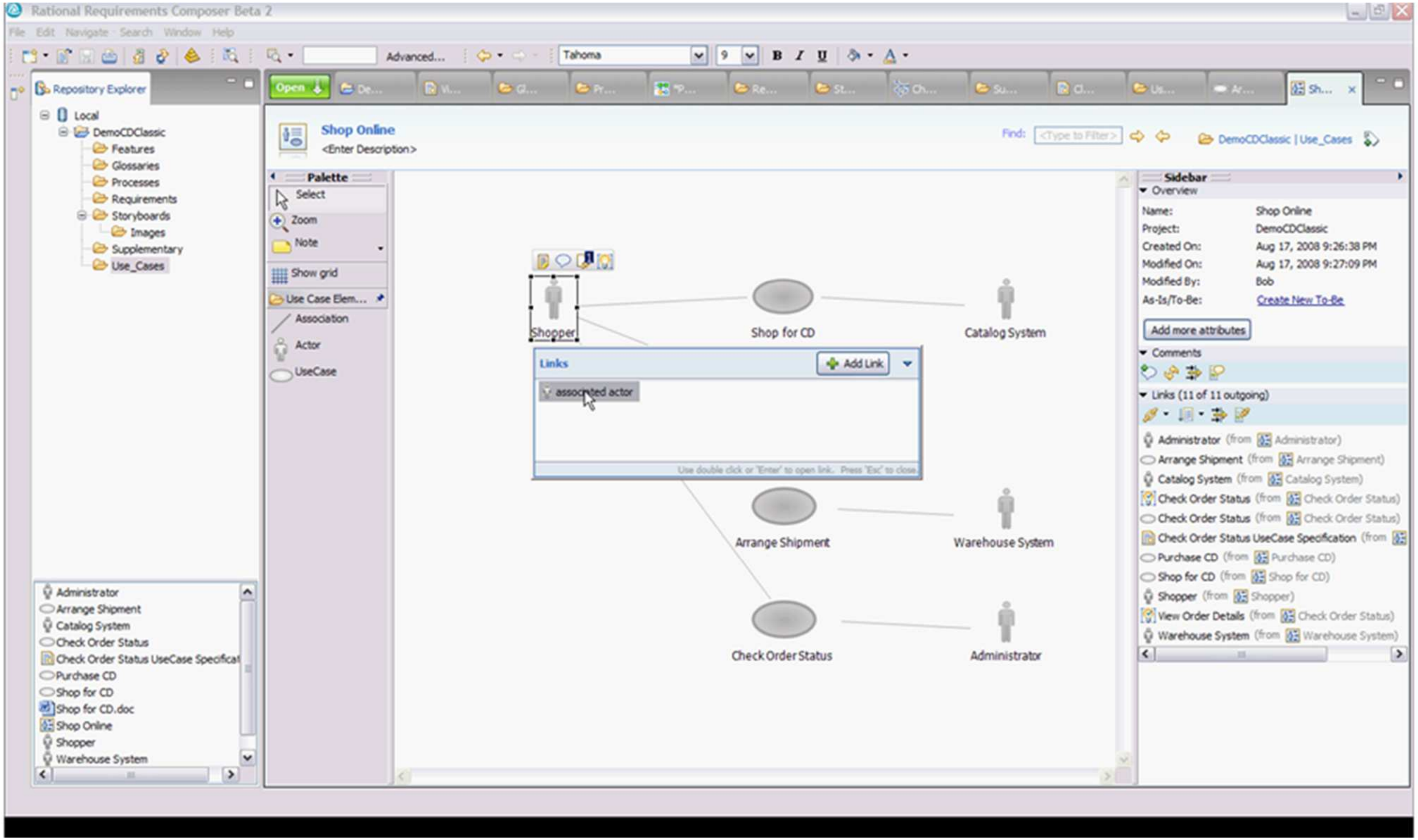
Tool: IBM Requisite Pro

- Very similar to Doors
  - Both are #1 and #2 on the market
- Offers full web client



The Jazz Team Server offers seamless ALM with shared resources – less reliance on point-to-point integrations

Tool: Requirements Composer

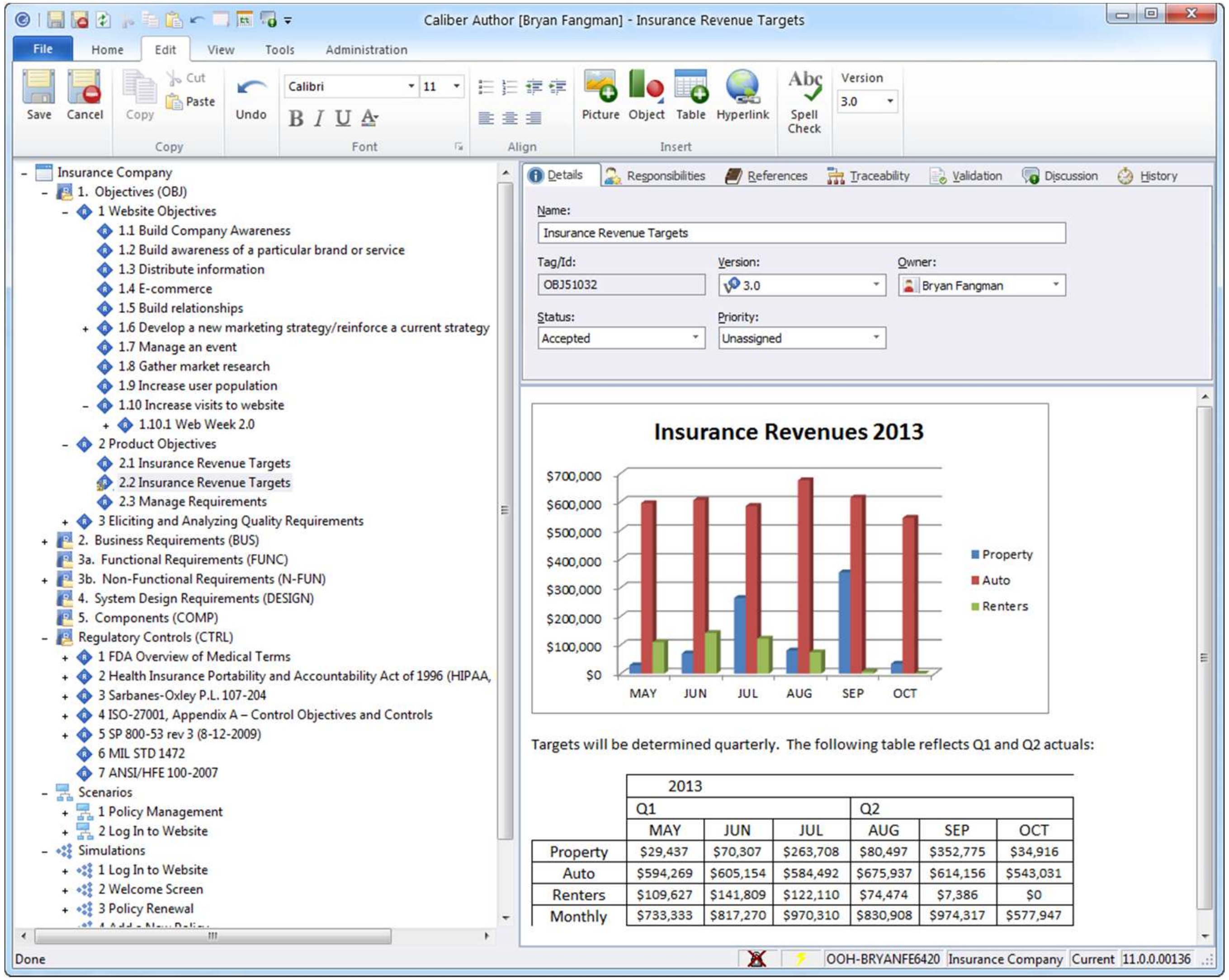


Use double click or 'Enter' to open link. Press 'Esc' to close.

Name:	Shop Online
Project:	DemoCDClassic
Created On:	Aug 17, 2008 9:26:38 PM
Modified On:	Aug 17, 2008 9:27:09 PM
Modified By:	Bob
As-Is/To-Be:	Create New To-Be

- Administrator (from Administrator)
- Arrange Shipment (from Arrange Shipment)
- Catalog System (from Catalog System)
- Check Order Status (from Check Order Status)
- Check Order Status (from Check Order Status)
- Check Order Status UseCase Specification (from
- Purchase CD (from Purchase CD)
- Shop for CD (from Shop for CD)
- Shopper (from Shopper)
- View Order Details (from Check Order Status)
- Warehouse System (from Warehouse System)

Tool: Borland Caliber



- Insurance Company
  - 1. Objectives (OBJ)
    - 1 Website Objectives
      - 1.1 Build Company Awareness
      - 1.2 Build awareness of a particular brand or service
      - 1.3 Distribute information
      - 1.4 E-commerce
      - 1.5 Build relationships
      - 1.6 Develop a new marketing strategy/reinforce a current strategy
      - 1.7 Manage an event
      - 1.8 Gather market research
      - 1.9 Increase user population
      - 1.10 Increase visits to website
        - 1.10.1 Web Week 2.0
    - 2 Product Objectives
      - 2.1 Insurance Revenue Targets
      - 2.2 Insurance Revenue Targets
      - 2.3 Manage Requirements
    - 3 Eliciting and Analyzing Quality Requirements
  - 2. Business Requirements (BUS)
  - 3a. Functional Requirements (FUNC)
  - 3b. Non-Functional Requirements (N-FUN)
  - 4. System Design Requirements (DESIGN)
  - 5. Components (COMP)
  - Regulatory Controls (CTRL)
    - 1 FDA Overview of Medical Terms
    - 2 Health Insurance Portability and Accountability Act of 1996 (HIPAA,
    - 3 Sarbanes-Oxley P.L. 107-204
    - 4 ISO-27001, Appendix A – Control Objectives and Controls
    - 5 SP 800-53 rev 3 (8-12-2009)
    - 6 MIL STD 1472
    - 7 ANSI/HFE 100-2007
  - Scenarios
    - 1 Policy Management
    - 2 Log In to Website
  - Simulations
    - 1 Log In to Website
    - 2 Welcome Screen
    - 3 Policy Renewal

Name: Insurance Revenue Targets

Tag/Id: OBJ51032

Version: 3.0

Owner: Bryan Fangman

Status: Accepted

Priority: Unassigned

Targets will be determined quarterly. The following table reflects Q1 and Q2 actuals:

	2013					
	Q1			Q2		
	MAY	JUN	JUL	AUG	SEP	OCT
Property	$29,437	$70,307	$263,708	$80,497	$352,775	$34,916
Auto	$594,269	$605,154	$584,492	$675,937	$614,156	$543,031
Renters	$109,627	$141,809	$122,110	$74,474	$7,386	$0
Monthly	$733,333	$817,270	$970,310	$830,908	$974,317	$577,947

Done | OOH-BRYANFE6420 | Insurance Company | Current | 11.0.0.00136

Tool: SoftWiki

- Research project for ‘agile‘ RE
- Uses WIKI approach for documentation of requirements
  - Central server, web-based interface
- Linking of requirements is based on an ontology
  - Links to documents and other requirements
  - Relations have defined semantic
  - Allows for logic reasoning

Fully Dressed Use Case Example (1/7)

[Larman]

- Use Case UC1: Process Sale

Scope: POS System; *Level:* User Goal

Primary Actor: Cashier

Stakeholders and Interest:

Cashier: Wants accurate, fast entry, and no payment errors, as cash drawer shortages are deducted from his/her salary.

Salesperson: Wants sales commission updated.

Customer: Wants purchase and fast service with minimal effort. Wants proof of purchase to support returns.

Company: Wants to accurately record transactions and satisfy customer interests. Wants to ensure that Payment Authorization Service payment receivables are recorded. Wants some fault tolerance to allow sales capture even if server components (e.g., remote credit validation) are unavailable. Wants automatic and fast update of accounting and inventory.

Government Tax Agencies: Want to collect tax from every sale. May be multiple agencies, such as national, state, and county.

Payment Authorization Service: Want to receive digital authorization requests in the correct format and protocol. Wants to accurately account for their payables to the store.

Preconditions: Cashier is identified and authenticated.

Success Guarantee (Postconditions): Sale is saved. Tax is correctly calculated. Accounting and Inventory are updated. Commissions recorded. Receipt is generated. Payment authorization approvals are recorded.

Fully Dressed Use Case Example (2/7)

Main Success Scenario:

1. Customer arrives at POS checkout with goods and/or services to purchase.
2. Cashier starts a new sale.
3. Cashier enters item identifier.
4. System records sale line item and presents item description, price, and running total. Price calculated from a set of price rules.

Cashier repeats steps 3-4 until indicates done.

5. System presents total with taxes calculated.
6. Cashier tells Customer the total, and asks for payment.
7. Customer pays and System handles payment.
8. System logs completed sale and sends sale and payment information to the external Accounting system (for accounting and commissions) and Inventory system (to update inventory).
9. System presents receipt.
10. Customer leaves with receipt and goods (if any).

Fully Dressed Use Case Example (3/7)

Extensions ***(Alternative flows):***

*a. At any time, System fails:

To support recovery and correct accounting, ensure all transaction sensitive state and events can be recovered from any step of the scenario.

1. Cashier restarts System, logs in, and requests recovery of prior state.
2. System reconstructs prior state.

2a. System detects anomalies preventing recovery:

1. System signals error to the Cashier, records the error, and enters a clean state.
2. Cashier starts a new sale.

3a. Invalid identifier:

1. System signals error and rejects entry.

3b. There are multiple of same item category and tracking unique item identity not important (e.g., 5 packages of veggie-burgers):

1. Cashier can enter item category identifier and the quantity.

3-6a: Customer asks Cashier to remove an item from the purchase:

1. Cashier enters item identifier for removal from sale.
2. System displays updated running total.

3-6b. Customer tells Cashier to cancel sale:

1. Cashier cancels sale on System.

3-6c. Cashier suspends the sale:

1. System records sale so that it is available for retrieval on any POS terminal.

Fully Dressed Use Case Example (4/7)

4a. The system generated item price is not wanted (e.g., Customer complained about something and is offered a lower price):

- 1. Cashier enters override price.
- 2. System presents new price.

5a. System detects failure to communicate with external tax calculation system service:

- 1. System restarts the service on the POS node, and continues.
  - 1a. System detects that the service does not restart.
    - 1. System signals error.
    - 2. Cashier may manually calculate and enter the tax, or cancel the sale.

5b. Customer says they are eligible for a discount (e.g., employee, preferred customer):

- 1. Cashier signals discount request.
- 2. Cashier enters Customer identification.
- 3. System presents discount total, based on discount rules.

5c. Customer says they have credit in their account, to apply to the sale:

- 1. Cashier signals credit request.
- 2. Cashier enters Customer identification.
- 3. Systems applies credit up to price=0, and reduces remaining credit.

6a. Customer says they intended to pay by cash but don’t have enough cash:

- 1a. Customer uses an alternate payment method.
- 1b. Customer tells Cashier to cancel sale. Cashier cancels sale on System.

Fully Dressed Use Case Example (5/7)

7a. Paying by cash:

1. Cashier enters the cash amount tendered.
2. System presents the balance due, and releases the cash drawer.
3. Cashier deposits cash tendered and returns balance in cash to Customer.
4. System records the cash payment.

7b. Paying by credit:

1. Customer enters their credit account information.
2. System sends payment authorization request to an external Payment authorization Service System, and requests payment approval.
   - 2a. System detects failure to collaborate with external system:
     1. System signals error to Cashier.
     2. Cashier asks Customer for alternate payment.
3. System receives payment approval and signals approval to Cashier.
   - 3a. System receives payment denial:
     1. System signals denial to Cashier.
     2. Cashier asks Customer for alternate payment.
4. System records the credit payment, which includes the payment approval.
5. System presents credit payment signature input mechanism.
6. Cashier asks Customer for a credit payment signature. Customer enters signature.

Fully Dressed Use Case Example (6/7)

7c. Paying by check…

7d. Paying by debit…

7e. Customer presents coupons:

1. Before handling payment, Cashier records each coupon and System reduces price as
   appropriate. System records the used coupons for accounting reasons.
   - 1a. Coupon entered is not for any purchased item:
     1. System signals error to Cashier.

9a. There are product rebates:

1. System presents the rebate forms and rebate receipts for each item with a rebate.

9b. Customer requests gift receipt (no prices visible):

1. Cashier requests gift receipt and System presents it.

Fully Dressed Use Case Example (7/7)

Special Requirements:

- Touch screen UI on a large flat panel monitor. Text must be visible from 1 meter.
- Credit authorization response within 30 seconds 90 % of the time.
- Somehow, we want robust recovery when access to remote services such the inventory system is failing.
- Language internationalization on the text displayed.
- Pluggable business rules to be insertable at steps 3 and 7.
- …

Technology and Data Variations List:

3a. Item identifier entered by bar code laser scanner (if code is present) or keyboard.

3b. Item identifier may be any UPC, EAN, JAN, or SKU coding scheme.

7a. Credit account information entered by card reader or keyboard.

7b. Credit payment signature captured on paper receipt. but within two years, we predict many customers will want digital signature capture.

Frequency of Occurrence: Could be nearly continuous.

Open Issues:

- What are the tax law variations?
- Explore the remote service recovery issue.
- What customization is needed for different businesses?
- Must a cashier take their cash drawer when they log out?
- Can the customer directly use the card reader, or does the cashier have to do it?

Pass/Fail Tests for Use Cases (1/3)

[Cockburn]

- Answers should produce a „Yes“, some less important items are omitted
  - don‘t forget the boss and the coffee break test for user goal use cases

- Use Case Title
  - 1. Is it an active-verb goal-phrase that names the goal of the primary actor?
  - 2. Can the system deliver that goal?
- Scope
  - 4. Does the use case treat the system mentioned in the Scope as a black box? (The answer must be "Yes" if it is a system requirements document.)
  - 5. If the system in Scope is the system to be designed, do the designers have to design everything in it and nothing outside it?
- Level
  - 6. Does the use case content match the stated goal level?
- Primary Actor
  - 8. Does he/she/it have behavior?
  - 9. Does he/she/it have a goal against the SuD that is a service promise of the SuD
- Preconditions
  - 10. Are they mandatory, and can they be set in place by the SuD?
  - 11. Is it true that they are never checked in the use case?
    Stakeholders and interests
  - 12. Are they named and must the system satisfy their interests as stated? (usage varies by formality and tolerance).

Pass/Fail Tests for Use Cases (2/3)

- Minimal Guarantees
  - 13. Are all the stakeholder's interests protected?

- Success Guarantees
  - 14. Are all the stakeholder's interests satisfied?

- Main Success Scenario
  - 15. Does it have 3-9 steps?
  - 16. Does it run from trigger to delivery of the success guarantee?

- Each step in any scenario
  - 18. is it phrased as a goal that succeeds?
  - 19. Does the process move distinctly foward after its successful completion?
  - 20. Is it clear which actor is operating the goal--who is "kicking the ball"?
  - 21. Is the intent of the actor clear?
  - 22. Is the goal level of the step lower than the goal level of the overall use case? Is it, preferably, just a bit below the goal level?
  - 23. Are you sure the step does not describe the user interface design of the system?
  - 24. Is it clear what information is being passed in the step?
  - 25. Does it "validate," as opposed to "check" a condition?

Pass/Fail Tests for Use Cases (3/3)

- Extension Conditions
  - 26. Can and must the system both detect and handle it?
  - 27. Is it what the system actually needs?

- Technology and Data Variation List
  - 28. Are you sure this is not an ordinary behavioral extension to the Main Success Scenario?

- Overall use Case Content
  - 29. To the sponsors and users: "Is this what you want?„
  - 30. To the sponsors and users: "Will you be able to tell, upon delivery, whether you got this?„
  - 31. To the developers: "Can you implement this?"