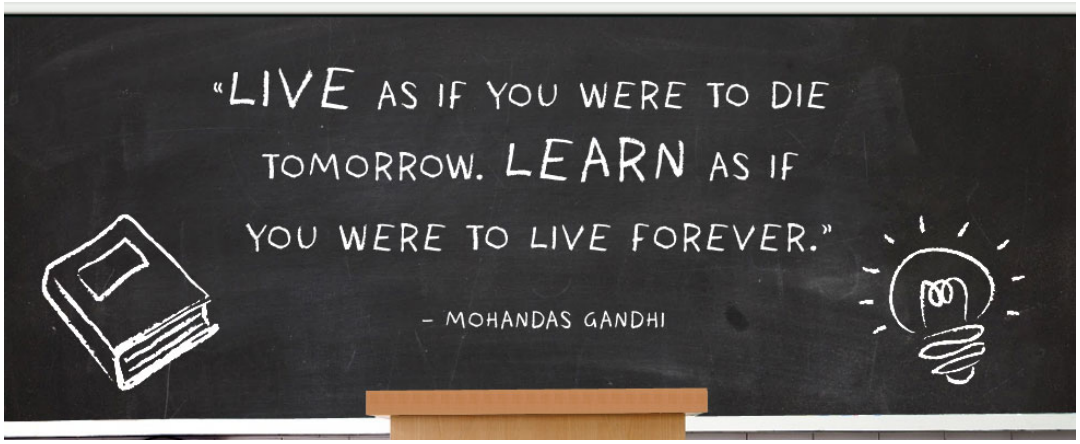
# Vorlesung Softwaretechnik
# SS 2019 / Teil II

Prof. Dr.-Ing. Samuel Kounev
M.Sc. Johannes Grohmann



# 2. Requirements

Prof. Dr.-Ing. Samuel Kounev

# Acknowledgements


- Lecture materials from Prof. Dr. Oliver Hummel (University of Mannheim) have been used as a basis

- Further lecture materials that have been used
  - Craig Larman
  - Prof. Walter Tichy (KIT)

# Requirements

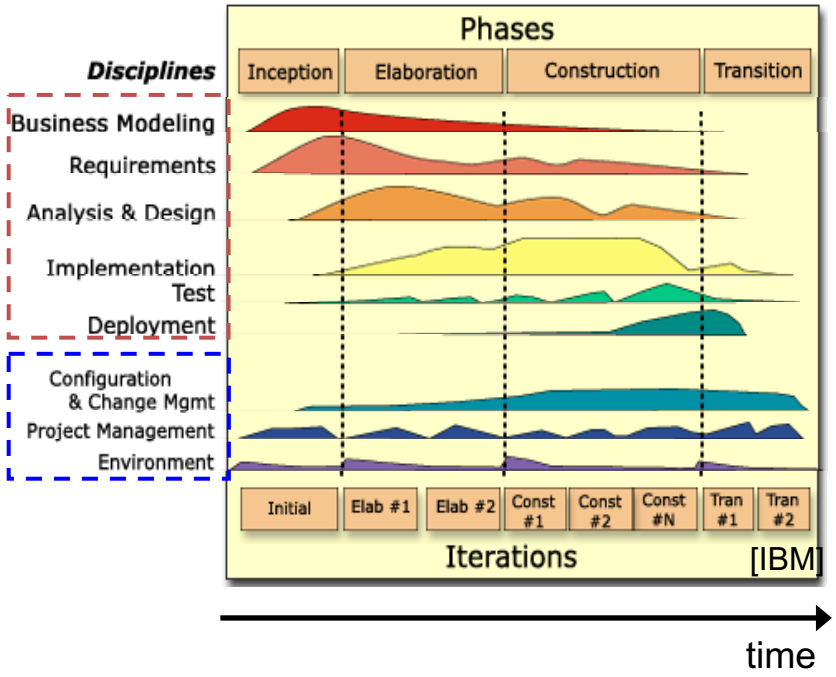
*„The indispensable first step to getting the things you want out of life is this: decide what you want“.*

-- Ben Stein


[Neshan Naltchayan, Wikipedia]

# Roadmap

1. What are software requirements?
   - Functional vs. non-functional
2. How to capture requirements?
   - Requirements artifacts
   - Use case model



# Requirements

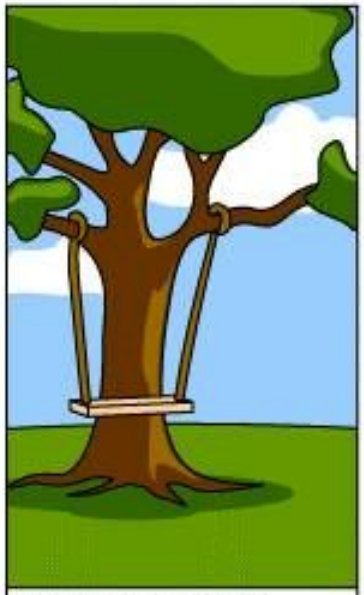
- ~60% of software defects are coming from incorrect requirements
  [Boehm, 1981]
  - *Requirements are missing*
  - *Requirements are wrong or missunderstood*
- Solving these problems during later phases is very expensive
  - Boehm estimates a **factor of ten per development phase**

---

- *You might want to listen to the*
  - *Christof Ebert's three biggest risks in requirements engineering*
    - wrong requirements
    - missing requirements
    - changing requirements

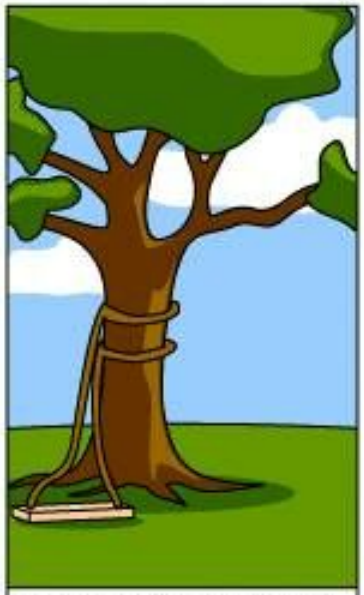

*http://www.se-radio.net/2008/10/episode-114-christof-ebert-on-requirements-engineering*

How the customer explained it



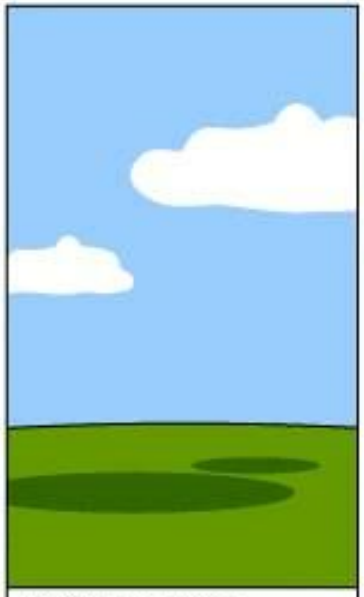
How the Project Leader understood it

How the Analyst designed it

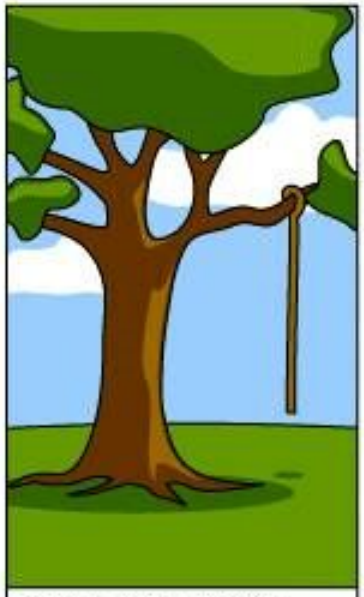


How the Programmer wrote it

How the Business Consultant described it

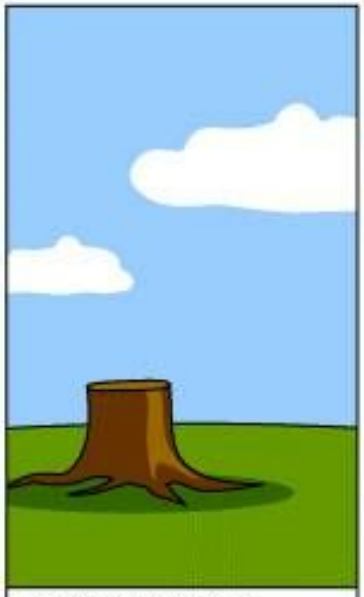


How the project was documented



What operations installed

How the customer was billed



How it was supported

What the customer really needed

# What are Requirements?

***Requirement***:

1) *A condition or capability needed by a user to solve a problem or achieve an objective.*

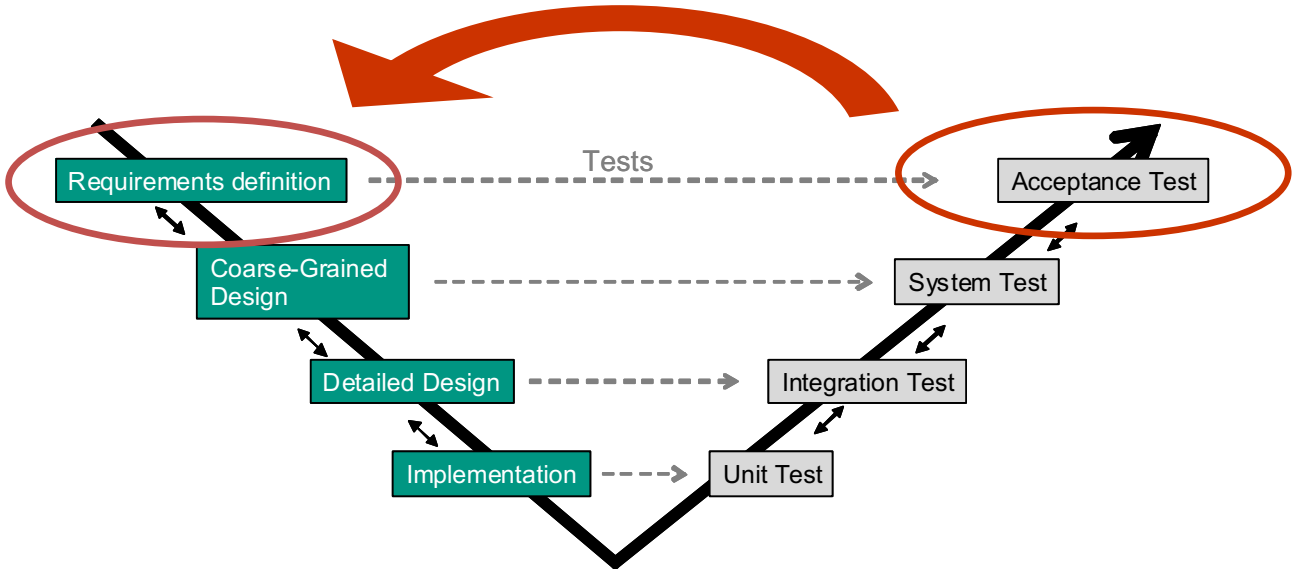
2) *A condition or capability that must be met or possessed by a system or system component to satisfy a contract, standard, specification, or other formally imposed documents.*

3) *A documented representation of a condition or capability as in 1) or 2).*

[IEEE 610.12-1990]

# Software Requirements Specification (SRS)

- Both starting point and end point for a software dev. project
- Probably the most important artifact in a project
- Ideally they should be
  1. clear and concise
  2. unambiguous
  3. consistent
  4. complete
  5. verifiable
  6. prioritized
  7. traceable



# The Requirements Engineering (RE) Process

- Cooperative, iterative and incremental process of –

1. **Requirements elicitation**
   - documentation of goals
   - scenarios
   - problem analysis
2. **Requirements documentation**
   - create Software Requirements Specification (SRS)
3. **Cross-cutting actions**
   - requirements **validation**
   - requirements **management**

---

➔ Identify **all** relevant requirements in necessary degree of detail

➔ Have the requirements acknowledged by the involved stakeholders

# Stakeholder

- A person or organisation that (in)directly **influences the requirements**
  - users of the system
    - labor union?
  - operator of the system
  - purchaser / sponsor / controller
  - software developer
  - software architect
  - tester

- Identification of relevant stakeholders and their relationships is critical for RE success
  - **missing stakeholders may lead to missing requirements**

- Also take care of potential political interests of stakeholders!

# The Requirements Engineer

- **Central role** in the development process
  - translates between users and developers
  - sometimes also called business analyst

- Is responsible for requirements **elicitation** and **documentation**
  - maintains the requirements document as well

- Needs **methodological skills**
  - thinks analytically
  - has empathy
  - has communication skills
  - is conflict solving
  - has discussion moderation skills
  - is cogent (überzeugend)

# Functional Requirements

- … statements of services the system should provide
  1. How the system should react to particular inputs
  2. How the system should behave in particular situations

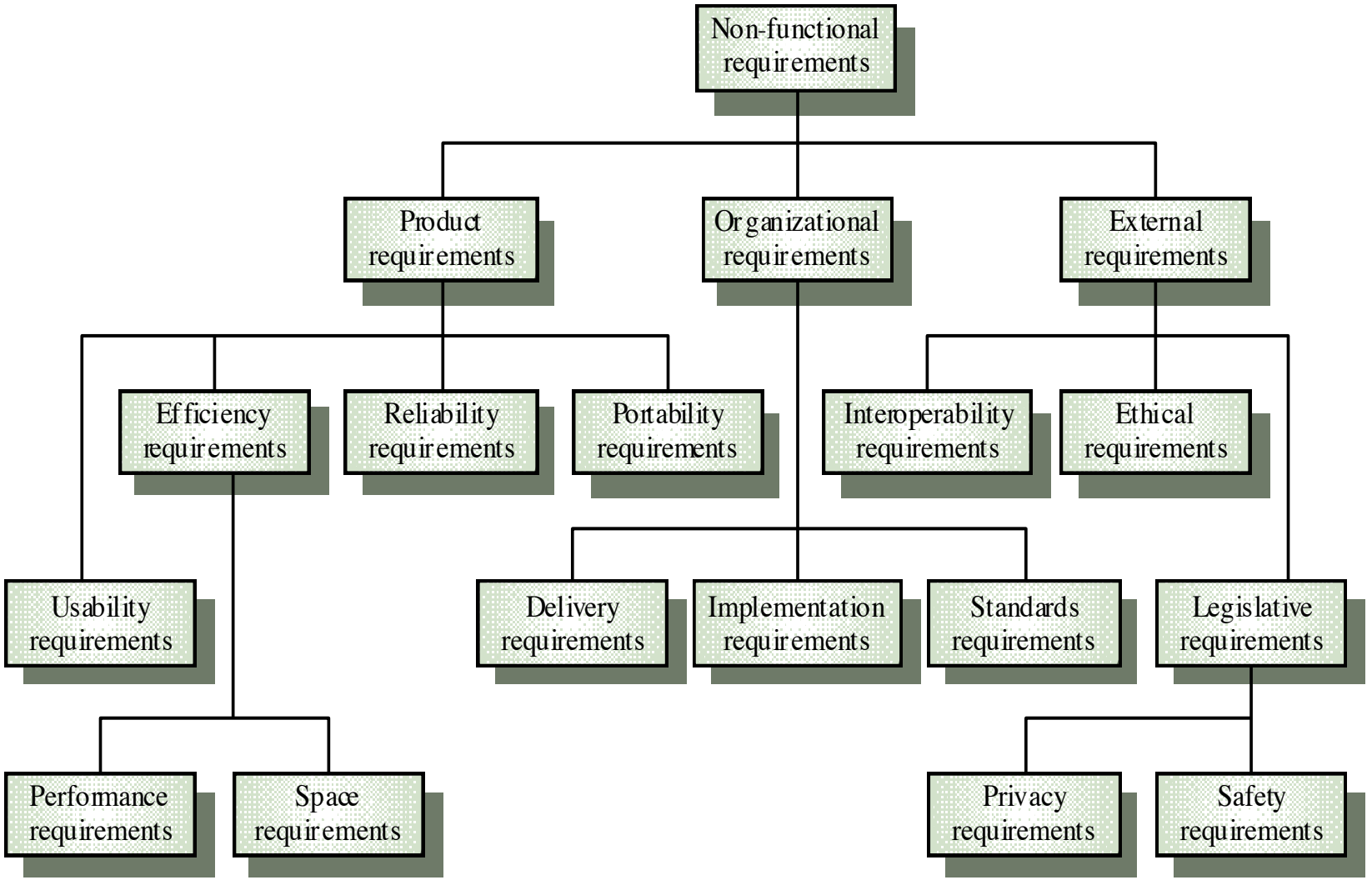
- … depend on the type of software, expected users, and the environment in which the software is used

- Examples
  - The user **shall** be able to search either all of the initial set of databases or select a subset from it
  - The system **shall** provide viewers for the user to read documents in the document store

# Non-functional Requirements


- Constraints on the services offered by the system –
  - e.g., timing constraints
- Process requirements may also be specified mandating a particular –
  - environment, programming language, or dev. method
  - fulfilment of particular standards
- Examples
  - The system shall respond to user requests within 2 seconds
  - The system dev. process and deliverable documents shall conform to the process and deliverables defined in XYZCo-SP-STAN-95

# Non-functional Requirement Types




[Sommerville]

# UP's FURPS+ Requirements Categorization

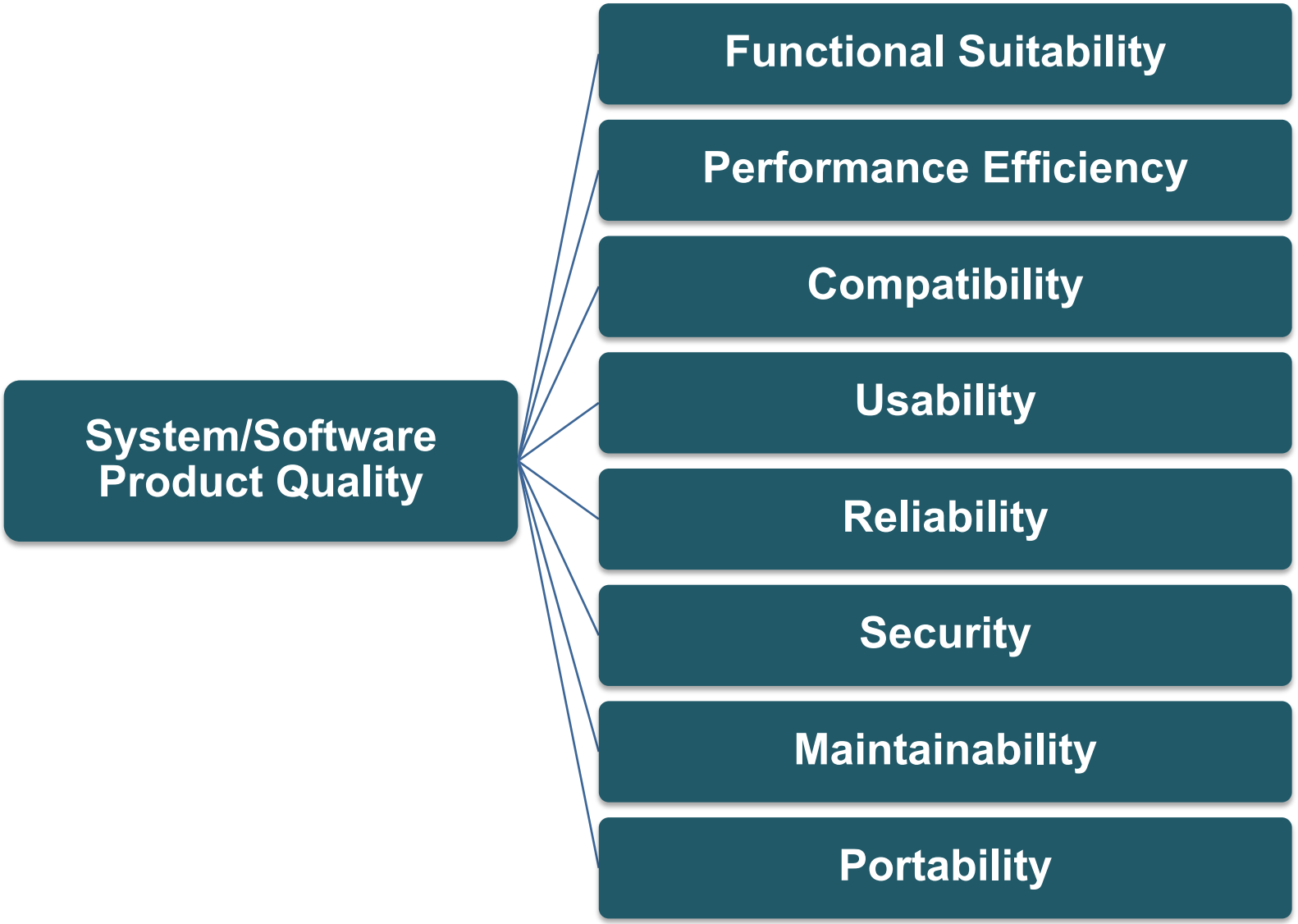
- **Functional** – features, capabilities, security
- **Usability** – human factors, help, documentation
- **Reliability** – frequency of failure, recoverability, predictability
- **Performance** – response times, throughput, accuracy, resource usage
- **Supportability** – adaptability, maintainability, internationalization, configurability

\+ *ancillary and sub-factors, such as*

- **Implementation** – resource limitations, languages and tools, hardware,…
- **Interface** – constraints imposed by interfacing with external systems
- **Operations** – system management in its operational setting
- **Packaging** – for example, a physical box
- **Legal** – licensing and so forth

**Quality requirements („-ilities“)**: usability, reliability, performance, …

# Software Quality per ISO/IEC 25010:2011



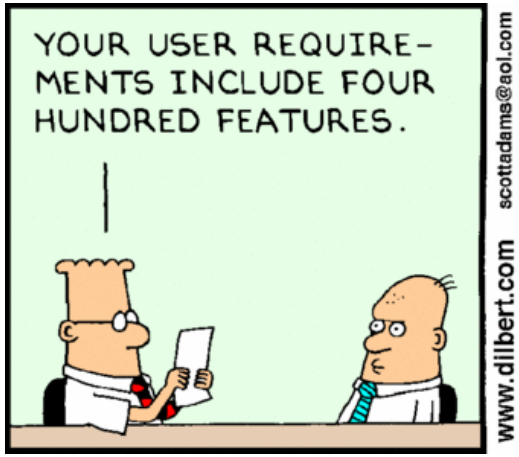
# Traditional Feature Lists

Traditional requirements analysis methods typically result in detailed, low-level feature lists

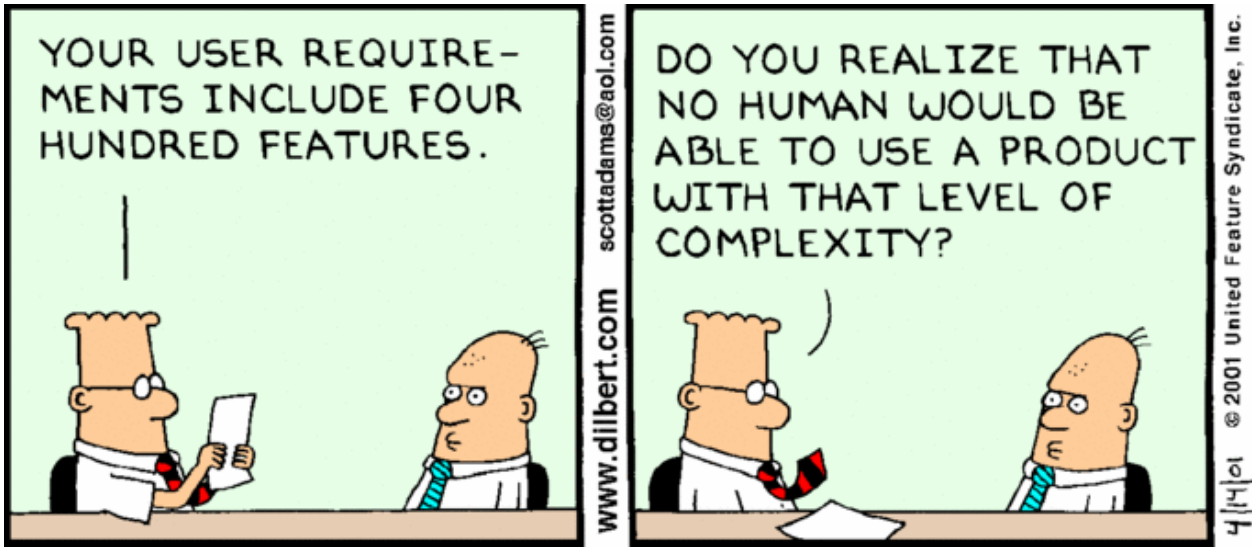
| ID | Feature |
|---|---|
| FEAT1.9 | The system shall accept entry of item identifiers |
| …. | |
| FEAT2.4 | The system shall log credit payments to the accounts receivable system |
| …. | |

Especially in the context of enterprise applications such long, detailed lists –

- do not describe requirements in a cohesive way
- have the feel of a “laundry list” (unstructured)



# The Dilbert Solution ☺

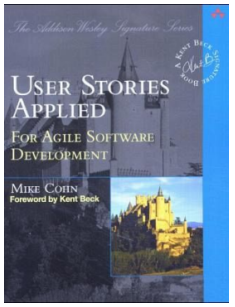




http://www.dilbert.com/

# Modern Requirements Capture

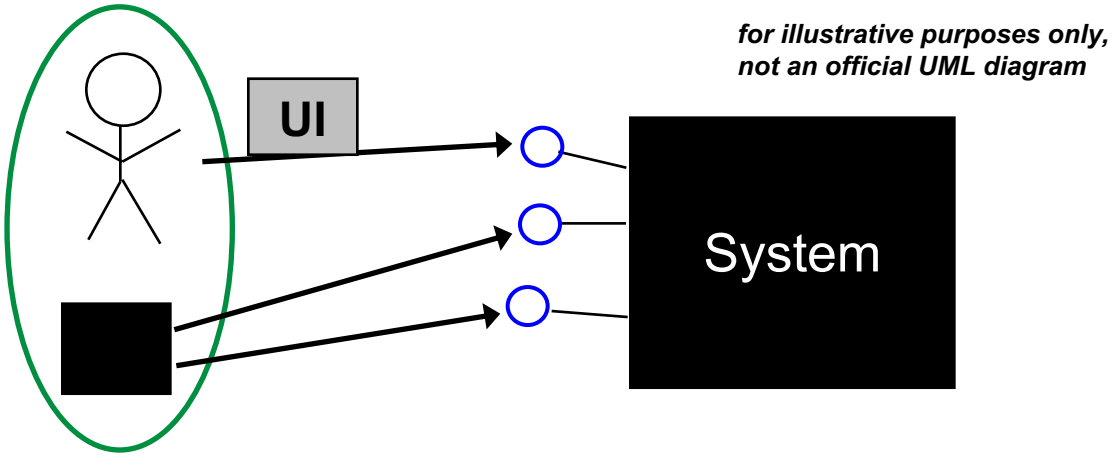
- Common approaches for specifying requirements are –
  - use cases and user stories
- Both focus on **interaction** of the user and the software
  - place requirements in the context of the stories and goals of the system
  - chief virtues are simplicity and utility
- Both are not always the most appropriate approach
  - some applications still call for a feature-driven approach
  - e.g., operating systems, database management systems




- Brief history
  - **Use cases** were introduced in 1986 by Ivar Jacobson [Jacobson92]
  - Fleshed out in “Writing Effective Use Cases” by A. Cockburn [Cockburn00]
  - **User stories** are the predominant approach in agile processes

# Use Cases

- A notation to capture an interaction of (typically) two parties
  - A story of how a system is used by a user in order to reach a goal
- The system is perceived as a **black-box** with which **users** (humans or other systems) interact
  - via well-defined interfaces called **system operations**
  - use cases describe **what** the system should do (**its responsibilities**) as opposed to **how** it should do it



# Use Case Terminology (1/3)

- **Actor**
  - an entity with behavior, e.g., a person (role), another computer system, an organization
  - typically an entity interacting with the ***system under discussion (SuD)*** with behavior outside of it
- **Primary Actor**
  - has user goals fulfilled through services of the SuD
- **Supporting Actor**
  - provides a service to the SuD, e.g., payment authorization service
- **Offstage Actor**
  - has an interest in the behavior of the SuD, but is not primary or supporting; e.g., a government tax agency

# Use Case Terminology (2/3)

- **Scenario** → a specific sequence of actions and interactions between actors and the SuD
  - a.k.a. (“also known as”) use case instance
  - one particular story of using the system

Cashier

Checkout Counter

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...

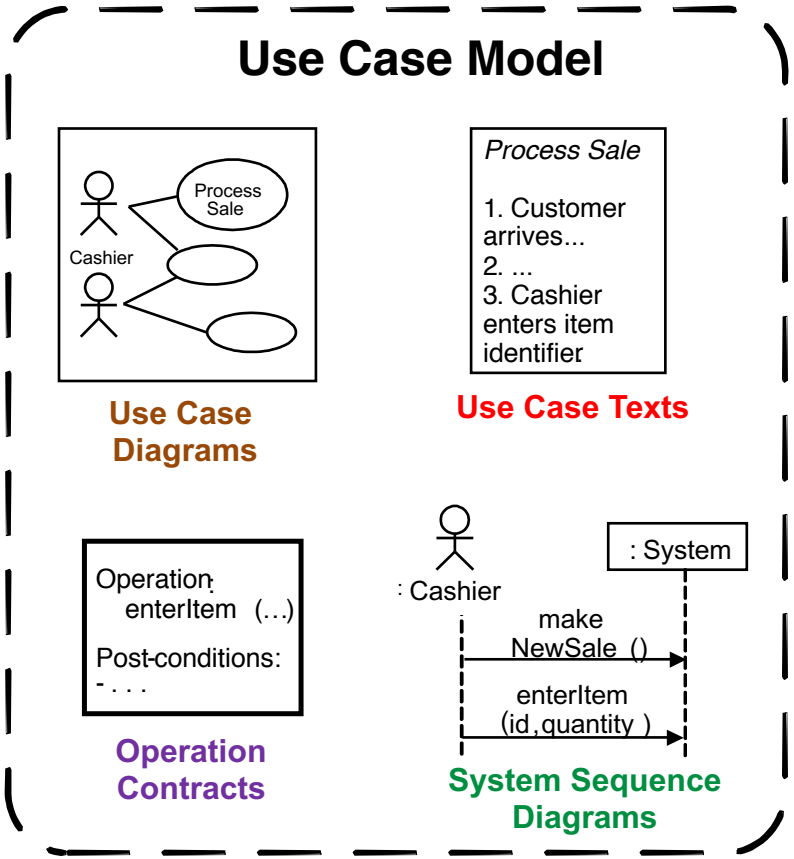
# Use Case Terminology (3/3)

- **Use Case**
  - a collection of related success and failure scenarios that describe actors using the SuD to support a goal
  - a use case is a set of use case instances, i.e., scenarios

- **Use Case Model**
  - the set of all use cases and related diagrams
  - may also include sequence or activity diagrams to illustrate flows

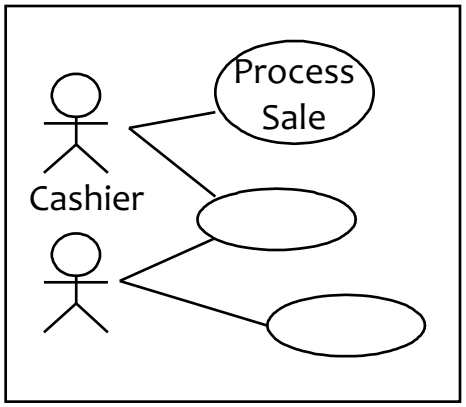
# UP's Use Case Model

- … comprises 4 essential elements closely related with each other
  1. Use Case Texts
  2. Use Case Diagrams
  3. System Sequence Diagrams
  4. Operation Contracts




- Some important remarks / guidelines
  - Requirements are captured by writing text → Use Case Texts
  - Diagrams are only used for illustration
  - Use Case Texts are merely a notation
    - …can be used for various purposes and on various abstraction levels

# UP's Use Case Model (1/2)




**Use Case Diagrams**

*Process Sale*

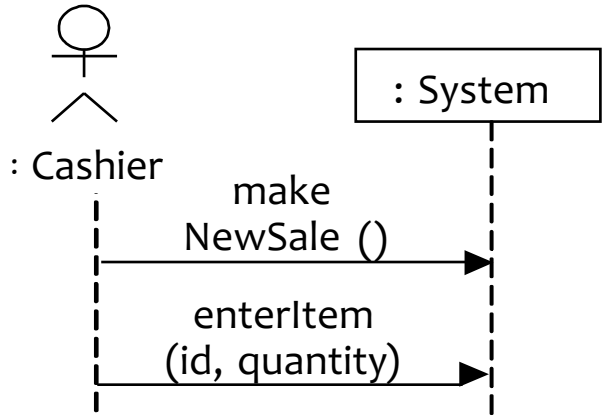
1. Customer arrives ...
2. ...
3. Cashier enters item identifier.

**Use Case Texts**

Operation :
enterItem (...)

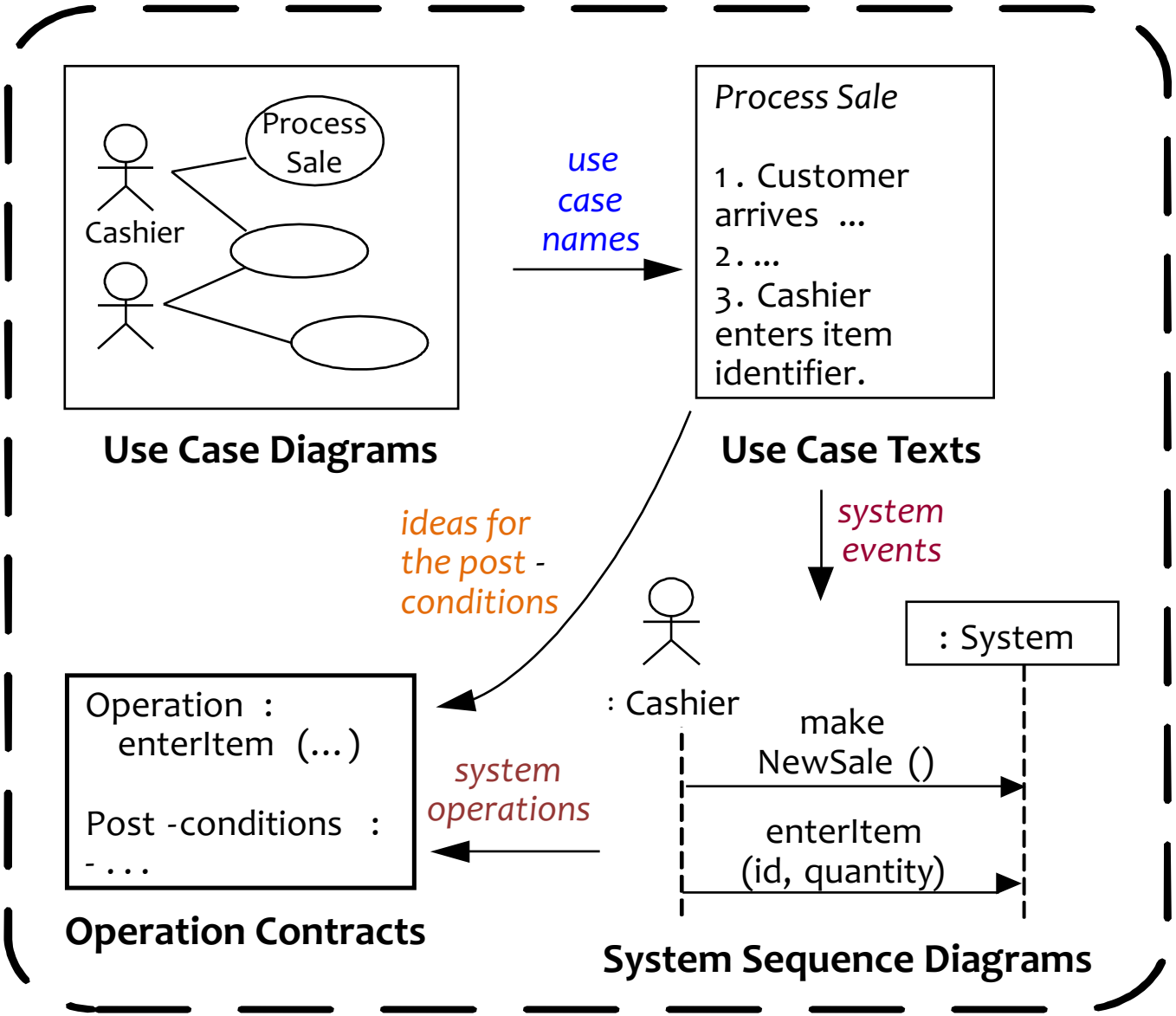
Post -conditions :
- ...

**Operation Contracts**




**System Sequence Diagrams**

# UP's Use Case Model (2/2)



# Further UP Requirements Artifacts

- **Supplementary Specification**
  - Everything not covered in the use cases
  - Non-functional requirements + features not expressed as use cases
- **Glossary**
  - Defines noteworthy terms
  - Encompasses a data dictionary (value ranges, etc.)
- **Vision**
  - Executive summary of the vision of the project
  - Summarizes the high-level requirements and business case
- **Business Rules** (a.k.a. Domain Rules)
  - General requirements or policies that transcend one project
  - E.g., government tax laws, company policies

# Sample UP Artifacts and Timing

| Discipline | Artifact | Incep. | Elab. | Const. | Trans. |
|---|---|---|---|---|---|
| | Iteration → | I1 | E1..En | C1..Cn | T1..T2 |
| Business Modeling | Domain Model | | s | | |
| Requirements | ***Use-Case Model*** | s | r | | |
| | ***Vision*** | s | r | | |
| | ***Supplementary Specification*** | s | r | | |
| | ***Glossary*** | s | r | | |
| | ***Business Rules*** | s | r | | |

s = start
r = refine

# Running Example: NextGen POS System

- **Point of Sale (POS) System**
  - records sales and handles payments
  - typically used in retail stores
  - includes hardware components such as computer and bar code scanner and software to run the system
  - interfaces to various services such as third party tax calculator

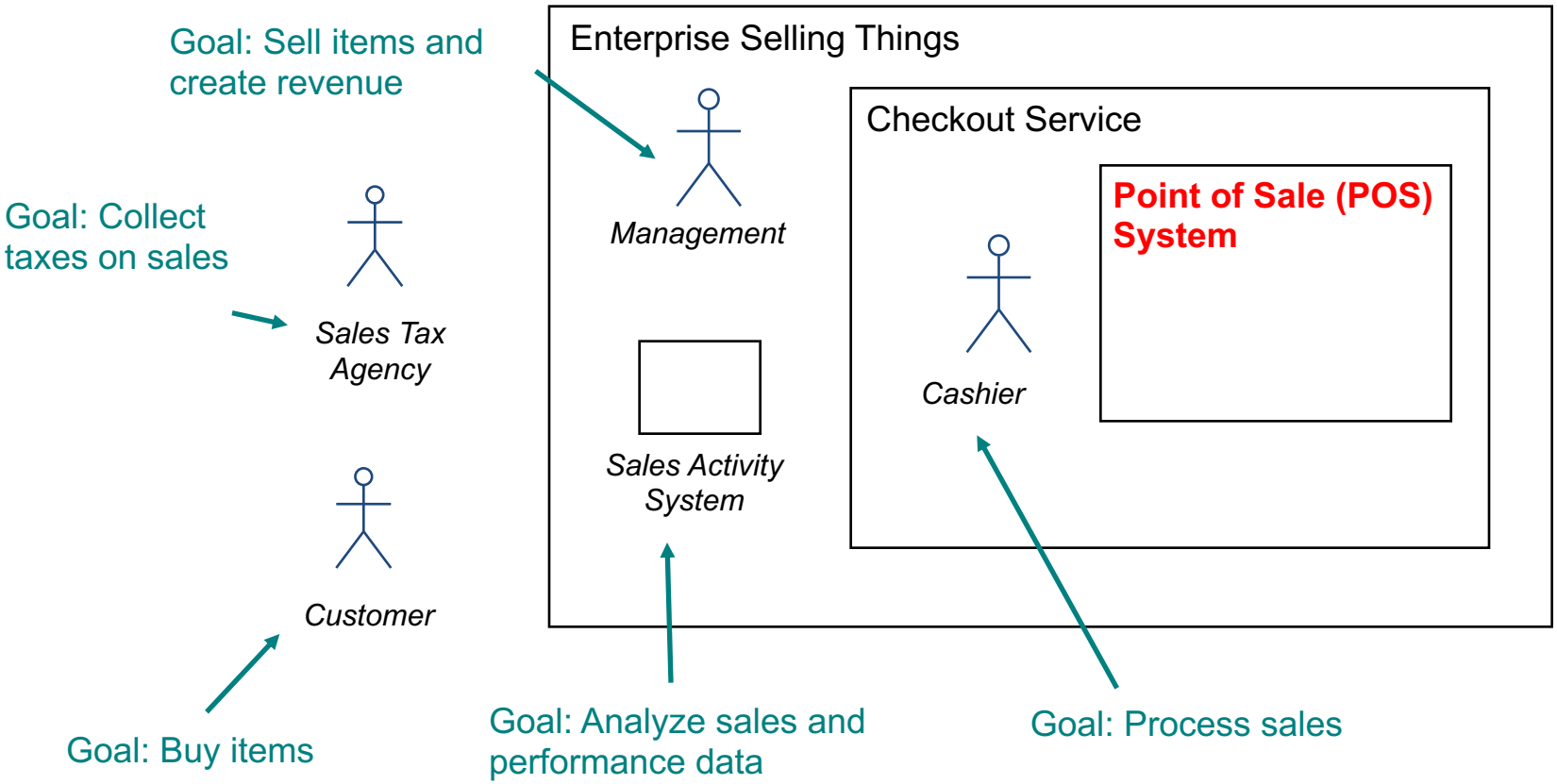
# System Context and Scope

- **System Boundary (Scope)**
  - border between the system under discussion and the environment
  - interface of the system to the environment
- **System Context**
  - part of the environment of a system relevant for definition and understanding of requirements
- **Context Boundary**
  - separates system context from irrelevant environment

*Most boundaries are usually unsharp in the beginning*

# Goals, Actors and Scopes

It is important to identify the boundary of the “system” you are modeling



# Black-box User Goal Use Cases

- Typically, the focus is on black-box **user goal use cases**
  - where one user is working with the system for limited amount of time to reach a specific business goal
- Other **goal levels** and modeling scopes are also feasible
- **Activities** can be grouped at many **levels of granularity**
  - from individual small steps to enterprise level activities
  - which of the following is a valid use case?
    - negotiate a supplier contract
    - handle returns
    - log-in
    - withdraw money
  - arguably, these are all use cases at different levels, depending on the system boundary, actors and goals

# Use Case Goal Levels [Cockburn]

1. **(High-Level) Summary**
   - groups together a number of user goal and sub-function use cases
   - often equals a **business process**

2. **User-Goal**
   - describes how a **user's goal** is reached by interacting with the system

3. **Sub(-function)**
   - used to factor out **"sub-goals" (sub-tasks)** required to achieve a goal
   - typically without a direct business value

4. **Too-low** (= feature or system operation)
   - everything **"smaller" than a sub-function** is considered "too low"
   - e.g., describing the internals of a login operation

# Elementary Business Process (EBP)

- A good rule of thumb is to focus on **elementary business processes** (EBPs) → usually form good **user-goal use cases**

- 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 steps within an EBP

# User Goal Use Case Heuristics

Two rules of thumb for identifying user-goal use cases

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 logging-in …*
     - *I have been searching customer data …*
     - *I have been creating customer accounts …*

2. **The coffee break test** [Cockburn]
   - finish a use case when you would 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?*

# Let‘s practice a bit …

- *Identity the right 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. verify credit card data in an ATM

High-Level Summary  User-Goal  Sub-Function  Too-Low

# Use Case Elaboration

1. **Choose the system boundary**
   - e.g., the *Point of Sale (PoS)* 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 level that represents an EBP
   - *process sale*, *print report* …

4. **Define use cases** that satisfy user goals
   - name them according to their goal in an imperative style

# Iterative Use Case Elaboration

- Fully elaborated uses cases are complex
  - hence it is common to refine them iteratively

- Use case formats (precision levels)
  1. name & brief main success scenario → *brief use cases*
  2. add extension conditions → *casual use cases*
  3. add extension handling steps + more → *fully dressed use cases*
     - usually written according to a standard template

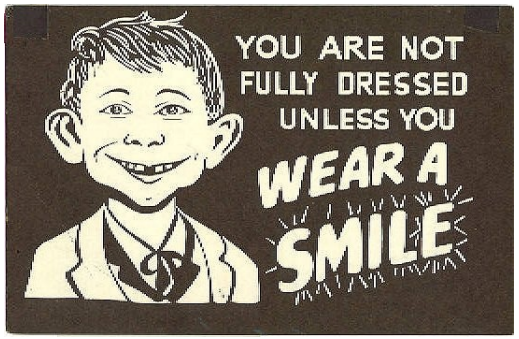
➔ You may need to extract or merge sub-use-cases afterwards

# Examples

1. **Brief Use Case**
   - 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.

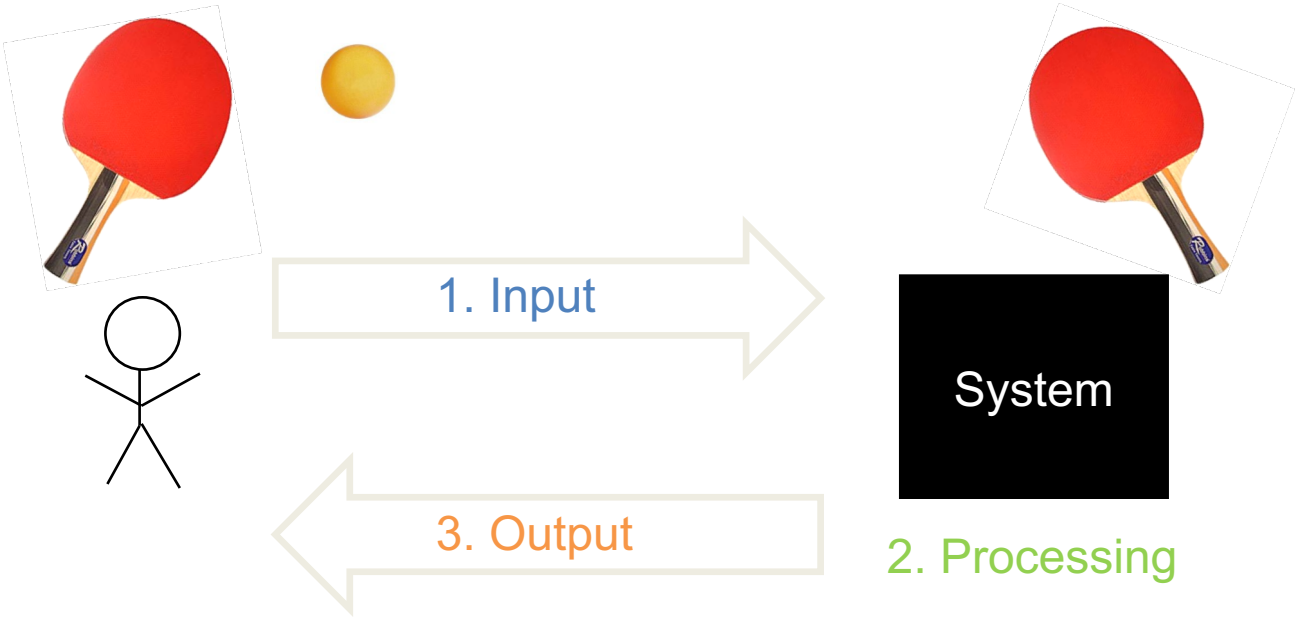
2. **Casual Use Case (additionally)**
   - ✚ an item identifier may be incorrectly recognized
   - ✚ the cashier may cancel the sale process
   - ✚ the cashier might need to remove an item from the sale
   - ✚ the customer may chose to pay by credit card …

3. **Fully Dressed Use Case**
   - see Appendix



# How to write Fully Dressed Use Cases

- User plays “ping pong” with the system



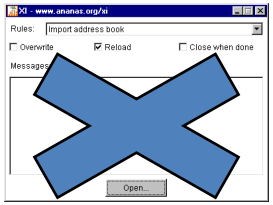

- Focus on **responsibilities**!

  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*

# Writing Guidelines

- 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
- Common practice: start actor names with uppercase letter
- Write use cases in essential style especially during early iterations
  - Essential style: No User Interface (UI) details in use cases
    - no clicking buttons or entering text into text fields, etc.
    - however, need to define the data exchanged
      - *such as PIN, account number, etc.*
  - Concrete style: UI decisions embedded in the use case text



# Further Guidelines

- Data descriptions
  1. Precision Level 1: Data nickname
     - item identifier
  2. Precision Level 2: Data fields associated with the nickname
     - item identifier : String
  3. Precision Level 3: Field types, lengths, and validations
     - item identifier : String, a 13 character EAN barcode, validations specified in Std 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…

# Use Case Relationships

- Three possible kinds of relationships between use cases
  - include, extend, generalize
  - but only include is widely used in practice

- ***Include*** is used for two main reasons
  - to factor out partial behavior common across several base use cases
    - e.g., paying by credit card appears in Process Sale and Process Rental
  - to break an overwhelmingly long use case into subunits
    - the base use case is aware and in control of the “sub use case call”

# Sub-function Include Example (1/2)

## UC1: Process Sale

***Main Success Scenario***

1. Customer arrives at POS checkout with goods and/or services to purchase

…

7. Customer pays and System handles payment

.....

***Extensions:***

7b. Paying by credit: *Include UC12: Handle Credit Payment*

7c. Paying by check: *Include UC13: Handle Check Payment*

## UC7: Process Rental

.....

***Extensions:***

6b. Paying by credit: *Include Handle Credit Payment*

# Sub-function Include Example (2/2)

## UC12: Subfunction Handle Credit Payment

….

***Level:*** Subfunction

***Main Success Scenario***

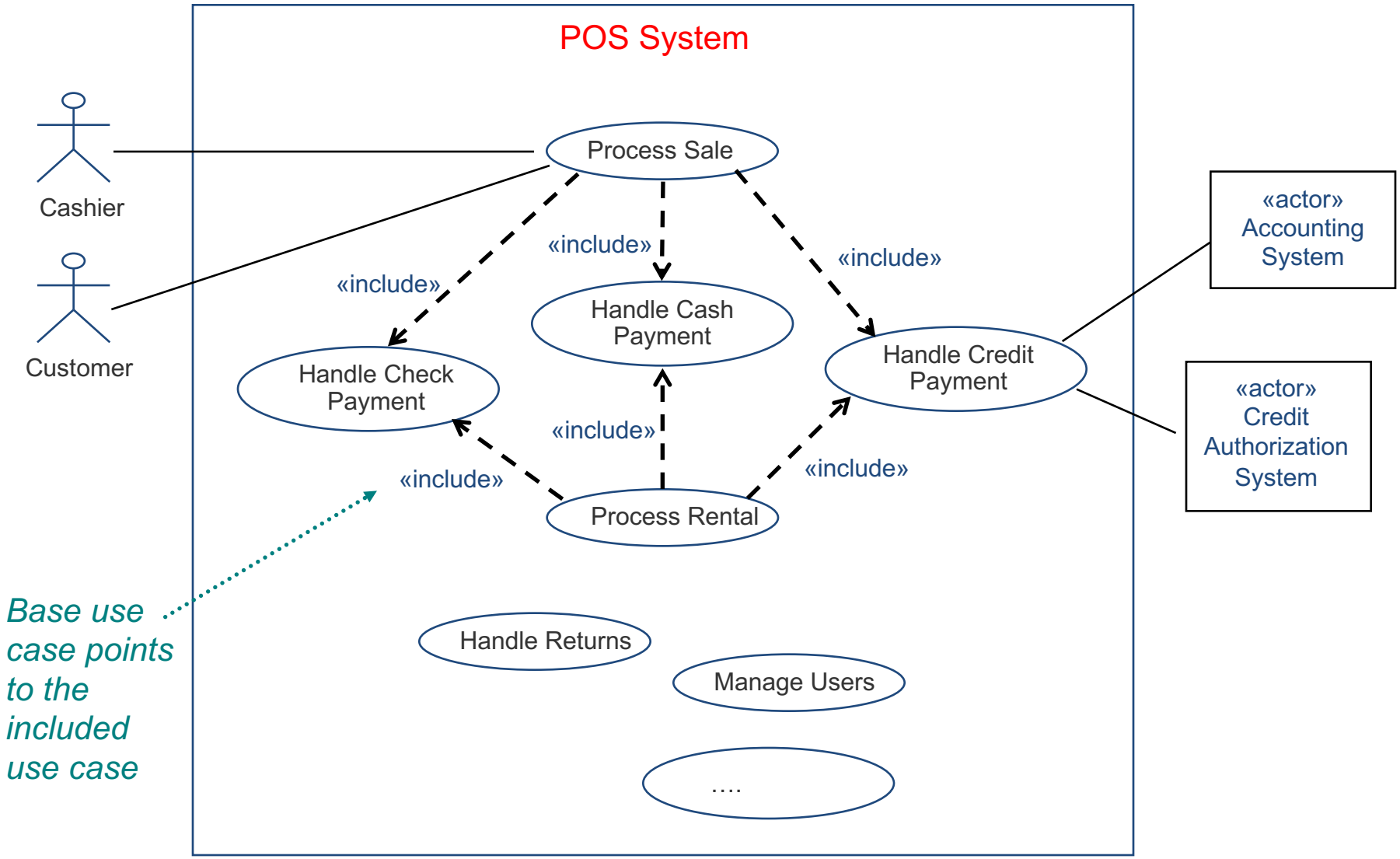
1. Customer enters their credit card information
2. System sends payment authorization request to an external payment Authorization System, and requests payment approval
2. System receives payment approval and signals approval to Cashier
3. ....

***Extensions:***

2a. System detects failure to collaborate with external system

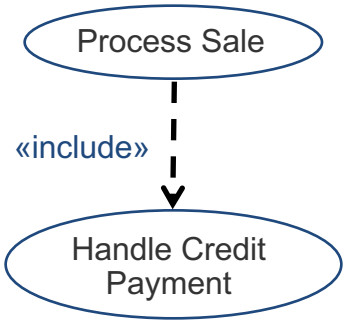
   1. System signals error to Cashier
   2. Cashier asks Customer for alternate payment

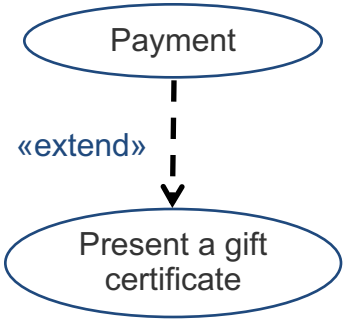
# “Include” in Use Case Diagrams



# Use Case Relationships

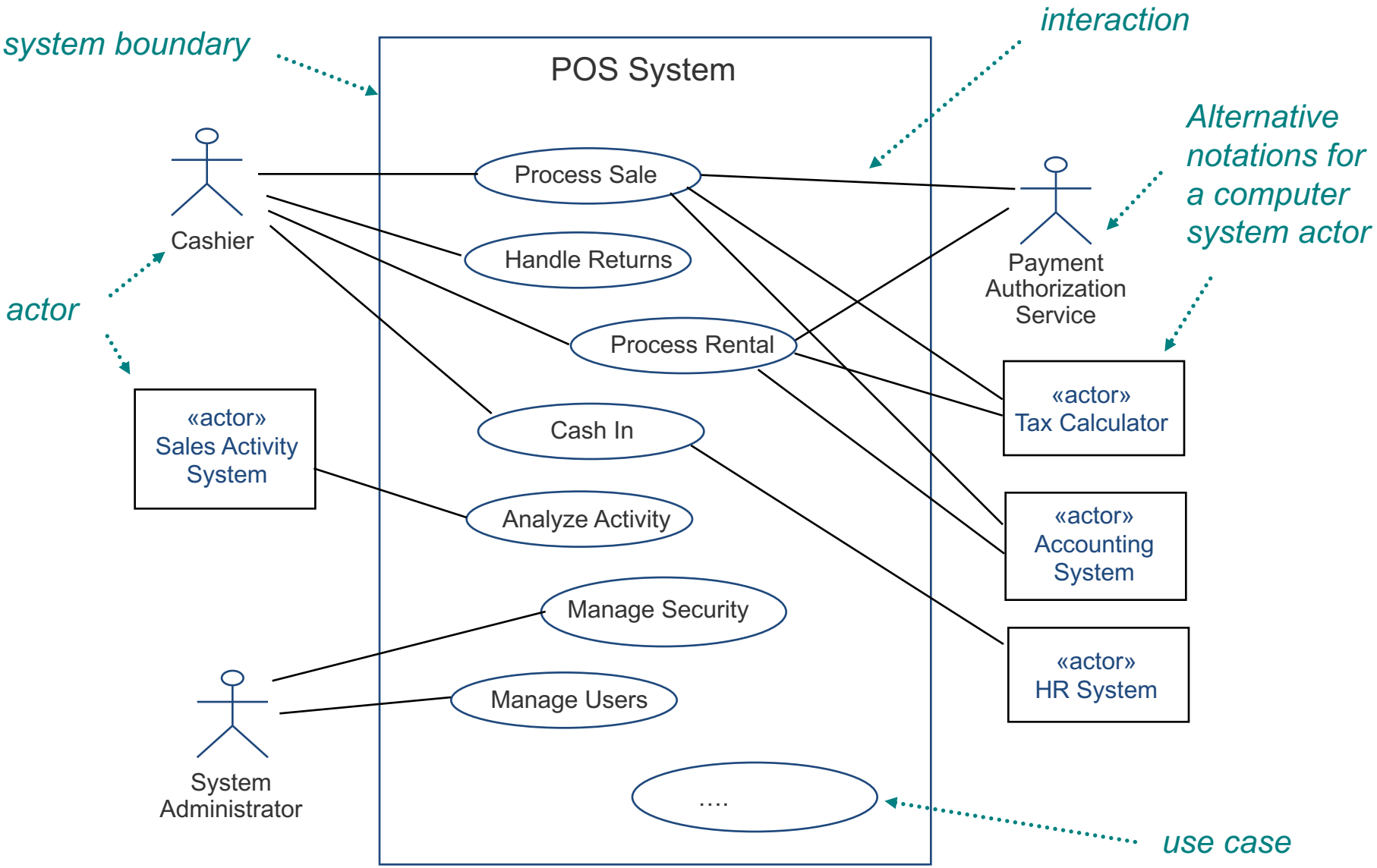
- ***Extend*** can be also used to factor out common behavior
  - however, it is typically used for external “interrupts”
    - e.g., if a customer presents a gift certificate during payment
- Difference between ***Include*** and ***Extend***



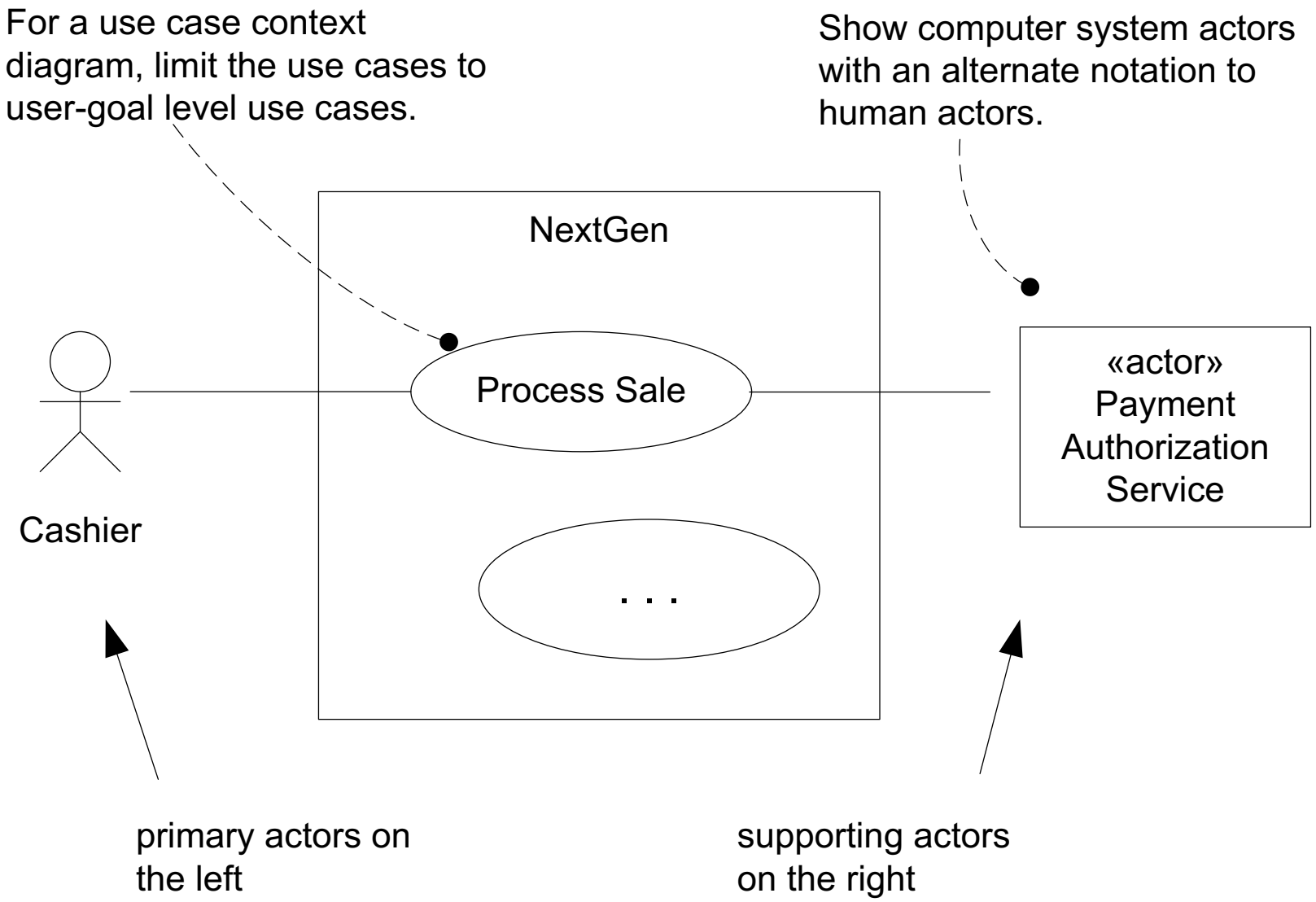




[See http://stackoverflow.com/questions/1696927/whats-is-the-difference-between-include-and-extend-in-use-case-diagram]

# Use Case Diagrams



# Diagramming Recommendations (1/2)



# Diagramming Recommendations (2/2)

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

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

# User Stories

- Collected on index cards
  - not suitable for UI requirements
- They describe requirements from the **end-user's point of view**
  1. a name (and ideally an ID)
  2. a brief textual description
  3. acceptance criteria
- The description should contain
  - the user's role
  - the goal of the story
  - optionally the benefit

**Front side**

US17: Create Meeting

As a base user, I need to create a new meeting, enter its name, room and purpose, its date as well as its start and end time.

Priority: high

**Back side**

Acceptance Criteria

- start time must not be before end time
- the meeting must be successfully saved
- the date must not be in the past

# Agile Requirements Elicitation

1. Product Owner initially populates the product backlog with **coarsely described requirements**
   - derived from the product concept and discussed with the customer
   - the goal is to quickly collect some requirements
2. Product Owner **clusters** the requirements according to their themes
3. Product Owner and team **prioritize** the requirements
   - usually according to business value and perhaps risk
4. Product Owner **refines** the requirements with the highest priorities
   - in requirements workshops with the customer and the team as far as possible
   - the product backlog should now contain enough information to start with the first sprint
5. The requirements are **updated, refined or even removed** as needed

# Priority: MoSCoW

[Минеева Ю., Wikipedia]

- … is the capital of Russia ☺
- … but also a way of prioritizing requirements

1. **M**UST have
   - a critical requirement that must be present in the product
2. **S**HOULD have
   - important, but not absolutely necessary requirement
3. **CO**ULD have
   - nice to have requirement that could increase customer satisfaction
4. **W**ON'T have / **W**OULD like
   - requirement of relatively low importance that still increases business value

[IEEE-830]: *Mandatory, Optional, Nice to have…*

# Priority: Cost Value Analysis

One dimensional priorization is often not sufficient

- contrastive pairs are often more helpful
- using value/importance vs. –
  - cost / time
  - risk
  - volatility

  etc…

| Requirement | Value | Risk | → Prio. |
|---|---|---|---|
| A | 4 | 2 | 2 |
| B | 3 | 3 | 1 |
| C | 3 | 2 | 1.5 |

# Software Requirements Specification (SRS)

- 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 standarised use case and document templates

- Documents all interfaces btw. the system and its environment
  - describes **what**, **not how**
    - system remains a **black-box**
    - BUT: design decisions taken by constraints can be included
      - e.g., choice of hardware platform

# SRS 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… → see 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

  $\Rightarrow$ Traceability

# RE Tools

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

# Summary

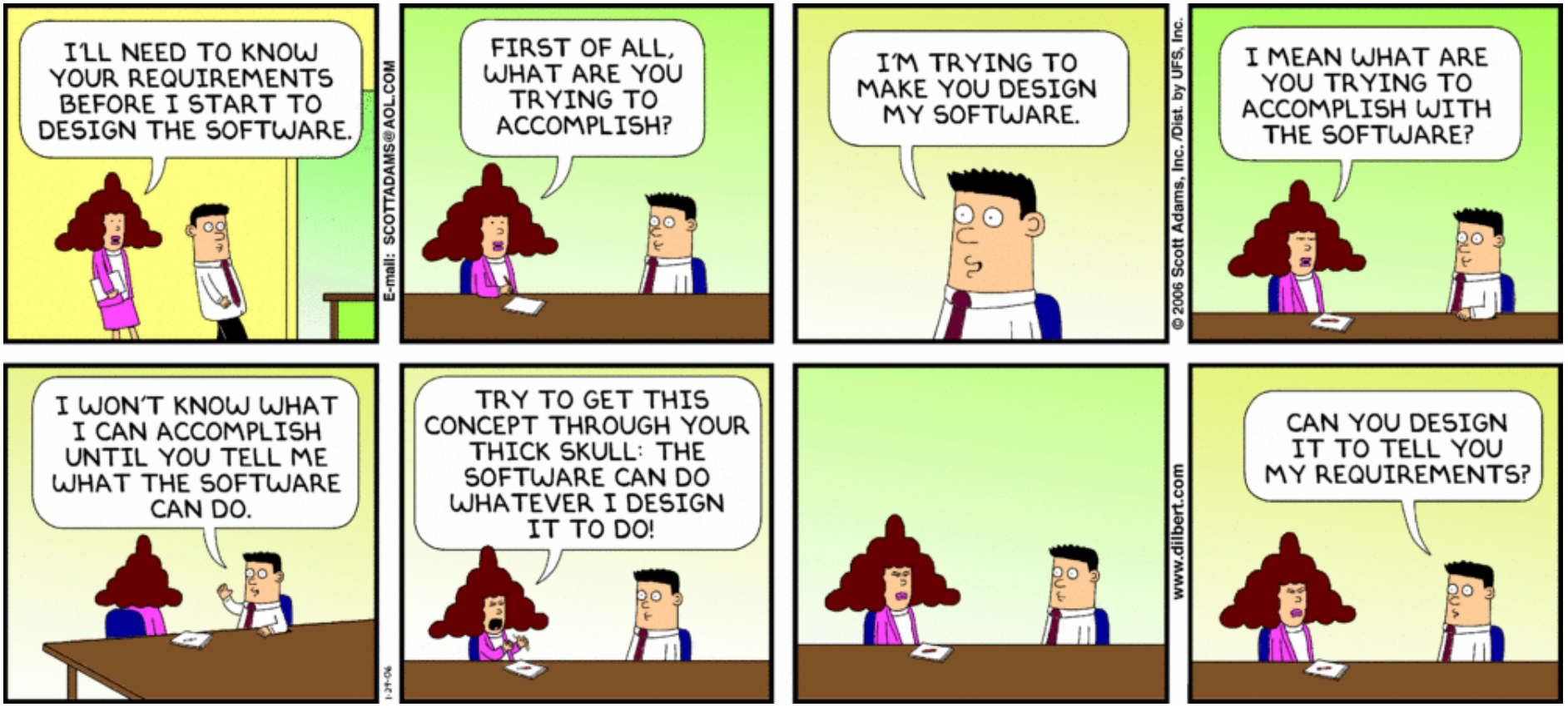
- The functional requirements to interactive software systems are typically captured in user-goal use cases
  - comprise the interactions of the system with its users
  - captured early in a brief format
  - refined during the development process

- The use case model is the foundation for a number of diagrams used to model the system
  - as we shall see in the next lectures

# Today‘s 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
- [Cohn04] M. Cohn
  *User Stories Applied: For Agile Software Development*
  Addison-Wesley, 2004
- [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
- [Heineman/Council01] G. Heineman, W. Council
  *Component-based Software Engineering*
  Addison-Wesley, 2001
- [Jacobson92] I. Jacobson et al.
  *Object-Oriented Software Engineering: A Use Case Driven Approach*
  Addison-Wesley, 1992
- Pohl, K. (2007), Requirements Engineering: Grundlagen, Prinzipien, Techniken, dpunkt, Heidelberg
- Patton, J. (2005) How You Slice it, Better Software.
  http://agileproductdesign.com/writing/how_you_slice_it.pdf
- Lamsweerde, A. V. (2001), Goal-oriented requirements engineering: A guided tour

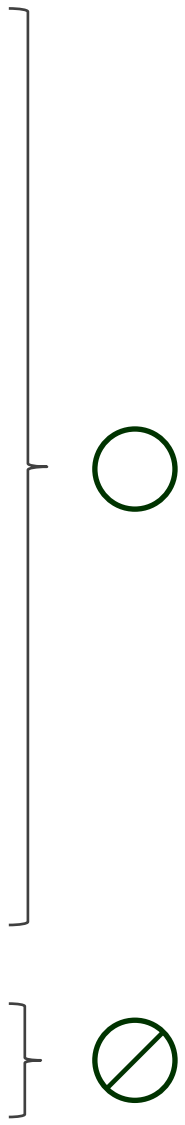
# Requirements Engineering



# Appendices

# Table of Contents

- Basic Writing Style Guidelines
- Examplary Requirements Templates
  - RUP (FURPS+)
  - IEEE
- Requirements Tools
  - DOORS
  - Requisite Pro
  - Requirements Composer
  - Caliber
  - SoftWiki
- Use Case Checklist
- Use Case Examples (Brief, Casual, Fully Dressed)

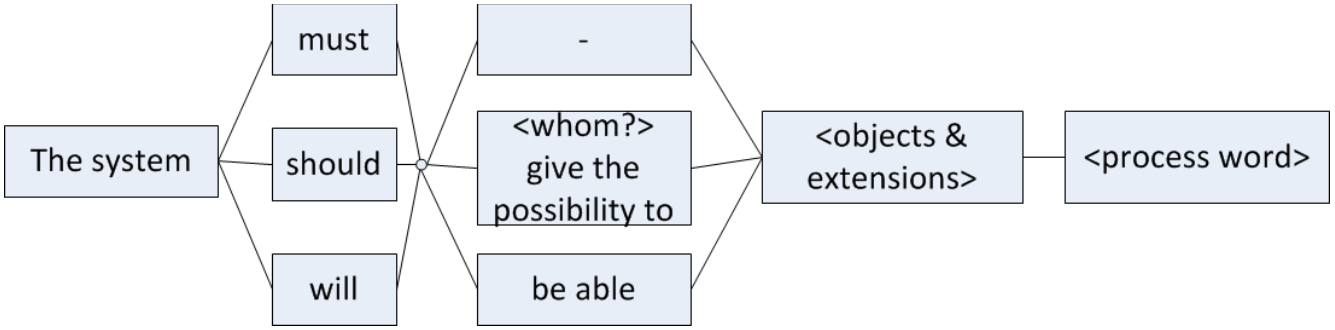


# Writing Style Guidelines (1/2)

- Short sentences, one clear requirement per sentence
  - *BAD: The Navigation system navigates to a destination with comfortable usability*
  - GOOD: The navigation system must allow in all modes to set the destination
- Active language that makes clear who is responsible for what
  - *BAD: When the PIN was validated, money can be withdrawn from the account*
  - GOOD: When the ATM has validated the PIN, the customer can withdraw money from his account
- Avoid „weak“ words, such as –
  - *effective, user-friendly, easy, quickly, timely, reliable, appropriate*
- Maintain a glossary of terms

# Writing Style Guidelines (2/2)

- Use sentence templates, such as –




- or: [Trigger] Actor Action Object [Condition] [Intel]
  - Trigger: *When the Navigation System is set into on road mode*
  - Actor: *the Navigation System*
  - Action: *displays*
  - Object: *the distance to the destination*
  - Condition: *until another mode is chosen.*

- *or others…* *see e.g. [Ebert 2008, p. 148]*

# 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 a
   - 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**

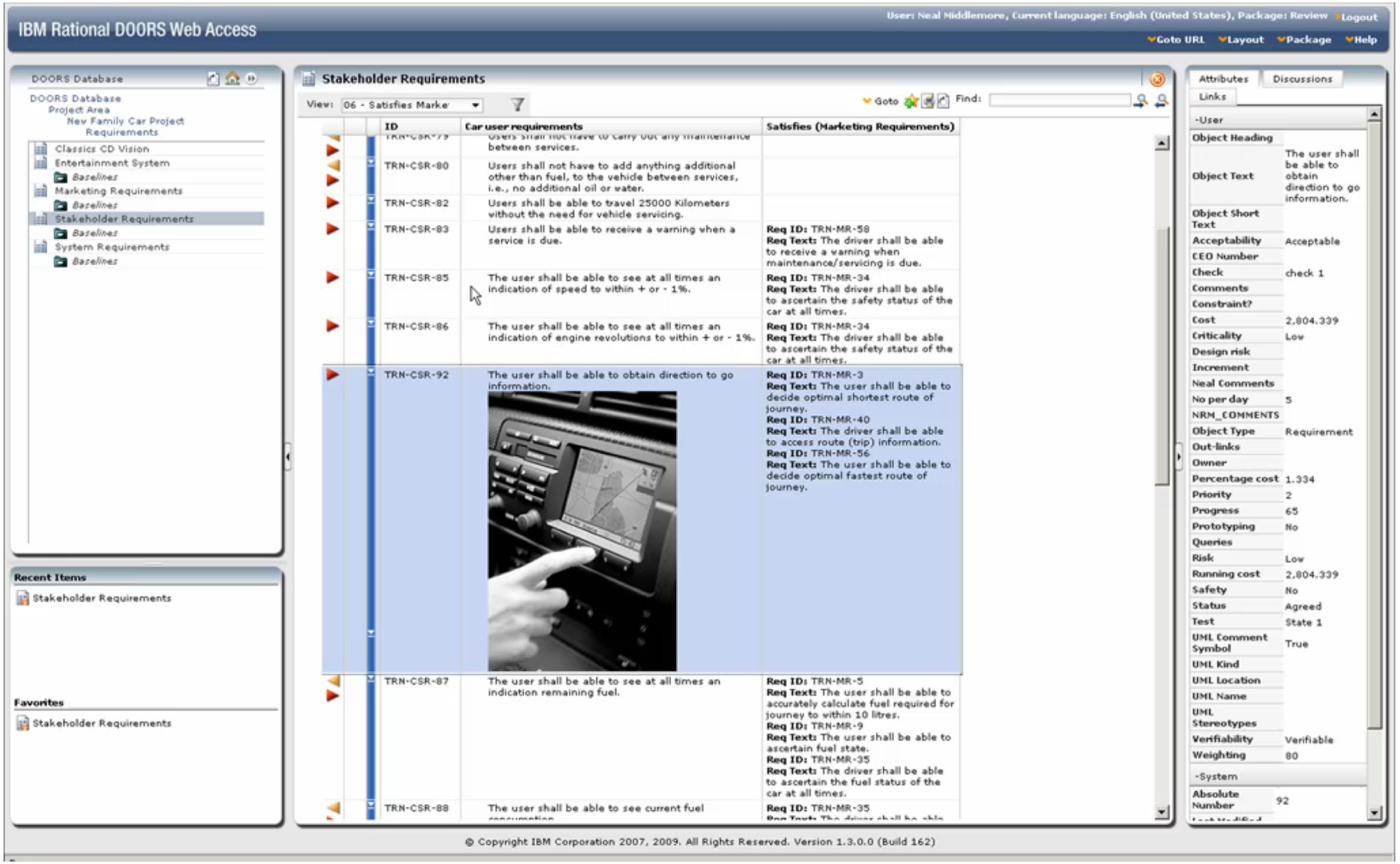
# IEEE SRS Document Structure [IEEE 830-1998]

- **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

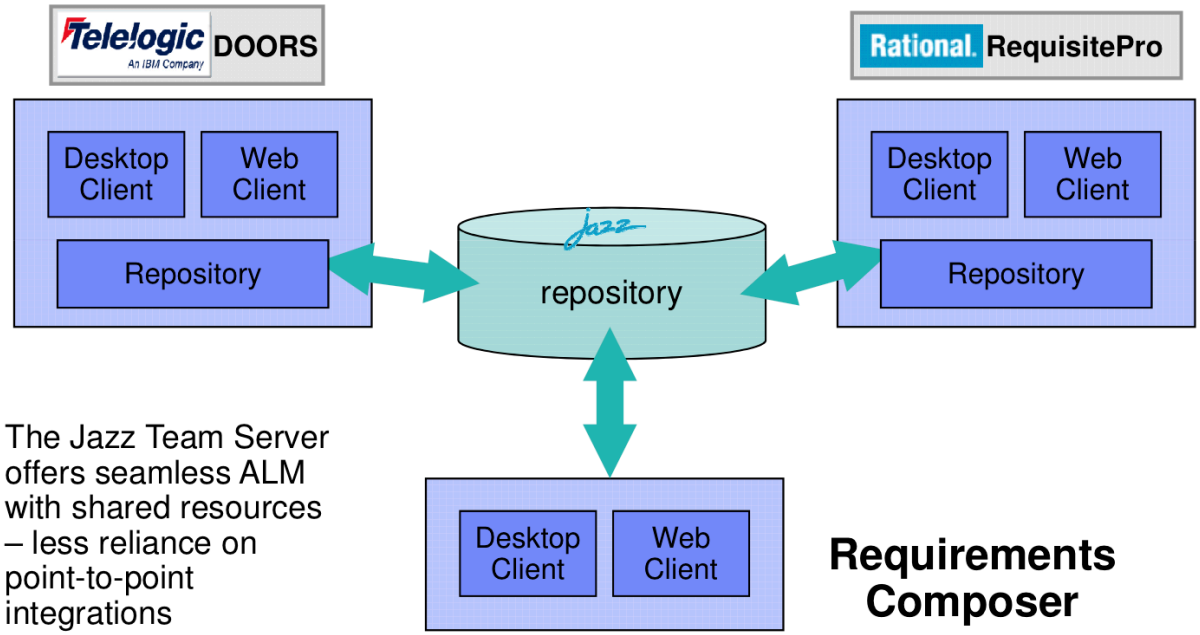
- Formely know as Telelogic DOORS
- Dedicated to requirements change management
  - Supports custom requirements processes / workflows
- Server based
- Reviewer web client
- Integrated with many other development tools
  - Rational Requirements Composer, etc.
- Can generate requirements documents

# DOORS Web Access

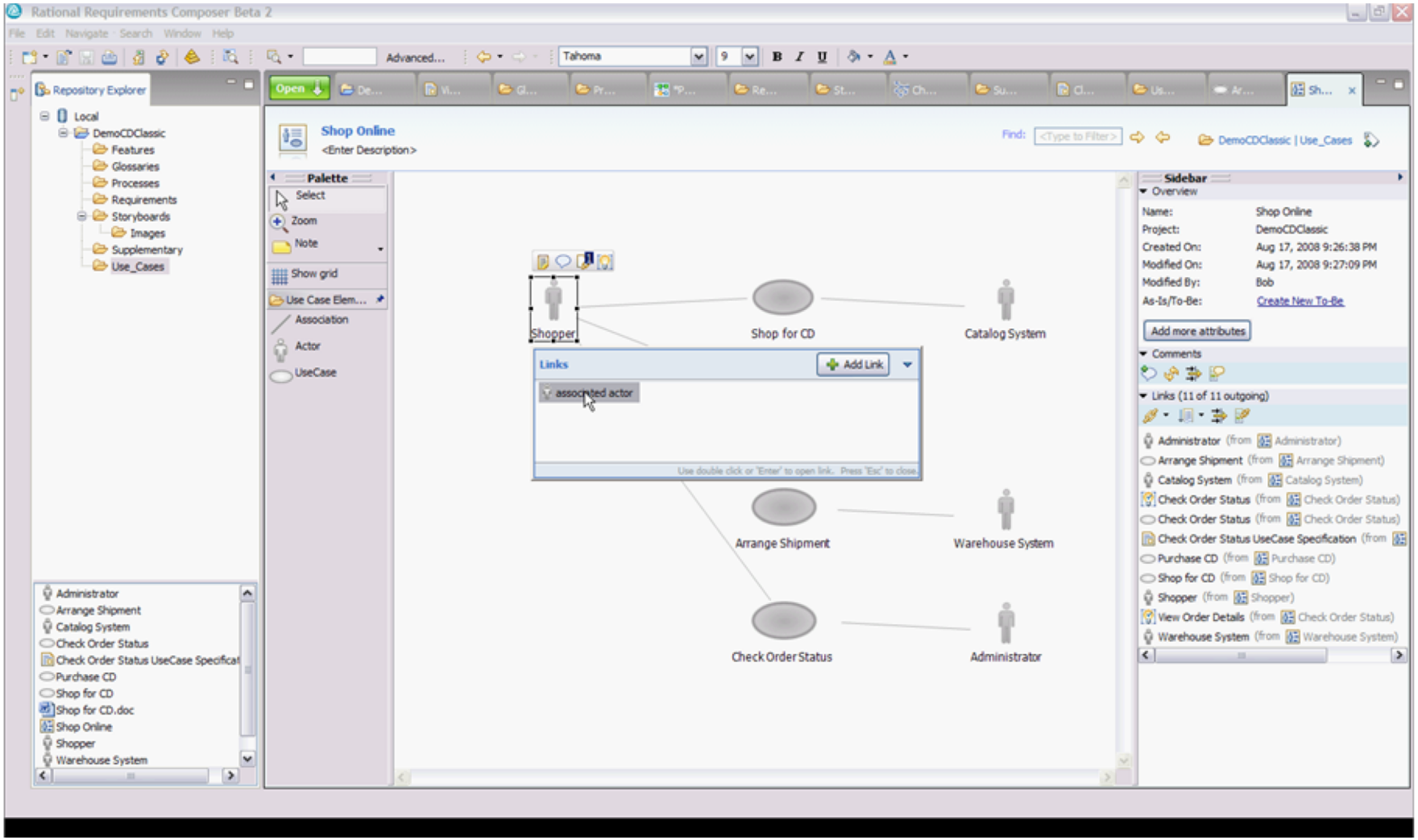


# Tool: IBM Requisite Pro

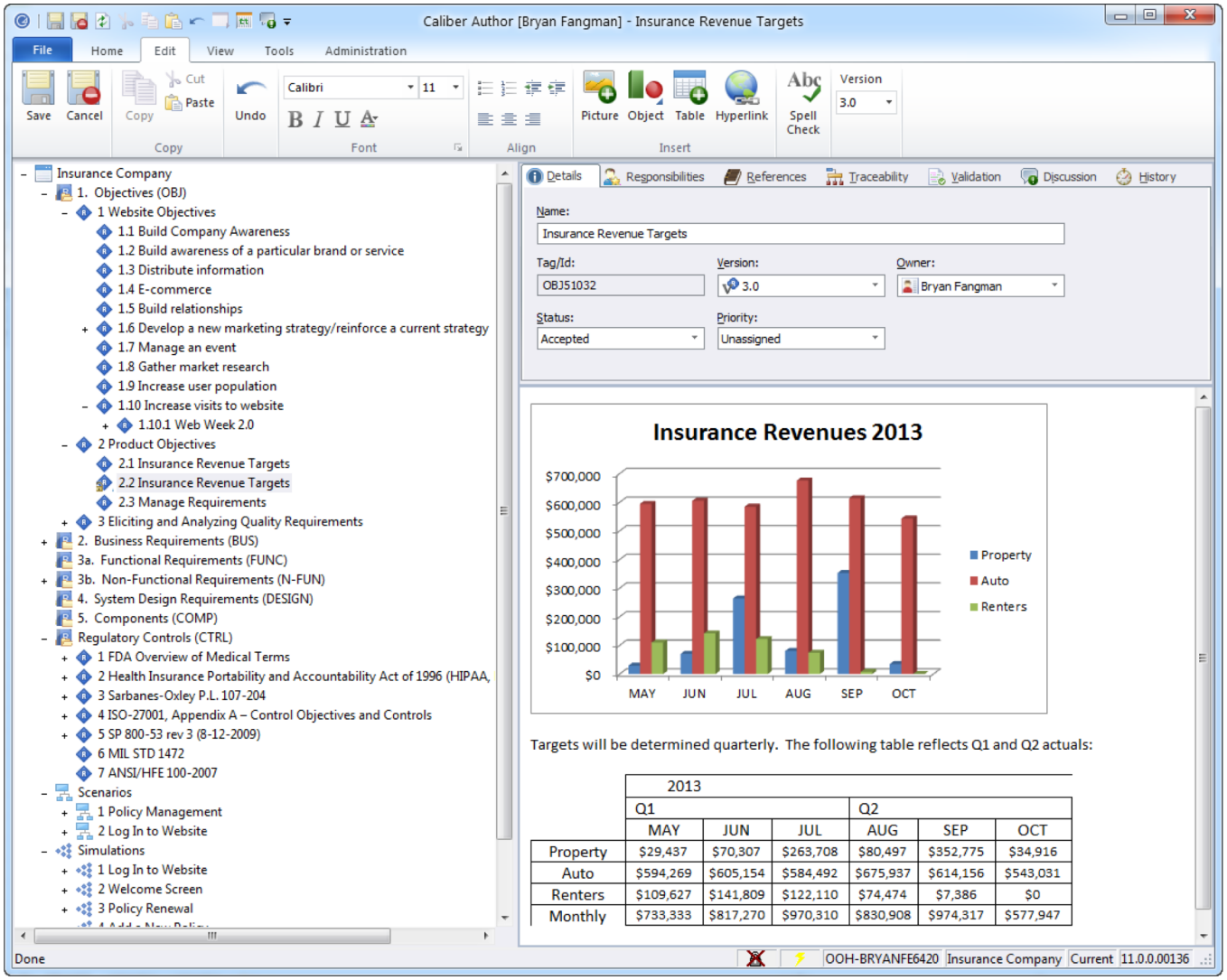
- Very similar to Doors
  - Both are #1 and #2 on the market
- Offers full web client



# Tool: Requirements Composer



# Tool: Borland Caliber



# 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

# 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?"

# Use Case Formats / Precision Levels [Larman]

- **Brief Use Cases**
  - Terse one paragraph summary, usually of the main success scenario
  - Used early in the development process to get an overview
- **Casual Use Cases**
  - Informal paragraph format
  - Multiple paragraphs that cover various scenarios
  - Added when requirements are analyzed in more detail
- **Fully Dressed Use Cases**
  - Most elaborate
    - All steps and variations written in detail
    - Supporting sections such as preconditions and success guarantees
  - Help provide a deep understanding of the goals, tasks and requirements
  - Usually written according to a standard template
    - E.g., the one by Cockburn http://alistair.cockburn.us/Basic+use+case+template

# Example of Brief Format Use Case

- Process Sale

  *"A customer arrives at a checkout point with items to purchase. The cashier uses the POS to record each purchased item. The system presents a running total and line-item details. The cashier enters payment information, which the system validates and records. The system updates inventory. The customer receives a receipt from the system and then leaves with the items."*

- Essence is discovering and recording functional requirements by writing stories of using a system to help fulfill stakeholder's goals

- Similar to **user stories** in the agile world [Cohn04]
  - *"As a cashier I want to checkout customers so that they pay their purchased items and the store inventory is updated."*
  - User stories are typically less elaborate
    - and merely considered being a reminder for requirements
    - often written to the following template:
      *As a (role) I want (something) so that (benefit).*

# Example of Casual Format Use Case

## Process Sale

- **Main Success Scenario**
  1. *A customer arrives at a checkout point with items to purchase.*
  2. *The cashier uses the POS to record each purchased item.*
  3. *The system presents a running total and line-item details.*
  4. *The cashier enters payment information, which the system validates and records.*
  5. *The system updates inventory.*
  6. *The customer receives a receipt from the system and then leaves with the items.*
- **Alternative Scenarios**
  1. *If the credit authorization is rejected, inform the customer and ask for alternative payment method*
  2. *If the item identifier is not found in the system, notify the Cashier and suggest manual entry of the identifier code (perhaps it is corrupted)*
  3. *If the system detected failure to communicate with the external tax calculator system, …*

# Fully Dressed Use Case Sections (1/4)

## 1. Preface elements

- many optional preface elements are possible
  - only important elements should be placed here (e.g., scope and goal level)
- also common is the primary actor element
  - identifies the principal actor that calls upon system services to fulfill a goal

## 1. Stakeholders and interest list

- suggests and bounds what the system must do
- provides the originating source for each responsibility

## 2. 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
- E.g., 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: should be written as something that can be detected
     - Handling: 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
  - e.g., a constraint imposed by a stakeholder concerning 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

# Fully Dressed Use Case Example (1/7)

## 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 (Basic Flow):***

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.

***Miscellaneous (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?