Karlsruhe Institute of Technology

Software Engineering II

Prof. Dr. Raffaela Mirandola

Topic 14
Use Cases

SASIS – SELF-ADAPTIVE SOFTWARE-INTENSIVE SYSTEMS
KASTEL – INSTITUTE OF INFORMATION SECURITY AND DEPENDABILITY

sasis.kastel.kit.edu

Overview on Topic

Content today

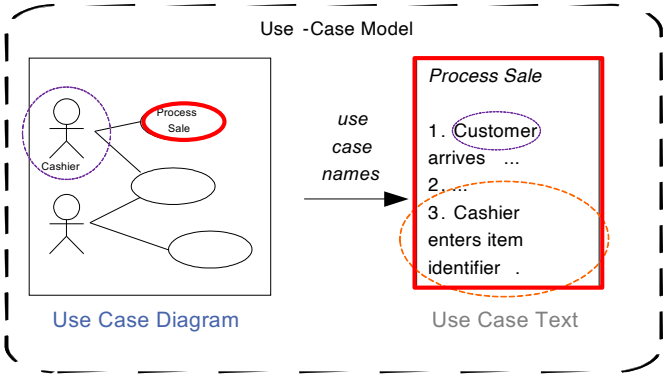
- Model-Based Requirements Elicitation
  - Use Cases and Goal Levels
  - Fully Dressed Use Cases

Learning Goals – Be able to

- understand the principles of model-based requirements capturing approaches
- understanding different levels of abstraction
- find the “right” level of abstraction
- describe the purpose and the elements of Use Case Models
- classify use cases according to their level and goal
- to create use case diagrams and fully dressed use cases

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

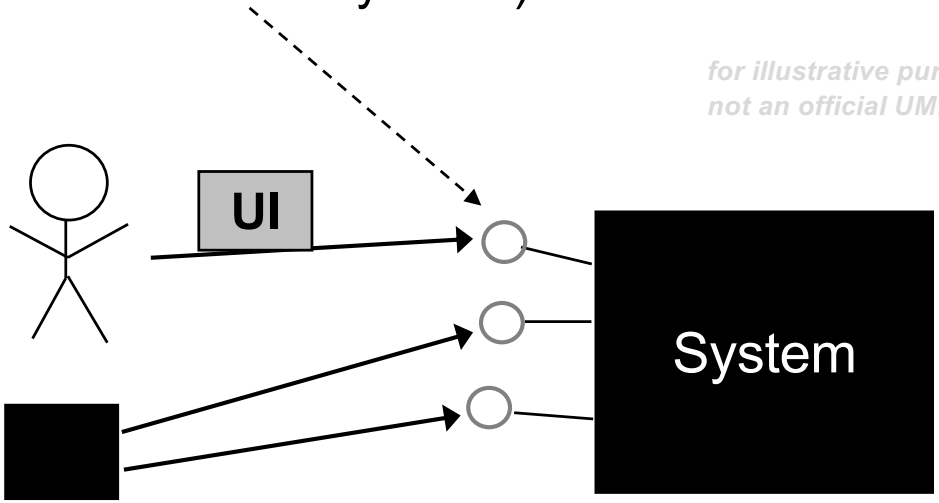
➔ *diagrams are only used for illustration*

➔ 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 Modelling Idea

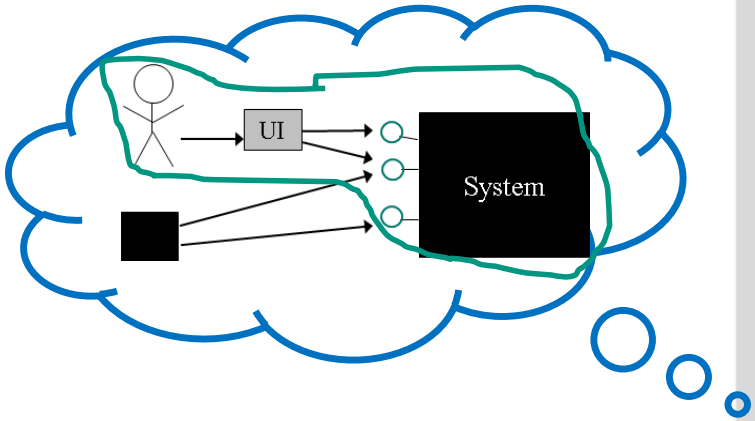
- Contract about behaviour
- Often, the computer system is perceived as a **black box**
  - with which the users (humans or other systems) interact



- Operational requirements only part of requirements document
- Other parts: data definitions, business rules, user interface design, business domain model, sections on quantitative and qualitative requirements, and so on

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
  - can also be another computer system
  - use cases therefore focus on operational requirements (often functional)



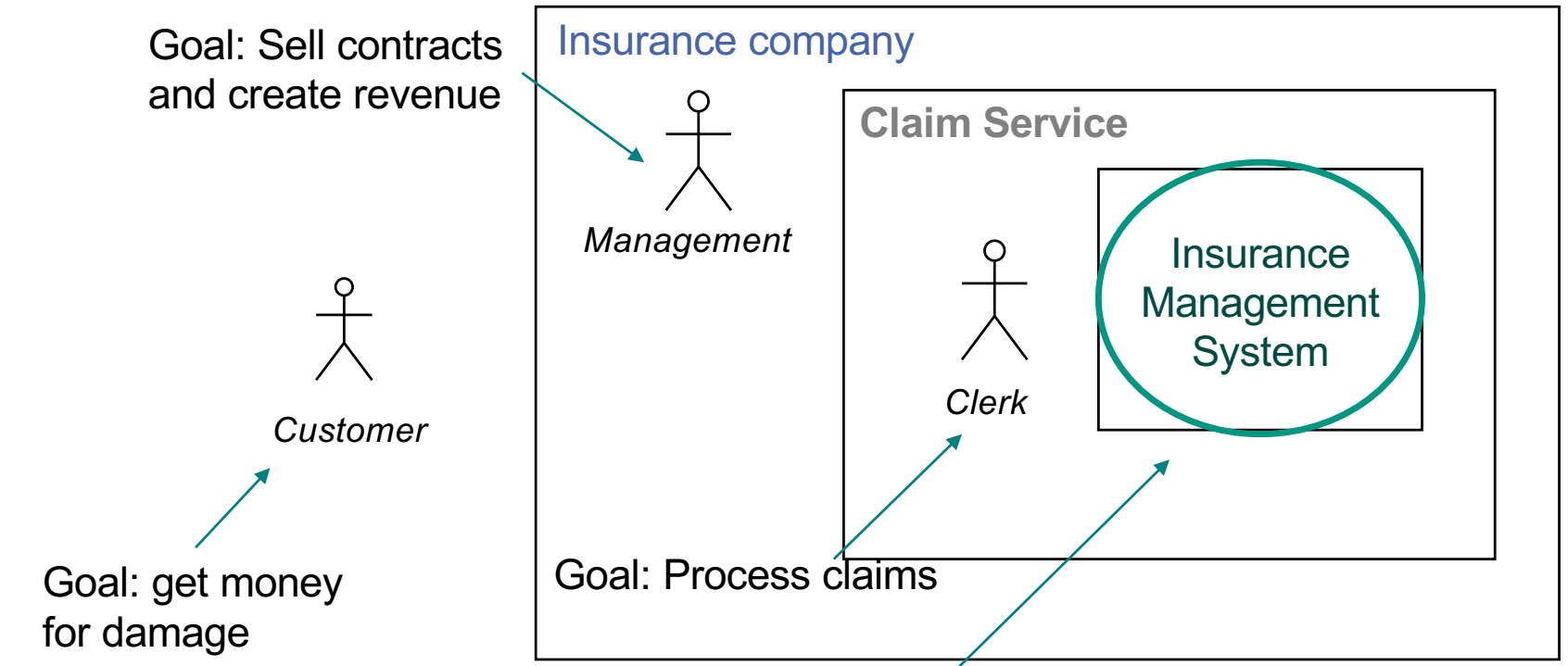
- In the context of software development, the term usually denotes a **“black-box user goal use case”**

- **Other goal levels and modeling scopes** are also feasible
  - however, in practice people often ignore this fact
  - Use cases are than specified at very different levels of granularity, which can cause a lot of confusion

Goals, Actors and Scopes

Important: understand what should be modeled as a use case

- identify the boundary of the "system" under consideration [Larman], [Cockburn]



→ Depending on scope, different goals and actors will be identified

System Scope and Context

System Boundary (Scope) [Cockburn]

- border between system under discussion and environment
  - often: system to be designed
- multiple scopes can be relevant for a project, but just one at a time
- interface of the system to the environment
  - information sources and sinks
  - hardware and software

System Context

- part of the environment of a system relevant for definition and understanding of requirements
- can be modelled using context diagrams

Context Boundary

- separates system context from irrelevant environment

➔ *Most boundaries are usually unclear in beginning*

- and only sharpened during requirements elicitation

What makes a good Use Case?

- It is common to be unsure whether use case is valid (or useful)
- 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

➔ Emphasis on the goal to be achieved

➔ For *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)

[Larman]

- 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 as simple example

- A common mistake is defining many use cases at too low a level
  - these should steps, subfunctions or subtasks within an EBP
  - 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 –

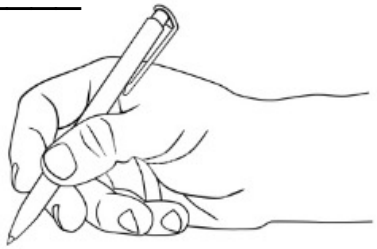
1. the boss test [Larman]
   - imagine your boss asks you what you have been doing all day?
   - how pleased would he be if you answer –
     - *I have been logging in …*
     - *I have been searching customer data …*
     - *I have been creating customer accounts …*

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

3. The size test [Larman]
   - (usually) not just a single step

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)*
     - glues together a number of user goal and subfunction use cases
       - *e.g. …*
  2. **User Goal** *(= EBP)*
     - describes how 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. …*

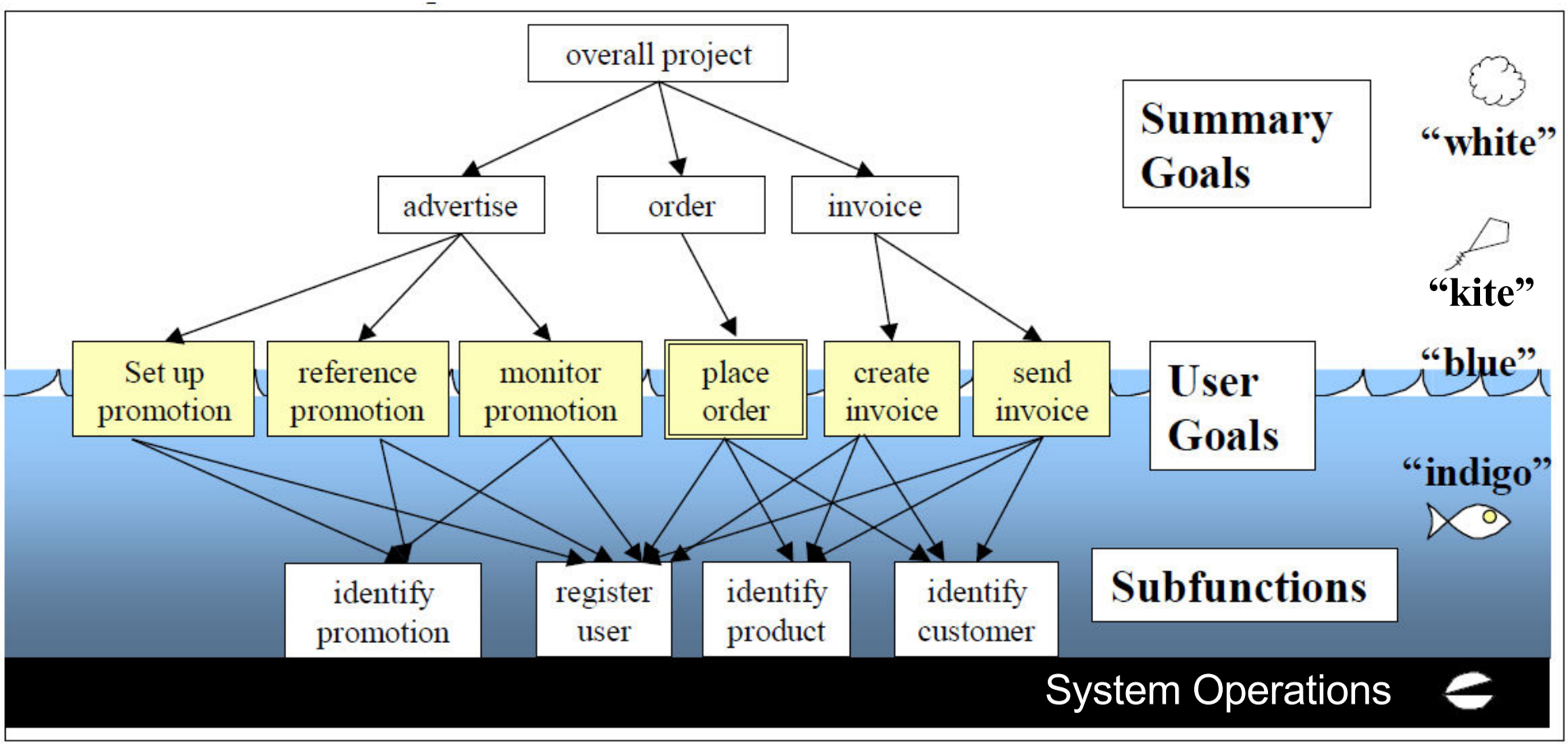


---

  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. …*

Picturing Use Case Levels

[Cockburn]



- To move up a level, ask: Why?
- To move down a level, ask: How?

Let‘s practice a bit …

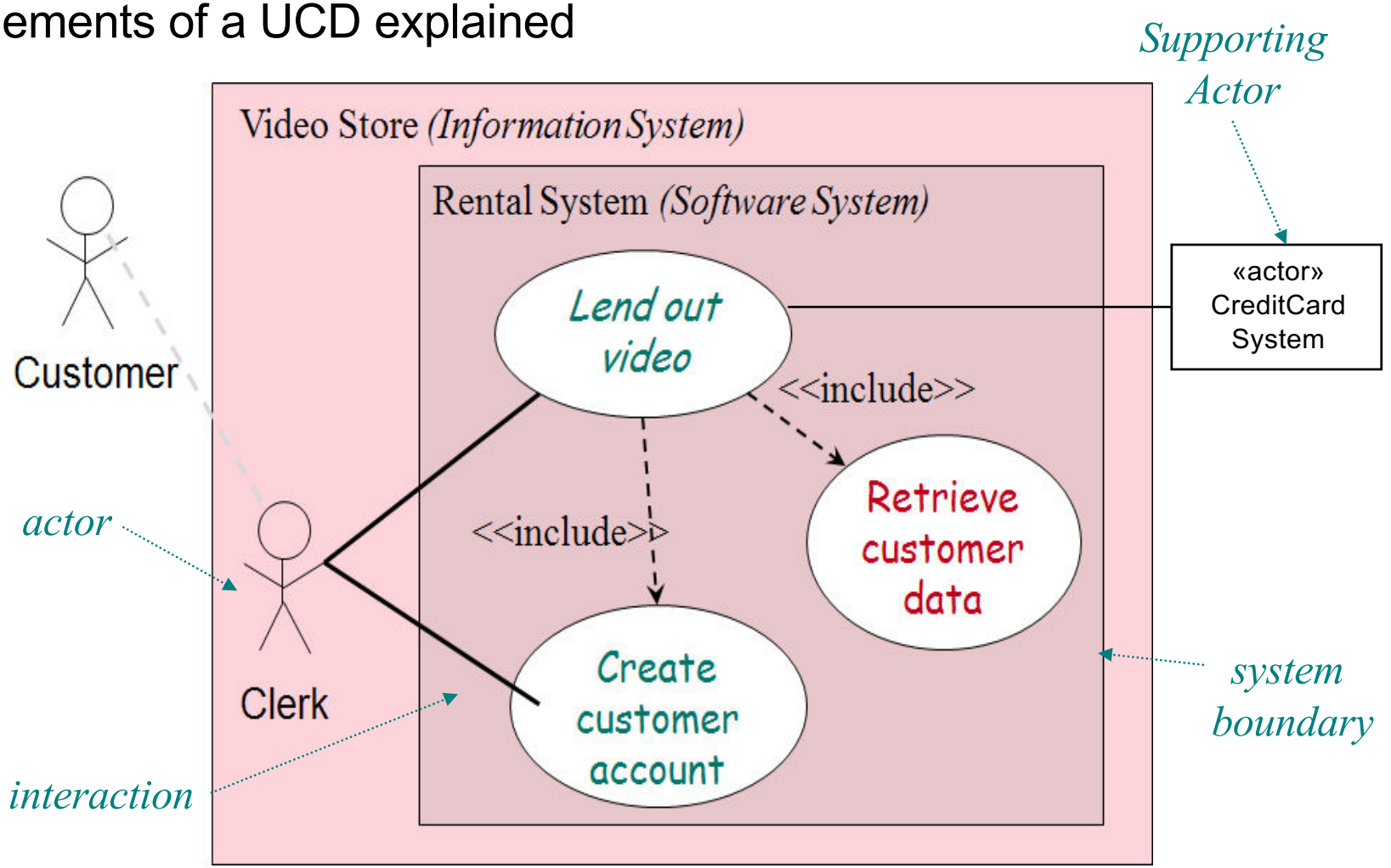
- *Identity the correct goal level for each activity*

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>> ~ sub use case "call"

<<extend>> ~ a possible extension, the extended use case is "unaware"

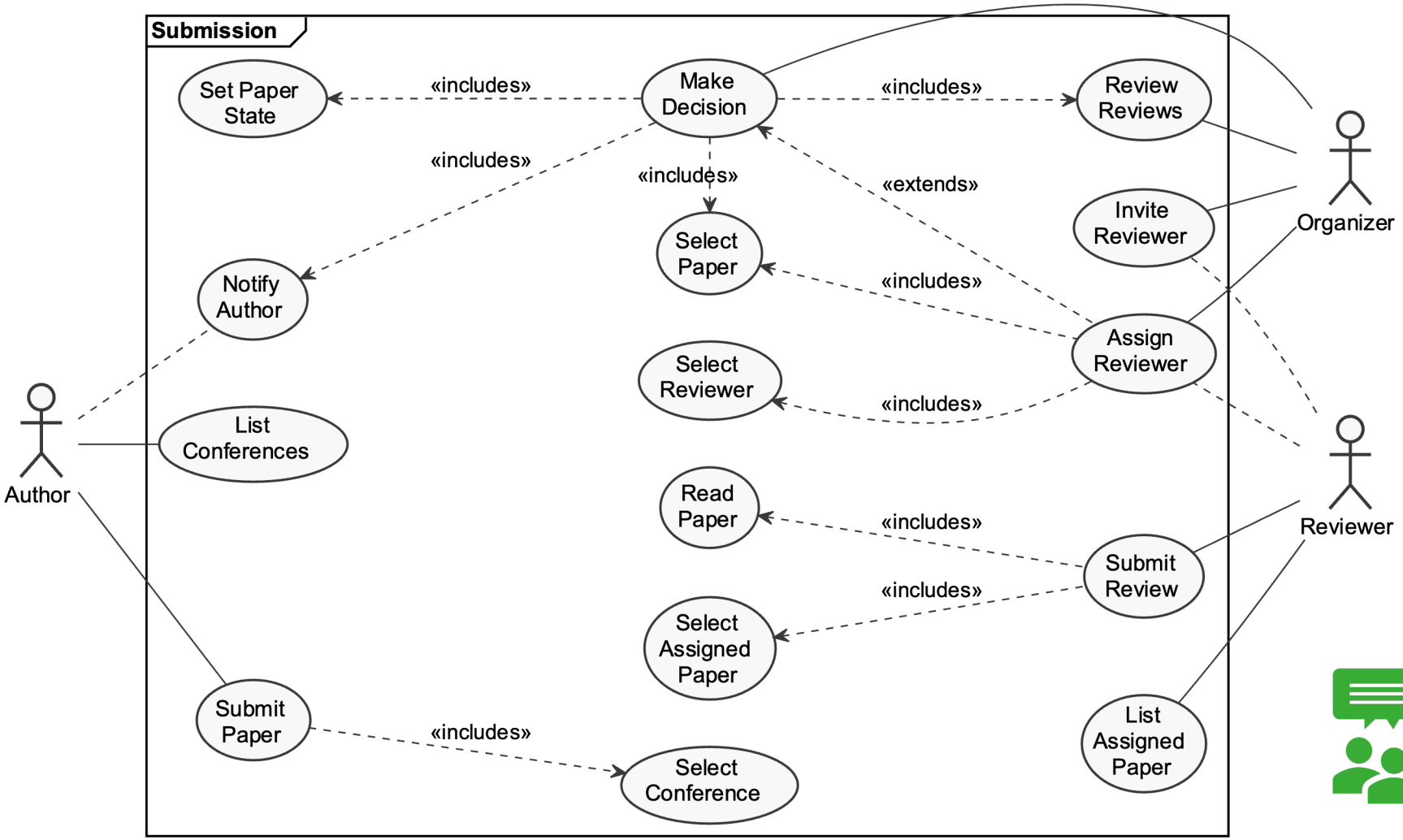
Use Case Diagramming Recommendations

- ***Diagrams are of secondary importance to writing text at this point***
  - ➔ don't get bogged down into too much diagramming work
  - ➔ only needed to highlight relations and provide outline

- Recommendations
  - for a use case diagram, usually use user-goal level use cases
    - table of contents, context diagram with primary and supporting actors
  - possibly "big picture" how higher-level use cases relate to lower-level ones
  - show computer system actors with an alternative notation to human actors
  - place the primary actor 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 if you use them

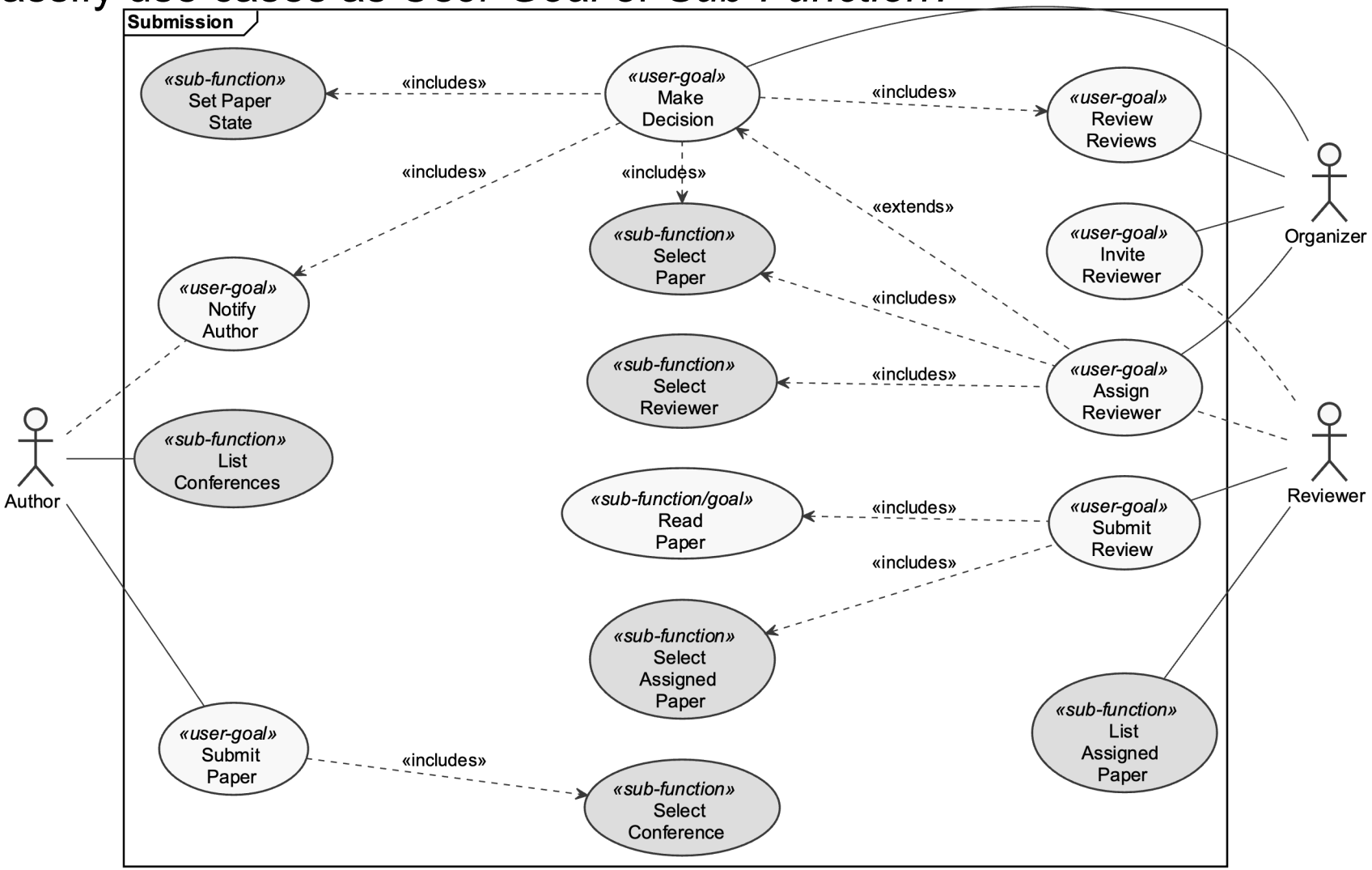
Let‘s practice a bit …

Classify use cases as *User Goal* or *Sub-Function?*



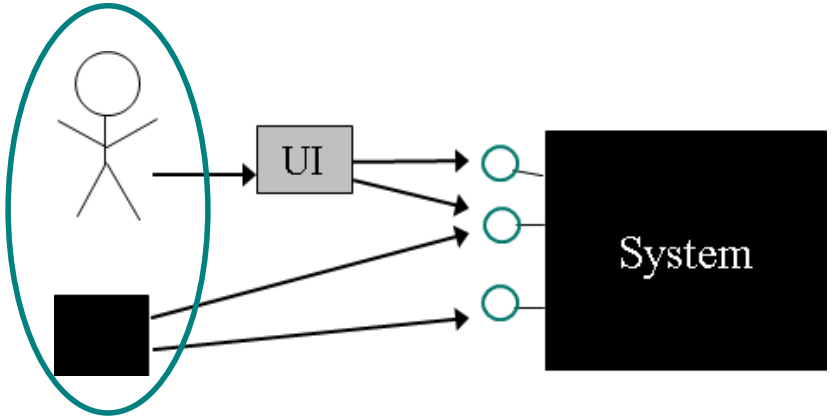
Let‘s practice a bit …

Classify use cases as *User Goal* or *Sub-Function?*



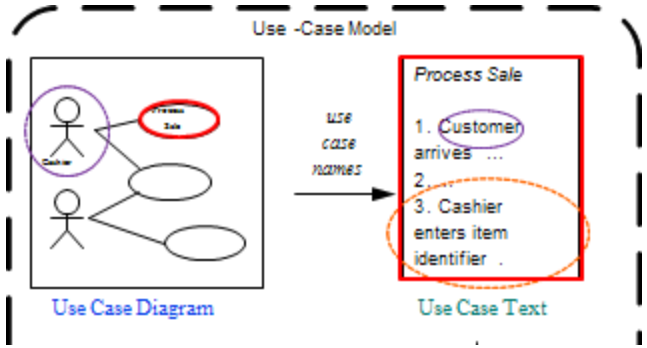
Key Use Case Terminology I

- **Stakeholder:** person or organisation that (in)directly influences requirements of a system, because they have an interest in the system
- **Actor:** entity with behaviour outside the system under discussion (SuD)
  - e.g. a person (i.e. role), computer system, organization
  - i.e. typically a user of the system
  - appears in a step of the use case



- **Primary actor** initiates interaction

- **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

- **Scenario:** specific sequence of actions and interactions between actors and SuD (system under development / discussion)
  - 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...

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

How to Find Use Cases

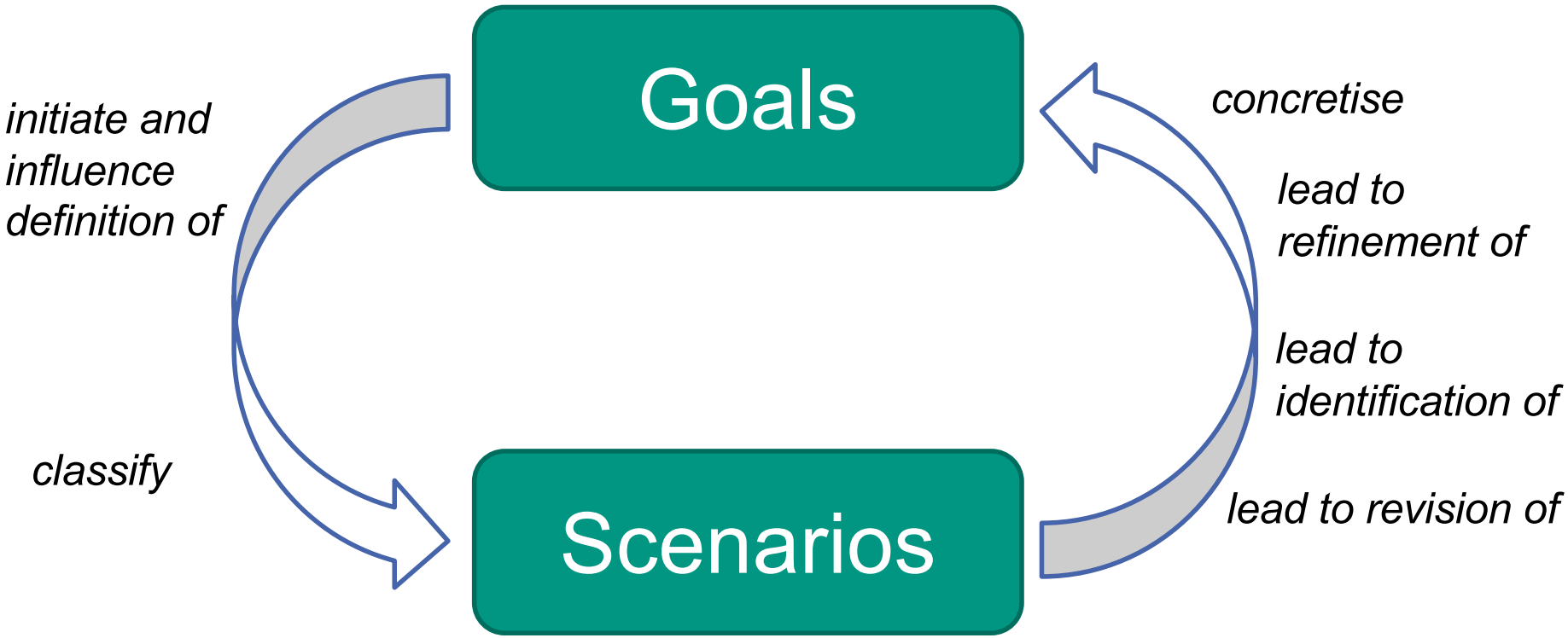
[Larman, p.83]

1. Choose the system boundary
   - e.g. *Point of Sale* system
2. Identify 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, p.17. p.221]

- Fully elaborated uses cases are complex
  - common to refine them iteratively
  - [Cockburn] recommends to work breadth-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
  2. use case briefs or main success scenario
  3. add extension conditions (failure conditions)
  4. add extension handling steps (failure handling) + more

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

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

Examples

- Actor and Goal
  - *As a cashier I would like to process sales in order to earn money for the store*
- Main Success Scenario
  - *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.*
- Failure Conditions
  - *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 …*
- Failure Handling:
  - *See Fully Dressed Use Case in Appendix 1*

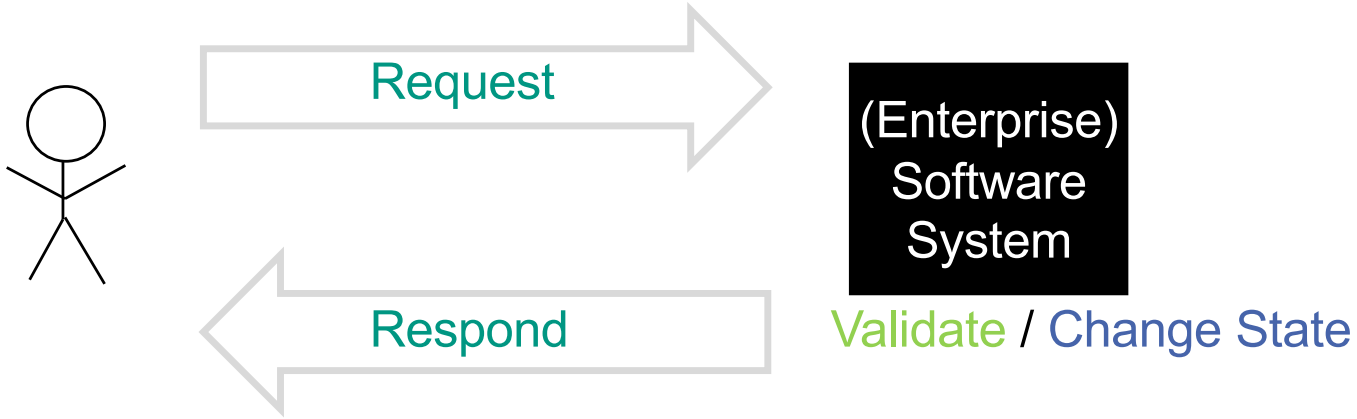
Use Case Templates

- Fully Dressed (see later)
  - Context of use, Scope, (Goal) level, Primary Actor,
  - Stakeholders and interests, Preconditions, Postcondition
  - Trigger, Main Success Scenario
  - Extensions
  - Special requirements
  - Technology and Data Variation Lists
- Casual
  - Less than the above, plain text, often no particular structure

- Others
  - One-Column Table
    - Like fully dressed, but in table
  - Two Column Tables
    - One column for actor, one for system
  - RUP Style
    - Similar to fully dressed

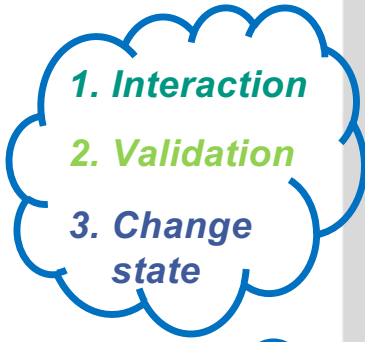
How to write Use Cases [Cockburn]

- Steps describe interactions, validation, or internal change



➔ an exemplary use case transaction for an ATM might contain these steps

1. *The Customer inserts his debit card*
2. *The ATM validates the debit card*
3. *(…)*
4. *The ATM deducts the amount from balance*
5. *(…)*

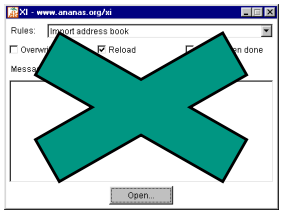


Further Writing Guidelines [Cockburn]

- Important that each step
  - shows (sub) goal succeeding – process moves forward
  - captures actor's intent – not movement on GUI
  - has an actor –
    - passing information and/or control (interaction) **OR**
    - validating a condition **OR**
    - updating system state

- 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 text fields etc.
    - ➔ *this is the job of UI designers*
  - however, need to define the data exchanged
    - *such as PIN, account number etc.*



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 main scenario free from if statements
  - rather they should be defined in 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
       - should be written as **something that is detectable by the system**
     - 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 extension points are complex, express extension as 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

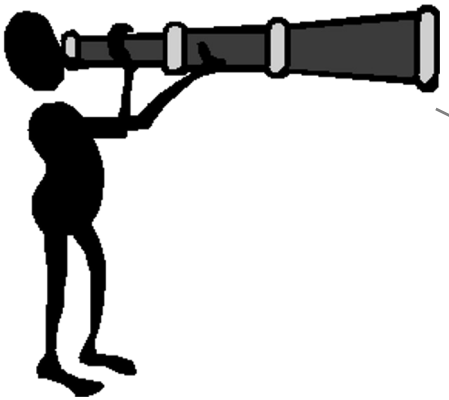
- record that a non-functional requirement, quality attribute or constraint relates specifically to a use case
  - *e.g., performance, reliability, usability design constraints*
- non-functional requirements can later be collected within a Supplementary Specification

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 an I/O device*
  - generally, such design constraints should be avoided
  - but if they are unavoidable they should be included
- record variations in data schemes
  - *e.g. using UPCs or EAN's for item identifiers*
  - you may add your data descriptions here

Conclusion

- Use cases as established best practice for requirements elicitation
  - though simple at a first glance there are many subtle details to consider –
    - goal level, clear assignment of responsibilities, etc.
    - see common mistakes and things to keep in mind
  - know elements of a use case and relations
    - links between use cases, extensions



Software Quality

By MB-one & Marc Lacoste - This file was derived from: ILA 2018, Schönefeld (1X7A5288).jpg with background clipped, (CC BY-SA 4.0)

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

APPENDIX

Appendices

- Common mistakes and summary statements from Cockburn
- Helpful stuff for real life
- Examplary Requirements Templates
  - RUP (FURPS+)
  - IEEE
- Requirements Tools
  - DOORS
  - Requisite Pro
  - Requirements Composer
  - Caliber
  - Enterprise Architect
  - SoftWiki
- Fully Dressed Use Case Example
- Use Case Checklist

Common Mistakes [Cockburn]

- No system
- No primary actor
- Too many user interface details
- Very low goal levels
- Purpose and content not aligned

Summarizing Statements I [Cockburn]

- For each use case
  - A use case is a prose essay
  - Make the use case easy to read
  - Just one sentence form
  - “Include” sub use cases
  - Who has the ball? (no passive voice)
  - Get the goal level right (ask why to shift levels)
  - Keep the GUI out
  - Two endings
  - Stakeholders need guarantees
  - Preconditions

Summarizing Statements II [Cockburn]

- For a use case set
  - An ever-unfolding story
  - Both corporate scope and system scope

- For working on use cases
  - Only a part of the requirements specification
  - Work breadth first
  - Know the cost of mistakes (how much precision is needed, what is risk?)
  - Rather write too little than too much (keep document readable)
  - Handle failures (check completeness of extensions)
  - Use case diagrams are not use cases

- Two major benefits of use cases (Section 1.4)
  - Goal-driven, enables communication among stakeholders
  - Uncover failure conditions and think about responses

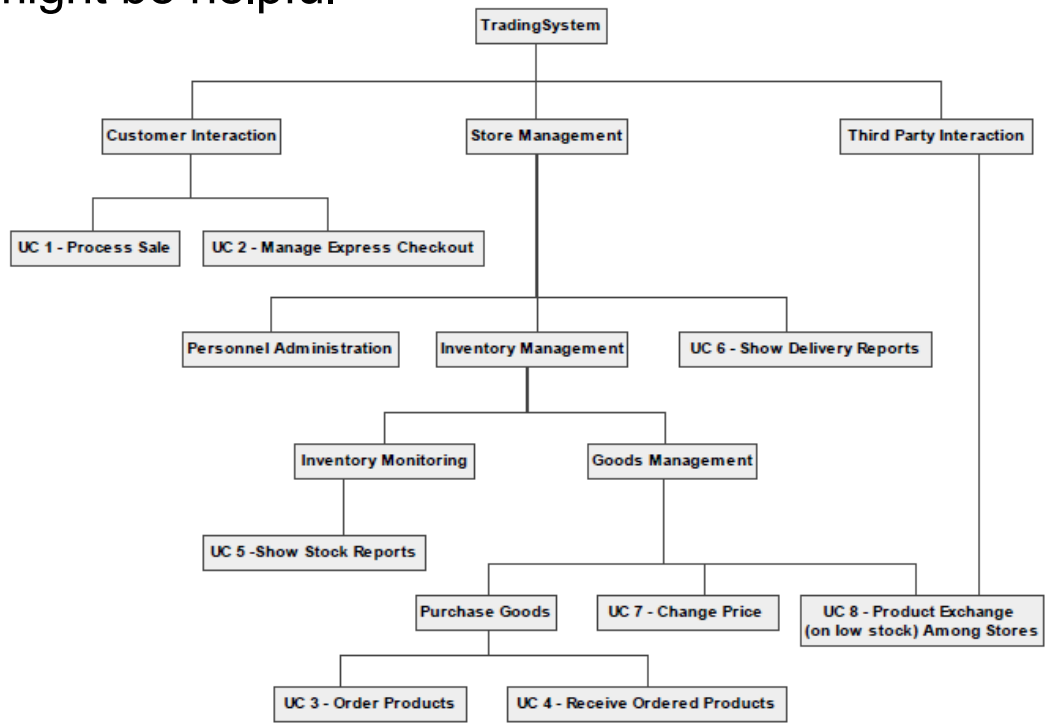
Helpful Stuff for Real Life I

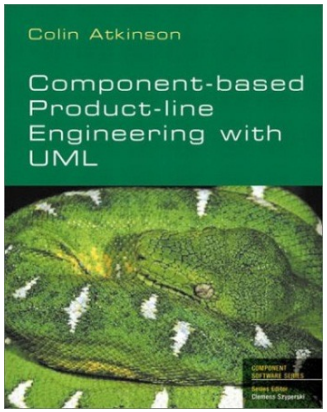
- Checklists can become handy for verifying requirements
  - e.g. the one from Cockburn listed in Appendix 2
- Special tools to manage use cases are available
  - such as –
    - Borland Caliber
    - Serena Dimensions RM
    - …
- However, use case texts can be written with every word processor
  -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

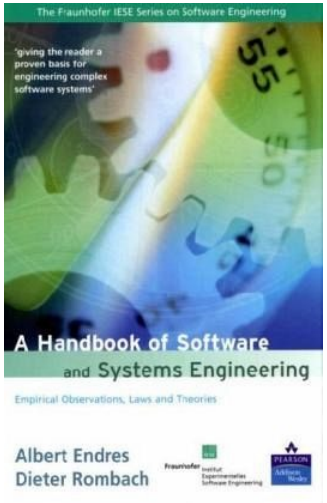
ID	Lev.	Area	Name	Prior.	Freq. of Use	Impor.	State
UC1	UG	CRM	Cr. Cust. Account	VH	M	Must	Appr.
UC2	SF	CRM	Retr. Cust. Acc.	H	VH	Should	Written
UC7	UG	CRM	Send Advert. Mail	M	L	Nice	Open

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







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

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]

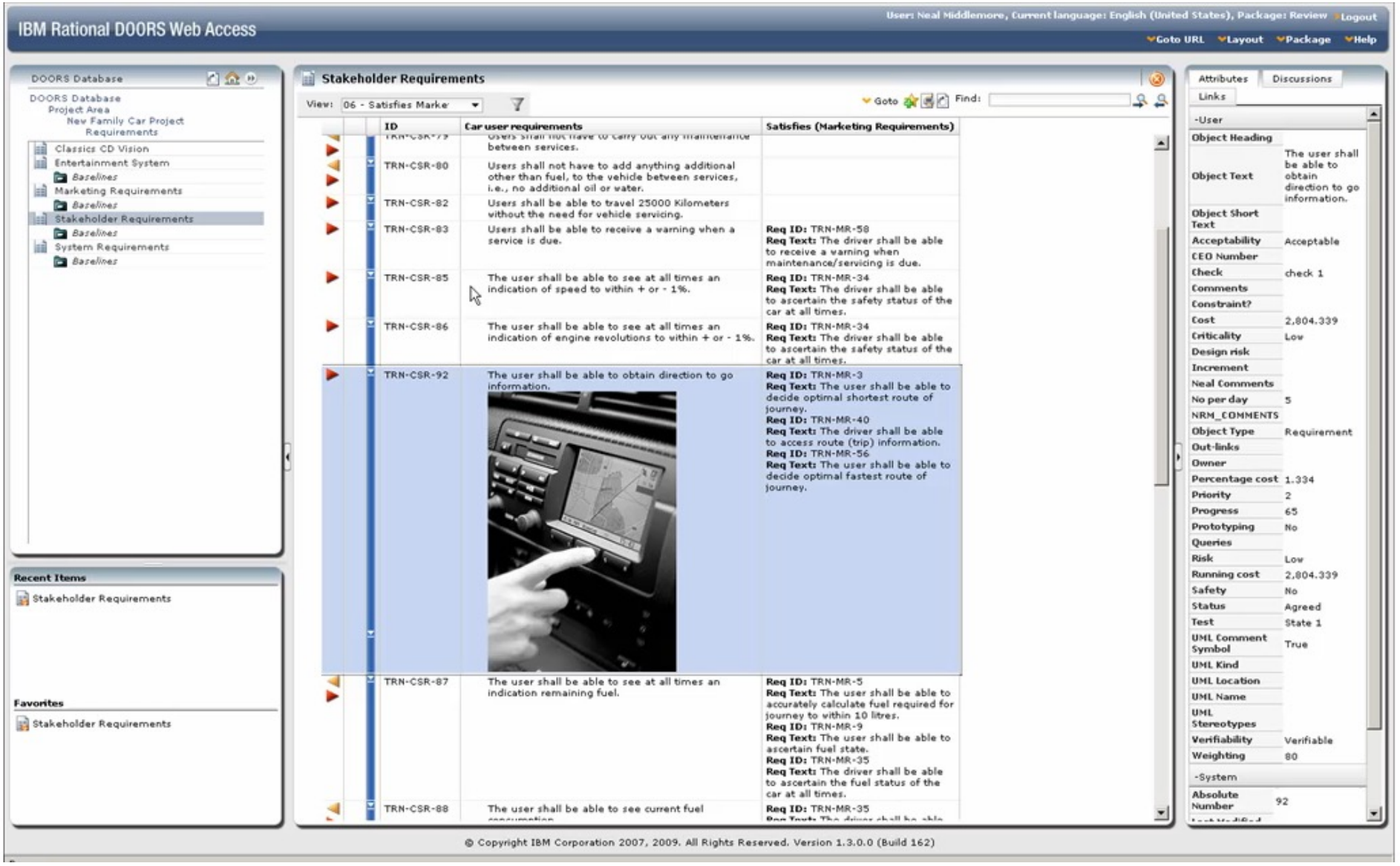
- Table of Contents
- Introduction
  - Purpose
  - Scope
  - Definitions
  - References
  - Overview
- General Description
  - Product Perspective
  - Product Function
  - User Characteristics
  - General Constraints
  - Assumptions and Dependencies
- Specific Requirements
  - Functional Requirements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

Tool: IBM Rational DOORS

- Formerly 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.

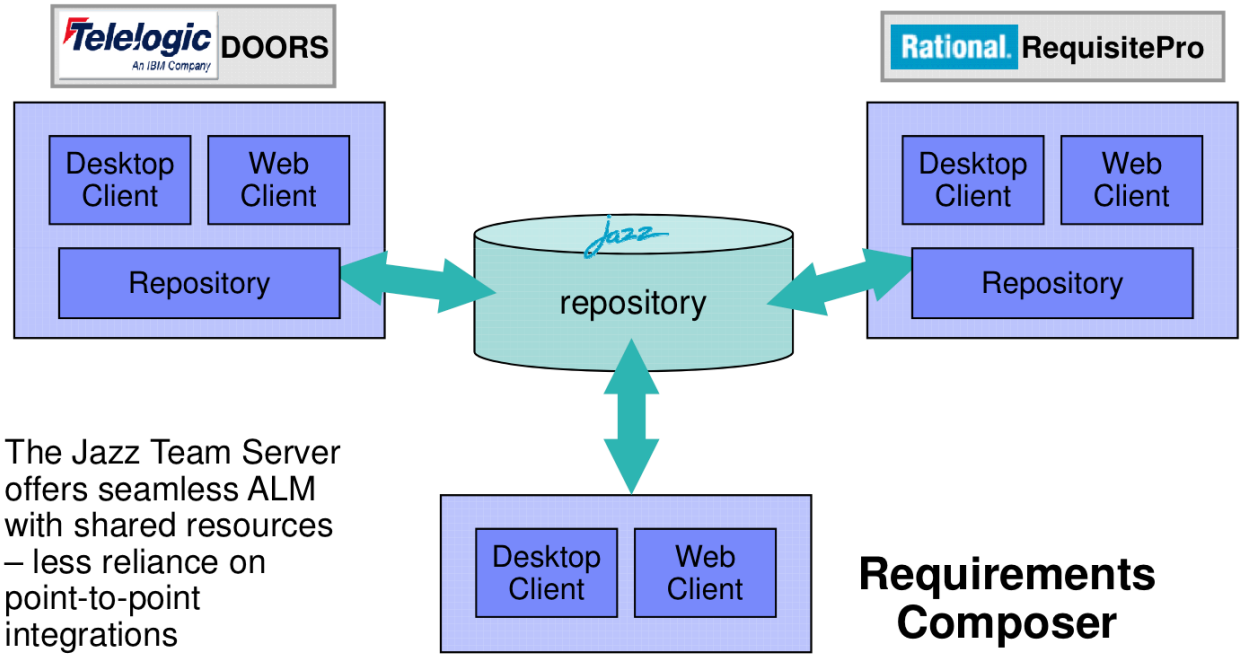
- Can generate requirements documents

DOORS Web Access



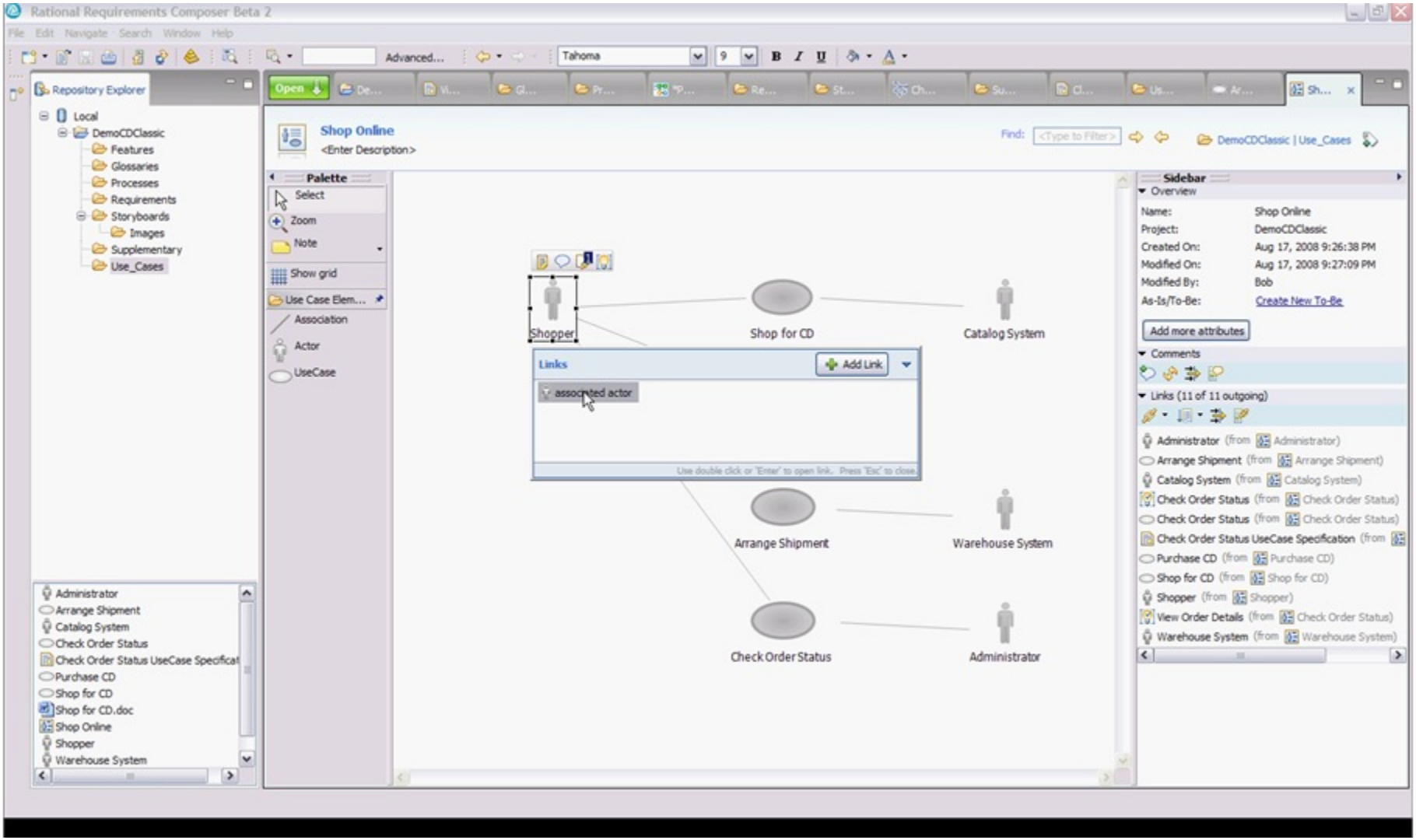
Tool: IBM Requisite Pro

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

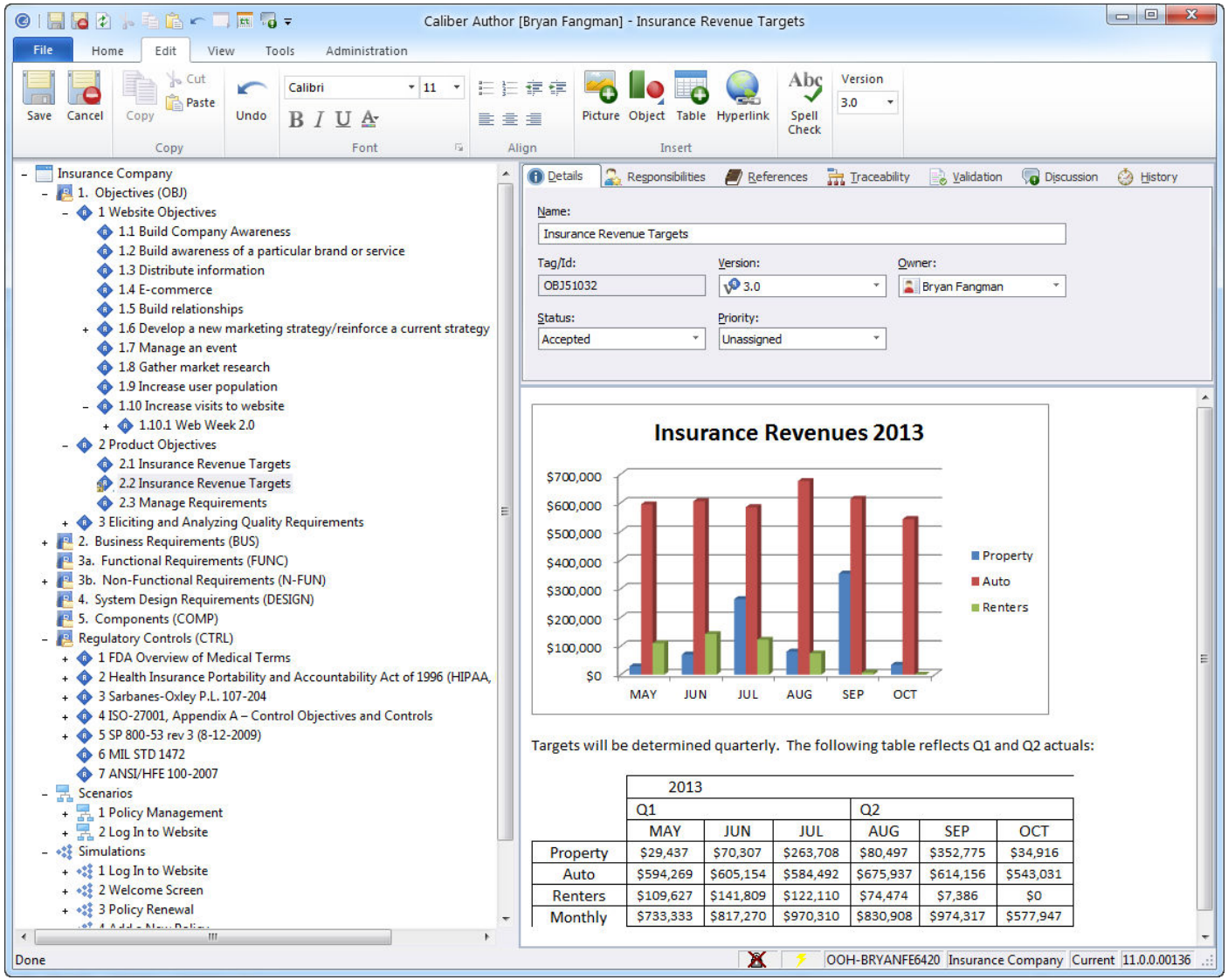


The Jazz Team Server offers seamless ALM with shared resources – less reliance on point-to-point integrations

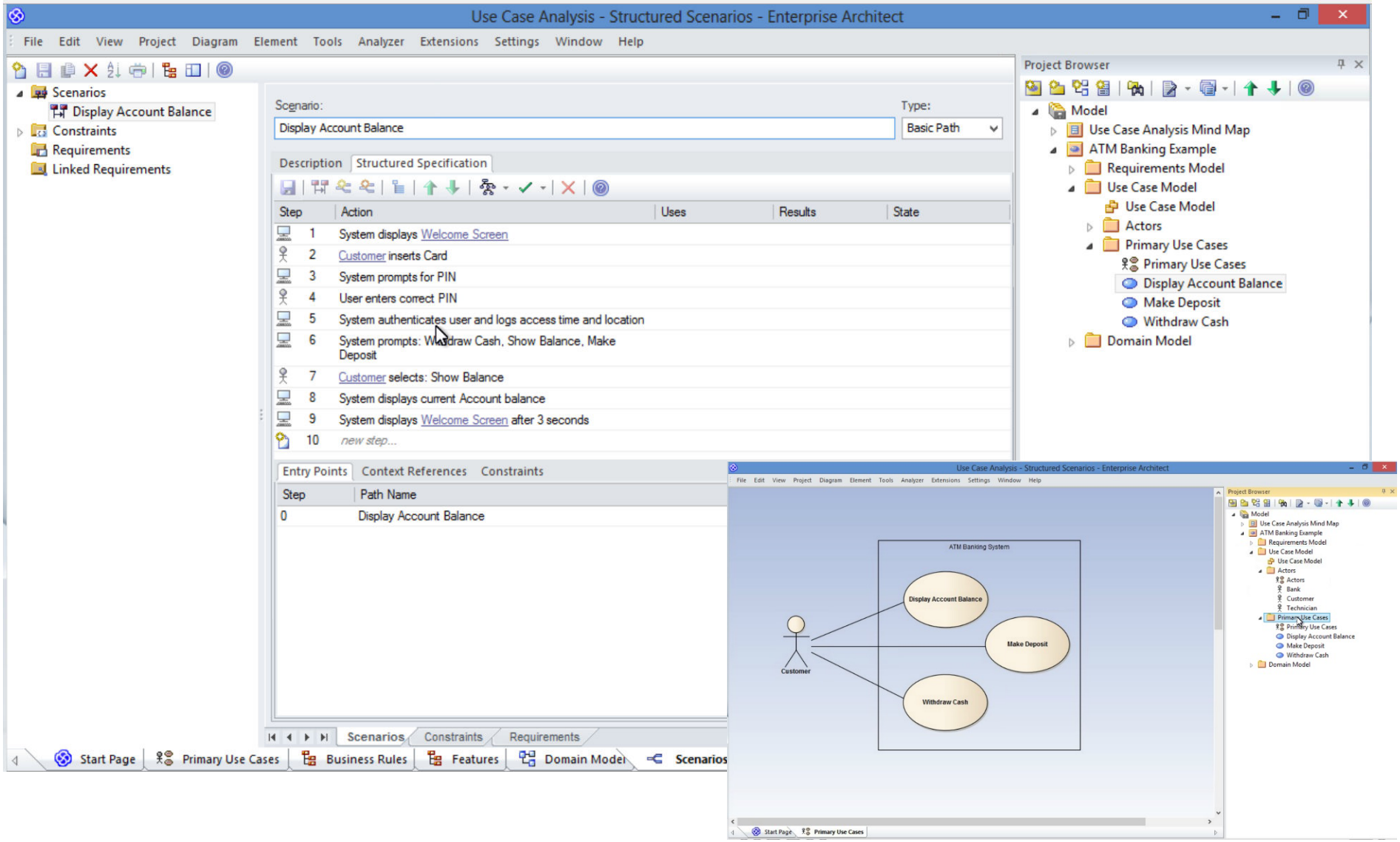
Tool: Requirements Composer



Tool: Borland Caliber



Tool: Enterprise Architect



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)

[Larman]

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)

[Larman]

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): [Larman]
    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)

[Larman]

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)

[Larman]

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)

[Larman]

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)

[Cockburn]

- 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)

[Cockburn]

- 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?"

Agile stories

[Cohn 2004]

- A single sentence about a requirement
- Written from a stakeholder's perspective
- Optionally including the expected benefit
- Accompanied by acceptance criteria for requirement
- Acceptance criteria make the story more precise

Standard template:

As a <role> I want to <my requirement> [so that <benefit>]

A sample story

As a skier, I want to pass the chairlift gate so that I get access without presenting, scanning or inserting a ticket at the gate.

Author: Dan Downhill Date: 2013-09-20 ID: S-18

Sample acceptance criteria

Acceptance criteria:

- Recognizes cards worn anywhere in a pocket on the left side of the body in the range of 50 cm to 150 cm above ground
- If card is valid: unlocks turnstile and flashes a green light for five seconds or until the turnstile is moved
- If card is invalid: doesn't unlock gate and flashes a red light for five seconds
- Time from card entering the sensor range until unlock and flash red or green is less than 1.5 s (avg) & 3 s (max)
- The same card is not accepted twice within an interval of 200 s