Requirements Engineering

Security Engineering

David Basin
ETH Zurich

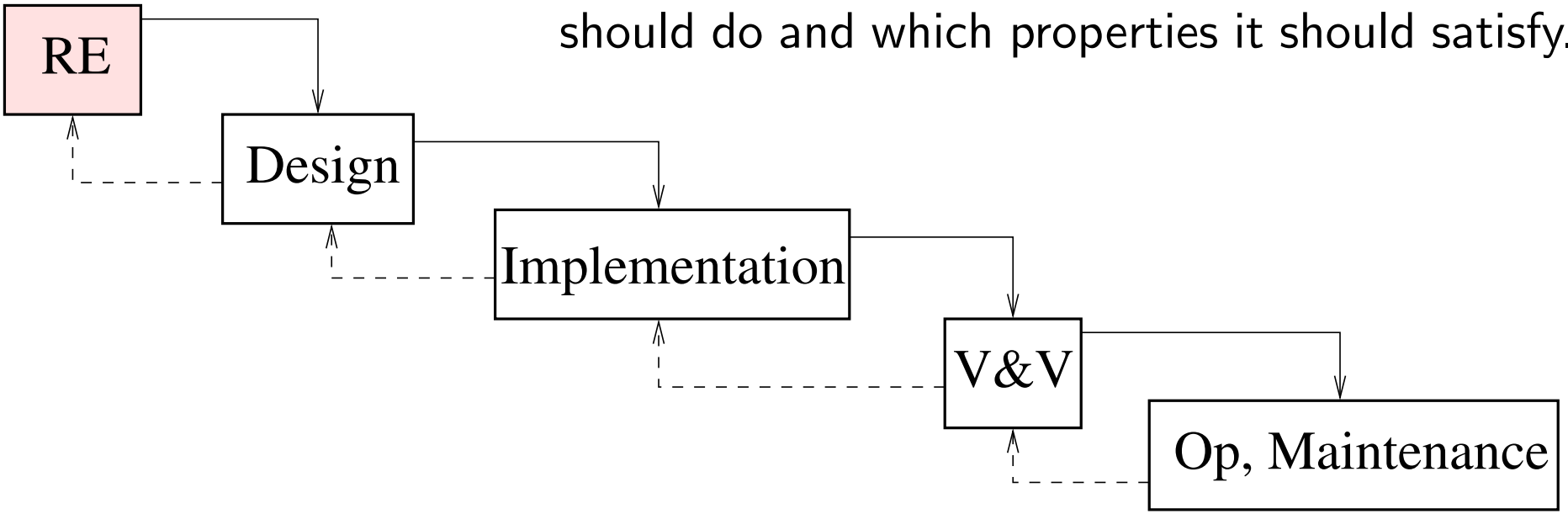
Motivation

"We would like European countries to be able to electronically share data about citizens and businesses. For instance, someone relocating from Switzerland to Germany should not have to physically go to different administrations to provide the necessary data. Of course, we must respect national and EU legislations, in particular privacy regulations. Existing legacy systems must be integrated. Oh, and you should know that nation states don't like interference with their businesses."

What would you do?

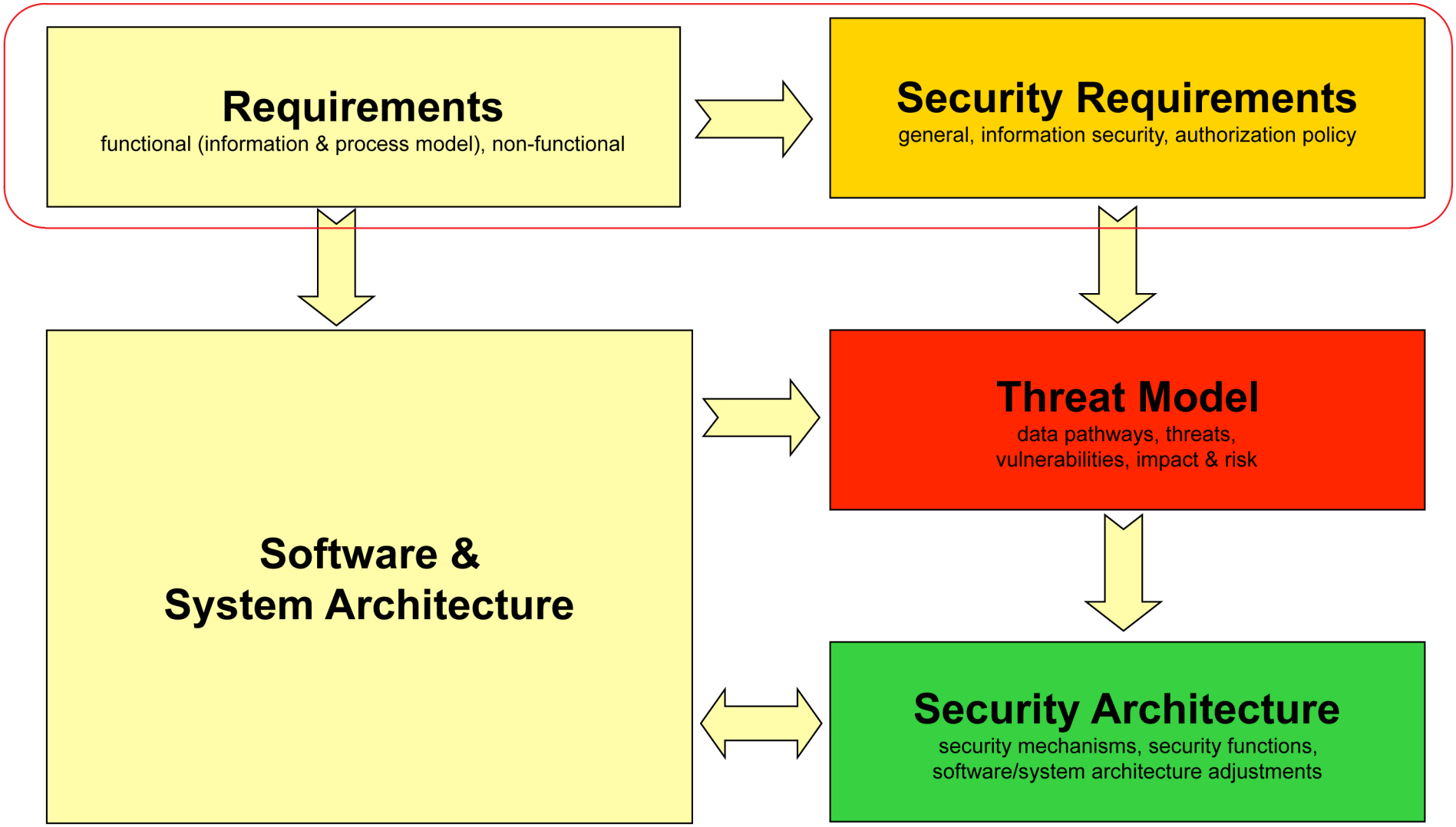
Context

Requirements engineering is about eliciting, understanding, and specifying what the system should do and which properties it should satisfy.



Requirements engineering is the branch of software engineering concerned with the real-world goals for, **functions** of, and **constraints** on software systems. It is also concerned with the relationship of these factors to precise specifications of software behavior, and to their evolution over time and across software families. **(Zave 1997)**

Security context



Road map

☞ **Requirements and Requirements Engineering**

- Security-specific requirements
- Use cases and misuse cases
- Data and process requirements
- Discussion

Goals for lecture

- Understand the notion of requirement
- See the need for different activities in requirements engineering
- Understand the relationships between functional and security requirements
- Learn several modeling techniques

Requirements

- Hardware/software systems are built to **solve problems**
- The **problem domain** is the part of the world where the system should produce desired effects, together with the means by which it produces them.
- **Requirements** specify the **effects** the system should have in the problem domain as well as **assumptions** on the domain itself.
- . . . and also what should **not happen**!

In other words, requirements specify how the system should and should not behave in its intended environment!

Kinds of requirements

- Many kinds!

  System, development process, data-format, structural, system deployment, documentation, ...

- **Functional requirements** describe what the system should do
  - Goal and I/O-oriented descriptions, defining system's purpose.
  - Can also be described in terms of system states.
  - **Example:** the system allows bank transfers.
  - Think "use cases"

- **Non-functional requirements** describe constraints
  - **Security**, usability, scalability, maintainability, testability, ...
  - **Example:** System must handle 1000 SSL connections/second.
  - Think "quality requirements"

Functional versus non-functional requirements

Boundary often fuzzy

- Function(al) = purpose?

  1000 SSL/connections second might be main **purpose** of an SSL accelerator even if is only a constraint on the **function** computed.

- Negative constraints can be functional requirements

  Example 1 pure function from input to output

  $f(x) \in \{y \mid y \leq x\}$ is equivalent to $f(x) \notin \{y \mid y > x\}$

  Example 2: behavior defined by action sequences (traces)

  "there is no payment without prior authentication"

  So for correct systems, authentication always precedes payment

Functional versus non-functional (cont.)

- More examples

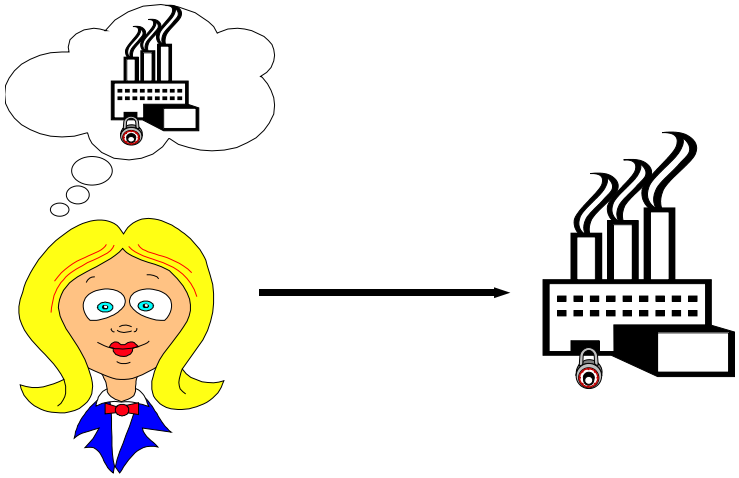
  1. "Account is blocked after 3 wrong PIN codes" (F)
  2. "Whenever button pressed, elevator stops"
     or "Elevator stops only if button pressed" (F)
  3. "System must produce a response within 10 milliseconds" (NF)
  4. "Only authenticated users may access services" (Compare w/ 2)
  5. "All communication must be encrypted using IPsec" (F)
  6. "Attackers should not be able to read network traffic" (NF?)

- N.B. negative nonfunctional constraints (like 6) can often be converted to positive functional requirements (like 5).

Requirements desiderata

- Quantifiability and precision
  - "SW shall be reliable" and "SW should be secure" are useless
  - "reliability of $< 10^{-12}$ errors per hour of execution" is better
  - **Security** appears more difficult than reliability in this respect
- Consistency and prioritization
  - Sometimes requirement conflict. Different stake-holders want incompatible things
  - **Security** almost always conflicts with usability and cost
  - Requirements must then be prioritized

Relevance



- Everything is built on the requirements!

  The earlier errors occurs, the greater the consequences and costs

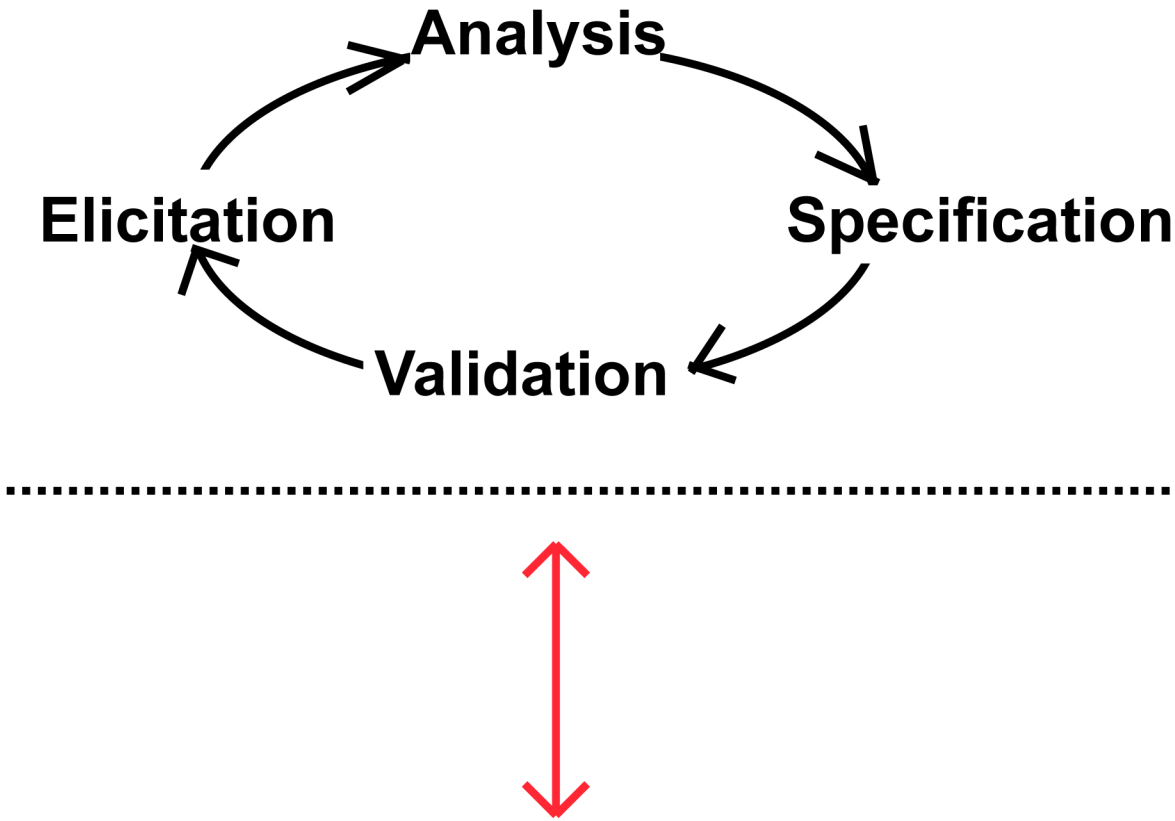
- Why is this so notoriously difficult?
  - Requirements are a fuzzy domain: imprecisions, incompleteness
  - Requirements are moving targets
  - One must be clear about all possible ways a system can be used, **including by the adversaries**

Road map

☞ **Requirements and Requirements Engineering**

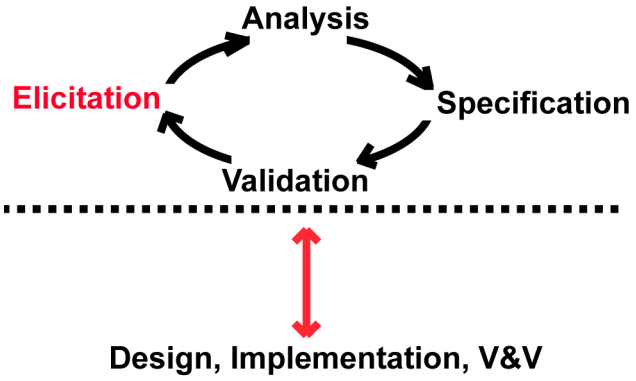
- Security-specific requirements
- Use cases and misuse cases
- Data and process requirements
- Conclusions

Requirements engineering activities



(1) Elicitation

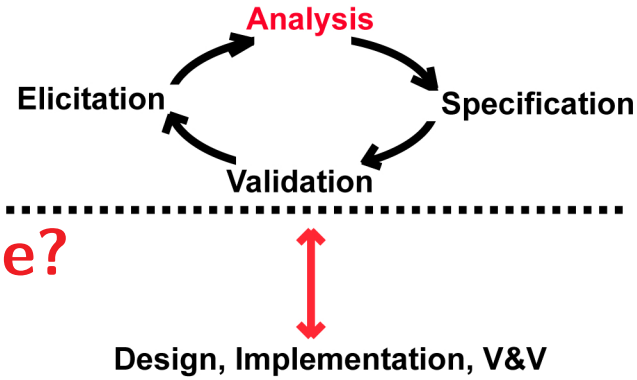
Determine requirements with stakeholders



- Requirement sources (**example:** banking)
  - Stake-holders (bank directors, different system users)
  - High-level system objectives (banking functions)
  - Domain knowledge (investment vs. private banking)
  - Operational context
  - Organizational context

- Elicitation techniques
  - Interviews with stake-holders
  - Prototypes and models
  - Scenarios: ask "what if" questions
  - Facilitated group meetings (brainstorming)
  - Observation: watch how users work with existing system

(2) Analysis



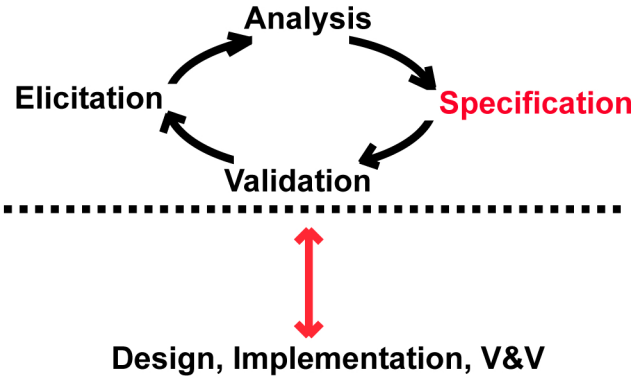
Are requirements clear, consistent, complete?

- Elicited requirements may be imprecise, incomplete, or contradictory
  - What exactly does "inhibit replay" mean?
  - Or "strong authentication"?
- To improve precision, build **models** of the system.

  E.g., data flow, transition system, class models, etc.
- Carry out risk analysis, both security and business related
- N.B. distinction between **analysis** and **validation** not always made

(3) Specification

Document desired system behavior



- Documents what system should do (functional and nonfunctional)
  - Emphasis on **what**, not **how**
  - Often written just in natural language
  - May be more precise and include system models
  - Or even formal specifications of interface descriptions
- Different organizations have standards for structuring document

  Example: ANSI/IEEE Std 830-1984 (next slides)
- Basis for contracts, schedules, and cost planning
- Ideally tracked throughout development

IEEE Guide to Software Requirements Specification
Topics addressed

Functionality: what the software should do

External interfaces: how it interacts with people, the system's hardware, other hardware, and other software

Performance: its speed, availability, response time, recovery time of various software functions, etc.

Attributes: portability, correctness, maintainability, security, and other considerations

Design constraints imposed on an implementation: implementation language, resource limits, operating environment, any required standards in effect, etc.

IEEE Guide to Software Requirements Specification

Structure

1. Introduction

   (a) Document purpose (identify product/revision/release)
   (b) Product scope (main objectives, relate to enterprise goals)
   (c) Definitions, acronyms, abbreviations
   (d) References (related documents, contracts, standards, ...)

2. General description

   (a) Product perspective (context, e.g., new product or replacement)
   (b) Product function (high-level description, pictures)
   (c) User characteristics (main classes of system users)
   (d) General constraints (op. environment, design, implementation)
   (e) Assumptions and dependencies (e.g., 3rd party components)

IEEE example (cont.)

3. External interface requirements

   Subsections for user, software, hardware, and communication interfaces. Communication includes security requirements.

4. Functional requirements (system features)

   For each feature:

   (a) Description and priority
   (b) Specific functional requirements

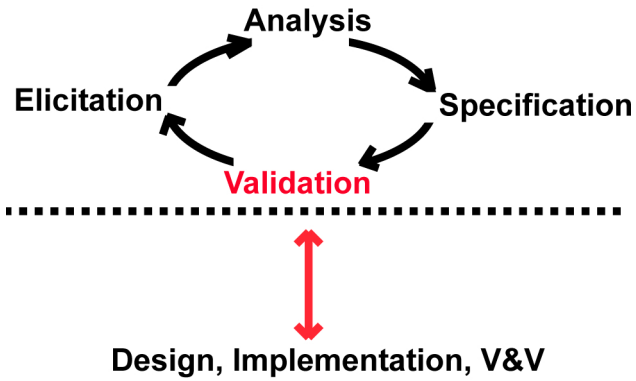
5. Nonfunctional requirements

   safety, **security**, and other quality attributes, e.g., adaptability, maintainability, portability, testability, usability, etc.

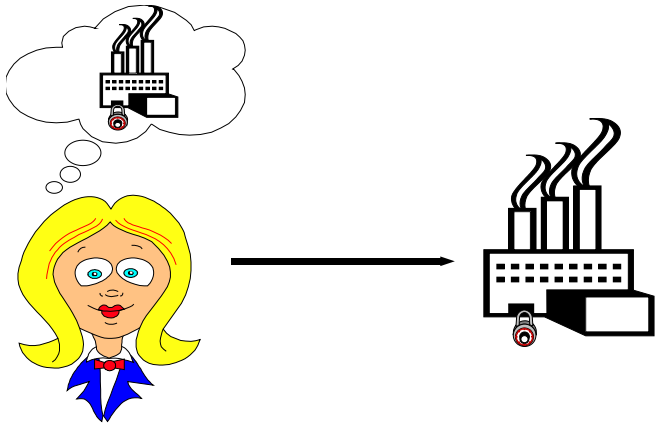
6. Appendices (including glossary and analysis models) and index

(4) Validation

Are we building the right system?



- Check if requirements specified are really what stake-holders want
- Difficult!
  - Stake-holder requirements are intangible (in their heads)
  - Customers often change their minds and the system context may also change, e.g., when suppliers fail to deliver
  - You can only write down what you have thought of beforehand
- Techniques
  - Use prototypes or flip-chart mockups
  - Use informal or formal models
  - Ongoing interaction with customers



Road map

- Requirements and Requirements Engineering

☞ **Security-specific requirements**

- Use cases and misuse cases
- Security requirements and their modeling
- Conclusions

Security requirements relate to ...

- Confidentiality
- Integrity
- Availability
- Identification
- Authentication
- Authorization
- Intrusion detection
- Non-repudiation
- Privacy
- Security auditing
- Survivability
- Physical protection
- Usage

- Security at different levels
  - Beyond the company, e.g., earthquakes and fire
  - Technical
  - Protocols
  - Single modules
  - Applications, systems
  - Networks
  - Socio-technical (people+machines)
  - Guidelines
- Usually given by negative formulation

Source: D. Firesmith, *Engineering Security Requirements*, JOT 2(1):53-68, 2003

Impact of security on RE

- Security is a requirement that must be made precise
- Negotiation/prioritization when conflicts with other requirements
  - Security vs. usability, performance, costs, ...
- Impact on system architecture and programming language/libraries
- Adversarial context is critical (i.e., possible threats)
  - for understanding what security functionality is required
  - problem even more acute later during design phase
- Difficult problem: aided by industry standards, catalogs of common problems, code-level advisories, etc.

Sources of general security requirements

- Laws or sector-specific provisions.
  - Federal Data Protection Act, e.g., in Switzerland[1] and Germany[2]
  - PCI Data Security Standard for Payment Card Industry[3]
- **Example:** PCI standard gives requirements along with testing procedures (see next slides)
- General guidelines and organizational rules
  - For algorithms, standard technologies, password complexity, ...
  - E.g., NIST special papers.[4] See talk on standards.

[1] www.admin.ch/ch/e/rs/2/235.1.en.pdf
[2] www.gesetze-im-internet.de/bdsg_1990
[3] www.pcisecuritystandards.org
[4] csrc.nist.gov/publications/PubsSPs.html

Example: payment card industry requirements

- Requirements for enhancing security of payment account data
- Developed by major players of payment card industry

  American Express, Visa, Mastercard, ...
- Helps vendors adopt consistent data security measures to protect customer account data
- Includes requirements for security management, policies, procedures, network architecture, software design, and other critical protective measures
- Associated council provides certification (see certification module)

PCI example (cont.)

- **Build and Maintain a Secure Network**

  1. **Install and maintain a firewall configuration to protect cardholder data**
  2. **Do not use vendor-supplied defaults for system passwords and security parameters**

- **Protect Cardholder Data**

  3. **Protect stored cardholder data**
  4. **Encrypt transmission of cardholder data across open, public networks**

- **Maintain a Vulnerability Management Program**

  5. **Use and regularly update anti-virus software or programs**
  6. **Develop and maintain secure systems and applications**

- **Implement Strong Access Control Measures**

  7. **Restrict access to cardholder data by business need to know**
  8. **Assign a unique ID to each person with computer access**
  9. **Restrict physical access to cardholder data**

- **Regularly Monitor and Test Networks**

  10. **Track and monitor all access to network resources and cardholder data**
  11. **Regularly test security systems and processes**

- **Maintain an Information Security Policy**

  12. **Maintain a policy addressing information security for employees and contractors**

PCI — each point is refined

- **Example:** 3. Protect stored cardholder data

  3.1. Keep cardholder data storage to a minimum. Develop a data retention and disposal policy. Limit storage amount and retention time to that required for business, legal, or regulatory purposes, as documented in the data retention policy.

- Testing procedures given as well (e.g., for 3.1)

  Obtain and examine the company policies and procedures for data retention and disposal, and perform the following:

  - Verify that policies and procedures include legal, regulatory, and business requirements for data retention, including specific requirements for retention of cardholder data.
  - Verify that policies and procedures include provisions for disposal of data when no longer needed...

From general to specific

- Such guidelines are good starting points. But they leave open:
  - Which data is critical?
  - Who is authorized to do what?
  - How should controls be used?
- These requirements must be distilled from system-specific models

  Use case models provide information on authorization

  Misuse case models provide (negative) security requirements

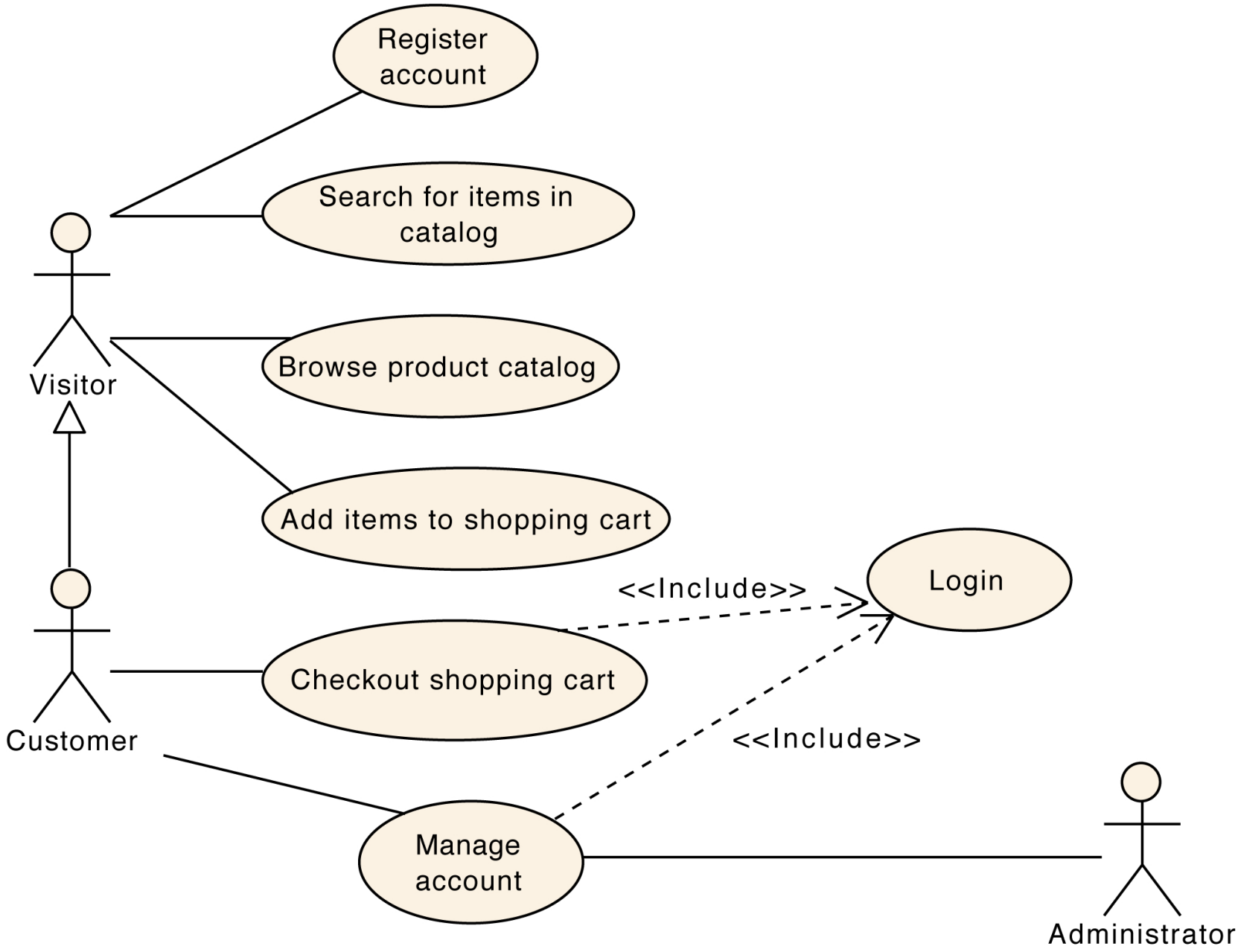
  Data models can indicate data criticality and authorization
- There is some tension on when this is done
  - Waterfall process requires complete requirements prior to design
  - Iterative processes are more flexible in this regard

Road map

- Requirements and Requirements Engineering
- Security-specific requirements
- ☞ **Use cases and misuse cases**
  What must and what must not happen?[5]
- Data and process requirements
- Conclusions

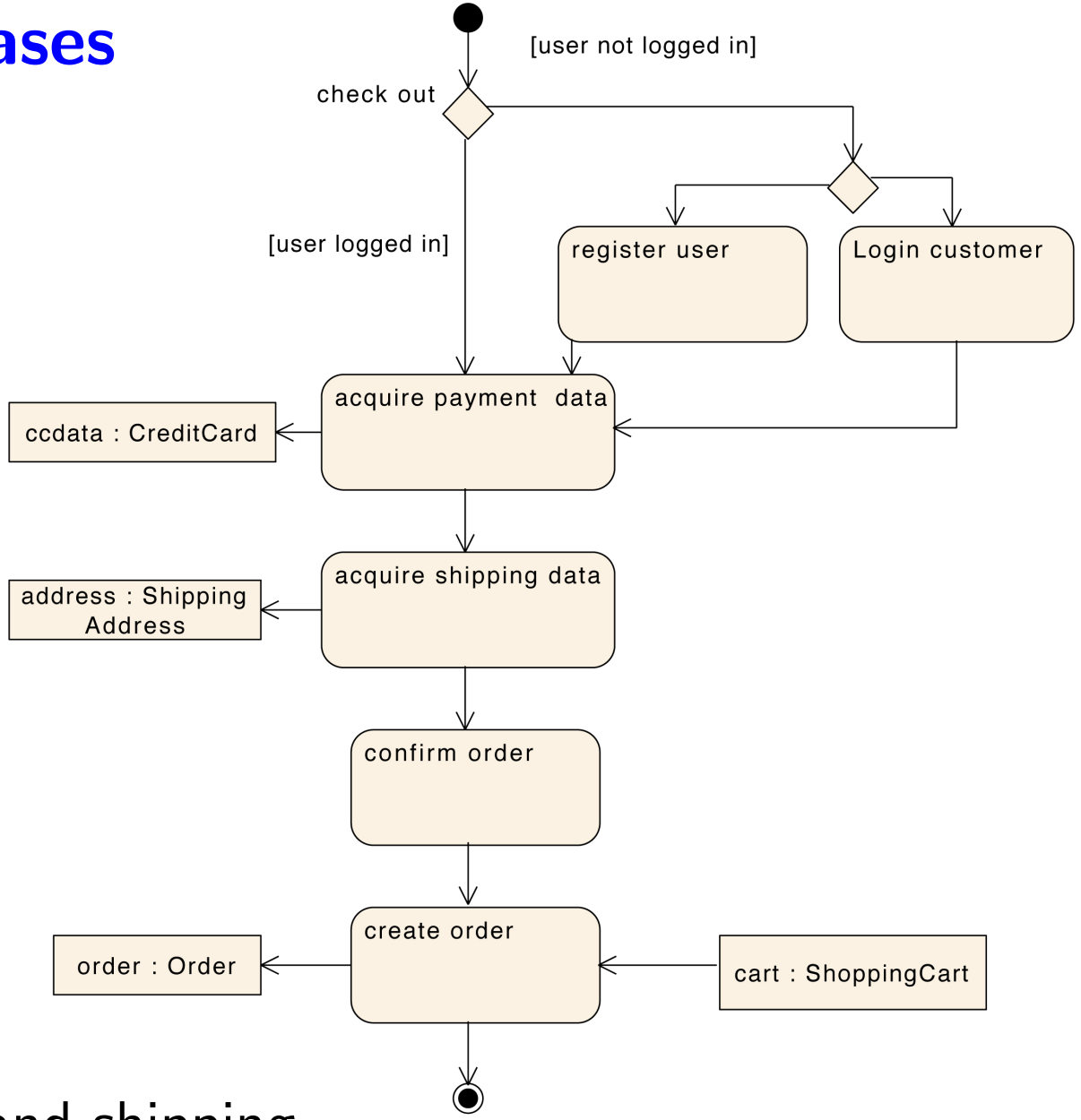
[5] We use and motivate here modeling abstractions introduced in subsequent modules.

Elicitation of functional requirements: use cases



Elaboration of use cases via activity diagrams

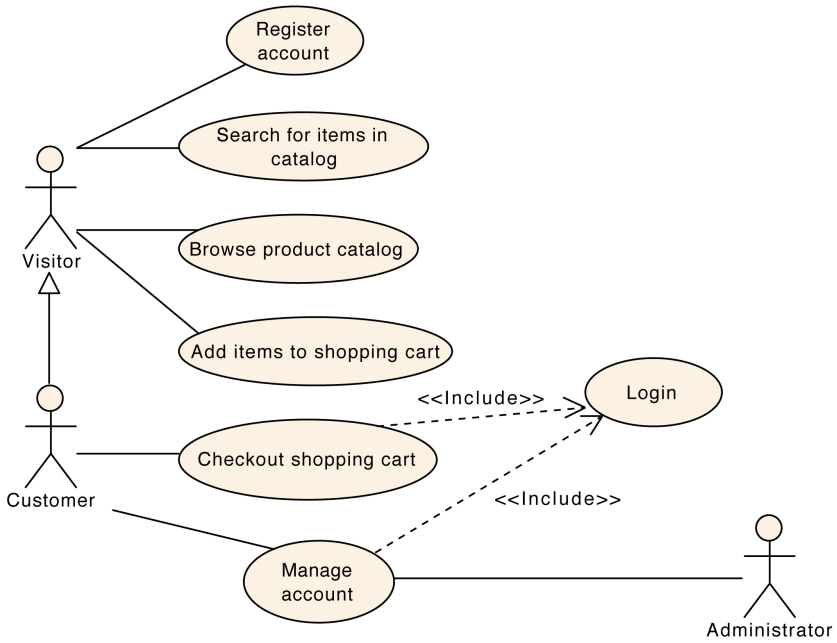
Scenario: steps describing an interaction between a system and a user



Example (check out):

Customer enters payment and shipping data into the system. Afterwards, the order is confirmed and processed.

Essence of use case diagrams



- Use Cases: scenarios for using system
- Used to elicit functional requirements

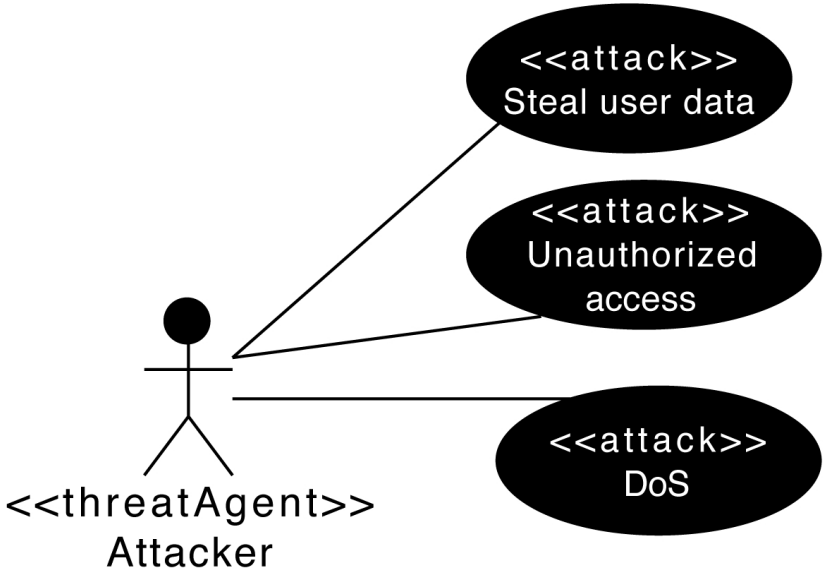
  Who can do what with the system

- They get you to think about your problems and spawn discussions
- If you find the little men silly, just don't draw them!

  Idea of usage scenarios is, however, fundamental

But what about security requirements?

Eliciting security requirements: misuse cases[6]

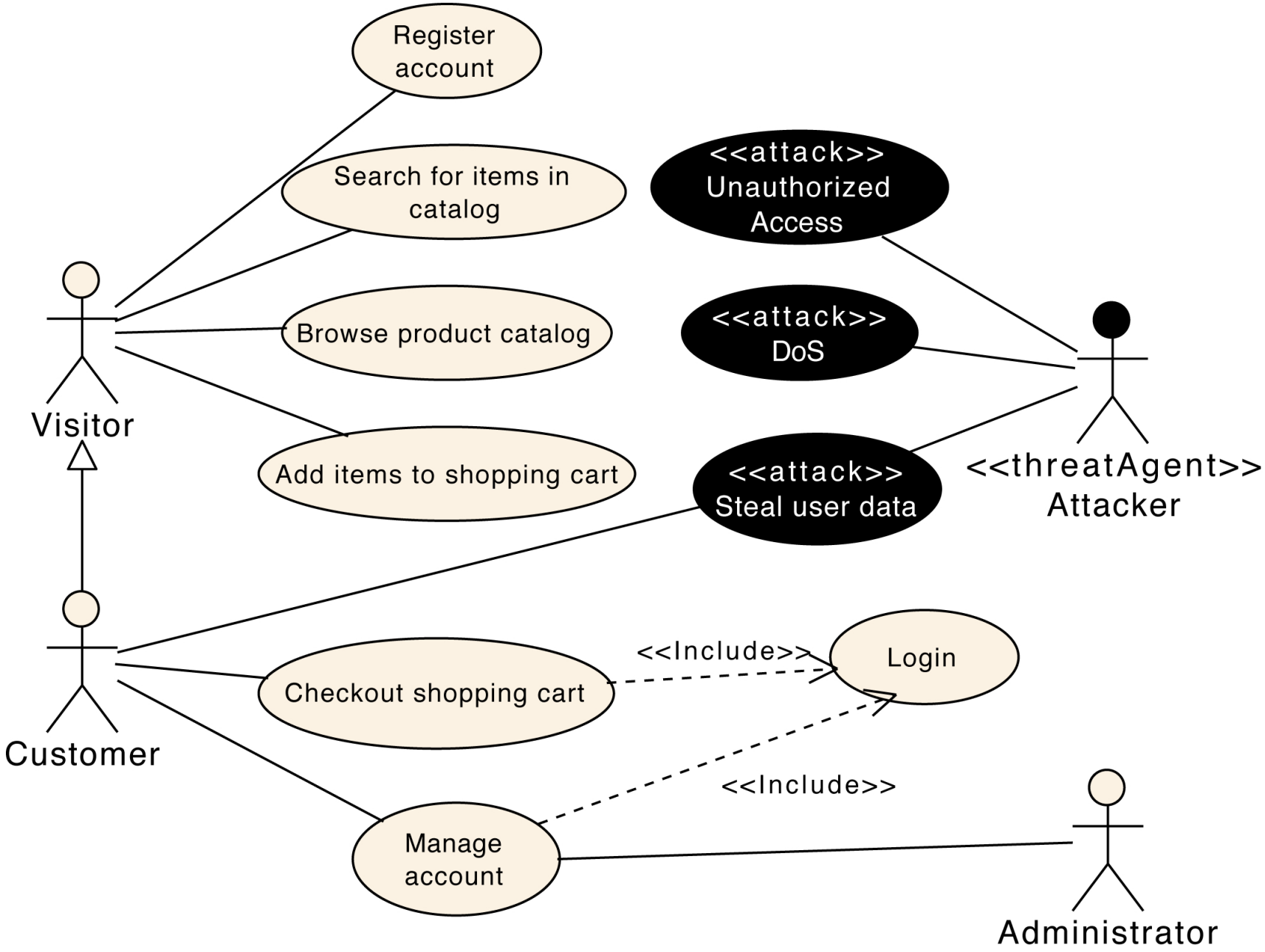


- Specify non-intended functionality
- Sometimes desired behavior is used for malicious intentions

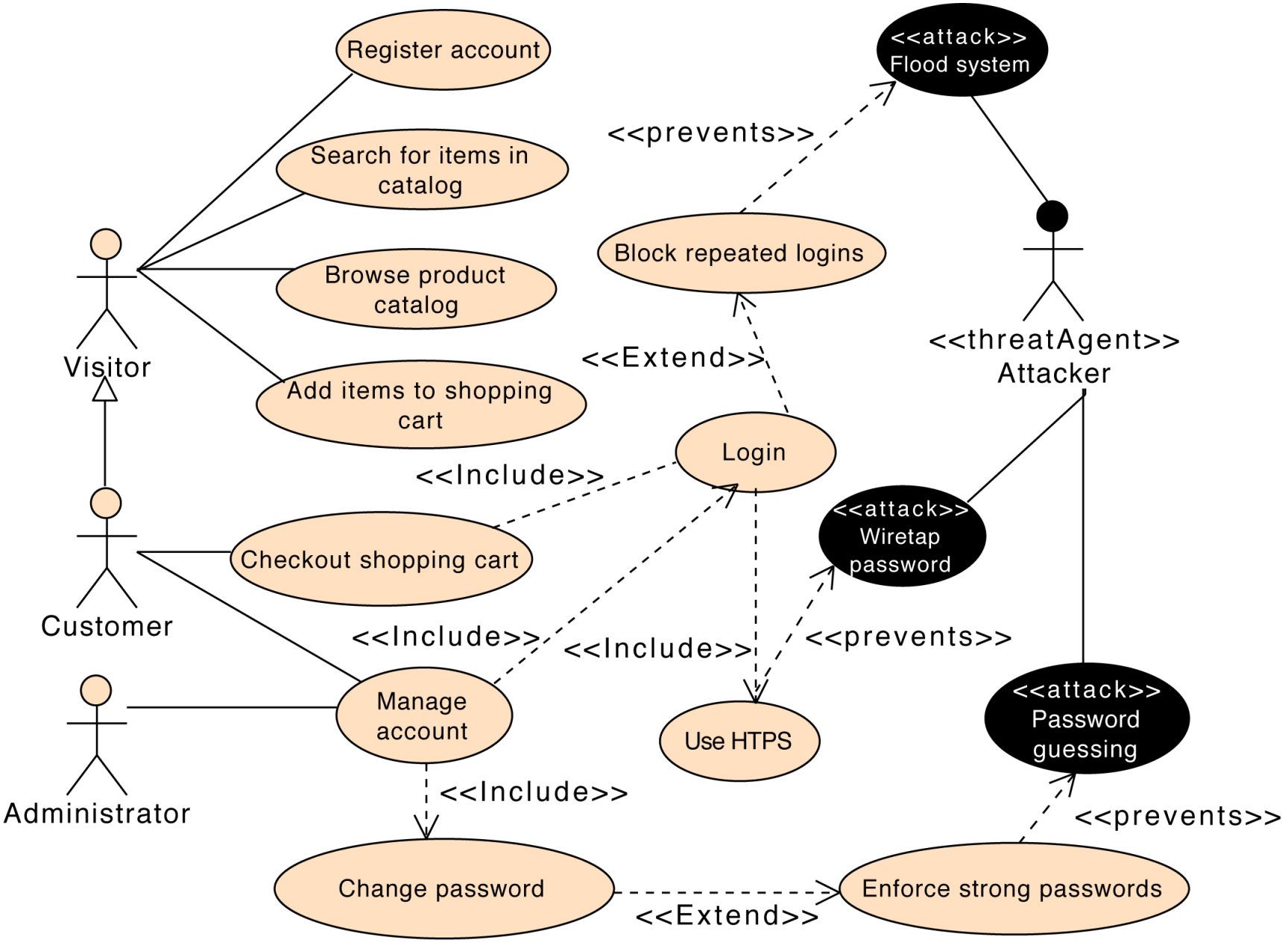
  Mixed include-relationships (example follows)
- Sometimes, legitimate actors do things unintentionally

[6] G. Sindre, A. Opdahl, *Eliciting Security Requirements by Misuse cases*. Proc. TOOLS-Pacific, pp. 120-131, 2000

Integration within one diagram



Example: more involved



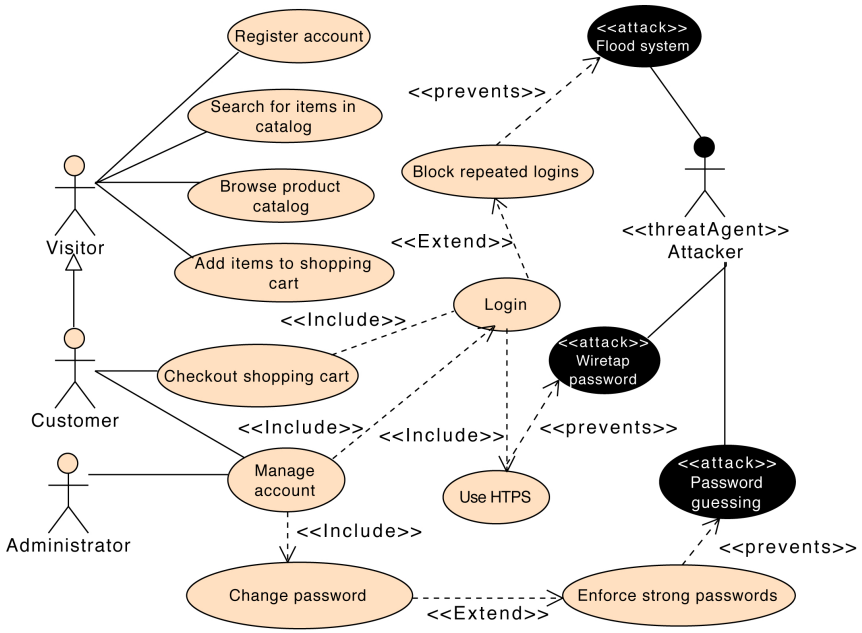
Relationships

- Those for use cases plus additional ones:

  prevents: functionality used to prevent misuse case
  detects: functionality used to detect abusive action

- Can add new relations, such as **may prevent**
- Can have multiple attackers: e.g., "attacker", "bad luck", ..
- Good guys can also have a dark side, representing insider attacks
- Problems
  - Precise semantics
  - Complexity of diagrams

Methodology



1. Normal actors $\rightsquigarrow$ main use cases
2. Main mis-actors $\rightsquigarrow$ main misuse cases
3. Potential relationships between use cases and misuse cases; concentrate on "includes" relationships

   E.g., login **includes** use of HTTPS to **prevent** wire tapping
4. Introduce new use cases that aim to detect or prevent misuse cases

   E.g., administrator monitors system, performs backups, etc.
5. Continue requirements analysis/specification, possibly iterating

3–5 are part of threat modeling & security design (coming up)

When do you stop?

- Misuse cases convey **what should not happen** at a high level

  E.g., steal credit card data, obtain password, or flood system

- How can these things happen?
  - Break them down into sub anti-goals and find causes
  - Do this recursively, thereby refining requirements

- Analysis involves assets, threats, vulnerabilities, technology, organization

- Limit to how far one goes during requirements analysis
  - Point is not to work out design here
  - Further refinement later, after design is given

We return to this when studying threat modeling and risk analysis.

Road map

- Requirements and Requirements Engineering
- Security-specific requirements
- Use cases and misuse cases
- ☞ **Data and process requirements**
- Conclusions

Domain and application specific requirements

- Standards and guidelines provide good starting points
- But they must be refined and augmented to cover concrete systems and the information they process
- This requires understanding the **criticality of data and systems**
- Usually formulated in terms of[7]
  - standard security objectives: confidentiality, integrity, availability
  - categories: high, moderate, low
- Requirements for systems typically based on "high-water marks" for the data they manipulate

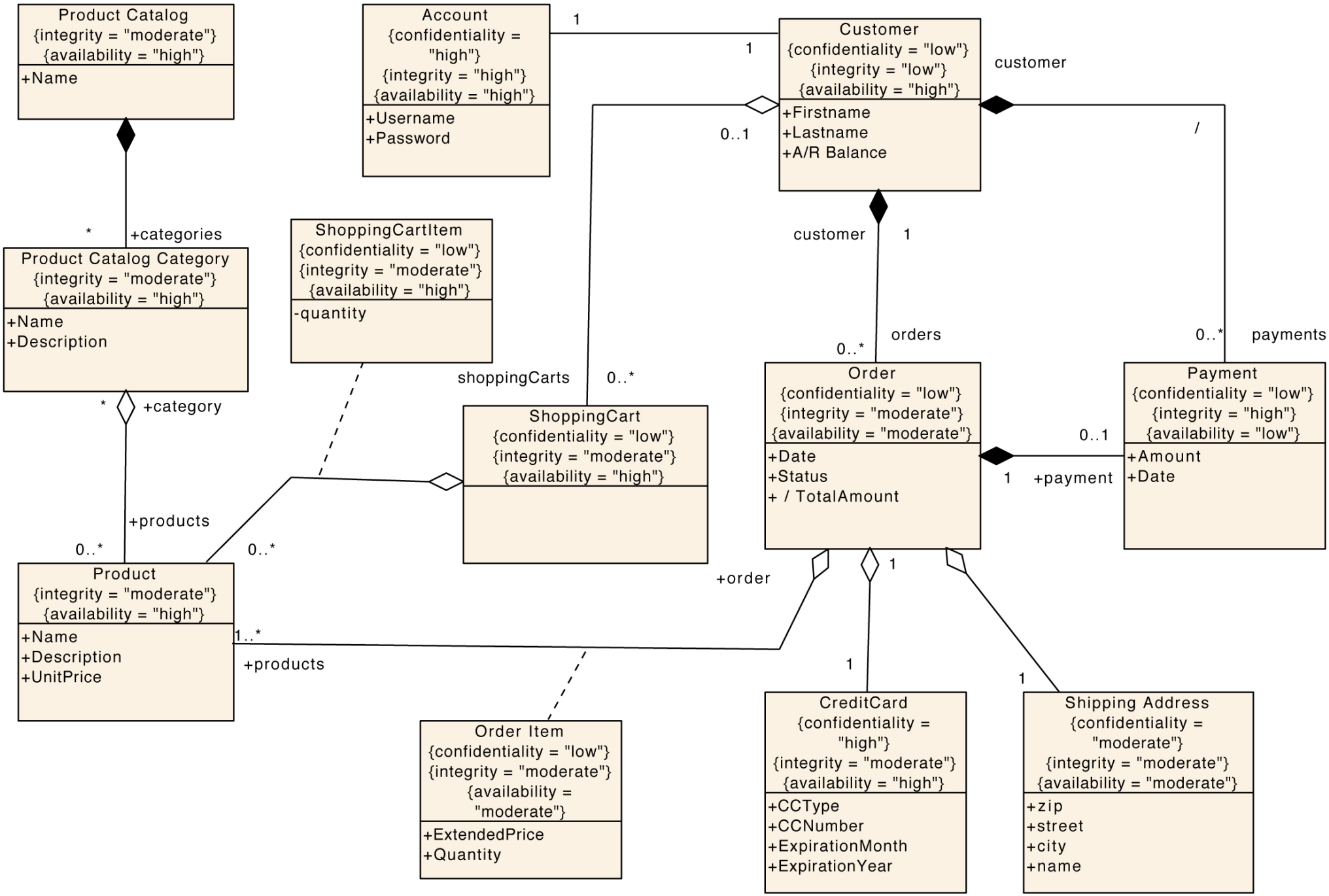
[7] See FIPS Standards for Security Categorization of Federal Information and Information Systems, csrc.nist.gov/publications/fips/fips199/FIPS-PUB-199-final.pdf.

Data criticality

Data Item	Confidentiality	Integrity	Availability
Catalog	N/A	Moderate	High
Category	N/A	Moderate	High
Product	N/A	Moderate	High
Customer	Low	Low	High
Account	High	High	High
Shopping Cart	Low	Moderate	High
Credit Card	High	Moderate	High
Payment	Low	High	Low
⋮	⋮	⋮	⋮

- Only possible after some design done, e.g., class-diagrams
- Table form or graphical model possible (next slide)
- Basis for security architecture for system/channels (later)
- Also for risk analysis (later)

Data criticality depicted using class models



Authorization policy

- Knowing which data is critical is not enough

  Which parties are authorized to perform which actions?

- Information access policy
  - **Confidential** information has restricted read access.
  - **Integrity** requirements yields restricted write access.
- Good default policy is based on least-privilege
  - Access only granted to those requiring it for their tasks (as indicated by use cases and other behavioral models)
  - User roles/functions taken from models
- Process/service execution: also as constrained by models

Authorization policy: information

Information	Visitor	Customer	Warehouse Clerk	Customer Care Clerk
Catalog	read	read		read
Category	read	read		read
Product	read	read		read
Item	read	read		read
Person		read & modify (own)		read, modify
OnlineAccount.Password		read & modify (own)		
OnlineAccount.A/R Balance				read
OnlineAccount.default		read (own)		read
ShoppingCart		read & modify (own)		
CreditCard		read & modify (own)		read
Payment		read (own)		read
Order		read (own)	read (resp), modify (resp)	read
ItemOrdered		read (own)	read (resp)	read
ItemPurchased				read

Authorization policy: processes

Process	Sub Process	Visitor	Customer	Warehouse Clerk
Browse Products		**X**	*X*	
	Use Catalogue search engine	*X*	*X*	
	Display catalogue	*X*	*X*	
	Display catalogue category	*X*	*X*	
	Display product	*X*	*X*	
	Display product items	*X*	*X*	
	Add item to shopping cart	*X*	*X*	
Register user account		**X**		
Login			**X**	
Checkout shopping cart				
	Maintain shopping cart		**X (own)**	
	Maintain order billing		**X (own)**	
	Maintain order shipping		**X (own)**	
	Confirm Order		**X (own)**	
Maintain User Account				
	Maintain user account data		**X (own)**	
	Confirm user account maintenance		**X (own)**	
Process order				**X**

Road map

- Requirements and Requirements Engineering
- Security-specific requirements
- Use cases and misuse cases
- Data and process requirements

☞ **Conclusions**

Conclusions

- Security requirement are both functional and nonfunctional!
- Standards and guidelines help with high-level formalization
- Models help to concretize the details

  But full details usually only present later after design
- Models also useful for risk analysis

  And for updating requirements for future releases

Literature

- B. Kovitz: Practical Software Requirements, Manning, 1999
- Advisories such as http://www.cert.org/advisories/
- I. Jacobson: Object Oriented Development in an Industrial Environment, Proc. OOPSLA, pp. 183-191, 1987
- G. Sindre, A. Opdahl: Eliciting Security Requirements by Misuse cases. Proc. TOOLS-Pacific, pp. 120-131, 2000
- J. Musa: Software Reliability Engineering, AuthorHouse, 2004