SE463
Software Requirements Specification & Analysis

User Requirements

Readings:

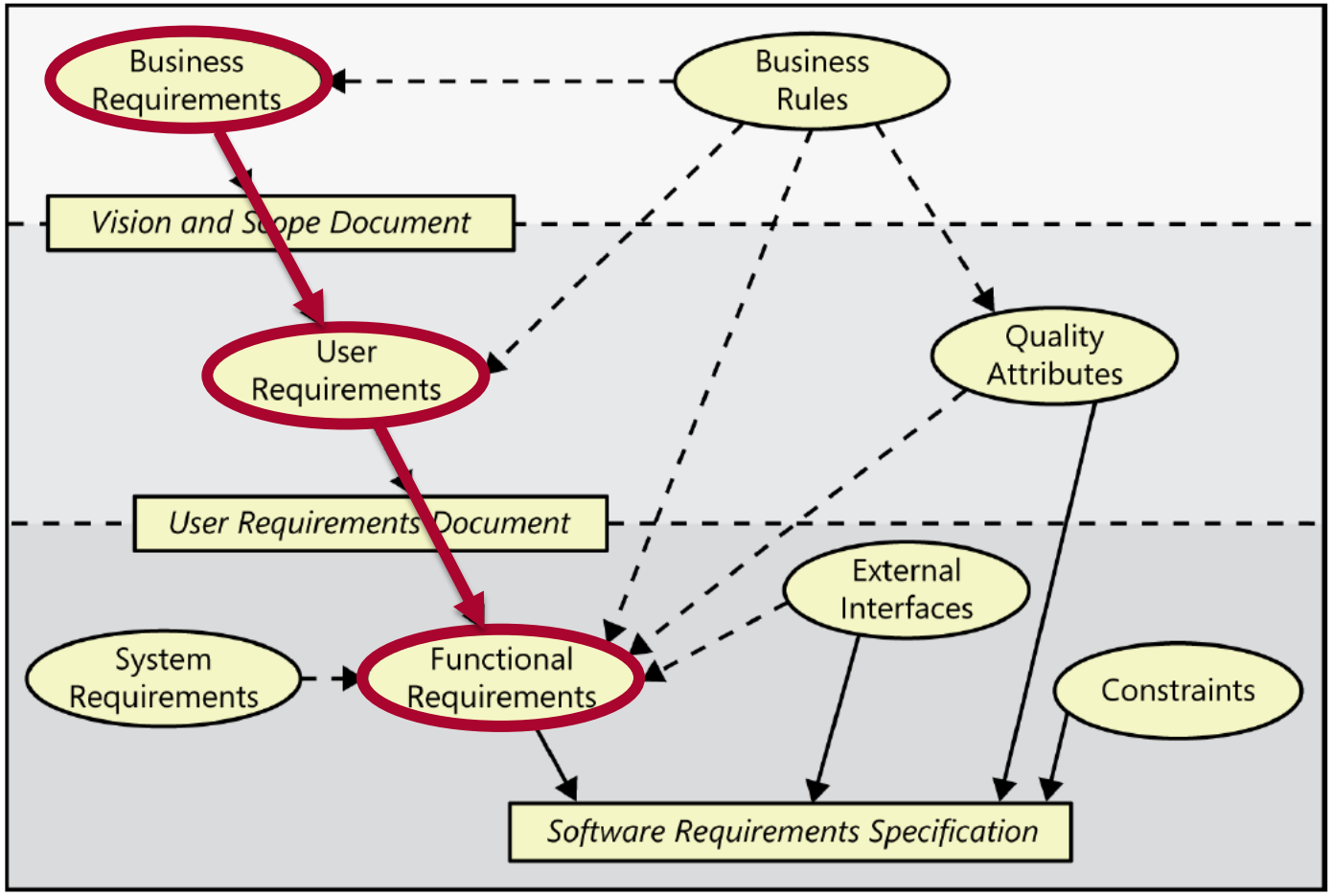
Karl E. Wiegers and Joy Beatty. *Software Requirements, 3ed.* Microsoft Press, 2013. Chapter 8: "Understanding user requirements"

Larman, C., *Applying UML and Patterns*, 3ed, 2004. Chapter 6: "Use Cases"

Module Objectives

- Types of requirements
- Use cases, context diagrams, user stories
- Avoiding scope creep

Types of Requirements



Airline wants to reduce counter staff costs by 25%

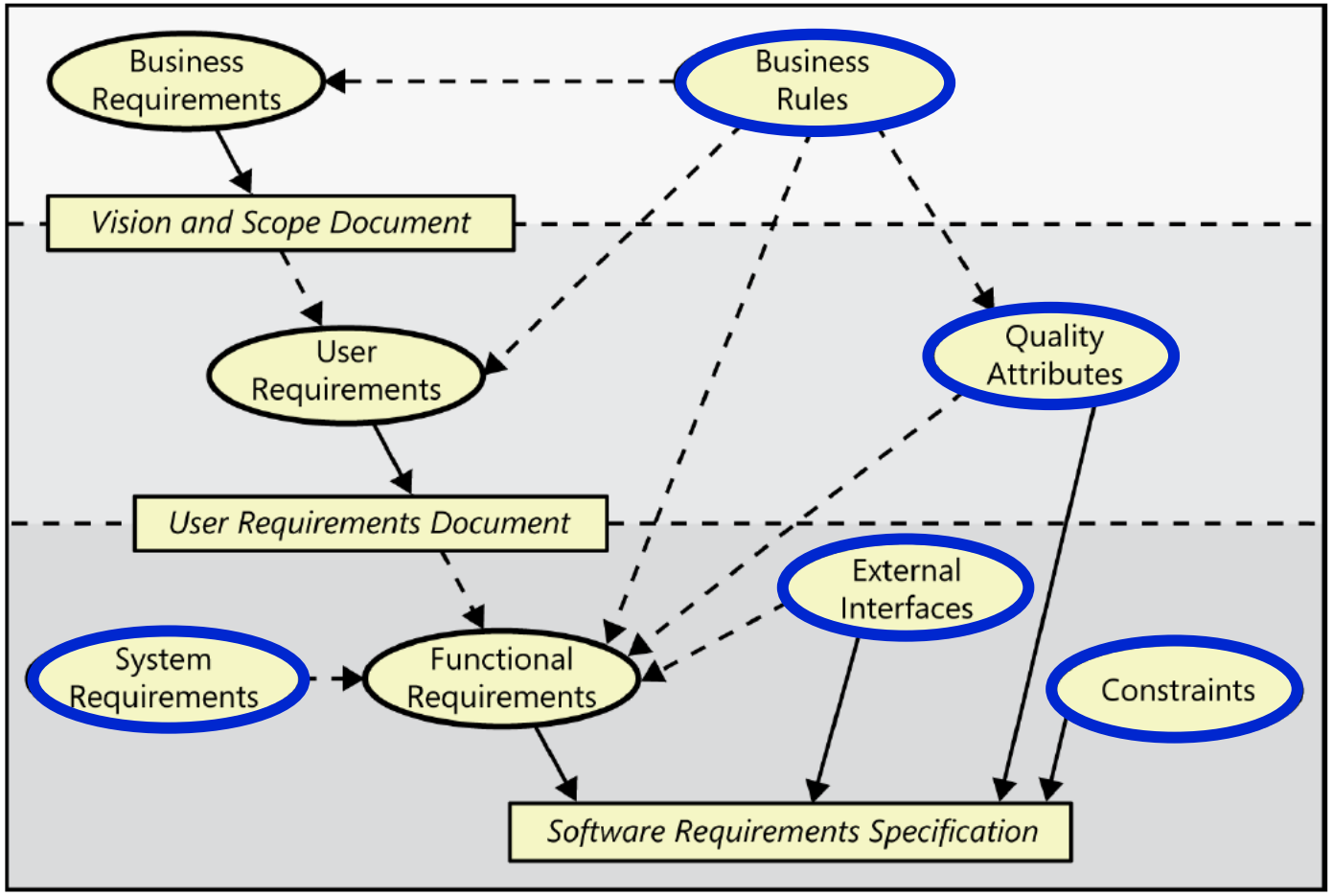
Passengers check in for a flight using an airport kiosk

Kiosk prints boarding passes upon successful check-in

FIGURE 1-1 Relationships among several types of requirements information. Solid arrows mean "are stored in"; dotted arrows mean "are the origin of" or "influence."

Karl E. Wiegers and Joy Beatty. *Software Requirements, 3ed.* Microsoft Press, 2013.

Types of Requirements



Airline wants to reduce counter staff costs by 25%

Passengers check in for a flight using an airport kiosk

Kiosk prints boarding passes upon successful check-in

FIGURE 1-1 Relationships among several types of requirements information. Solid arrows mean "are stored in"; dotted arrows mean "are the origin of" or "influence."

Karl E. Wiegers and Joy Beatty. *Software Requirements, 3ed.* Microsoft Press, 2013.

Use Cases

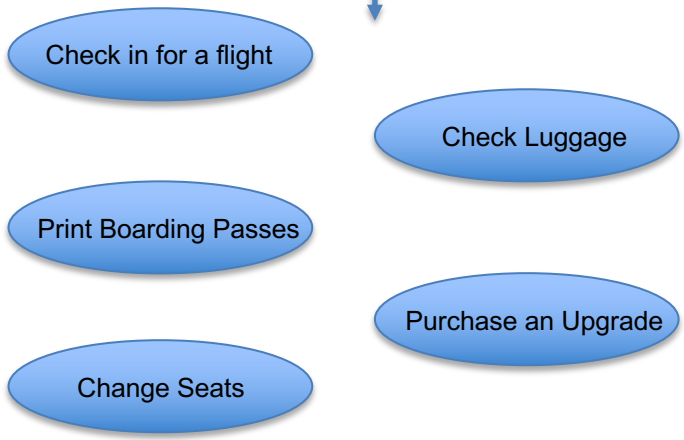
Decompose The Work into vertical slices to reduce complexity.

Each slice is called a use case

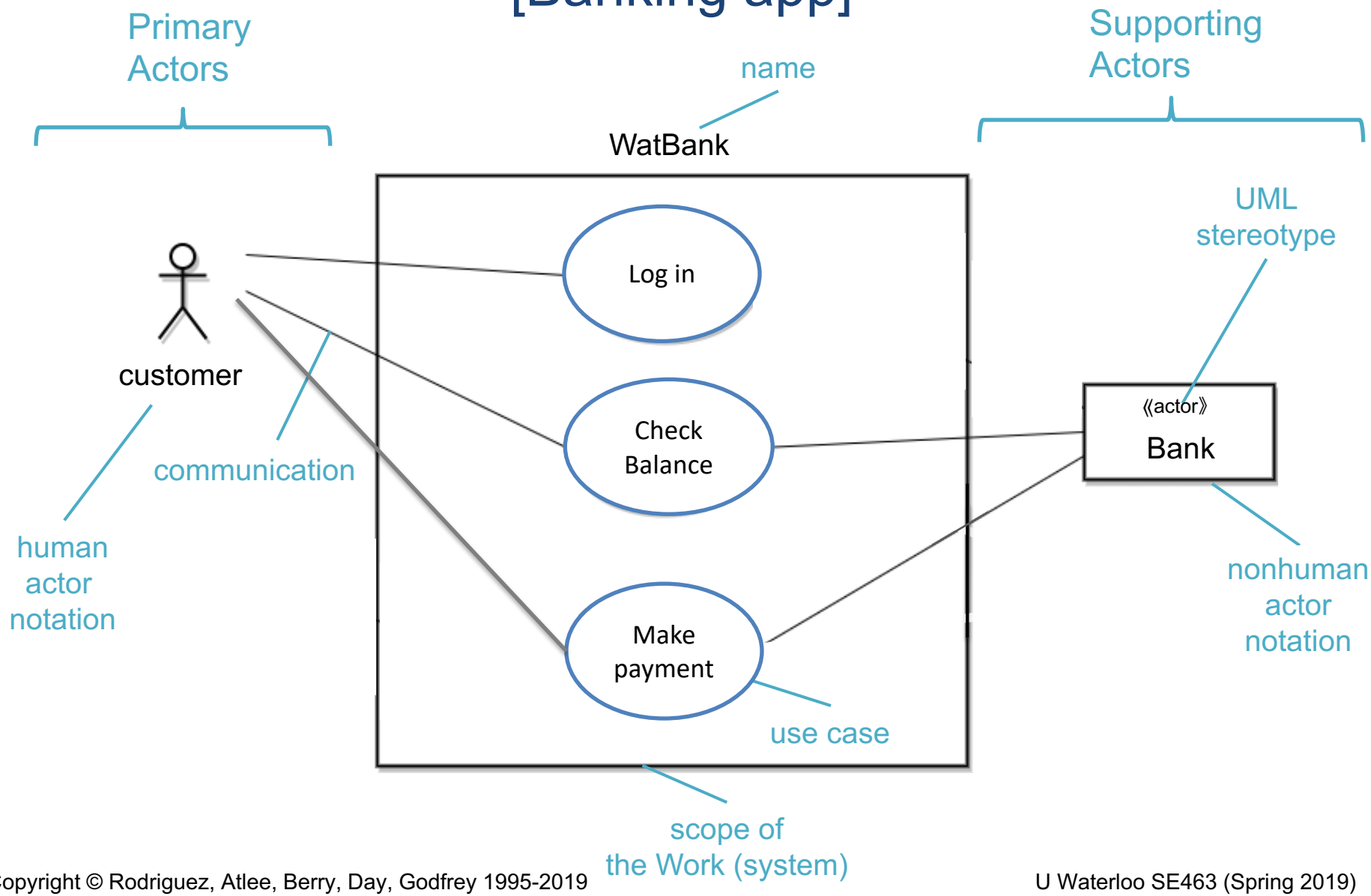
- represents some end-to-end functionality
- triggered by an external event (e.g., from adjacent system)
- captures a complete response to a triggering event
- use cases are (ideally) orthogonal to one another

Use Case Diagrams

System: Airport check-in Kiosk	Use cases
	Check-in for a flight
	Print boarding pass
	Change seats
	Check Luggage
	Purchase an upgreade



Use Case
[Banking app]

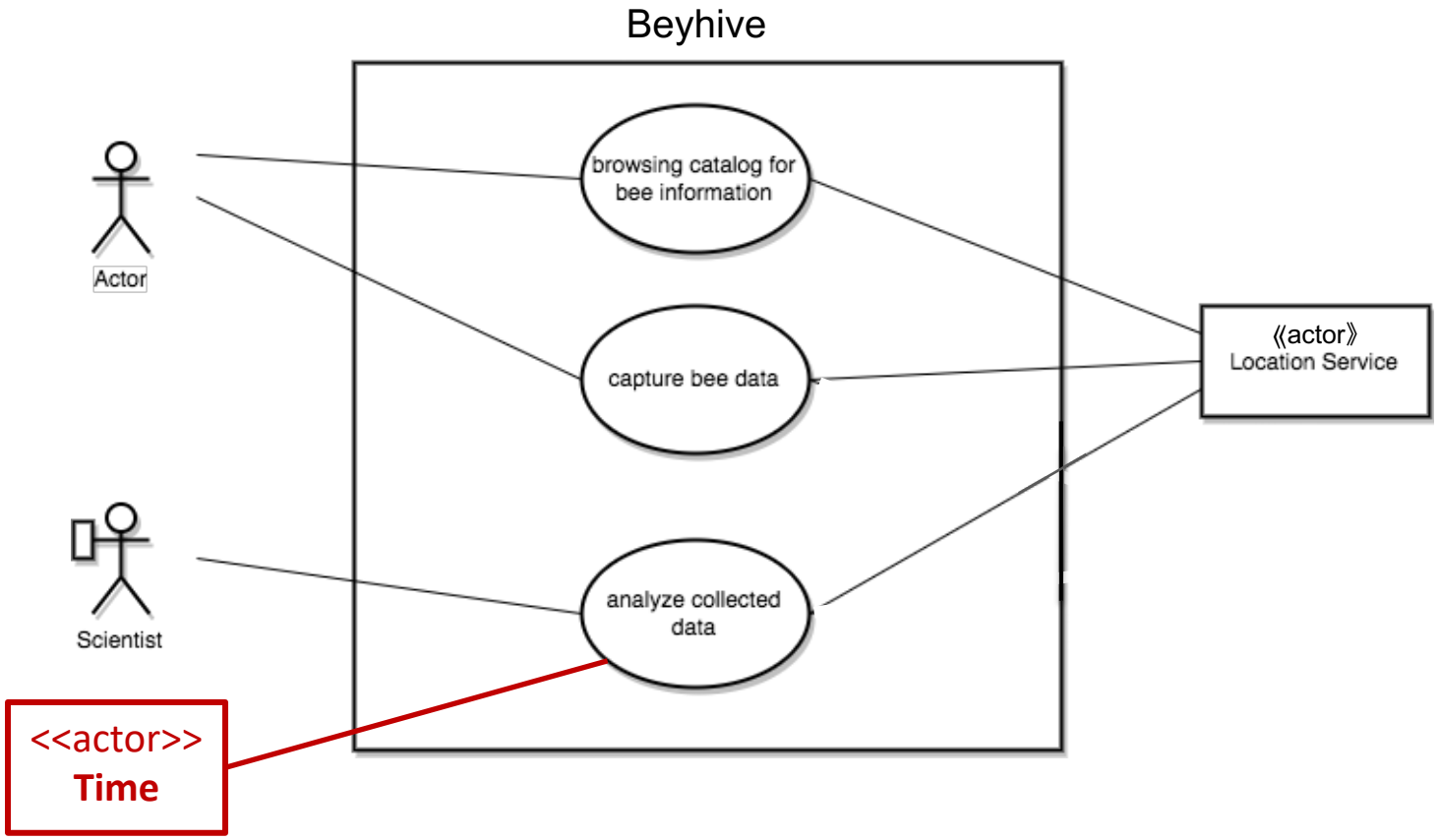


Actors

- A *primary actor* is one that initiates a use case
  - E.g., Users, Time
- A *supporting actor* provides services to the use case (Work)
  - May be invoked by the Work, or may monitor the Work and react to events triggered by the use case
- Same person can play multiple roles
- The actors are the real-world actors – sources of environmental phenomena, expressed in the requirements, not the specification.
  - So, prefer “Scientist” and “User” to “UI”
- Focus on roles rather than individuals
  - E.g. user, administrator, maintenance

Time-Triggered Use Case

Time-triggered use cases are activated when a date or time comes to pass.



Example: Patient Monitoring System

Patients in an intensive-care ward in a hospital are monitored by electronic analog devices attached to their bodies by sensors of various kinds.

- Through the sensors, the devices measure the patient's vital factors: pulse rate, temperature, blood pressure, and so on.
- A program is needed to read the factors, at a frequency specified for each patient, and store them in a database.
- The factors read are to be compared with safe ranges specified for each patient, and readings that exceed the safe ranges are to be reported by alarm messages displayed on the screen of the nurse's station.

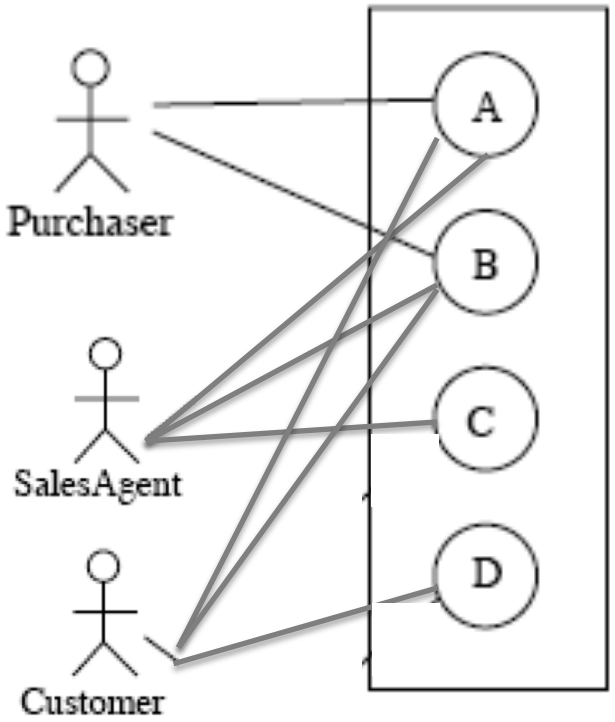
Stevens, Myers, and Constantine, *IBM Systems Journal*, 1974

Actor Generalization

- Use actor generalization when actors have common *interesting* behaviour

  i.e., they interact with many of the same use cases

- Factor out common behaviour as an abstract actor
  - Children inherit all relationships with use cases of the parent

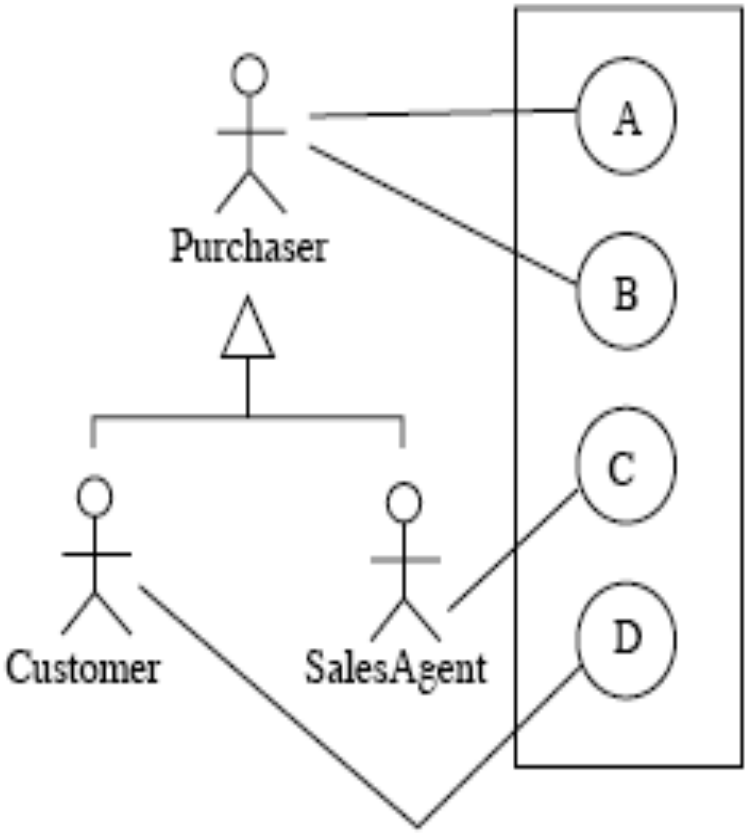


Actor Generalization

- Use actor generalization when actors have common *interesting* behaviour

  i.e., they interact with many of the same use cases

- Factor out common behaviour as an abstract actor
  - Children inherit all relationships with use cases of the parent



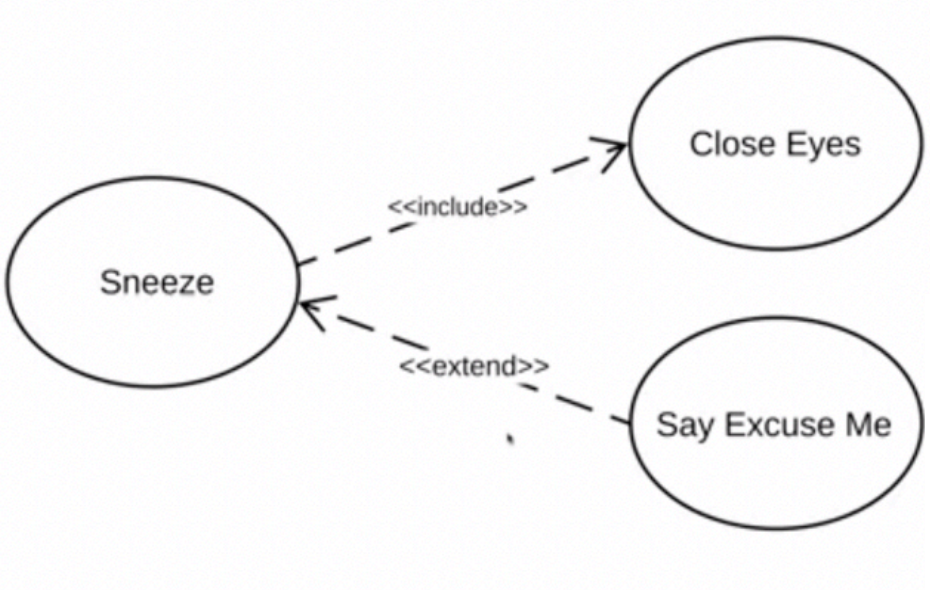
«include»

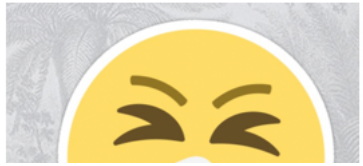
- <<include>> - a sub use case that is used within multiple other use cases (like a procedure call)
  - Purpose is to highlight essential functionality that is part of multiple use cases
  - Avoids repetition of the same steps in multiple use cases, improving readability
  - Specify point of inclusion in the base use case
  - When sub use case completes, control returns to the base use case

«extend»

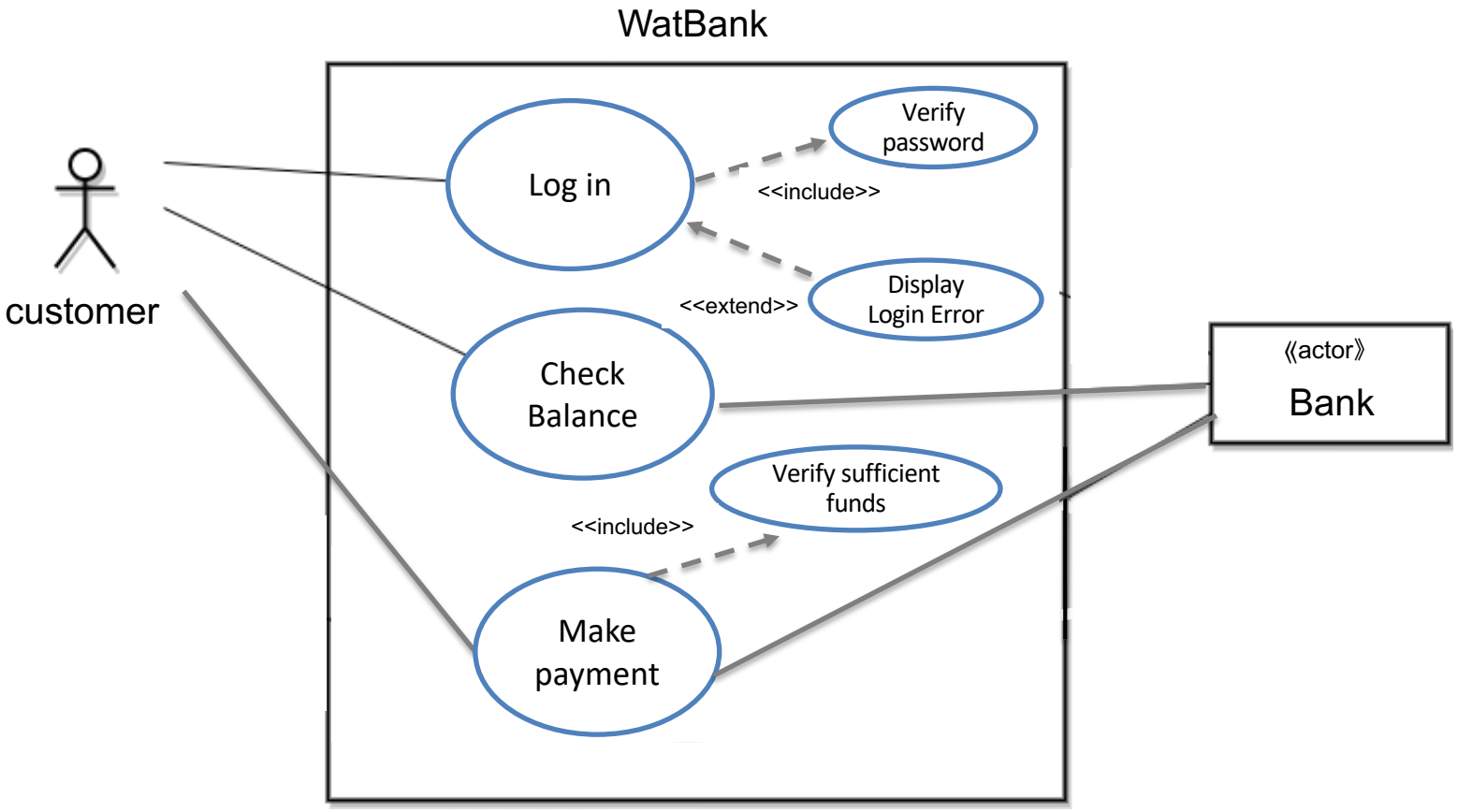
- <<extend>> - a sub use case that extends or replaces the end of an existing use case
  - Purpose is to highlight new functionality that extends an existing use case (cf. adding a new use case)
  - Base use case has hooks where it can be extended
  - Unlike «include», base case is complete without extension use case

Example





Example

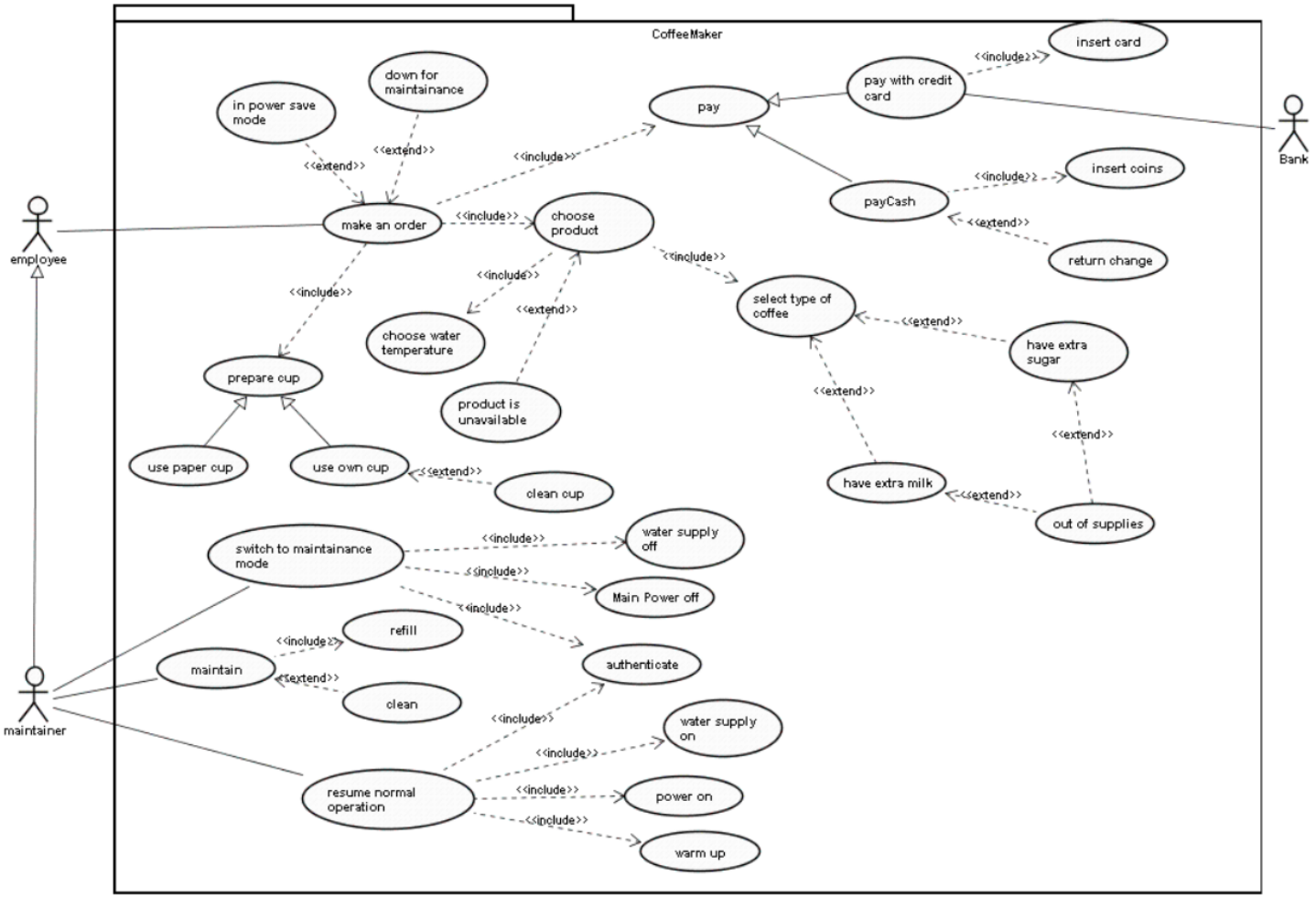


Use Case Modelling: Goals

- Use case modelling is really as simple as you think. That way, nontechnical people can understand it.
- A good use case model is simple, without too much detail. Be careful and not make this mistake!! Abstraction is your friend
- The main purpose is “discovery”, mapping out the base-level of the system with the client
  - Tho invention and negotiation are key too, not just “discovery”

Bad Example (from the Web)

Use <<include>> and <<extend>> sparingly.



http://www.iai.uni-bonn.de/III/lehre/vorlesungen/SWT/OOSC06/exercises/exercise2.html

Use-Case Description

We augment use-case diagrams with use-case descriptions (a textual description)

"Brief" format:

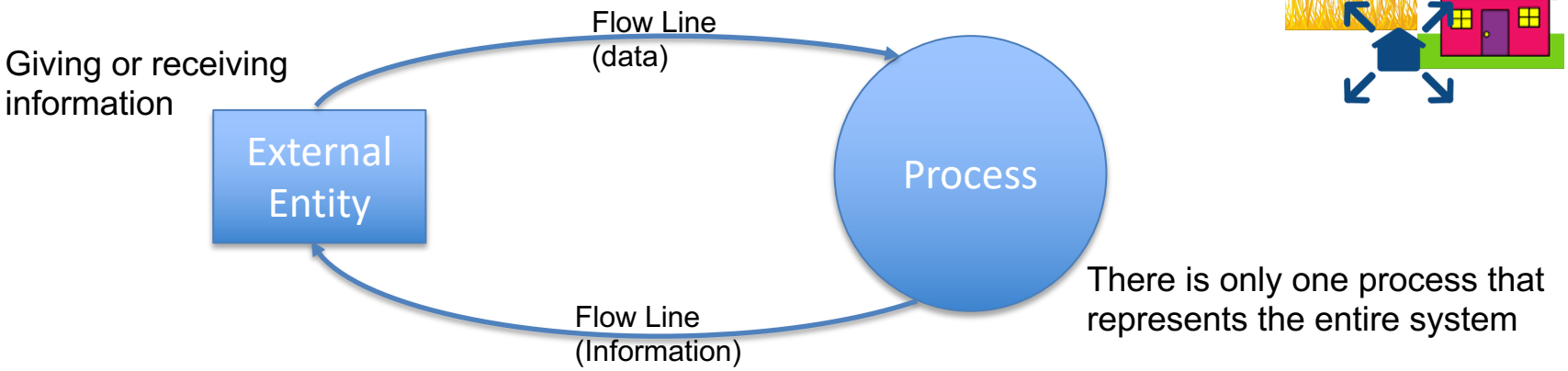
- *UC3: Order blood – Customer submits blood order and payment info. System processes payment and sends shipment order to ShippingDept.*

"Casual" format:

- *UC3: Order blood – Customer submits blood order and payment info (invoice or credit card). System verifies availability of blood. If availability OK, System processes payment. If payment by CC, then checks with CCAuthService. If payment by invoice, then verifies Customer status with AccountingDept. If payment OK then System sends shipment order to ShippingDept. If availability or payment problems, then notify customer of details.*

"Fully dressed" format: scenario (in another lecture)

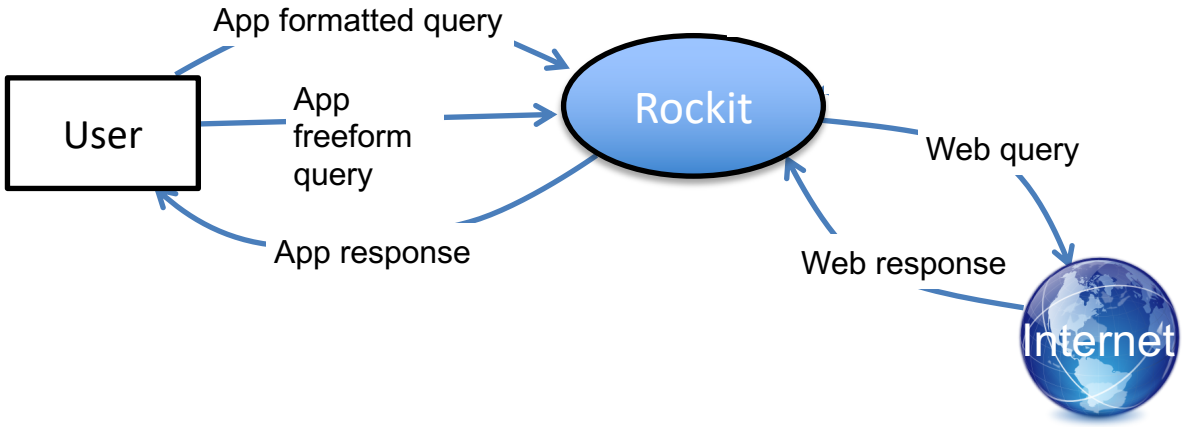
Context Diagram



A context diagram is a graphical representation of

- the boundary between the Solution and external entities
- information flows between them

Context Diagram
[Rockit]

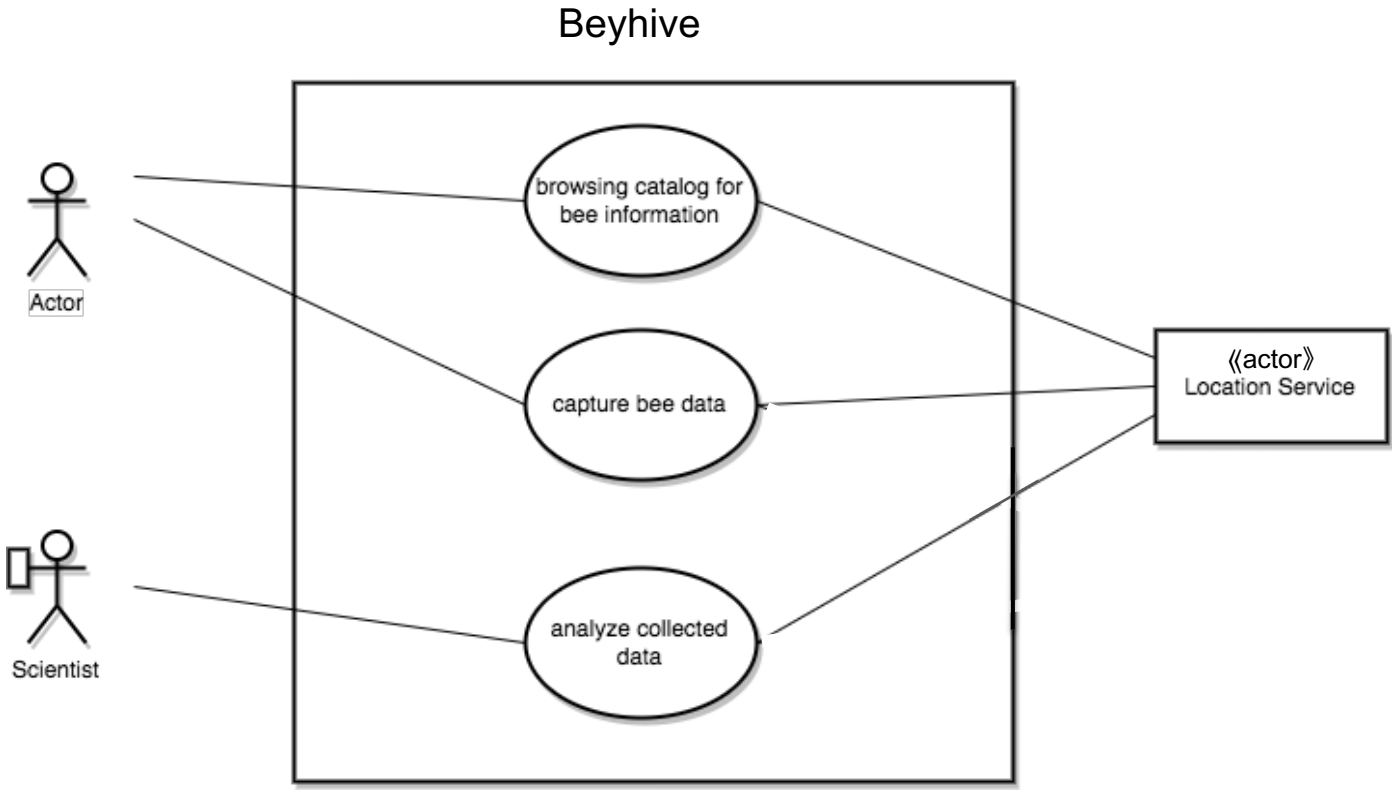


Context Diagram:Exercise

Patients in an intensive-care ward in a hospital are monitored by sensors of various kinds (i.e., by electronic analog devices attached to their bodies). Through the sensors, the devices measure the patient's vital factors: pulse rate, temperature, blood pressure, and so on. A program is needed to read the factors, at a frequency specified for each patient, and store them in a database. The factors read are to be compared with safe ranges specified for each patient, and readings that exceed the safe ranges are to be reported by alarm messages displayed on the screen of the nurse's station.

Context Diagram: Exercise

Users learn about important bee facts. They contribute to the repository of bee information with pictures of bees.
Scientist can access the repository to view aggregated data.



User Stories

User stories are an alternative approach to describing use cases.

Each user story is a short simple "story" (description) of one thing that the user wants to be able to do.

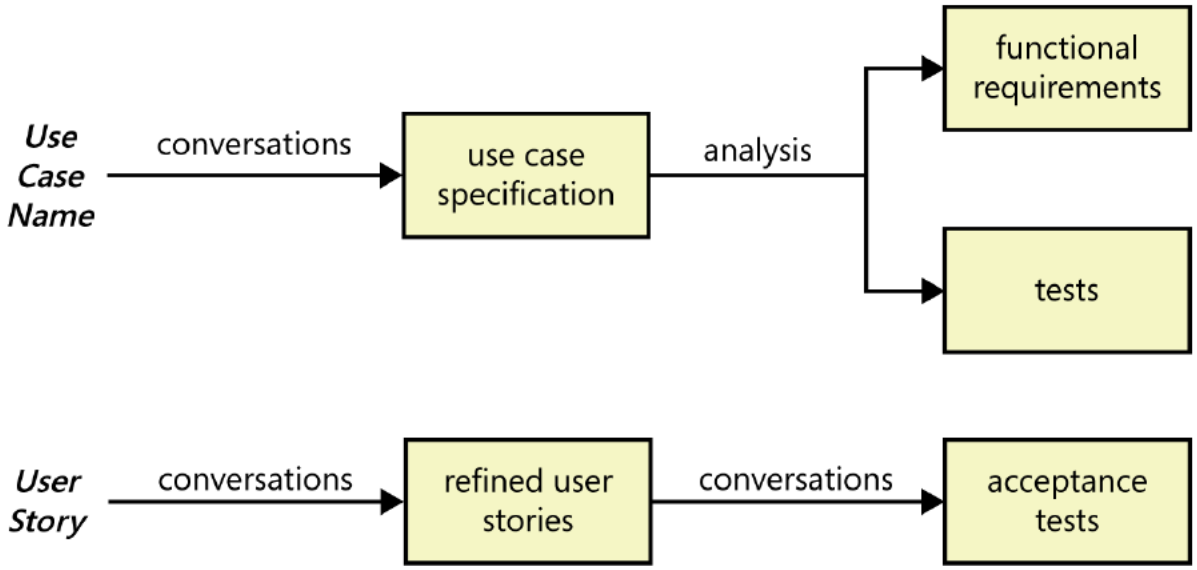


FIGURE 8-1 How user requirements lead to functional requirements and tests with the use case approach and the user story approach.

Karl E. Wiegers and Joy Beatty. *Software Requirements, 3ed.* Microsoft Press, 2013.

User Stories

User stories provide a light-weight approach to managing requirements:

- Short statement of new functionality or feature
- Written from the point of view of the user
- Its details are fleshed out just in time before the story is placed in the sprint.

Three C's of User Stories

- **Card** - stories are traditionally written on note cards, using a structured syntax:

  As a <role>, I want <something>, so that <benefit is achieved>

As a vacation traveller,
I want to see photos of hotels,
so that I can get a sense of the quality of the hotel

As a user, I want to cancel a reservation, so that my credit card is not charged

Three C's of User Stories

- **Card** - stories are traditionally written on note cards, using a structured syntax:

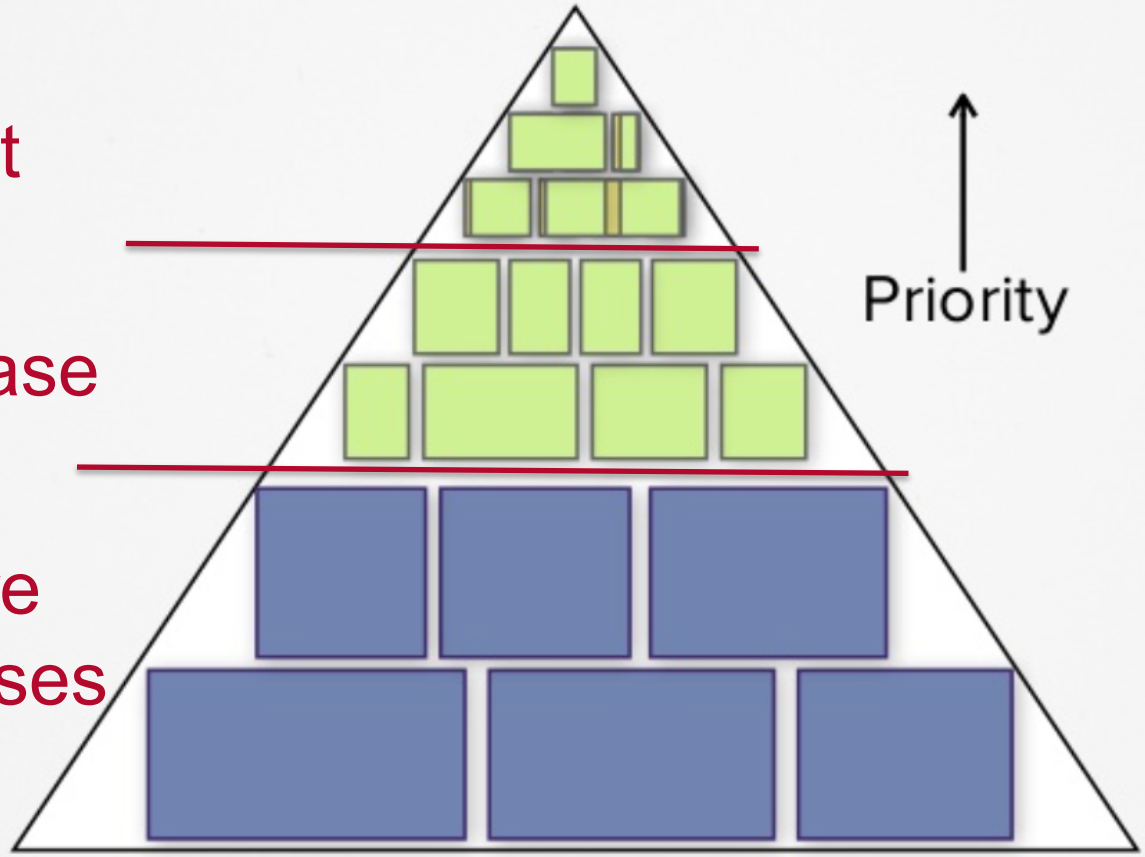
  As a <role>, I want <something>, so that <benefit is achieved>

> *As a vacation traveller,*
> *I want to see photos of hotels,*
> *so that I can get a sense of the quality of the hotel*

> *As a user, I want to cancel a reservation, so that my credit card is not charged*

- **Conversation** – discussions with the product owner reveal details of the requirements

The product backlog iceberg



Three C's of User Stories

- **Card** - stories are traditionally written on note cards, using a structured syntax:
  As a <role>, I want <something>, so that <benefit is achiev

> *As a vacation traveller, I want to see photos of hotels, so that I can get a sense of the quality of the hotel*

> *As a user, I want to cancel a reservation, so that my credit card is not charged*

- **Conversation** – discussions with the product owner reveal details of the requirements
- **Confirmations** – acceptance criteria for objectively determining whether an implementation meets the requirements.

Examples

- As a first time book buyer I want to find the perfect mystery novel by reading staff reviews so I won't waste my money reading bad books
- As a staff member I want to enter book reviews so I can delight book buyers
- As a first time book buyer I want to see only books that get five star reviews so I don't waste my time reading reviews about weaker books
- As a first time book buyer I want to select a single book for purchase so that I can read it *(Mark L. – the team have elegantly avoided solving the whole shopping cart problem for the moment by handling just a single book. Clearly they will later need to handle multiple books but if they can't get one book home they can't get more home).*
- As a first time book buyer I want to ship my book home within two days so that I can get it before my next flight

Where are the details?

- As a user, I can cancel a reservation.
  - Does the user get a full or partial refund?
    - Is the refund to her credit card or is it site credit?
  - How far ahead must the reservation be cancelled?
    - Is that the same for all hotels?
    - For all site visitors? Can frequent travelers cancel later?
  - Is a confirmation provided to the user?
    - How?

Details as Conditions of Satisfaction

User stories are told from the point of the user, and represent user requirements

Conditions of Satisfaction are told from the point of the system. They represent functional requirements of the system that help to ensure that the system meets the user requirements.

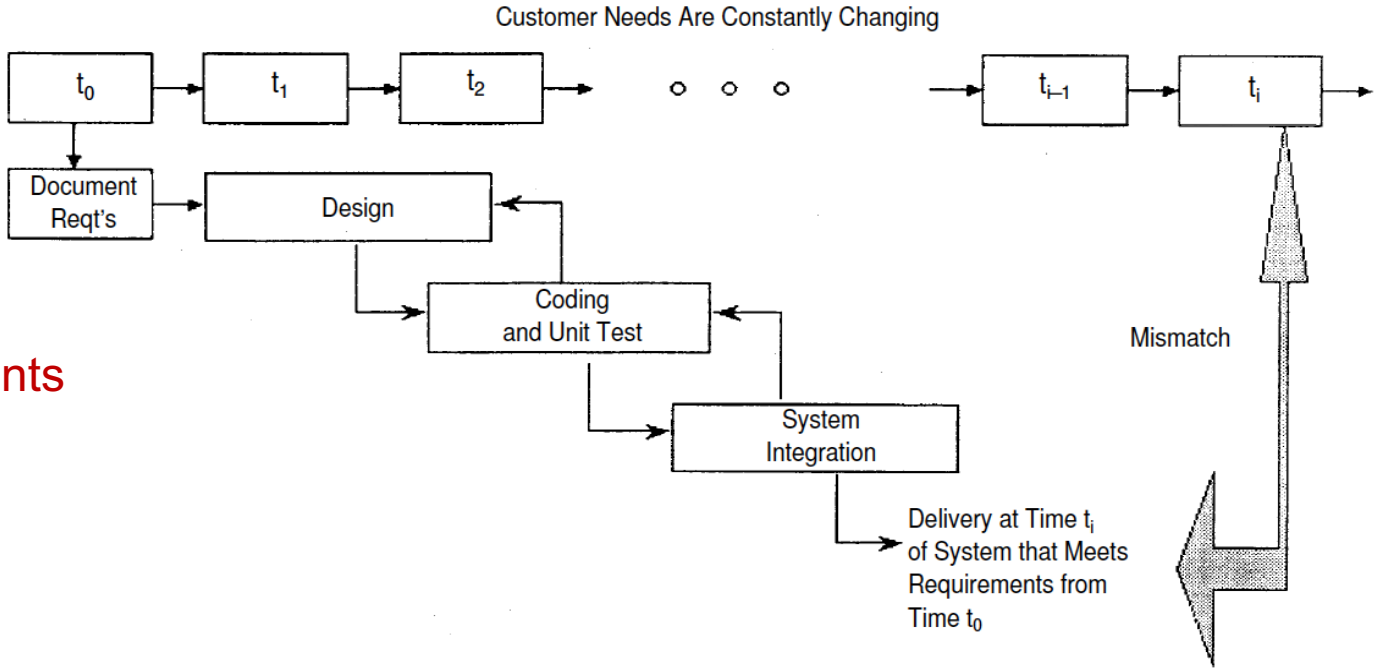
- we'll cover functional requirements later in the term

Why User Stories?

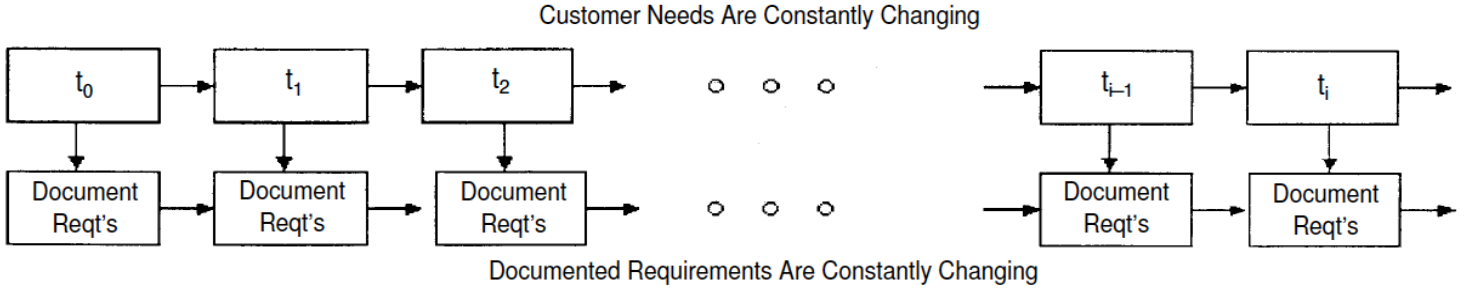
- Easy for stakeholders to understand, and to remember.
- Shift the focus from written requirements documentation to discussion.
- Encourage iterative development, with stories being appropriate sized increments for planning
- Delay the elicitation of requirements details until just before development.
- Support participatory elicitation

Changing Requirements

Requirements Mismatch



Analysis Paralysis



Alan Davis, *Great Software Debates*, Wiley, 2004

Changing Requirements

Three aides for dealing with changing requirements

1) Requirements baseline

- Good enough to proceed to design with an acceptable level of risk
- Formally reviewed and agreed on
- Subsequent changes managed through change-control process
- Rough guide: limit changes to < 0.5% per month (6% per year)

2) Unique Value Proposition

3) Project scope

Keeping the Scope in Focus

Unique value proposition, project priorities, and project scope should be used to vet each new requirement:

- If out of scope, then should file away for a future release or project
- If in scope, then can consider incorporating it, if it is high priority relative to already-committed requirements
- If out of scope but too good to ignore or defer, can consider broadening the project scope, and make updates to
  - Project objectives and scope
  - Project budget, schedule, and/or staff

Scope in Agile Projects

In agile projects, scope creep is addressed by keeping prioritized requirements in the backlog (future topic) and allocating a few requirements to each release.

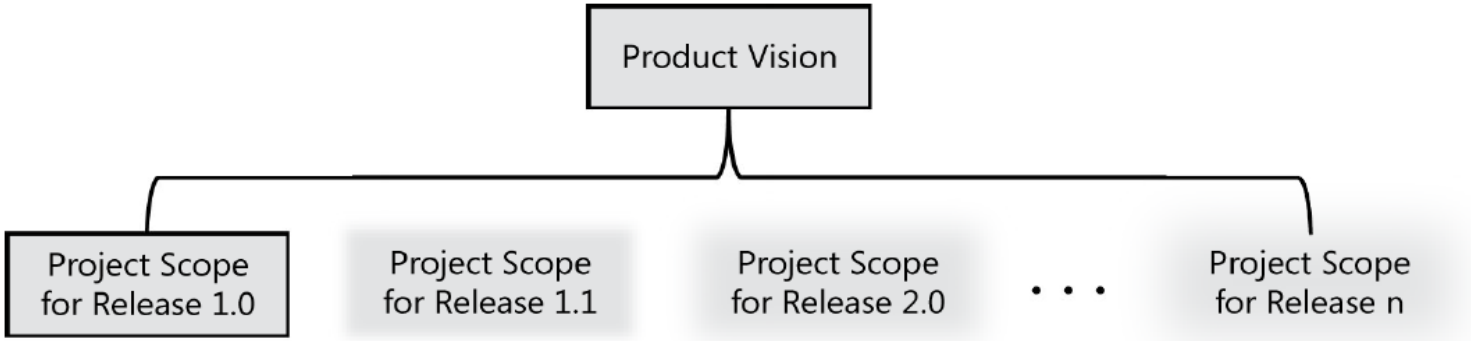


FIGURE 5-1 The product vision encompasses the scope for each planned release, which is less well defined the farther out you look.

Karl E. Wiegers and Joy Beatty. *Software Requirements, 3ed.* Microsoft Press, 2013.

Summary

Use Cases – decompose the work into work-pieces to manage complexity

Use-Case Diagram – expresses use cases in a manner that is easy for all stakeholders to visualize

- Big picture view of the problem

Context Diagrams – shows information flow between the Solution and external entities

User Stories – express user requirements from the perspective of the user and their motivations