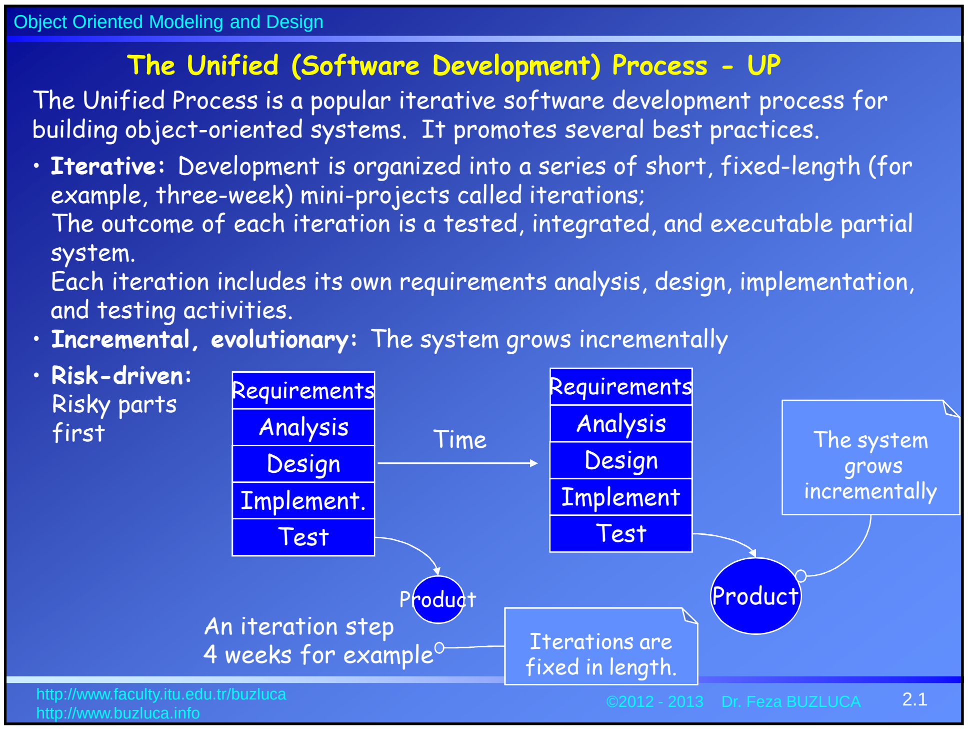
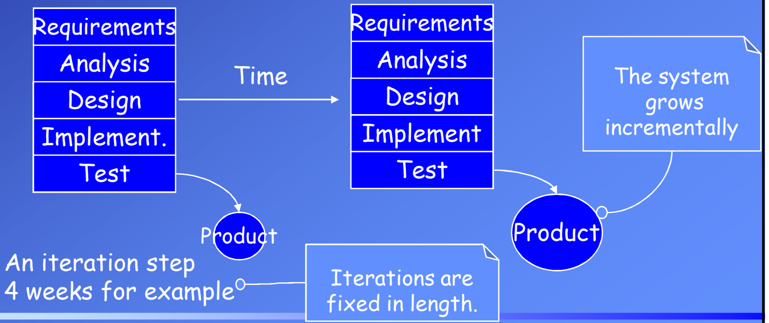
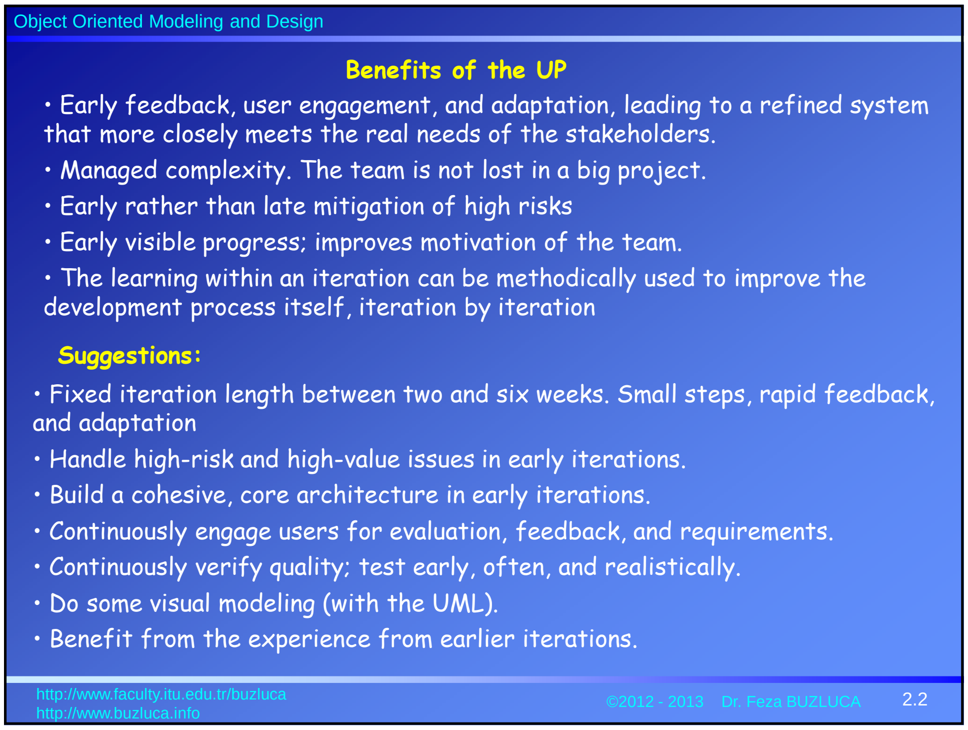


The Unified (Software Development) Process - UP

The Unified Process is a popular iterative software development process for building object-oriented systems. It promotes several best practices.

- **Iterative:** Development is organized into a series of short, fixed-length (for example, three-week) mini-projects called iterations;
  The outcome of each iteration is a tested, integrated, and executable partial system.
  Each iteration includes its own requirements analysis, design, implementation, and testing activities.
- **Incremental, evolutionary:** The system grows incrementally
- **Risk-driven:** Risky parts first


Requirements
Analysis
Design
Implement.
Test
Time
Requirements
Analysis
Design
Implement
Test
The system grows incrementally
Product
Product
An iteration step 4 weeks for example
Iterations are fixed in length.



Benefits of the UP

- Early feedback, user engagement, and adaptation, leading to a refined system that more closely meets the real needs of the stakeholders.
- Managed complexity. The team is not lost in a big project.
- Early rather than late mitigation of high risks
- Early visible progress; improves motivation of the team.
- The learning within an iteration can be methodically used to improve the development process itself, iteration by iteration

Suggestions:

- Fixed iteration length between two and six weeks. Small steps, rapid feedback, and adaptation
- Handle high-risk and high-value issues in early iterations.
- Build a cohesive, core architecture in early iterations.
- Continuously engage users for evaluation, feedback, and requirements.
- Continuously verify quality; test early, often, and realistically.
- Do some visual modeling (with the UML).
- Benefit from the experience from earlier iterations.

UP Phases

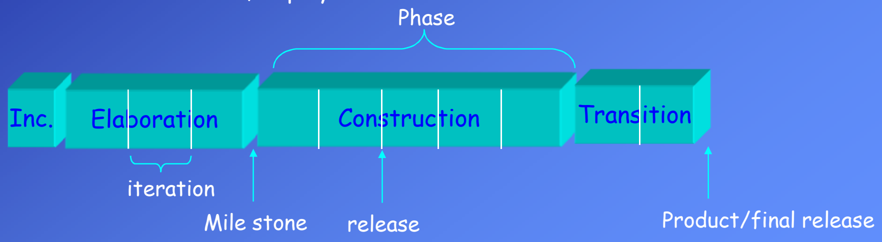
A software development project consists of 4 phases, each of them including some iterations.

1. **Inception:** Approximate vision, business case, scope, vague estimates, feasibility. Continue or stop?

2. **Elaboration:** Refined vision, iterative implementation of the core architecture, resolution of high risks, identification of most requirements and scope, more realistic estimates.

3. **Construction:** Iterative implementation of the remaining lower risk and easier elements, and preparation for deployment

4. **Transition:** Beta tests, deployment.



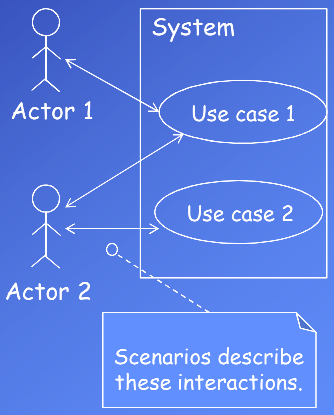
Defining Requirements, Use-Case Model

Use cases are text stories (collection of scenarios), used to discover and record requirements.

They are written with the user (customer) of software.

Use cases are not just documents. They ensure the understanding of the requirements, the cooperation with the customer and verification of the system.

Without use cases we cannot know what the system should do.



Originated by Ivar Jacobson (1939-), Swedish engineer.

He is also known as major contributor to UML, Rational Unified Process and aspect oriented programming.

http://blog.ivarjacobson.com/ivarblog/

Definitions

Use Case:

•"A use case specifies a sequence of actions, including variants, that a system performs and that yields an observable result of value to a particular actor."

(Three amigos: Jacobson, Booch, Rumbaugh 1999)

• "A use case is a collection of possible sequences of interactions between the system under discussion and its external actors, related to a particular goal."

(Cockburn 2000)

Scenario:

A scenario is a specific sequence of actions and interactions between actors and the system.

Each scenario is only one particular story of using a system.

There can be many possible scenarios in a system.

For example in a ticket selling system one scenario can consist of steps buying a ticket successfully; and another scenario can describe steps of failing to find a place to concert.

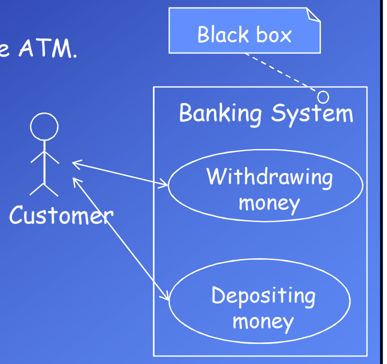
A use case is a collection of related success and failure scenarios that describe an actor using a system to support a goal.

Example:

Withdrawing money from a bank account using the ATM.

One of the possible scenarios:

1. Customer inserts the Bank Card.
2. The system prompts for a PIN.
3. Customer enters the PIN.
4. System validates the PIN.
5. System displays different transaction alternatives.
6. Customer selects money withdraw.
7. System prompts for amount.
8. Customer enters the amount.
9. Customer selects money withdraw.
10. System dispense money, and receipt.
11. System updates the account.
12. System returns the Bank Card.



This is only one of the possible scenarios.

Another scenario may performed if the PIN of the customer is not valid, or if the customer has not sufficient amount of money.

Definitions (cont'd)

Actor:

An actor is anything with behavior, such as a person (identified by role), computer system, or organization that interacts with the system under discussion.

There are three kinds of external actors in relation to the system under discussion.

- **Primary actor** is the main user who has goals fulfilled through using services of the system under discussion.
  For example, the cashier in the POS system.
- **Supporting actor** provides a service (for example, information) to the system.
  The credit card authorization service is an example. Often a computer system, but could be an organization or person.
- **Offstage actor** has an interest in the behavior of the use case, but is not primary or supporting.
  For example, a government tax agency.

Case Study: The NextGen POS System

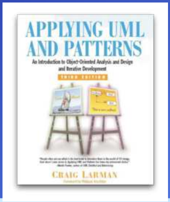
Most of examples in this course are given on a case study about a point of sale (POS) system named as "NextGen".

A POS system is a computerized application used to record sales and handle payments; it is typically used in a retail store.

It includes a keyboard, a bar code scanner, and a printer.

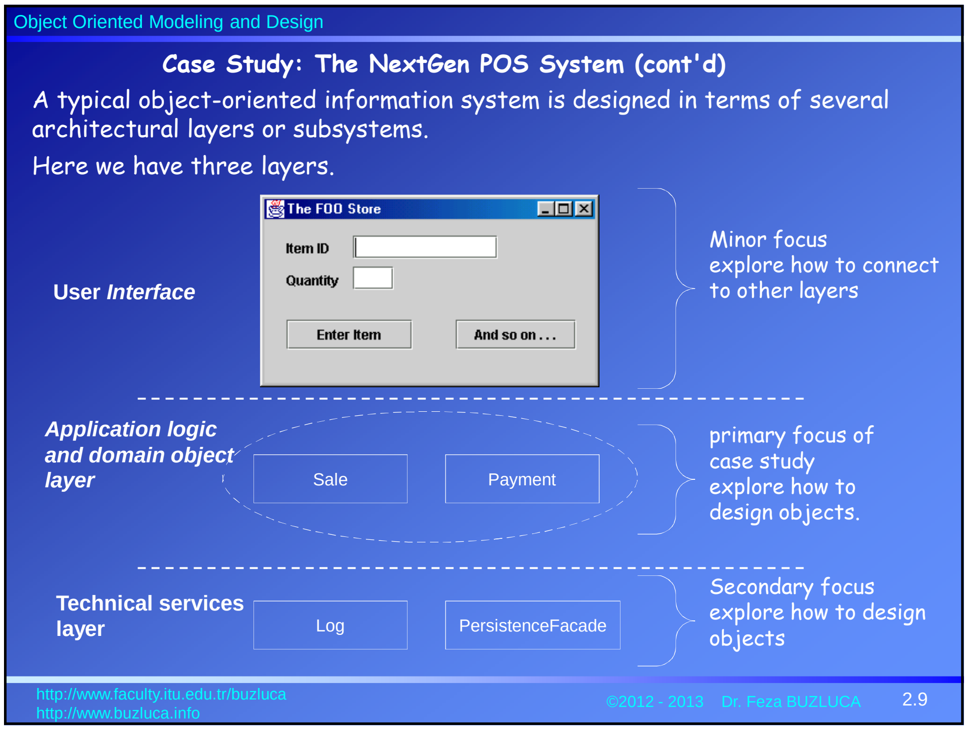
It interfaces to various service applications, such as a third-party tax calculator, inventory control and credit card authorization center.

Photo: http://www.asterpos.com.au/



The case study is from the book:

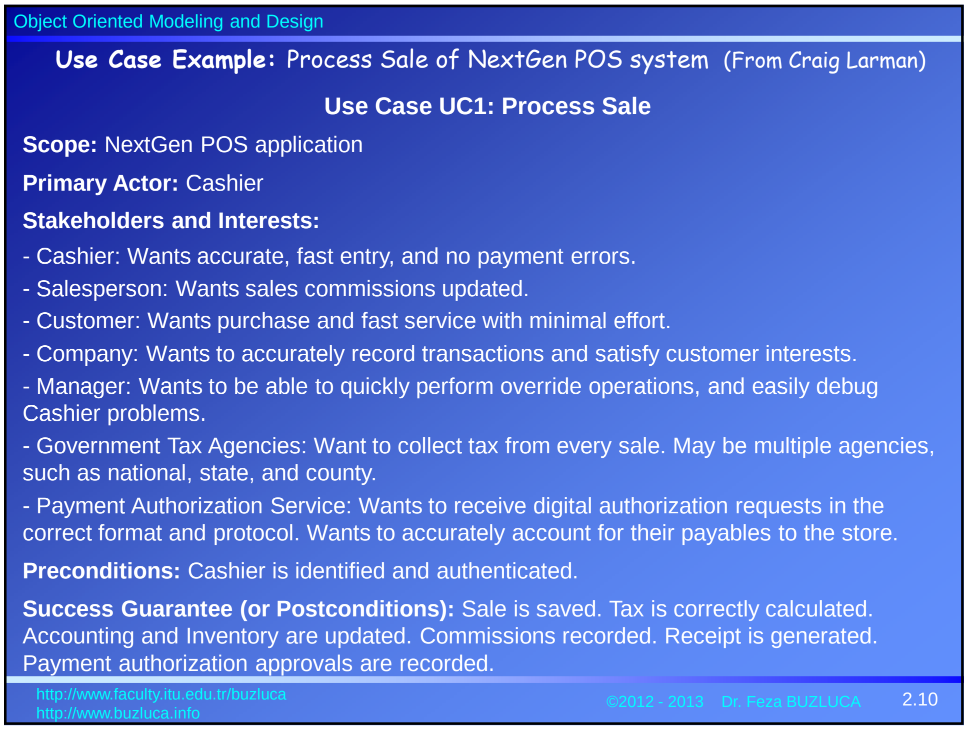
Craig Larman, Applying UML and Patterns , An Introduction to OOA/D and Iterative Development, 3/e, 2005.



Case Study: The NextGen POS System (cont'd)

A typical object-oriented information system is designed in terms of several architectural layers or subsystems.

Here we have three layers.



Use Case Example: Process Sale of NextGen POS system (From Craig Larman)

Use Case UC1: Process Sale

Scope: NextGen POS application

Primary Actor: Cashier

Stakeholders and Interests:

- Cashier: Wants accurate, fast entry, and no payment errors.

- Salesperson: Wants sales commissions updated.

- Customer: Wants purchase and fast service with minimal effort.

- Company: Wants to accurately record transactions and satisfy customer interests.

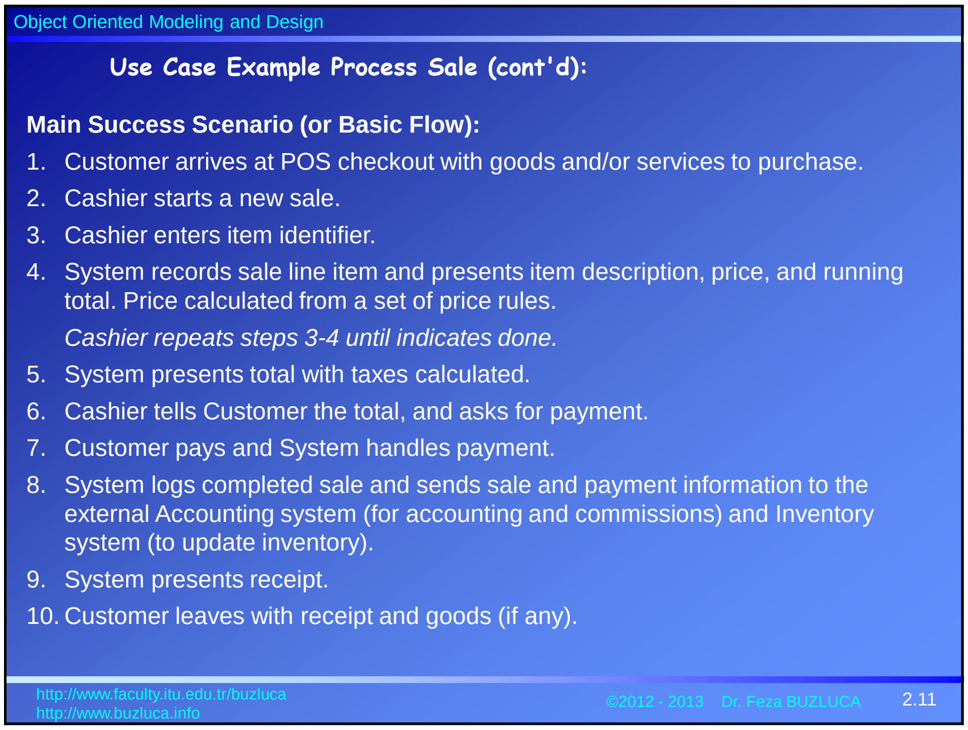
- Manager: Wants to be able to quickly perform override operations, and easily debug Cashier problems.

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

- Payment Authorization Service: Wants 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 (or Postconditions): Sale is saved. Tax is correctly calculated. Accounting and Inventory are updated. Commissions recorded. Receipt is generated. Payment authorization approvals are recorded.

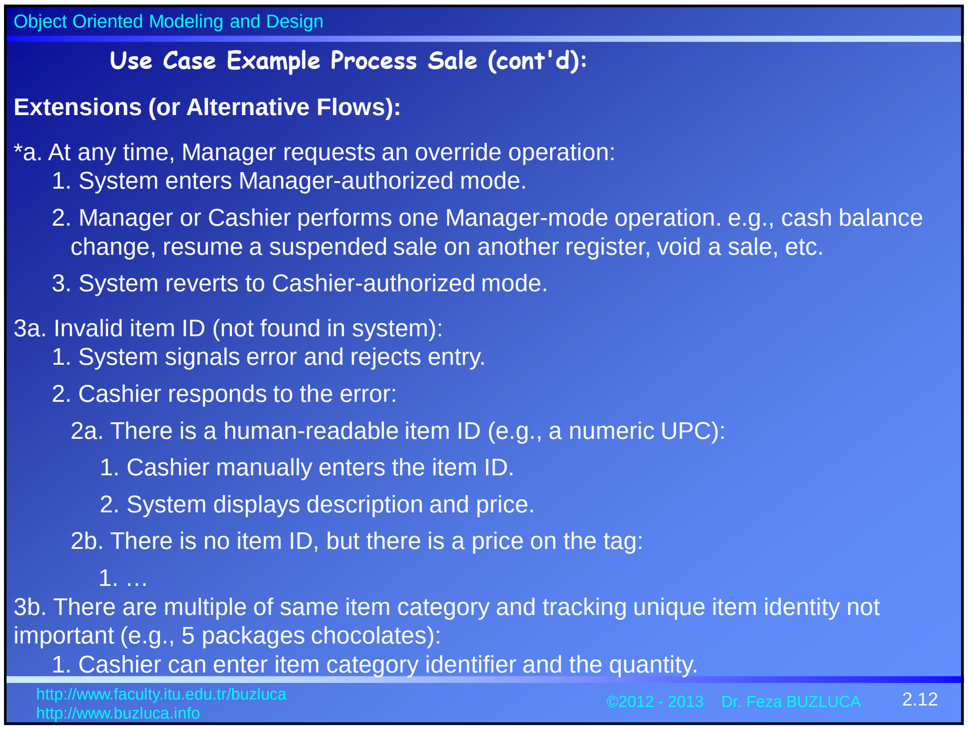


Use Case Example Process Sale (cont'd):

Main Success Scenario (or 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).



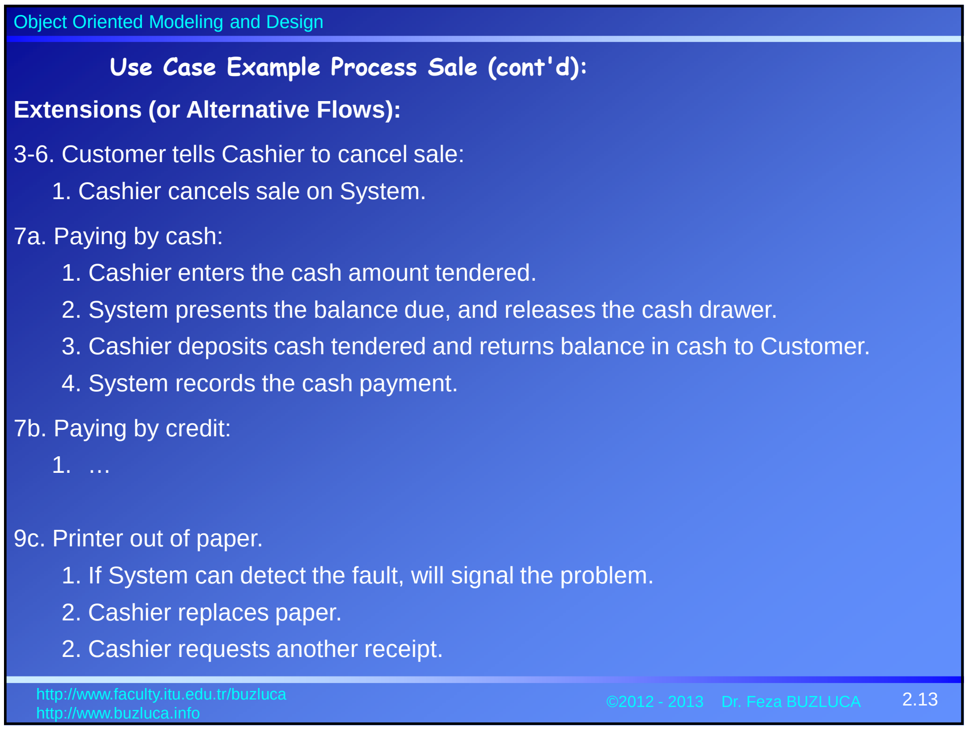
Use Case Example Process Sale (cont'd):

Extensions (or Alternative Flows):

*a. At any time, Manager requests an override operation:
- 1. System enters Manager-authorized mode.
- 2. Manager or Cashier performs one Manager-mode operation. e.g., cash balance change, resume a suspended sale on another register, void a sale, etc.
- 3. System reverts to Cashier-authorized mode.

3a. Invalid item ID (not found in system):
- 1. System signals error and rejects entry.
- 2. Cashier responds to the error:
  - 2a. There is a human-readable item ID (e.g., a numeric UPC):
    - 1. Cashier manually enters the item ID.
    - 2. System displays description and price.
  - 2b. There is no item ID, but there is a price on the tag:
    - 1. …

3b. There are multiple of same item category and tracking unique item identity not important (e.g., 5 packages chocolates):
- 1. Cashier can enter item category identifier and the quantity.



Use Case Example Process Sale (cont'd):

Extensions (or Alternative Flows):

3-6. Customer tells Cashier to cancel sale:

1. Cashier cancels sale on System.

7a. Paying by cash:

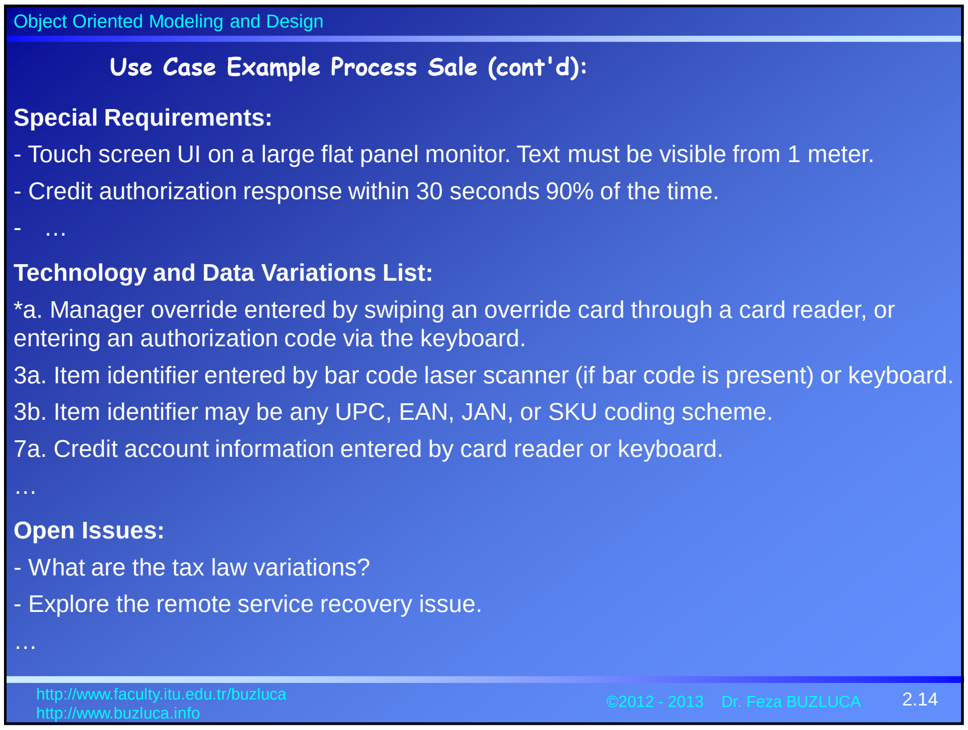
1. Cashier enters the cash amount tendered.
2. System presents the balance due, and releases the cash drawer.
3. Cashier deposits cash tendered and returns balance in cash to Customer.
4. System records the cash payment.

7b. Paying by credit:

1. …

9c. Printer out of paper.

1. If System can detect the fault, will signal the problem.
2. Cashier replaces paper.
2. Cashier requests another receipt.



Use Case Example Process Sale (cont'd):

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.
- …

Technology and Data Variations List:

*a. Manager override entered by swiping an override card through a card reader, or entering an authorization code via the keyboard.

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

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

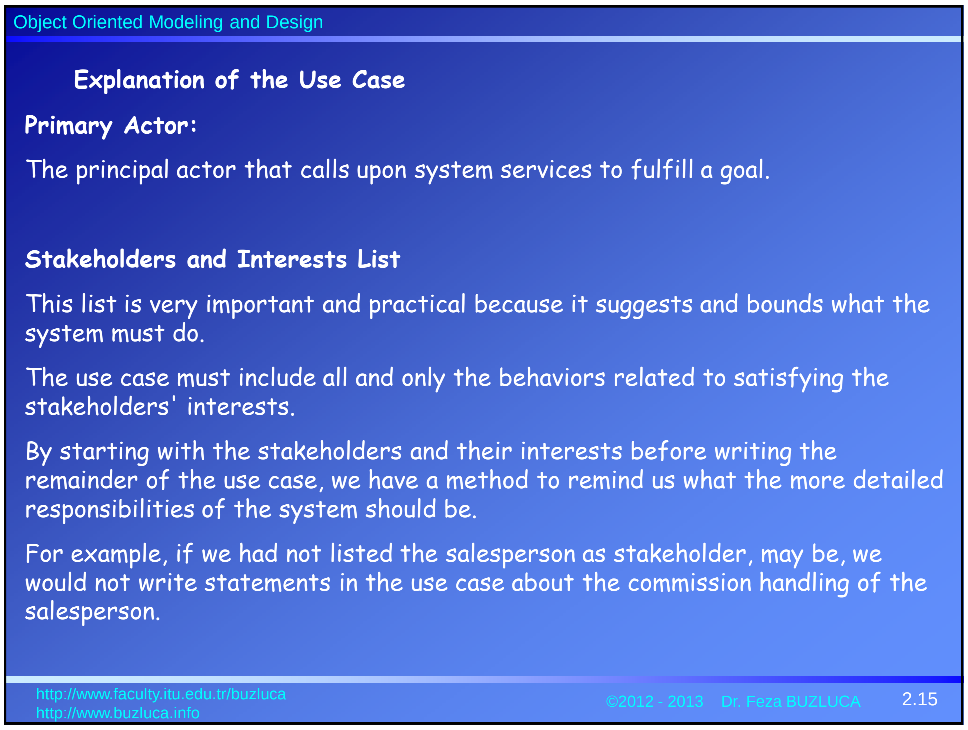
7a. Credit account information entered by card reader or keyboard.

…

Open Issues:

- What are the tax law variations?
- Explore the remote service recovery issue.

…



Explanation of the Use Case

Primary Actor:

The principal actor that calls upon system services to fulfill a goal.

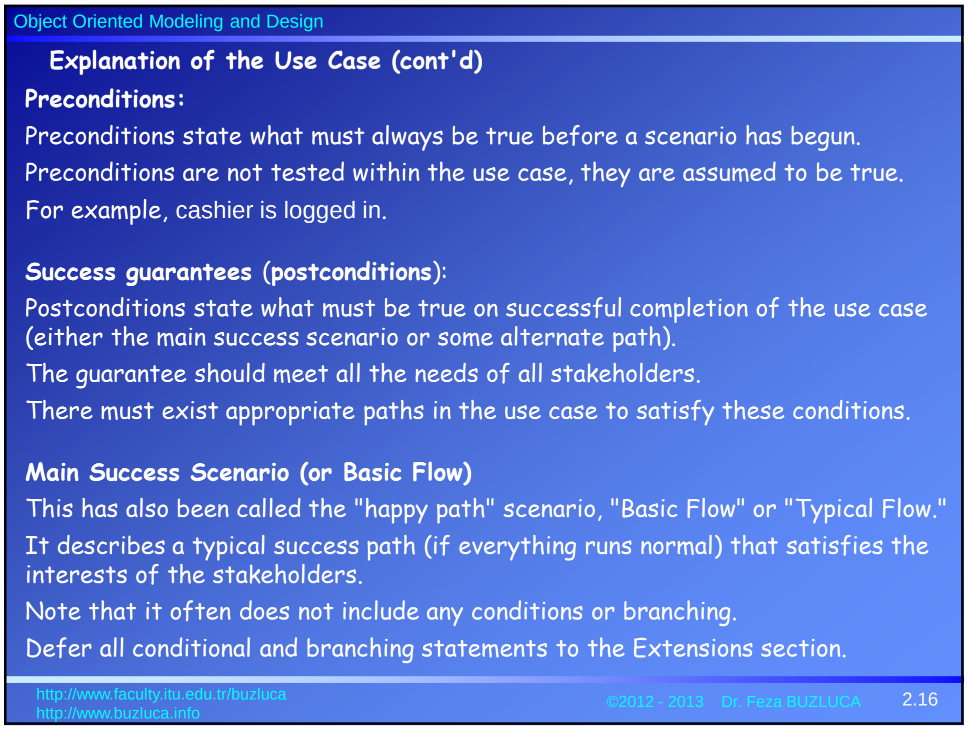
Stakeholders and Interests List

This list is very important and practical because it suggests and bounds what the system must do.

The use case must include all and only the behaviors related to satisfying the stakeholders' interests.

By starting with the stakeholders and their interests before writing the remainder of the use case, we have a method to remind us what the more detailed responsibilities of the system should be.

For example, if we had not listed the salesperson as stakeholder, may be, we would not write statements in the use case about the commission handling of the salesperson.



Explanation of the Use Case (cont'd)

Preconditions:

Preconditions state what must always be true before a scenario has begun.

Preconditions are not tested within the use case, they are assumed to be true.

For example, cashier is logged in.

Success guarantees (**postconditions**):

Postconditions state what must be true on successful completion of the use case (either the main success scenario or some alternate path).

The guarantee should meet all the needs of all stakeholders.

There must exist appropriate paths in the use case to satisfy these conditions.

Main Success Scenario (or Basic Flow)

This has also been called the "happy path" scenario, "Basic Flow" or "Typical Flow."

It describes a typical success path (if everything runs normal) that satisfies the interests of the stakeholders.

Note that it often does not include any conditions or branching.

Defer all conditional and branching statements to the Extensions section.

Explanation of the Use Case (cont'd)

Extensions (or Alternate Flows)

Extensions indicate all the other scenarios or branches, both success and failure.

Extension scenarios are branches from the main success scenario, and so can be notated with respect to its steps 1…N.

For example, at Step 3 of the main success scenario there may be an invalid item identifier.

An extension is labeled "3a"; it first identifies the condition and then states the response.

3a. Invalid item ID (not found in system):
    1. System signals error and rejects entry.

How to write use case, Guidelines

- Write use cases with the customer (user) of the software system.

  Write requirements focusing on the users or actors of a system, asking about their goals and typical situations.

  Sometimes software company can start a project without having a specific customer. Than the company must play both roles, customer and developer.

- Write black-box use cases.

  By defining system responsibilities with black-box use cases, one can specify **what** the system must do (the behavior or functional requirements) without deciding **how** it will do it (the design).

  During requirements analysis avoid making "how" decisions, and specify the external behavior for the system, as a black box.

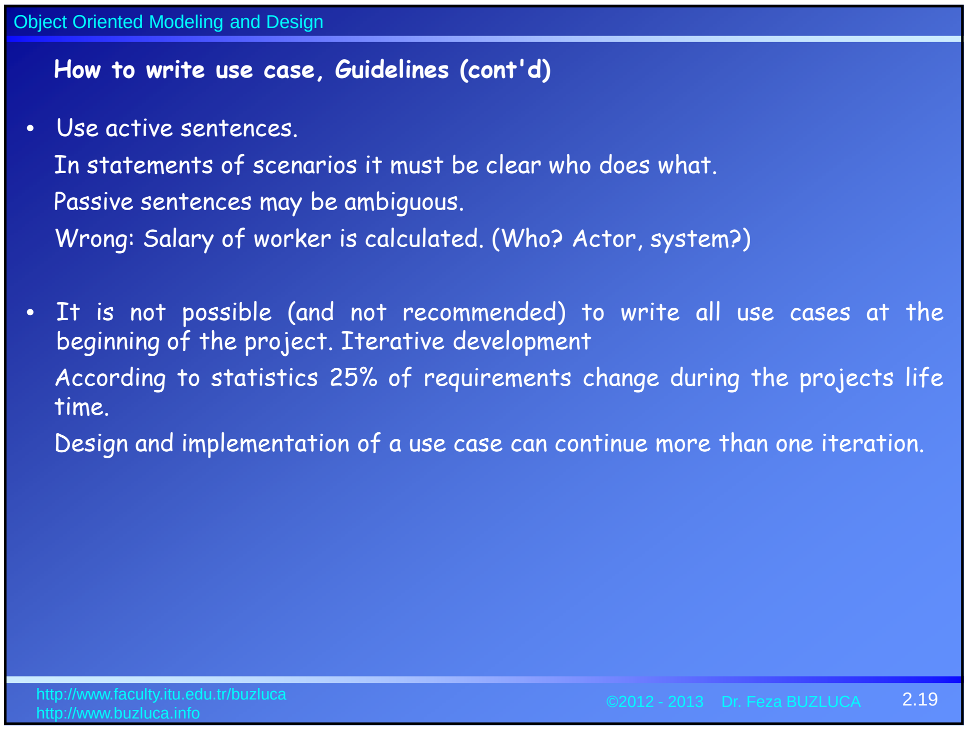
  Remember;

  Analysis: **Understanding. What?** Design: **Solution. How?**

  Black-box style (right): The system records the sale.

  Wrong: The system writes the sale to a database.

  Wrong: The system puts the sale in a double-linked list.



How to write use case, Guidelines (cont'd)

- Use active sentences.

  In statements of scenarios it must be clear who does what.

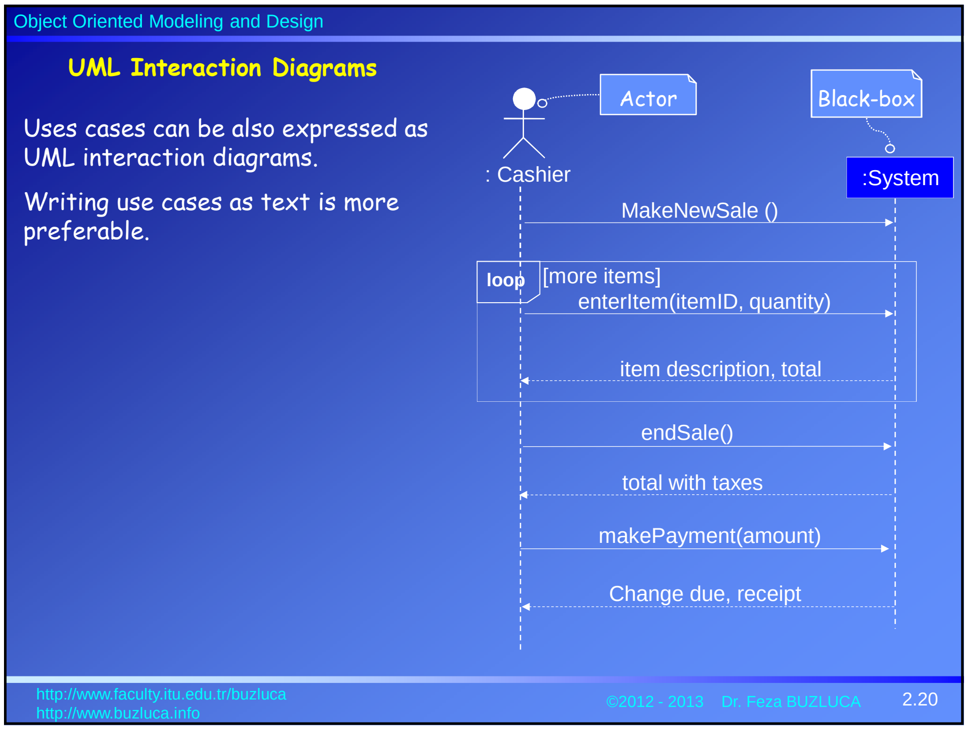
  Passive sentences may be ambiguous.

  Wrong: Salary of worker is calculated. (Who? Actor, system?)

- It is not possible (and not recommended) to write all use cases at the beginning of the project. Iterative development

  According to statistics 25% of requirements change during the projects life time.

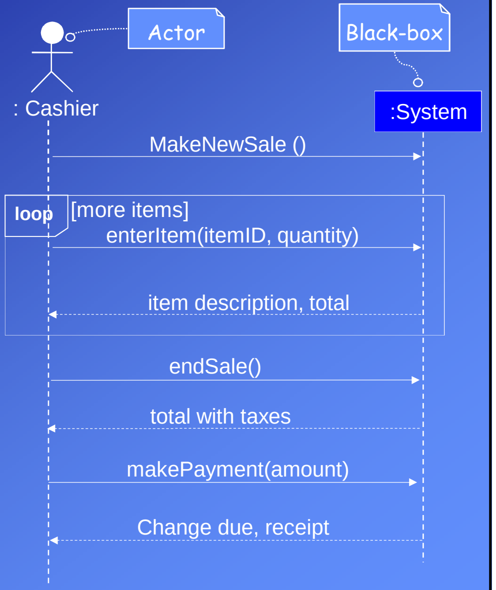
  Design and implementation of a use case can continue more than one iteration.

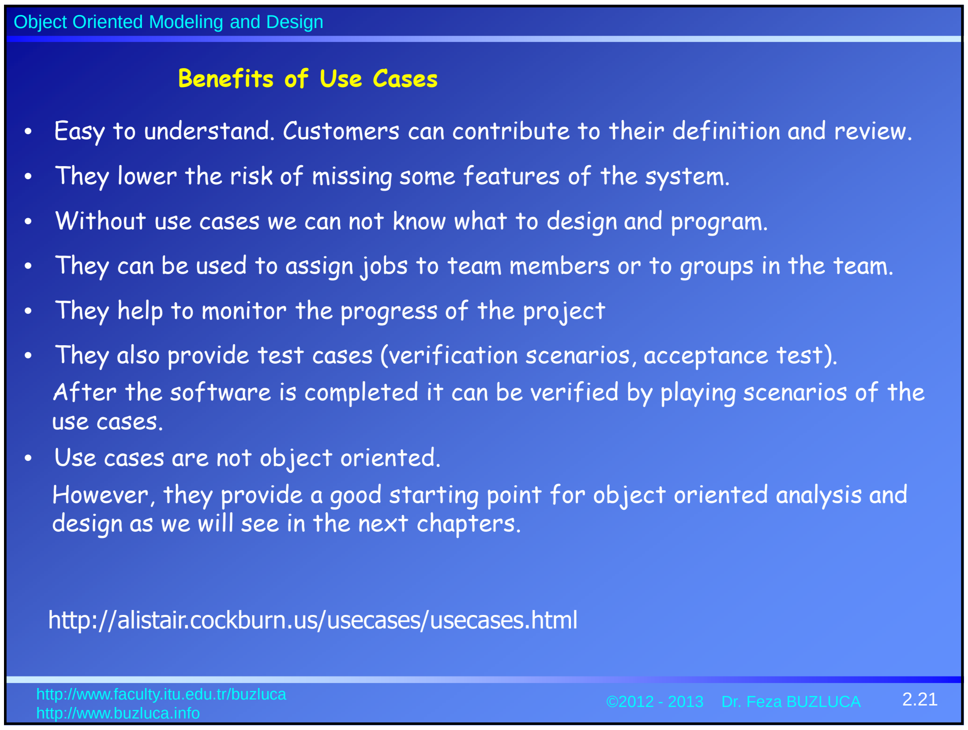


UML Interaction Diagrams

Uses cases can be also expressed as UML interaction diagrams.

Writing use cases as text is more preferable.





Benefits of Use Cases

- Easy to understand. Customers can contribute to their definition and review.
- They lower the risk of missing some features of the system.
- Without use cases we can not know what to design and program.
- They can be used to assign jobs to team members or to groups in the team.
- They help to monitor the progress of the project
- They also provide test cases (verification scenarios, acceptance test).

  After the software is completed it can be verified by playing scenarios of the use cases.
- Use cases are not object oriented.

  However, they provide a good starting point for object oriented analysis and design as we will see in the next chapters.

http://alistair.cockburn.us/usecases/usecases.html