CS189A - Capstone

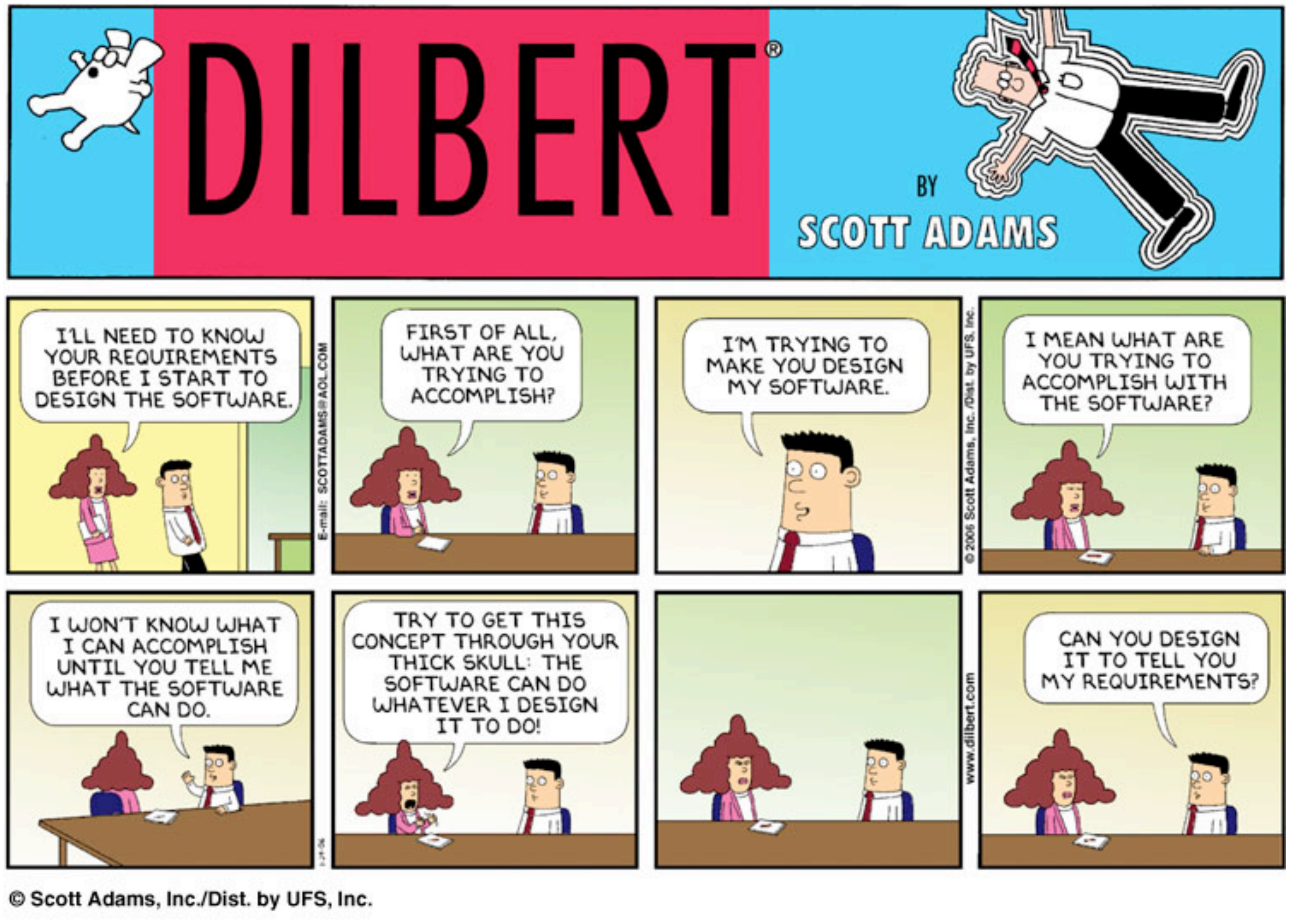
Christopher Kruegel
Department of Computer Science
UC Santa Barbara
http://www.cs.ucsb.edu/~chris/

Announcements

- **Next Project Deliverable:** Software Requirements Specifications (SRS) are due Wednesday, February 2nd
- Here are two approaches for the SRS
  - Submit a set of use cases (properly organized, with table of contents, page numbers)
  - Submit an SRS document following the IEEE standard outline (you do not have to include all the sections, just include the sections that relate to your project)
- Each team is free to choose an option (or a combination of them) that fits their project the best

Software Requirements

UC Santa Barbara


DILBERT®
BY SCOTT ADAMS
I'LL NEED TO KNOW YOUR REQUIREMENTS BEFORE I START TO DESIGN THE SOFTWARE.
E-mail: SCOTTADAMS@AOL.COM
FIRST OF ALL, WHAT ARE YOU TRYING TO ACCOMPLISH?
I'M TRYING TO MAKE YOU DESIGN MY SOFTWARE.
© 2006 Scott Adams, Inc./Dist. by UFS, Inc.
I MEAN WHAT ARE YOU TRYING TO ACCOMPLISH WITH THE SOFTWARE?
I WON'T KNOW WHAT I CAN ACCOMPLISH UNTIL YOU TELL ME WHAT THE SOFTWARE CAN DO.
TRY TO GET THIS CONCEPT THROUGH YOUR THICK SKULL: THE SOFTWARE CAN DO WHATEVER I DESIGN IT TO DO!
www.dilbert.com
CAN YOU DESIGN IT TO TELL YOU MY REQUIREMENTS?
© Scott Adams, Inc./Dist. by UFS, Inc.

Software Requirements

- Brooks in “No Silver Bullet” paper says:

  The hardest single part of building a software system is deciding precisely what to build. No other part of the conceptual work is as difficult as establishing the detailed technical requirements ... No other part of the work so cripples the resulting system if done wrong. No other part is as difficult to rectify later.

- Developers of the early Ballistic Missile Defense System observed [Alford, IEEE TSE, 1977]

  In nearly every software project that fails to meet performance and cost goals, requirements inadequacies play a major and expensive role in project failure

- In mission-critical defense systems identifies requirements as a major problem source in two thirds of the systems examined [US General Accounting Office, 1992]
- Other studies on projects in aerospace industry and NASA also found requirements to be a critical software development problem

Requirements errors are costly to fix

Stage	Relative Repair Cost
Requirements	1-2
Design	5
Coding	10
Unit Test	20
System Test	50
Maintenance	200

Relative cost to repair a requirements error in different stages of software life-cycle (cost increases exponentially)

Software Requirements Specification

- Software Requirements Specification (SRS):
  - Specification of a particular software product in a specific environment

- Basic goal of the SRS document is to specify what the software must do. To achieve this goal:
  - Understand precisely what is required of the software
  - Communicate the understanding of what is required to all the parties involved in the development
  - Provide a means for controlling the production to ensure that the final system satisfies the requirements (including managing the effects of changes)

What shall be in SRS?

- **Functionality:** What is the software supposed to do?
  - Example: *The software product shall sort a set of integers in ascending order. The software product shall write the sorted set of integers to an output file in the ASCII format. In the output file each integer shall be separated by a blank space.*

- **External Interfaces:** How does the software interact with people, the system’s hardware (there may be a hardware component within the system), other hardware and other software?
  - Example: *The software product shall read the set of integers from an ASCII file.*
    - We also have to specify the format of the input file and how the name of the file will be given.

What shall be in SRS?

- **Performance:** What is the speed, availability, response time, recovery time of various software functions, etc.?
  - Example: *For the input files with less than 1000 integers the software product shall produce the output file within 2 seconds.*

- **Design constraints imposed on an implementation:** Are there any required standards, implementation language restrictions, resource limits, operating environment(s) etc.?
  - Example: *The software product shall run on PCs that run Linux operating system.*
    - We should also specify which version of Linux, what type of PC (constraints on processor, memory etc.)

Classification of Requirements

- Requirements can be classified as:
  - **Functional requirements:** Requirements defining the behavior of the system, fundamental process or transformation that the software performs on inputs to produce outputs
  - **Nonfunctional requirements:** Requirements and constraints on external interfaces, performance, dependability, maintainability, reusability, security, etc.
  - **Domain requirements:** Requirements that come from the application domain of the system and reflect characteristics of the domain (can be functional or nonfunctional)

Who uses requirements?

- What are the uses of software requirement specification?
  - For customers it is a specification of the product that will be delivered, ***a contract***
  - For managers it can be used as a basis ***for scheduling and measuring progress***
  - For the software designers it provides a specification of ***what to design***
  - For coders it defines the range of ***acceptable implementations*** and the ***outputs that must be produced***
  - For quality assurance personnel it is used for ***validation, test planning, and verification***

Essential difficulties in SRS

- **Comprehension:** People do not know exactly what they want. They may not have a precise and detailed understanding of what the output must be for every possible input, how long each operation should take, etc.
- **Communication:** Software requirements are difficult to communicate effectively. The fact that requirements specification has multiple purposes and audiences makes this problem even more severe
- **Control:** It is difficult to predict the cost of implementing different requirements. Frequent changes to requirements make it difficult to develop stable specifications
- **Inseparable concerns:** Requirements must simultaneously address concerns of developers and customers. There may be conflicting constraints which may require trade-offs, compromises.

Eliciting Requirements

To elicit the requirements

- Interviews with the customer
- Use questionnaires if there are multiple users
- Investigate the environment the product will be used
  - investigate the customer’s business
- Scenarios: Walk through different scenarios of how the product will be used by the customer
  - understandable to the customer
  - can uncover additional requirements
- Rapid Prototyping: After an initial requirements analysis, build a prototype. Focus on aspects of the software that will be visible to the user such as input/output formats

Characteristics of a good SRS

- **Correct:** Every requirement stated in SRS should be one that the software shall meet. Correctness can be checked by customer or a higher level specification (system specification)

- **Unambiguous:** Every stated requirement in SRS should have only one interpretation
  - Natural languages are inherently ambiguous, they should be used carefully
  - Use of formal languages can help, however they may be hard for the customer to understand

- **Complete:**
  - All significant requirements, whether relating to functionality, performance, design constraints, attributes, or external interfaces should be included
  - Responses to all realizable classes of input data and situations should be included (responses to both valid and invalid input)

Characteristics of a good SRS

- **Consistent:** No subset of specified requirements should conflict. Possible conflicts:
  - There may be logical or temporal conflicts between two specified actions
  - Different part of SRS may use different terms to refer to the same object

- **Verifiable:** A requirement is verifiable if there exists some cost-effective process with which a person or machine can check that the software product meets the requirement
  - Your claims should be ***measurable***
  - Avoid subjective phrases such as "*works well*" which are not possible to measure/verify
  - A verifiable requirement: *Output of the program shall be produced within 20 seconds of event X 60% of the time; and shall be produced within 30 seconds of event X 100 % of the time*

Characteristics of a good SRS

- **Modifiable:** The style and structure of SRS should make it possible to change it easily, completely and consistently
  - No redundancy
  - Express each requirement separately (not intermixed)
- **Traceable:** SRS should facilitate referencing of each requirement in future development or enhancement documentation
  - Good indexing
    - **Do not forget the page numbers!**

An Example Outline for SRS
(Based on IEEE Recommended Practice)

Table of Contents

1. Introduction
   - 1.1 Purpose
     - Purpose of the SRS
     - Intended audience of the SRS
   - 1.2 Scope
     - List software products that will be produced
     - Summarize what software products will do
     - Describe the application of the software being specified, including relevant benefits, objectives and goals
   - 1.3 Definitions, acronyms, abbreviations
     - Definition of all terms, acronyms, abbreviations required to properly interpret SRS
   - 1.4 References
     - Provide a complete list of referenced documents
   - 1.5 Overview
     - Describe what is in the reminder of the document
     - Explain how SRS is organized

2. Overall description

   2.1 Product perspective

   - Identify the interface between the proposed software and existing systems, including a diagram of major system components.
   - A block diagram showing major components of the larger system, interconnections, and external interfaces can be helpful.

   2.2 Product functions

   - Provide a summary of the major functions that the software will perform
   - The functions should be organized in a way that makes the list of functions understandable to the customer or to anyone else reading the document
   - Diagrams can be used to explain different functions and their relationships

2.3 User characteristics

- General characteristics of the intended user of the product, level of expertise/training required to use the product

2.4 Constraints

- List all the constraints that will limit the developers options, interfaces to other applications, programming language requirements, hardware limitations, etc.

2.5 Assumptions and dependencies

- List the factors that affect the requirements in the SRS (assumptions on which operating system is available etc.)

3. Specific requirements (these are the detailed requirements)
    - This section of the SRS should contain the software requirements to a level of detail sufficient to enable designers to design a system to satisfy those requirements, and testers to test that the system satisfies those requirements

    3.1 External interface requirements
    - This section should specify various interfaces in detail: system interfaces, user interfaces, hardware interfaces, software interfaces, communications interfaces, etc.
    - A detailed description of all inputs and outputs from the software system should be given:
        - Should include: source of input and destination of output; valid range, accuracy and/or tolerance; units of measure; timing; screen formats/ organization; window formats/organization; data formats; command formats, etc.
    - Should complement but not repeat the information given in Section 2

3.1.1 User interfaces

– Screen formats, page or window layouts, error messages, etc. Some sample screen dumps can be used here to explain the interface

3.1.2 Hardware interfaces

– Interface between hardware and software product, which devices are supported

3.1.3 Software interfaces

– Specify use of other software products and interfaces with other application systems

3.1.4 Communication interfaces

– Interfaces to communications such as local network protocols, etc.

3.2 Functional requirements

- Functional requirements should define all the fundamental actions that the system must take place in the software in accepting and processing the inputs and in processing and generating the outputs
- Should include: validity checks on input; exact sequence of operations; responses to abnormal situations; relationship of outputs to inputs
- It can be organized in various ways, such as with respect to user classes, features, stimulus or a combination of those.
- Use-case diagrams, scenarios, activity diagrams can be used here
- ***This section is very important. You need to organize use-cases etc. very well so that they are comprehensible. You need to make sure that your functional requirements are unambiguous, complete and consistent.***

3.3 Performance requirements

- Speed, availability, response time, recovery time of various software functions, etc.
- Performance requirements should be specified in measurable terms. For example: *“95% of the transactions shall be processed in less than 1 second.”* rather than *“An operator shall not have to wait for the transaction to complete*
- There can be a separate section identifying the capacity constraints (for example amount of data that will be handled)

3.4 Design constraints

- Required standards, implementation language restrictions, resource limits, operating environment(s) etc.

3.5 Software system attributes

- Attributes such as security, portability, reliability

3.6 Domain requirements

- Explain the application domain and constraints on the application domain

Specification Languages

- Main issue: When you write code you write it in a programming language
  - How do you *write* the requirements?
  - How do you *write* the design?

- Specification languages
  - Used to specify the requirements or the design
  - As we have seen parts of SRS are necessarily in English (customer has to understand). To bring some structure to the SRS you can use semi-formal techniques such as use-case diagrams.
  - For design you can use UML class diagrams, sequence diagrams, state diagrams, activity diagrams
  - Some specification languages (such as UML class diagrams are supported with code generation tools)

Specification

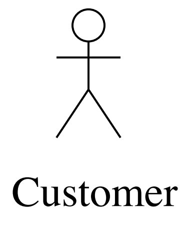
- Specifications can be
  - Informal
    - No formal syntax or semantics
      - for example in English
    - Informal specifications can be ambiguous and imprecise
  - Semiformal
    - Syntax is precise but does not have formal semantics
    - UML (Universal Modeling Language) class diagrams, sequence diagrams
  - Formal
    - Both syntax and semantics are formal
    - Z, Statecharts, SDL (Specification and Design Language), Message Sequence Charts (MSC), Petri nets, CSP, SCR, RSML

Ambiguities in Informal Specifications

- “The input can be typed or selected from the menu“
  - The input can be typed or selected from the menu or both
  - The input can be typed or selected from the menu but not both
- “The number of songs selected should be less than 10”
  - Is it strictly less than?
  - Or, is it less than or equal?
- “The user has to select the options A and B or C”
  - Is it “(A and B) or C”
  - Or, is it “A and (B or C)”

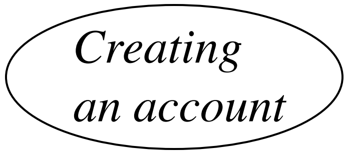
Use Cases

- Use cases document the behavior of the system from the users' point of view.
  - By user we mean anything external to the system
- An **actor** is a role played by an outside entity that interacts directly with the system
  - An actor can be a human, or a machine or program
  - Actors are shown as stick figures in use case diagrams



Use Cases

- A **use case** describes the possible sequences of interactions among the system and one or more actors in response to some initial stimulus by one of the actors
  - Each way of using the system is called a use case
  - A use case is not a single scenario but rather a description of *a set of scenarios*
  - For example: *Creating an account*
  - Individual use cases are shown as named ovals in use case diagrams



- A use case involves a sequence of interactions between the initiator and the system, possibly involving other actors.
- In a use case, the system is considered as a black-box. We are only interested in externally visible behavior

Use Cases

- To define a use case, group all transactions that are similar in nature
- A typical use case might include a main case, with alternatives taken in various combinations and including all possible exceptions that can arise in handling them
  - Use case for a bank: *Performing a Transaction at the Counter*
    - Subcases could include *Making Deposits*, *Making Withdrawals*, etc., together with exceptions such as *Overdrawn* or *Account Closed*
  - *Apply for a Loan* could be a separate use case since it is likely to involve very different interactions
- Description of a use case should include events exchanged between objects and the operations performed by the system that are visible to actors

Defining Use Cases

1. Identify the boundary of the application, identify the objects outside the boundary that interact with the system
2. Classify the objects by the roles they play, each role defines an actor
3. Each fundamentally different way an actor uses the system is a use case
4. Make up some specific scenarios for each use case (plug in parameters if necessary)
5. Determine the interaction sequences: identify the event that initiates the use case, determine if there are preconditions that must be true before the use case can begin, determine the conclusion
6. Write a prose description of the use case
7. Consider all the exceptions that can occur and how they affect the use case
8. Look for common fragments among different use cases and factor them out into base cases and additions

Online HR System

Use case: Update Benefits

Actors: Employee, Employee Account Database, Healthcare Plan System, Insurance Plan System

Precondition: Employee has logged on to the system and selected “update benefits” option

Flow of Events:

Basic Path:

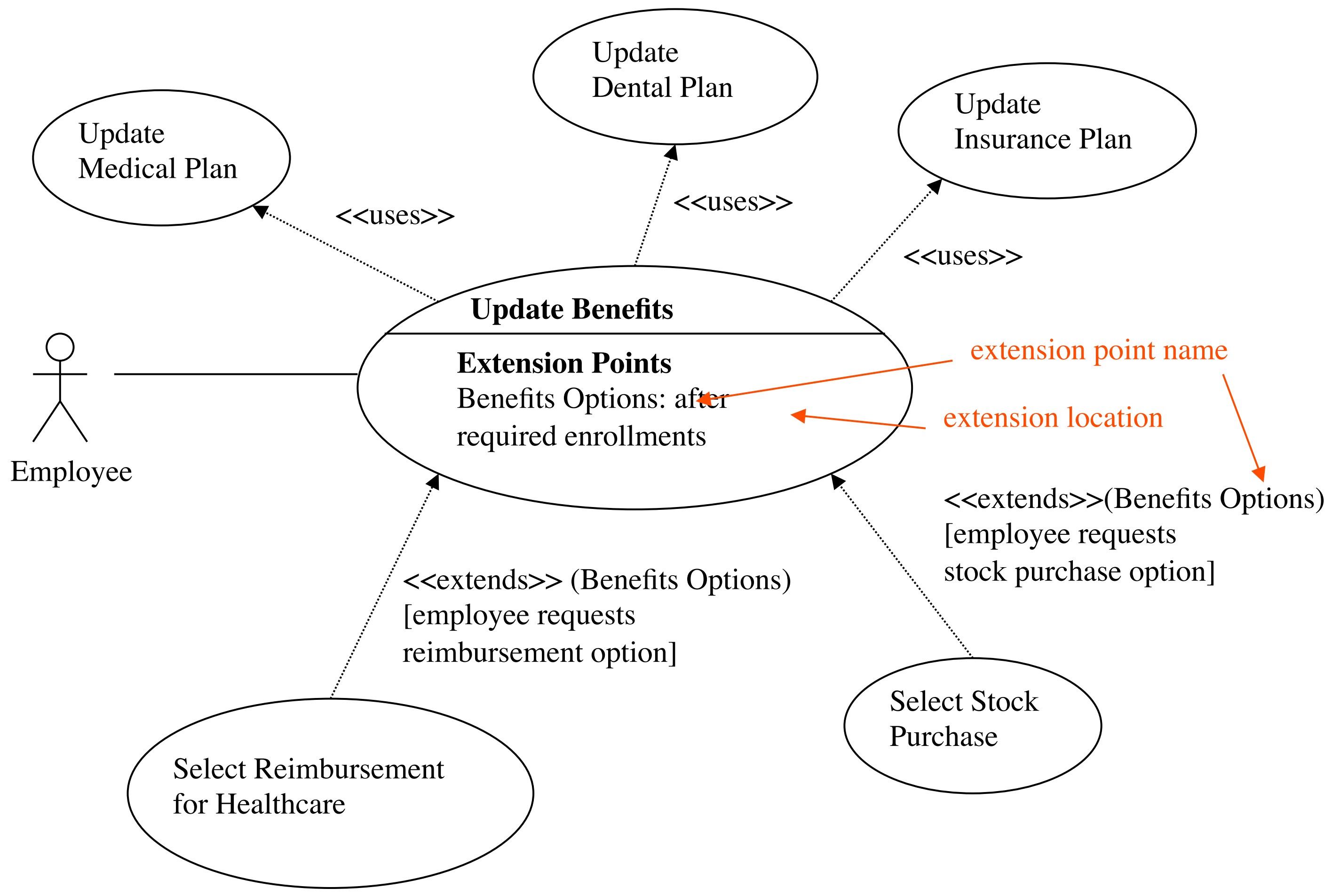
1. System retrieves employee account from Employee Account Database
2. System asks employee to select medical plan type; **uses** Update Medical Plan
3. System asks employee to select dental plan type; **uses** Update Dental Plan

...

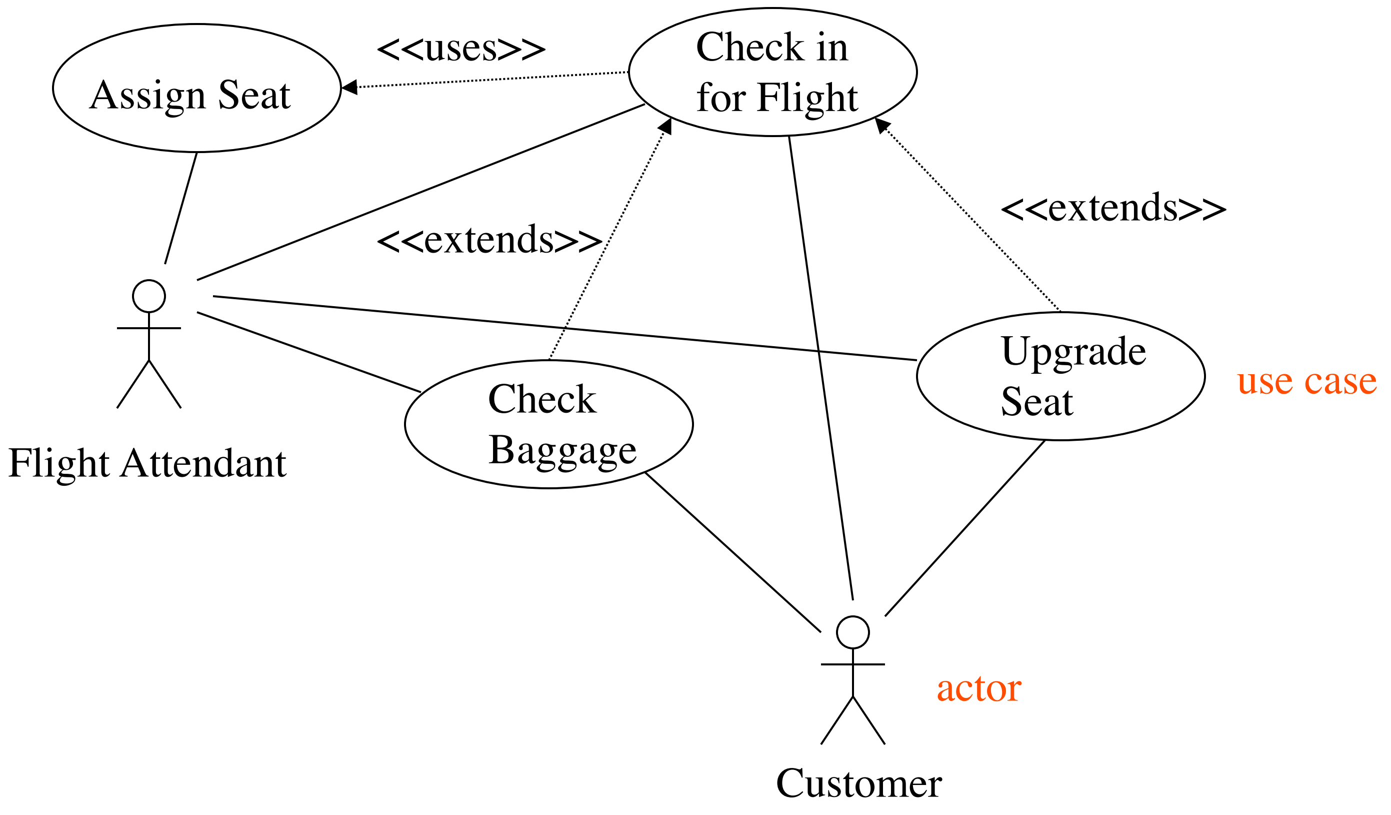
Alternative Paths:

If health plan is not available in the Employee’s area the employee is informed and asked to select another plan

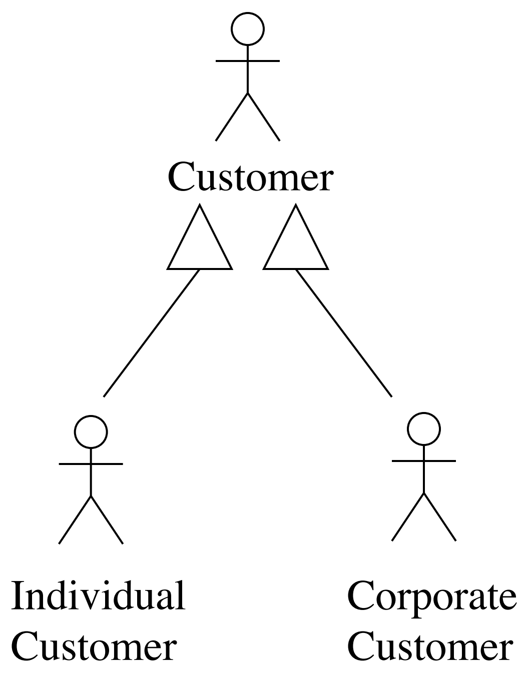
Online HR System

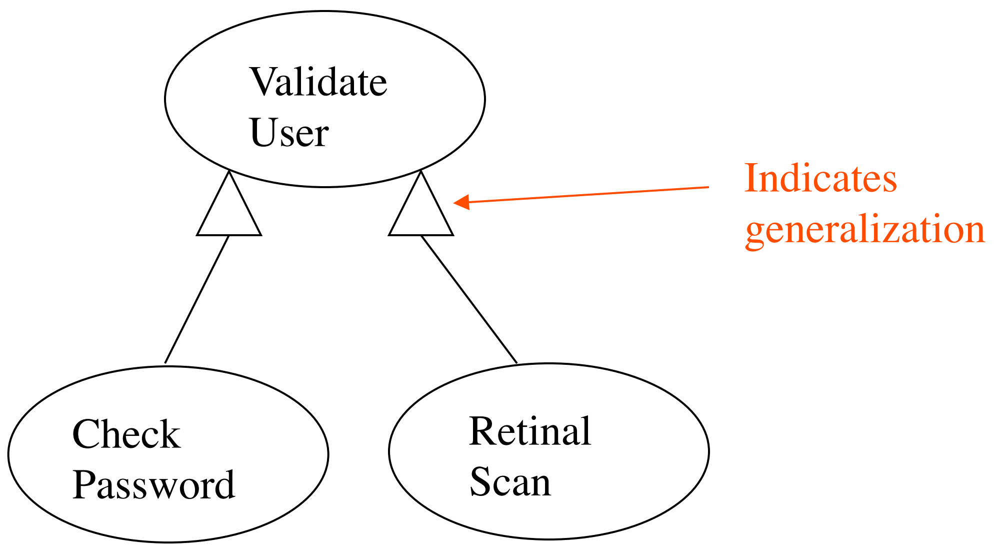

Update Medical Plan
Update Dental Plan
Update Insurance Plan
<<uses>>
<<uses>>
<<uses>>
Update Benefits
Extension Points
Benefits Options: after required enrollments
extension point name
extension location
Employee
<<extends>>(Benefits Options) [employee requests stock purchase option]
<<extends>> (Benefits Options) [employee requests reimbursement option]
Select Reimbursement for Healthcare
Select Stock Purchase

Combining Use Cases



Generalization in Use Case Diagrams


Customer
Individual Customer
Corporate Customer


Validate User
Check Password
Retinal Scan
Indicates generalization