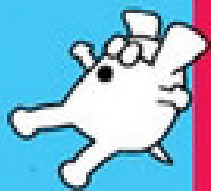

Requirement

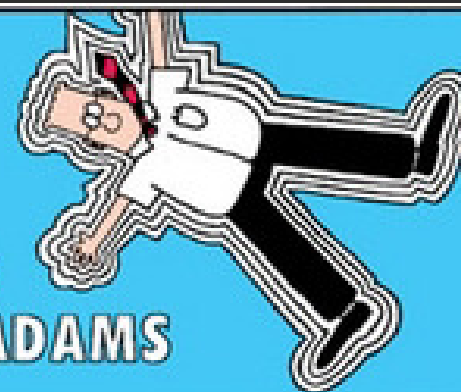




DILBERT®

BY

SCOTT ADAMS



I'LL NEED TO KNOW YOUR REQUIREMENTS BEFORE I START TO DESIGN THE SOFTWARE.



Email: SCOTTADAMS@AOL.COM

FIRST OF ALL, WHAT ARE YOU TRYING TO ACCOMPLISH?



I'M TRYING TO MAKE YOU DESIGN MY SOFTWARE.



© 2008 Scott Adams, Inc./Dist. by UFS, Inc.

I MEAN WHAT ARE YOU TRYING TO ACCOMPLISH WITH THE SOFTWARE?

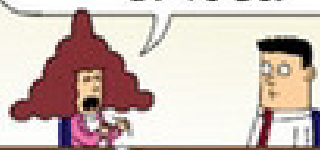


I WON'T KNOW WHAT I CAN ACCOMPLISH UNTIL YOU TELL ME WHAT THE SOFTWARE CAN DO.



UFS

TRY TO GET THIS THICK SKULL: THE SOFTWARE CAN DO WHATEVER I DESIGN IT TO DO!



www.dilbert.com

CAN YOU DESIGN IT TO TELL YOU MY REQUIREMENTS?



I want something
to get me across
town in the
shortest time



Meaning of Requirement

- ▶ In engineering, a **requirement** is a singular documented physical and functional need that a particular product or service must be or perform
- ▶ a feature of the system or a description of something the system is capable of doing in order to fulfill the system's purpose
- ▶ **Requirements engineering** is the set of activities that lead to the derivation of the system or software requirements.

[1]



Requirements engineering

- ▶ Requirement Engineering is the process of establishing the services that the customer requires from a system and the constraints under which it operates and develops.
- ▶ a systems and software engineering process which covers all of the activities involved in discovering, documenting and maintaining a set of requirements for a computer-based system.

[2]



Requirements engineering (cont'd)

- ▶ In the traditional waterfall model of the systems or software engineering process, requirements engineering is presented as the first stage of the development process, with the outcome being a requirements document or Software requirements specification.
- ▶ In fact, requirements engineering is a process that continues through the lifetime of a system as the requirements are subject to change and new requirements must be elicited and documented and existing requirements managed over the lifetime of the system.



Characteristics of good requirements

Characteristic	Explanation
Unitary (Cohesive)	The requirement addresses one and only one thing.
Complete	The requirement is fully stated in one place with no missing information.
Consistent	The requirement does not contradict any other requirement and is fully consistent with all authoritative external documentation.
Non- Conjugated (Atomic)	The requirement is <i>atomic</i> , i.e., it does not contain conjunctions. E.g., "The postal code field must validate American <i>and</i> Canadian postal codes" should be written as two separate requirements: (1) "The postal code field must validate American postal codes" and (2) "The postal code field must validate Canadian postal codes".
Traceable	The requirement meets all or part of a business need as stated by stakeholders and authoritatively documented.
Current	The requirement has not been made obsolete by the passage of time.
Feasible	The requirement can be implemented within the constraints of the project.
Unambiguous	The requirement is concisely stated without recourse to technical jargon, acronyms (unless defined elsewhere in the Requirements document), or other esoteric verbiage. It expresses objective facts, not subjective opinions. It is subject to one and only one interpretation. Vague subjects, adjectives, prepositions, verbs and subjective phrases are avoided. Negative statements and compound statements are prohibited.
Mandatory	The requirement represents a stakeholder-defined characteristic the absence of which will result in a deficiency that cannot be ameliorated. An optional requirement is a contradiction in terms.
Verifiable	The implementation of the requirement can be determined through one of four possible methods: inspection, demonstration, test or analysis.



Why are requirements important?

Projects fail because (source: Standish Group survey 1994)

- ▶ 13.1% incomplete requirements
- ▶ 12.4% lack of user involvement
- ▶ 10.6% lack of resources
- ▶ 9.9% unrealistic expectations
- ▶ 9.3% lack of executive support
- ▶ 8.7% changing requirements and specifications
- ▶ 8.1% lack of planning
- ▶ 7.5% system no longer needed



The Purpose of the Requirements Discipline

- ▶ The Requirements discipline intends to:
 - Find agreement on what the system should do.
 - Provide a better understanding of the system requirements.
 - Define the boundaries of the system.
 - Provide a basis for planning the technical contents of iterations.
 - Provide a basis for estimating cost.
 - Define a user-interface for the system.



Requirement

In systems and software engineering

- ▶ In systems engineering, a requirement can be a description of what a system must do, referred to as a **Functional Requirement**.
- ▶ Another type of requirement specifies something about the system itself, and how well it performs its functions. Such requirements are often called **Non-functional requirements**.



Functional Requirement

&

Non – Functional Requirement

Functional and Non-Functional means of classification

- ▶ the most frequently used means of requirements classification is Functional and Non-Functional.
- ▶ This classification helps identify whether a requirement will affect the functionality of the system (Functional) or whether it will constrain the system (Non-Functional).
- ▶ This classification is probably the most beneficial in that it helps define what system functions are being considered.



Functional Requirement

- ▶ A requirement specifies a function that a system or component must be able to perform
- ▶ Functional requirements may be calculations, technical details, data manipulation and processing and other specific functionality that define what a system is supposed to accomplish.
- ▶ Generally, are expressed in the form "system must do <requirement>"
- ▶ The plan for implementing is detailed in the system design
- ▶ specify particular results of a system
- ▶ drive the application architecture of a system



Examples

A **Functional Requirement** is a requirement that, when satisfied, will allow the user to perform some kind of function.

For example:

- ▶ display of the number of records in a database
- ▶ The customer must place an order
- ▶ Display the heart rate, blood pressure and temperature of patient connected to the patient monitor



Non – Functional Requirement

- ▶ A non – functional requirement is a statement of how a system must behave, it is a constraint upon the system behavior
- ▶ non-functional requirements are "constraints", "quality attributes", "quality goals", "quality of service requirements" and "non-behavioral requirements“
- ▶ tend to identify “user” constraints and “system” constraints
- ▶ non-functional requirements are "system shall be <requirement>“
- ▶ The plan for implementing is detailed in the system architecture.
- ▶ specify overall characteristics such as cost and reliability.
- ▶ drive the technical architecture of a system.



Qualities

that is non-functional requirements, can be divided into two main categories:

1. Execution qualities, such as security and usability, which are observable at run time.
2. Evolution qualities, such as testability, maintainability, extensibility and scalability, which are embodied in the static structure of the software system.



Examples

- ▶ How up-to-date this number needs
- ▶ The customer must be able to access their account 24 hours a day, seven days a week.
- ▶ The system must be unavailable from midnight until 1:00am for backups.
- ▶ The customer must place an order within two minutes of registering
- ▶ Display of the patient's vital signs must respond to a change in the patient's status within 2 seconds



Functional VS Non-Functional

- ▶ Functional requirements define what a system is supposed to do whereas non-functional requirements define how a system is supposed to be.
- ▶ Functional requirements are usually in the form of "system must do <requirement>", while non-functional requirements are "system shall be <requirement>".



characteristics

Characteristics of Functional Requirements and Non – Functional Requirement they have same following characteristics:

- uses simple language
- not ambiguous
- contains only one point
- specific to one type of user
- is qualified
- describes what and not how



Use Case

What is an "Use Case" ?

In software and systems engineering ,

use case

“is a list of steps, defining interactions between **role(actor)** and **system** to achieve a goal.”

****A role (known in UML as an "actor" which can be a human or an external system.)

[9]



Use Case with systems engineering

In systems engineering, use cases are used at a higher level than within software engineering, often representing missions or stakeholder goals. The detailed requirements may then be captured in SysML(Systems Modeling Language) or as contractual statements

[9]



History of Use case

Since Jacobson originated use case modeling in 1986 and many others have contributed to improving this technique, notably including Alistair Cockburn.

► [9]



Use case structure

Casual use case structure

Cockburn recognizes that projects may not always need detailed "fully-dressed" use cases. He describes a Casual use case with the fields:

- ▶ Title (goal)
- ▶ Primary Actor
- ▶ Scope
- ▶ Level
- ▶ (Story): the body of the use case is simply a paragraph or two of text, informally describing what happens.

▶ [9]



Do you agree with him?

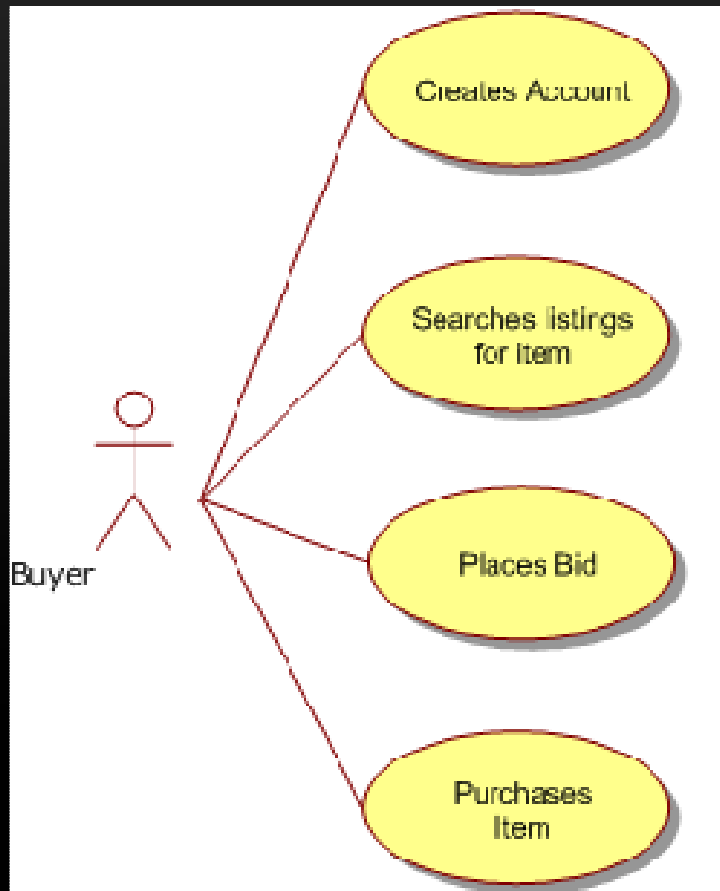
"There is no standard way to write the content of a use case, and different formats work well in different cases."

Martin Fowler states

[9]



Use case notation



- ▶ In order to represent an use case to be easy to understand, the relationships between all of the use cases and actors are represented in an use case diagram,

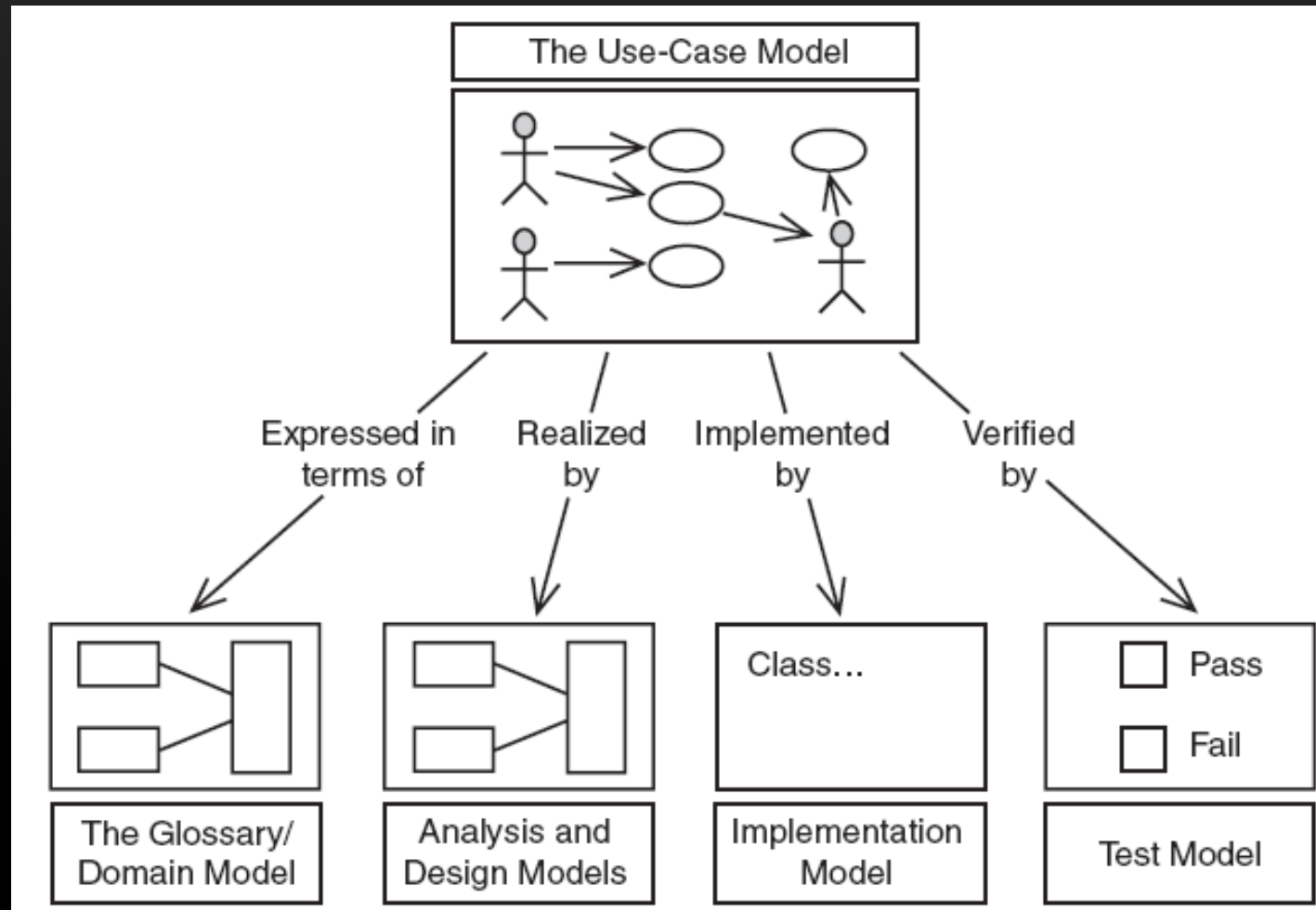
originally based upon Ivar Jacobson's Objectory notation.



Use Case Diagram



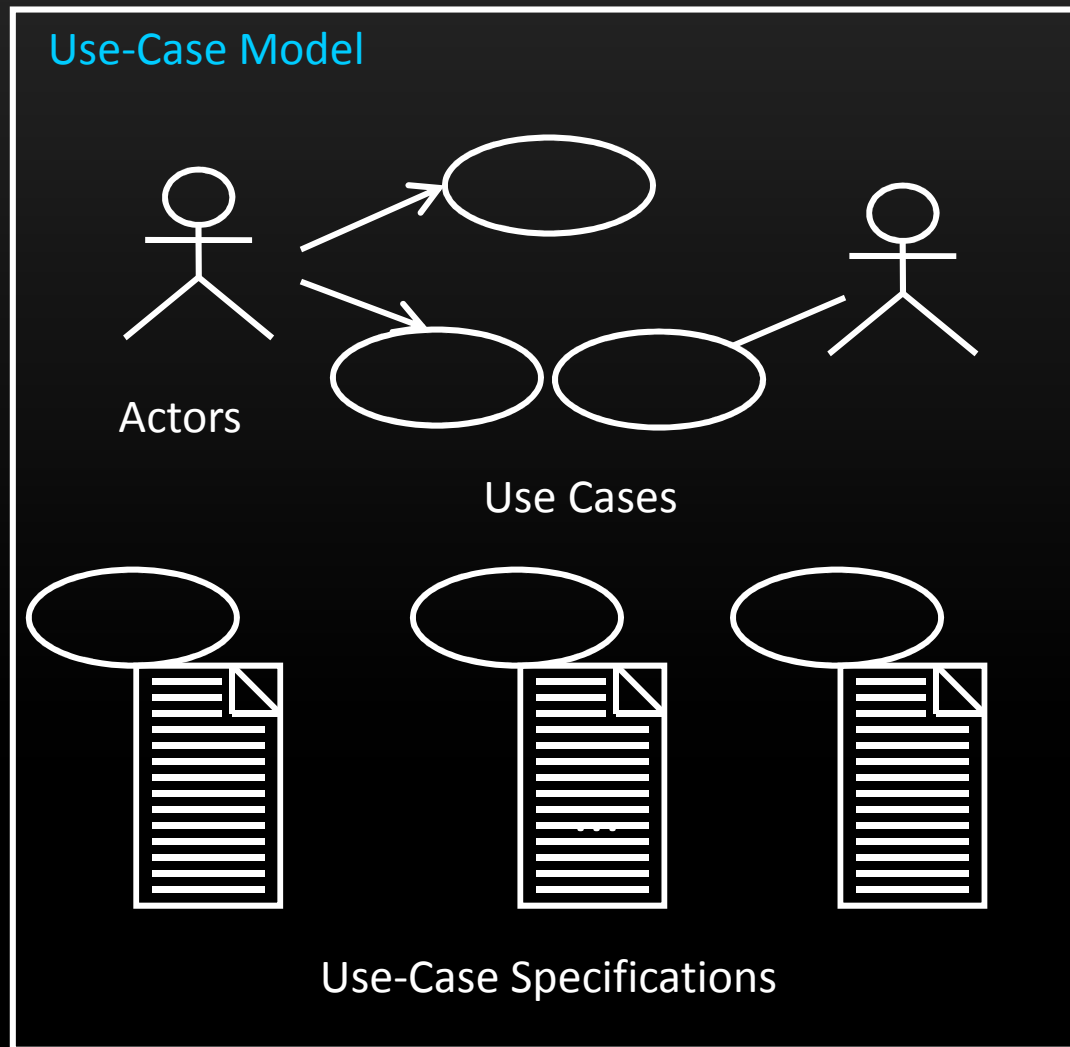
Usages of a use case diagram



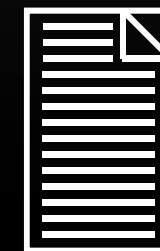
What Are the Benefits of Use-Case Diagram?

- ▶ Used to communicate with the end users and domain experts
 - ▶ Provides buy-in at an early stage of system development
 - ▶ Insures a mutual understanding of the requirements
- ▶ Used to identify
 - ▶ Who interacts with the system and what the system should do
 - ▶ The interfaces the system should have
- ▶ Used to verify
 - ▶ All requirements have been captured
 - ▶ The development team understands the requirements

Relevant Requirements Deliverables or Artifacts



Glossary



Supplementary
Specification

Major Concepts in Use-Case Diagram



Actor

An **actor** represents anything that interacts with the system eg. a human, hardware device, or another system can play.



Use Case

- ▶ A **use case** defines a set of use-case instances, where each instance is a sequence of actions a system performs that yields an observable result of value to a particular actor.

Useful Questions in Finding Actors

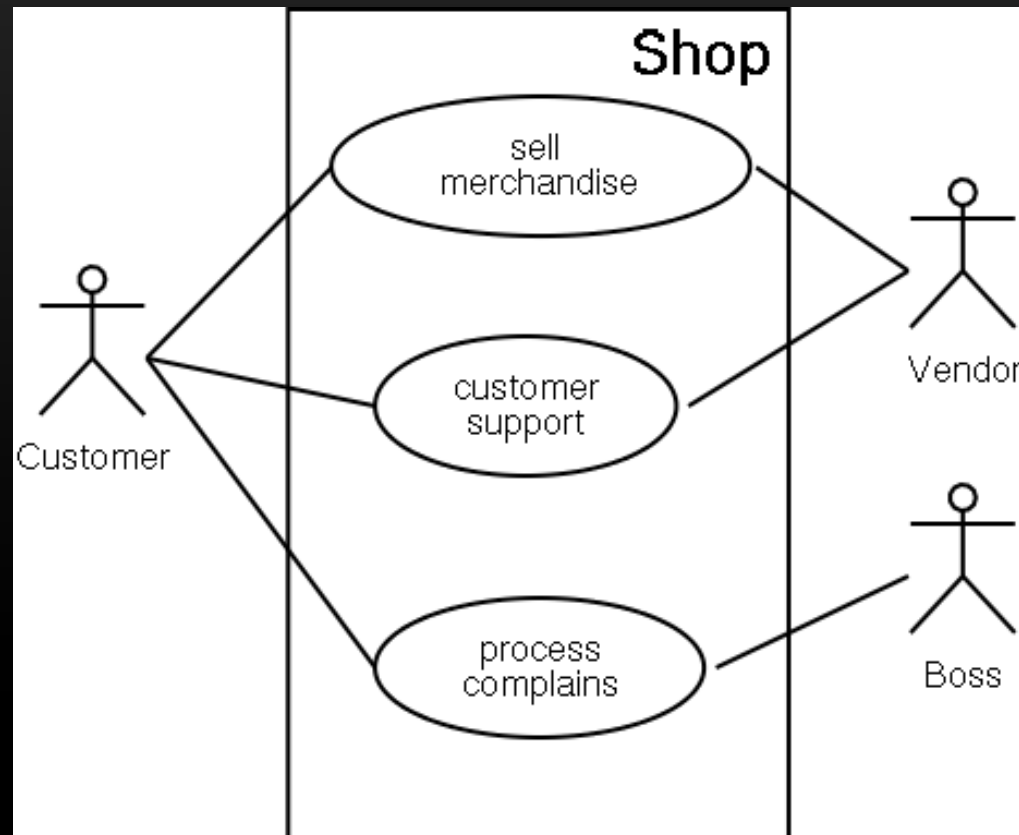
- Who will supply, use, or remove information?
- Who will use this functionality?
- Who is interested in a certain requirement?
- Where in the organization is the system used?
- Who will support and maintain the system?
- What are the system's external resources?
- What other systems will need to interact with this one?



Actor



Example 1 : of use case diagram



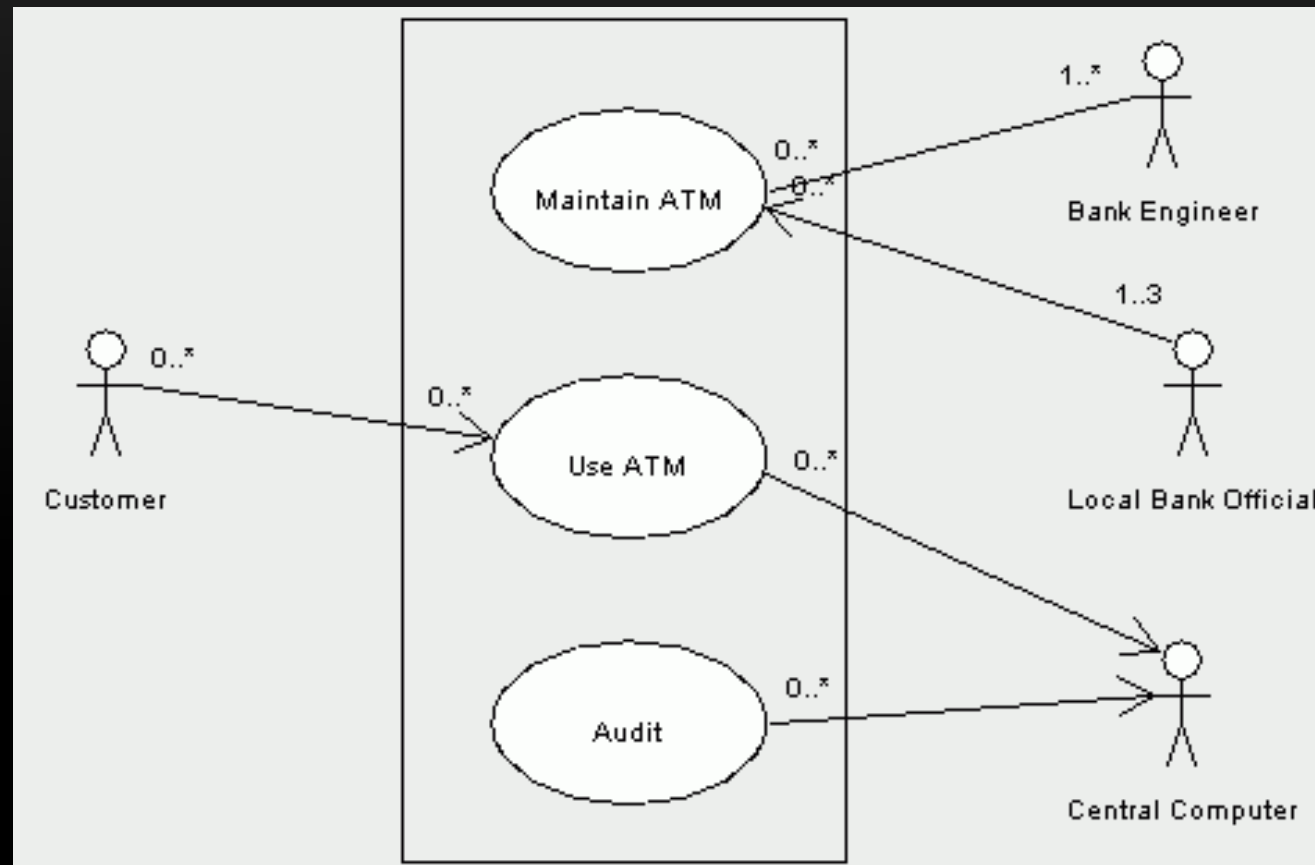
UML use cases for a simple shop model

<http://wrice.egloos.com/4847326>



Example 2 : of use case diagram

► Basic use case diagram for an ATM system



User Stories



Introduction to User Stories

- ▶ User stories serve the same purpose as use cases but are not the same.
- ▶ A good way to think about a user story is that it is a reminder to have a conversation with your customer , which is another way to say it's a reminder to do some just-in-time analysis.



What is a User Story?

- ▶ A user story describes desired functionality from the customer(user) perspective. A good user story describes the desired functionality, who wants it, and how and why the functionality will be used.
- ▶ They are in the format of about sentences of text written by the customer in the customers terminology without techno-syntax.

[4]



Usage

user stories define

- ▶ what is to be built in the software project. User stories are prioritized by the customer to indicate which are most important for the system and will be broken down in tasks and estimated by the developers.



Components of User Story

But user stories are not just these small snippets of text. Each user story is composed of three aspects:

1. Written description of the story, used for planning and as a reminder
2. Conversations about the story that serve to flesh out the details of the story
3. Tests that convey and document details that can be used to determine when a story is complete

[5]



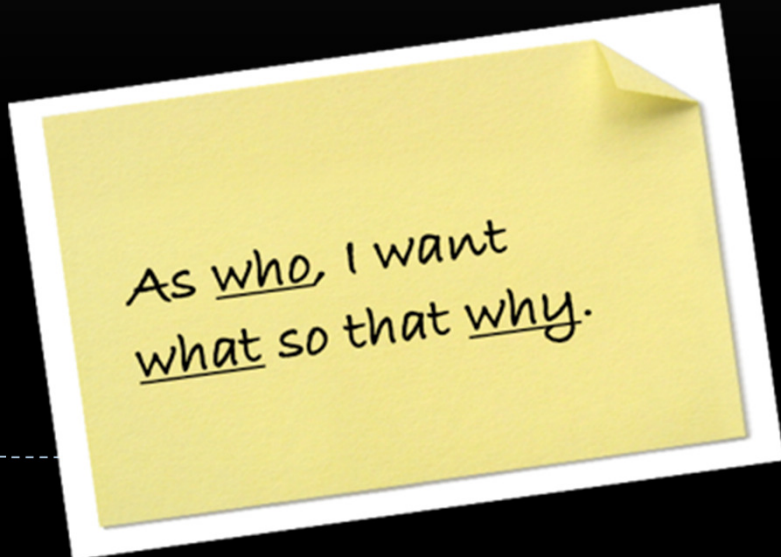
Creating user stories

User stories generally follow the following template:

"As a <role>, I want <goal/desire> so that <benefit>"

but the shorter version is commonly used as well:

"As a <role>, I want <goal/desire>"



As who, I want
what so that why.

Six Features of a Good User Story

A well-written user story follows the **INVEST** model

- ▶ Independent
- ▶ Negotiable
- ▶ Valuable
- ▶ Estimable
- ▶ Small
- ▶ Testable

[6]



Independent

- ▶ One user story should be independent of another (as much as possible). Dependencies between stories make planning, prioritization, and estimation much more difficult.

Negotiable

- A good story is negotiable. It is not an explicit contract for features; rather, details will be co-created by the customer and programmer during development.



Valuable

- ▶ Each story has to be of value to the customer. One good way of making stories valuable is to get the customer to write them. Once a customer realizes that a user story is not a contract and is negotiable, they will be much more comfortable writing stories.



Estimable

- ▶ The developers need to be able to estimate a user story to allow prioritization and planning of the story. If a story is too large to be estimated, then the scenario is not understood enough to prioritize or develop.



Small

- ▶ A good story should be small in effort, typically representing no more than 2-3 weeks of effort. Smaller units of work tend to receive smaller and more accurate estimations.

Testable

- We do not develop what we cannot test.
If you can't test it then you will never know when you are done.

Example: "software should be easy to use".



Examples

"As a <role>, I want <goal/desire> so that <benefit>"

- ▶ **As a student, I can** find my grades online **so that** I don't have to wait until the next day to know whether I passed.
- ▶ **As a non-administrative user, I want to** modify my own schedules but not the schedules of other users.
- ▶ **As a book shopper, I can** read reviews of a selected book **to** help me decide whether to buy it



COUNTEREXAMPLES

- ▶ “Write game rules.”
 - ▶ Drawbacks: no business *Value*, not *Estimable*.
 - ▶ Better: “As a newbie game player, I want to know who goes first so we can start the game.”
 - ▶ Better: “As a competitive gamer, I want a way to leapfrog my opposing players.”



Benefits

- ▶ They are very helpful to business because
 - ▶ they define what is to actually be built in the software project.
 - ▶ The customer is able to prioritize in his formulated user stories which ideas are most important for the developer
 - ▶ the tasks can be broken down into elements for better estimation.
- ▶ The user stories offer the chance for developers to directly converse with the customer about their needs and goals.
- ▶ The developer can test when the user stories are done, and report back to the customer with results.



Limitations

Some of the limitations of user stories in agile methodologies:

- ▶ They can be difficult to scale to large projects.
- ▶ They are regarded as conversation starters.

[7]



Use Case VS User Story

- ▶ A **user story** is a *lightweight* document that can be written on a card (In order to , as a , I want). A User Story doesn't capture all the details, it's an informal support for the discussion.
- ▶ A **use case** is an *heavyweight* document that needs a word document. It describes a "Normal Flow" of steps and/or actions and "Alternative Flows" which are detailed. A Use Case captures all the details, it's a formal specification.



Use Case VS User Story (cont'd)

- ▶ A **user story** is from a customer's point of view, sometime it's incorrect or incomplete. It may have no consideration on performance, on error handling, or nothing on the backend.
- ▶ A **use case** is from developer's point of view. It's accurate and complete. It should answer all the requirements from customers.



Use Case VS User Story (cont'd)

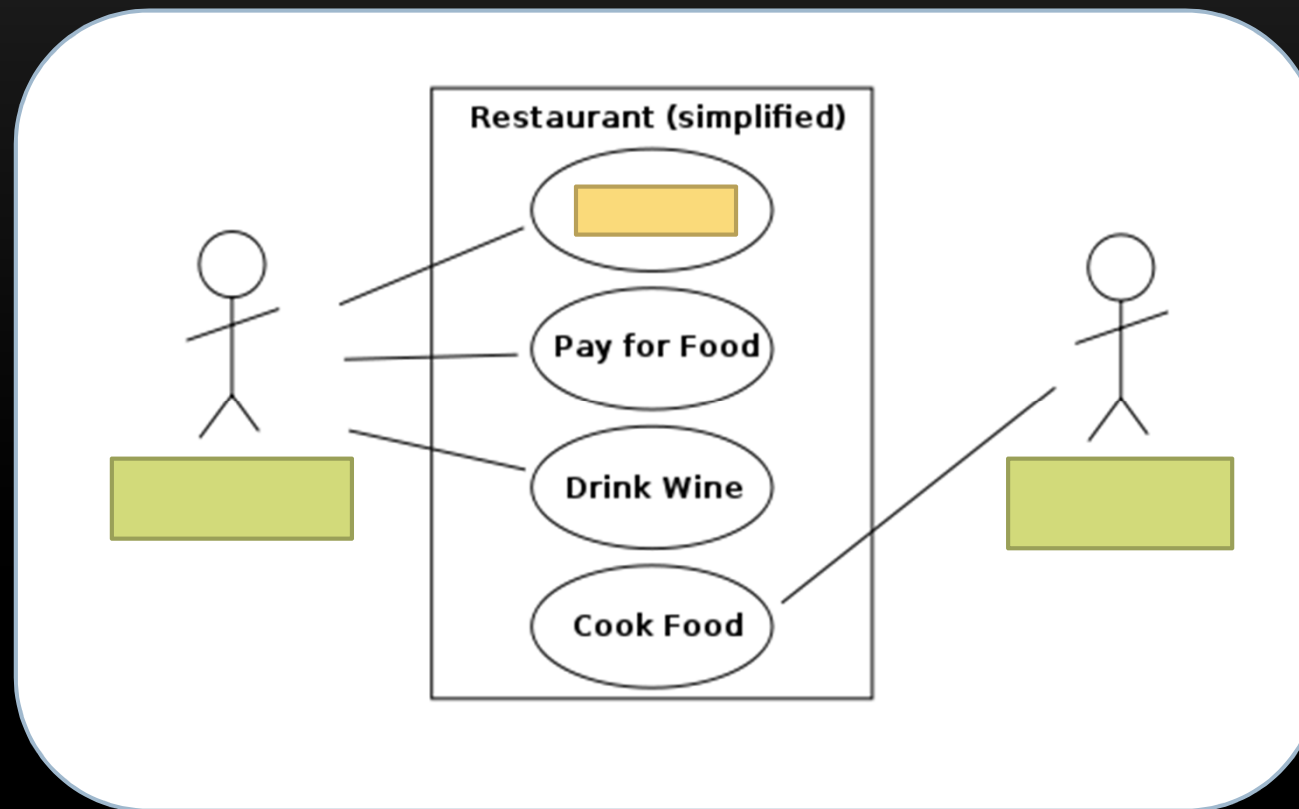
- ▶ **User stories are about needs.** You're describing is a "raw" user need. It's something that the user needs to do in his day-to-day job. If you never build any software for him, then that need will still exist!
- ▶ **Use cases are about the behavior you'll build into the software to meet those needs.** A developer should be able to read a use case that what the software needs to do. It has a lot of detail, and describes everything that the developer needs to build in order to meet the user's need. That's why it needs to be clear and unambiguous.

[8]

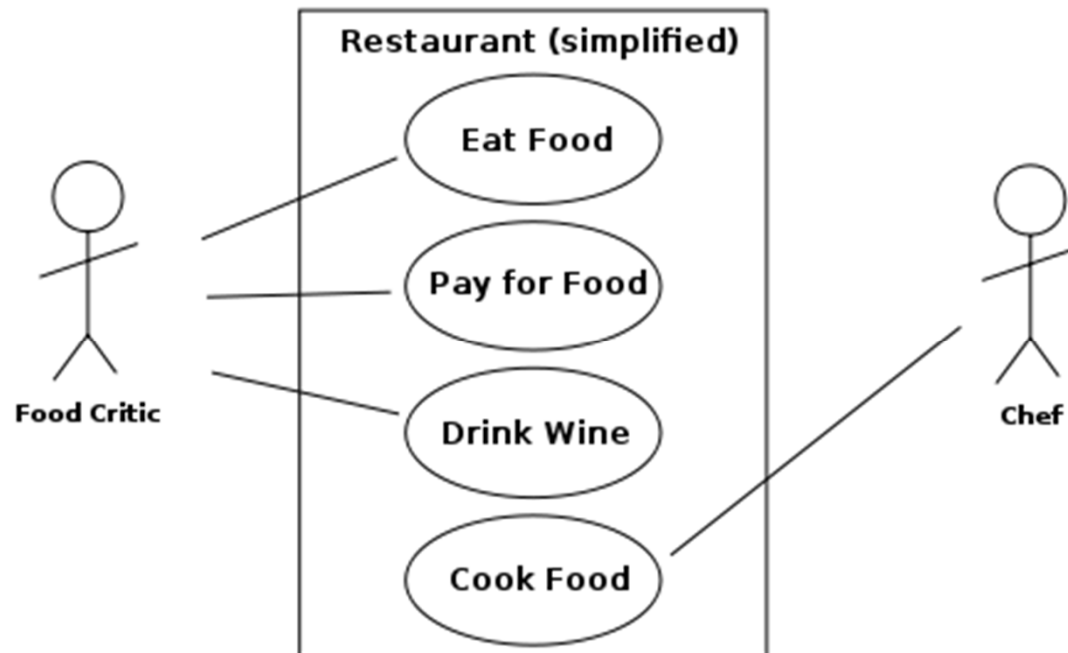


Exercise 1 : Making Use Case Diagram

Please design use case diagram for
“a *simple* restaurant model”



Answer exercise 1:



References

1. <http://en.wikipedia.org/wiki/Requirements>
2. [http://en.wikipedia.org/wiki/Requirements engineering](http://en.wikipedia.org/wiki/Requirements_engineering)
3. [s](#)
4. <http://www.subcide.com/articles/how-to-write-meaningful-user-stories/>
5. <http://www.mountangoatsoftware.com/articles/27-advantages-of-user-stories-for-requirements>
6. <http://agilesoftwaredevelopment.com/blog/vaibhav/good-user-story-invest>
7. [http://en.wikipedia.org/wiki/User story](http://en.wikipedia.org/wiki/User_story)
8. <http://www.stellman-greene.com/2009/05/03/requirements-101-user-stories-vs-use-cases/>



References

9. http://en.wikipedia.org/wiki/Use_case
10. Use Case -. Minder Chen, 1994-2004. UML. Use Case Modeling.
11. http://en.wikipedia.org/wiki/Non-functional_requirement
12. <http://www.requirementsauthority.com/functional-and-non-functional.html>
13. http://en.wikipedia.org/wiki/Functional_requirements
14. <http://www.lessons-from-history.com/node/83>

