Notations and Meta-Models

Notation:
• a graphical representation of a model, e.g. classes, associations, multiplicity
• appeals to intuition rather than formal rigor

Meta-model:
• a diagram which explains the notation
• UML diagrams explaining UML itself
0.1 The meta-model describes the syntax of the UML diagrams and thus defines what a well-formed model is.

Analysis and Design

- Final goal: “cutting code”
- UML diagrams etc are “just pretty pictures”
- HOWEVER:
  - Communication is of utmost importance, both inside the development team and outside with the customer/user
  - Build “the right” system – need for a “domain expert”
  - Need to concentrate and highlight the important details
  - Make extensive use of “use cases”
  - Build a road-map for the construction
  - Exploit o-o techniques
The UML Development Process

Inception → Elaboration → Construction → Transition

In the elaboration phase the requirements are refined and high-level analysis and design is carried out.

The construction phase consists of many iterations; it is the key phase.

The transition phase is mainly for Beta-testing, acceptance, performance optimization, and user training.

Elaboration

✦ Problem definition
  1. What is it we are going to build?
  2. How are we going to build it?

✦ What are the risks
  1. Requirement risks – did we understand what the customer wants?
  2. Technological risks – can it be done with current technology?
  3. Skill risks – can we do it with our expertise?
  4. Political risks – always around, be realistic, always look at the $$$ side.

✦ Plan the construction phase – iterations and milestones.
Views and Diagrams

❖ A model is an abstraction of a system or a context
❖ An architectural view is an abstraction of a model,
   • taken from a specific perspective / view
   • enabling the extraction of architecturally essential elements
❖ Starting point is always the “user view” as defined by the use cases
❖ The different model views:
   • structural view
   • behavioral view
   • environmental view
   • implementation view

Views and Diagrams - 2

❖ Structural view:
   • class diagrams (generic templates, “types”)
   • object diagrams – instances, “variables”

❖ Behavioral view:
   • sequence diagrams
   • collaboration diagrams
   • state chart diagrams
   • activity diagrams

❖ Implementation view:
   • component diagrams – modules

❖ Environment view:
   • deployment diagrams
UML Elements

Things – Entities
Relationships – Associations
(Diagrams – Views)

graphical symbols

relationships

associations

adornments

Use Cases

ȝ Scenarios, typical transactions and interactions have long been used for understanding requirements.
ȝ Use cases do this in a more systematic manner
ȝ Introduced by Jacobson initially for telecommunications systems
ȝ Based on “scenarios”:
  • a scenario is a sequence of steps describing the interaction between a user and a system;
  • a user is a “role” i.e. an external entity using the system in some way, for example by sending signals, entering data, etc.; it could be a human user or a device, external system;
  • example: “the sensor sends a report consisting of 16 bytes every 40 msec”
Use Cases - 2

» A scenario is one transaction that can happen, but it could fail or lead to an alternate transaction

» A use case is:
  • A set of scenarios tied together by a common user goal.
  • Frequently one finds a common “all-goes-well” case and many alternatives
  • These alternatives might cover special cases (things go well) and error situations (things go bad).

» Example use case: “Serving Dinner”
  • make use of “structured writing”

Example: Serving Dinner

1. The use case begins when the actor Guest enters the restaurant.
2. The actor Guest has the possibility of leaving his/her (!) coat in the cloakroom after which he is shown to a table and given a menu.
3. When the actor Guest has had sufficient time to make up his mind, he is asked to state his order. Alternatively, Guest can attract the waiter’s attention so that the order can be placed.
4. When the guest has ordered, the kitchen is informed what food and beverages contain.
5. In the kitchen certain basic ingredients such as sauces, rice, chilies, tacos, already have been prepared. Cooking therefore involves collecting together the basic ingredients, adding spices and so on and sorting out what needs to be done just before the dish is served. Also, the required beverages are fetched from its proper storage.
6. When the dish is ready, it is served to the actor Guest. When it has been eaten, the actor is expected to attract the waiter’s attention.
7. When the bill is paid, Guest can fetch his coat from the cloakroom and leave the restaurant. The use case is then completed.
Caveat: when writing the use cases do NOT concentrate on the functions but on the interactions!

A transaction is defined as an "atomic set of activities that are performed fully or not at all; it is invoked by a stimulus from an actor to the system or by a point in time being reached. A transaction consists of actions, decisions, and transmission of stimuli to the invoking actor or some other actor(s)."

An actor is a "role that someone or something in the environment can play in relation to a business"; alternatively, "an actor represents anything that needs to exchange information with the system. An "individual actor" sometimes referred to as user) is defined as an instance of class Actor. Further, the same person or other item can assume more than one role."
Use Cases – cont.

- Actors / roles carry out business use cases
- A business use case is typically broken down into a set of systems use cases
- Start with a set of actors and then work out the use cases for each actor.
- Many variations are possible, e.g.:
  - actors – every external system or user or just the initiator
  - show the one that “gets value”, the primary actor
  - do NOT include everything – instead make up separate use cases
  - add detail during the iterations

Fig 3-1,-2 Example Use Case and Diagram

Example – e-commerce

Buy a Product
1. Customer browses through catalog and selects items to buy
2. Customer goes to check out
3. Customer fills in shipping information (address; next-day or 3-day delivery)
4. System presents full pricing information, including shipping
5. Customer fills in credit card information
6. System authorizes purchase
7. System confirms sale immediately
8. System sends confirming email to customer

Alternative: Authorization Failure
At step 6, system fails to authorize credit purchase
Allow customer to re-enter credit card information and re-try

Alternative: Regular Customer
3a. System displays current shipping information, pricing information, and last four digits of credit card information
3b. Customer may accept or override these defaults
Return to primary scenario at step 6
Actors

- Actor is a “role” with respect to the system, e.g. “Trader”, “Waiter”, “Guest”, “Subsystem”, “NetworkInterface”, …
- Possible actors: humans, external systems, system components, devices, …
- Actors carry out use cases, (transactions, not functions)
  - First step: define the list of actors,
  - Second step: define the use cases for each actor.
- The goal is a description of the use cases, not the actors.
- External events are an excellent source for identifying use cases. A given event:
  - may cause a system reaction,
  - may cause a reaction from the user.
- A business use case is a response to the user or an event.
- A system use case is an interaction with the software.

*** remember: actors can also be receivers of output!

Example – cont.

Figure 3-2: Use Case Diagram
UC - Relationships

The include relationship factors common activities in a separate sub-use case which is referred to from the more general use cases.

Similarities between use cases can be handled by a use case generalization which is another way to capture alternative scenarios.

When more rules have to be observed in handling alternatives, a use case may be extended, which adds behavior to the base use case, but only at declared extension points.

UC – Relationships – cont.

Both generalization and extension allow splitting up a use case. This frequently happens during elaboration when a use case gets too complicated to be handled within one iteration. Handle the normal case first and the variations later:

- Use include when repetition occurs in two or more separate use cases [and you want to avoid repetition]
- Use generalization when describing a variation of normal behavior [and you wish to describe it casually]
- Use extend when describing a variation of normal behavior and the more controlled form declaring extension points in the base use case is preferred.
When to use Use Cases

"I can't imagine a situation in which I would not use use cases" – Fowler

- They are an essential tool in requirements capture and in planning and controlling an iterative project
- Capturing use cases is one of the primary tasks of the elaboration phase.
- Try to do use cases and conceptual modeling at the same time
- Use Cases represent an external view of the system
- Do not expect, hence, to see correlations between use cases and the classes inside the system.

Fred Brooks – The Mythical Man Month

"The hardest single part of building a software system is deciding precisely what to build" [p.199]

No other part of the conceptual work is so difficult as:
- Establishing the detailed technical requirements
- Including all interfaces to
  - people
  - machines / systems
  - other software systems / components

No other part so cripples the resulting system if done wrong and is more difficult to rectify later. (security!)

Therefore the most important function that software builders do for their clients is the
- Iterative extraction
- Iterative refinement
  of the product requirements

For the truth is the clients usually do not know what questions must be answered.
Building a Software System

- Elaborate the user/customer requirements - build the base use cases
- Construct the system in a number of iterations and refinements, building a series of models
  - requirements model
  - analysis model
  - design model
  - implementation model
  - test model
- Incremental development - the first approach is just a running skeleton executing a set of dummy methods; the first prototype performs mainline tasks, leaving out handling of exceptions, alternatives, incorrect inputs, clean-up and clean abort.
- Each model captures some part of the system and are the output of the individual activities

What are Use Cases?

- What is the purpose of a Use Case?
  - to gather user stories, or build requirements? (the values are stories, or requirements)?
- What is the contents of a Use Case?
  - required to be consistent, or can they be self-contradicting? If consistent, are they in plain prose or are they in a formal notation (the values are contradicting, consistent prose, formal content).
- What is the plurality of a Use Case?
  - is a use case really just another name for a scenario, or does a use case contain more than one use case? (the values are 1 or multiple)
- What is the structure of a Use Case?
  - Does a collection of use cases have a formal structure, an informal structure, or do they form an unstructured collection (the values are unstructured, semi-formal, formal structure).
Use Cases

- **Purpose** = requirements
- **Contents** = consistent prose
- **Plurality** = multiple scenarios per use case
- **Structure** = semi-formal

UML Use Case diagrams only show the relationships

What are Use Cases?

- **What does a Use Case look like?** - what is a Use Case?
  - A use case captures a contract between the stakeholders of a system and the system’s behavior.
  - The use case describes the system’s behavior under various conditions as the system responds to a request from one of the stakeholders, called the primary actor.
  - The primary actor initiates an interaction with the system to accomplish some goal. The system responds, protecting the interests of all the stakeholders.
  - Different sequences of behavior, or scenarios, can unfold, depending on the particular requests made and the conditions surrounding the requests.
Interactions

In the simplest case, an interaction is simply the sending of a message.

An interaction also could be a sequence of interactions.
- This is a recursive definition; at the bottom level, it consists of messages.
- Sometimes a sequence of messages is bundled into a single interaction item.

A sequence has no branching or alternatives.
- It is used to describe the past or a definite future, with conditions stated.
- Such a sequence is known as a scenario.

"Sequence of Interactions" is the same as "scenario".

In order to describe a system, we need to collect all the scenarios that might occur during one interaction.

Example

A collection of scenarios is a use case

An interaction with a banking machine may consist of the following scenarios.
- A use case answers a question of the form, "How do I get money out of that banking machine?"
- You, the primary actor, have a goal, to get money out the system, which is that banking machine.
- There are a number of situations you could find yourself in.
- The use case collects them into one place.
- It breaks down your goal into subgoals, eventually individual message actions, plus the interactions between various actors as they try to reach that goal, or eventually fail to reach it.

Scenarios and use cases go until goal success or abandonment.
When to stop

How do you know when to stop writing a scenario or use case?

There are two clauses that have to be made explicit to get the proper bounds on a use case and a scenario. The clauses are the same for each:

- Clause 1. All the interactions relate to the same goal.
- Clause 2. Interactions start at the triggering event and end when the goal is delivered or abandoned, and the system completes its responsibilities with respect to the interaction.

Definitions

Scenario. A sequence\(^\text{1}\) of interactions happening under certain conditions, to achieve the primary actor's goal, and having a particular result with respect to that goal. The interactions start from the triggering action and continue until the goal is delivered or abandoned, and the system completes whatever responsibilities it has with respect to the interaction.

- \(^\text{1}\) "Single-rooted strict partial ordering" is the correct phrase; messages can happen in parallel - write it however you prefer or the tool permits.

Use Case. A collection of possible scenarios between the system under discussion and external actors, characterized by the goal the primary actor has toward the system is declared responsibilities, showing how the primary actor's goal might be delivered or might fail.
Characteristic Information

The characteristic information for a use case is
- Primary Actor or actors
- Goal
- Scenarios used

The characteristic information for a scenario is
- Primary actor
- Goal
- Conditions under which scenario occurs
- Scenario result - goal delivery or failure.

Sample Scenario

System under discussion: the insurance company
Primary Actor: me, the claimant
Goal: I get paid for my car accident
Conditions: Everything is in order
Outcome: Insurance company pays claim
1. Claimant submits claim with substantiating data.
2. Insurance company verifies claimant owns a valid policy
   (failure here probably means goal failure)
3. Insurance company assigns agent to examine case.
4. Agent verifies all details are within policy guidelines.
   (an interaction between the agent and secondary actors)
5. Insurance company pays claimant
   (implies all preceding goals managed to pass)
Sample Use Case

System under discussion: the insurance company

Primary Actor: me, the claimant

Goal: I get paid for my car accident

Conditions: Everything is in order

Outcome: Insurance company pays claim

1. Claimant submits claim with substantiating data.
2. Insurance company verifies claimant owns a valid policy
3. Insurance company assigns agent to examine case.
4. Agent verifies all details are within policy guidelines.
5. Insurance company pays claimant

Extensions - - >
Sample Use Case

Variations
1. Claimant is
   a. a person
   b. another insurance company
   c. the government

5. Payment is
   a. by check
   b. by interbank transfer
   c. by automatic prepayment of next installment
   d. by creation and payment of another policy

Observations and Conclusions

The explosion of scenarios is prevented because a subordinate use case contains and conceals a possibly large number of alternative paths.

Thanks to the recursive approach, at each point in the scenario, everything that might possibly happen next gets reduced to a single pair - the next subgoal succeeds, or it fails.

All the conditions that force one or another recovery scenario in the subordinate use case are concealed within the use case and need not be dealt with there.

What has to be distinguished are the conditions that separate success from failure for the subordinate use case.

Thus, the outer scenario will name some conditions in the world, but ignore certain others. The inner use case’s scenarios will introduce some new conditions that are relevant to those scenarios only.

The list of conditions grows and gets more detailed as we proceed down the chain.
Observations and Conclusions

In the case of a subordinate use case that is used by many higher-level use cases, it will probably ignore some of the conditions referenced by the higher-level use case.
The actors usually do not know the conditions of the scenario at the start.
The conditions are those that will drive the actors down a particular path.

For example, a bank withdrawal scenario may have as conditions, "insufficient funds". The customer does not know this when he or she walks up to the ATM. However, that is the scenario that plays when he/she requests more money than he/she has in the bank account.

Example 1

Use Case "Get Paid for Car Accident"
Primary Actor: Claimant
Scope: Insurance Company ("MyInsCo")
Level: Summary
Stakeholders and Interests:
Claimant — to get paid the most possible.
MyInsCo — to pay the smallest appropriate amount.
Department of Insurance — to see that all guidelines are followed.
Precondition: None.
Minimal Guarantees: MyInsCo logs the claim and all activities.
Success Guarantees: Claimant and MyInsCo agree on amount to be paid; claimant gets paid that.
Trigger: Claimant submits a claim.
- - ->
Example 1

Main Success Scenario:
1. Claimant submits claim with substantiating data.
2. Insurance Company verifies claimant owns a valid policy.
3. Insurance Company assigns agent to examine case.
4. Insurance Company verifies all details are within policy guidelines.
5. Insurance Company pays claimant and closes file.

Extensions:
1a. Submitted data is incomplete:
   1a1. Insurance Company requests missing information.
   1a2. Claimant supplies missing information.
2a. Claimant does not own a valid policy:
   2a1. Insurance Company denies claim, notifies claimant, records all this, terminates proceedings.

Example 1

2a. Claimant does not own a valid policy:
   2a1. Insurance Company denies claim, notifies claimant, records all this, terminates proceedings.

3a. No agents are available at this time.
   3a1. (What does the insurance Company do here?)

4a. Accident violates basic policy guidelines:
   4a1. Insurance Company denies claim, notifies claimant, records all this, terminates proceedings.

4b. Accident violates some minor policy guidelines:
   4b1. Insurance Company begins negotiation with claimant as to amount of payment to be made.
What are Use Cases?

- Use cases are fundamentally a text form, although they can be written using flow charts, sequence charts, Petri nets, or programming languages.
- Under normal circumstances, they serve as a means of communication from one person to another, often among people with no special training. Simple text is, therefore, usually the best choice.
- The use case, as a form of writing, can stimulate discussion within a team about an upcoming system. The team might or might not document the actual requirements with use cases. Another team might document their final design with use cases.
- All of the above might be done for a system as large as an entire company or as small as a piece of a software application program.
- The same basic rules of writing apply to all of these situations, even though the teams will write with different amounts of rigor and at different levels of technical detail.

Use Cases and SUD

- Use cases document a certain "system under discussion" (SUD)
  - Use cases can document an organization's business processes;
    - The SUD is the organization itself.
    - The stakeholders are the company shareholders, customers, vendors, and government regulatory agencies.
    - The primary actors include the company's customers and perhaps their suppliers.
  - Use cases can record behavioral requirements for a piece of software;
    - here the SUD is the computer program.
    - The stakeholders are the people who use the program, the company that owns it, government regulatory agencies, and other computer programs.
    - The primary actor is the user sitting at the computer screen or another computer system.
Writing Use Cases

* A well-written use case is easy to read. It consists of sentences written in only one grammatical form — a simple action step — in which an actor achieves a result or passes information to another actor. Learning to read a use case should not take more than a few minutes.

* Learning to write a good use case is harder. The writer has to master three concepts that apply to every sentence in the use case and to the use case as a whole.

* The three concepts are
  - Scope: What is really the system under discussion?
  - Primary actor: Who has the goal?
  - Level: How high- or low-level is that goal?

Summary Definitions:

* Actor: anyone or anything with behavior.

* Stakeholder: someone or something with a vested interest in the behavior of the system under discussion (SuD).

* Primary actor: the stakeholder who or which initiates an interaction with the SuD to achieve a goal.

* Use case: a contract for the behavior of the SuD.

* Scope: identifies the system that we are discussing.

* Preconditions and guarantees (postconditions): what must be true before and after the use case runs.
Summary Definitions - cont:

- **Main success scenario**: a case in which nothing goes wrong.
- **Extensions**: what can happen differently during that scenario.
- **Numbers in the extensions**: refer to the step numbers in the main success scenario at which each different situation is detected (for instance, steps 4a and 4b indicate two different conditions that can show up at step 4).
- **When a use case references another use case**, the referenced use case is underlined (notation convention).

Examples:

---

**Example 2**

Example Use Case: "Register Arrival of a Box" - Shipping Dept.

- RA—"Receiving Agent"
- RO—"Registration Operator"
- **Primary Actor**: RA
- **Scope**: Night-time Receiving Registry Software
- **Level**: User goal

**Main Success Scenario**:

1. RA receives and opens box (box ID, bags with bag IDs) from Transport Company (TC)
2. RA validates box ID with TC registered IDs.
3. RA maybe signs paper form for delivery person.

- - >
Example 2

4. RA registers box’s arrival into system, which stores:
   RA ID
   Date, time
   Box ID
   Transport Company
   <Person name?>
   # bags (With bag IDs?)
   <Estimated value?>

5. RA removes bags from box, puts on cart, takes to RO.

Extensions: - - ->

Extensions:
2a. Box ID does not match transport company ID.
4a. Fire alarm goes off and interrupts registration.
4b. Computer goes down.
   Leave money on desk and wait for computer to come back up.

Variations:
4’. With and without Person ID.
4”. With and without estimated value.
5’. RA leaves bags in box.
Situations and Systems

Use cases can document different situations ad systems such as:
- A business work process
- Discussions about upcoming system requirements
- The functional requirements for a (software) system
- To document the design of the system

The purpose determines the writing style and the form (subform):
- “casual” vs “fully dressed” Use Cases (small/large group)
- Business use cases describe the operation of the organization/business
- System use cases document the design or break the requirements into smaller subsystems/subfunctions
- Black box use cases do not discuss the innards of the system
- White box use cases exhibit the internal structure and processes

Use Case Formats

The largest difference between use case formats is how "dressed up" they are.

Consider these two following scenarios:

- Case 1: A team is working on software for a large, mission-critical project. They decide that the extra ceremony is worth the extra cost, so
  - (a) the use case template needs to be longer and more detailed,
  - (b) the writing team should write in the same style to reduce ambiguity and misunderstanding, and
  - (c) the reviews should be tighter to more closely scrutinize the use cases for omissions and ambiguities.
- Having little tolerance for mistakes, they decide to reduce tolerances (variation between people) in the use case writing as well.
Use Case Formats

• Case 2: A team of three to five people is building a system whose worst damage is the loss of comfort, easily remedied with a phone call. They consider all the ceremony a waste of time, energy, and money. They therefore choose
  – (a) a simpler template,
  – (b) to tolerate more variation in writing style, and
  – (c) fewer and more forgiving reviews.
• The errors and omissions in the writing are to be caught by other project mechanisms, probably conversations among teammates and with users. They can tolerate more errors in their written communication and so more casual writing and more variation between people.

☞ Neither approach is wrong.
☞ Such choices must be made on a project-by-project basis.

Use Case Formats – cont.

☞ Neither approach is wrong. Such choices must be made on a project-by-project basis.
☞ The mistake is getting too caught up in precision and rigor when they are not needed, which will cost the project a team lot in time and energy.

☞ There must be at least two use case formats:
  • A casual one for low ceremony projects
  • A fully dressed one for high ceremony projects
  • Any one project will adapt one or a modification of the two forms.
Example 3 - Casual Version

Use Case 3a "Buy Something"

The Requestor initiates a request and sends it to her or his Approver.
The Approver checks that there is money in the budget, checks the price of the goods, completes the request for submission, and sends it to the Buyer.
The Buyer checks the contents of storage, finding the best vendor for goods.
The Authorizer validates Approver’s signature.
The Buyer completes request for ordering, initiates PO with Vendor.
The Vendor delivers goods to Receiving, gets receipt for delivery (out of scope of system under design).
The Receiver registers delivery, sends goods to Requestor.
The Requestor marks request delivered.

At any time prior to receiving goods, the Requestor can change or cancel the request. Canceling it removes it from any active processing (deletes it from system?). Reducing the price leaves it intact in processing.

Raising the price sends it back to the Approver.
Example 3 – Fully Dressed Version

Use Case 3b "Buy Something"

**Primary Actor:** Requestor

**Goal in Context:** Requestor buys something through the system, gets it. Does not include paying for it.

**Scope:** Business — the overall purchasing mechanism, electronic and nonelectronic, as seen by the people in the company

**Level:** Summary

**Stakeholders and Interests:**
- **Requestor:** Wants what he/she ordered, easy way to do that.
- **Company:** Wants to control spending but allow needed purchases.
- **Vendor:** Wants to get paid for any goods delivered.

**Precondition:** none

**Minimal Guarantees:** Every order sent out has been approved by a valid authorizer. Order was tracked so that company can be billed only for valid goods received.

---

**Success Guarantees:** Requestor has goods, correct budget ready to be debited.

**Trigger:** Requestor decides to buy something.

**Main Success Scenario:**

1. Requestor: initiate a request.
2. Approver: check money in budget, check price of goods, complete request for submission.
5. Buyer: complete request for ordering, initiate PO with Vendor.
6. Vendor: deliver goods to Receiving, get receipt for delivery (out of scope of system under design).
8. Requestor: mark request delivered.
Example 3 – Fully Dressed Version

Extensions:

1. a. Requestor does not know vendor or price: Leave those parts blank and continue.

1. b. At any time prior to receiving goods, Requestor can change or cancel request:
   - Canceling it removes it from active processing (Delete from system?).
   - Reducing price leaves it intact in processing.
   - Raising price sends it back to Approver.

2a. Approver does not know vendor or price: Leave blank and let Buyer fill in or callback.

2b. Approver is not Requestor’s manager: Still OK as long as Approver signs.

2c. Approver declines: Send back to Requestor for change or deletion.

3a. Buyer finds goods in storage: Send those up, reduce request by that amount, and carry on.

3b. Buyer fills in Vendor and price, which were missing: Request gets resent to Approver.

4a. Authorizer declines Approver: Send back to Requestor and remove from active processing. (What does this mean?)

5a. Request involves multiple Vendors: Buyer generates multiple POs.

5b. Buyer merges multiple requests: Same process, but mark PO with the requests being merged.

6a. Vendor does not deliver on time: System does alert of non-delivery.

7a. Partial delivery: Receiver marks partial delivery on PO and continues.

7b. Partial delivery of multiple-request PO: Receiver assigns quantities to requests and continues.

8a. Goods are incorrect or improper quality: Requestor refuses delivered goods. (What does this mean?)

8b. Requestor has quit the company: Buyer checks with Requestor’s manager: either reassign Requestor or return goods and cancel request.
Example 3 – Fully Dressed Version

Technology and Data Variations List: None.
Priority: Various
Releases: Several
Response Time: Various
Frequency of Use: 3/day
Channel to Primary Actor: Internet browser, mail system, or equivalent
Secondary Actors: Vendor
Channels to Secondary Actors: Fax, phone, car

Open Issues:
When is a canceled request deleted from the system?
What authorization is needed to cancel a request?

Example 3 – Fully Dressed Version

Open Issues:
When is a canceled request deleted from the system?
What authorization is needed to cancel a request?
Who can alter a request’s contents?
What change history must be maintained on requests?
What happens when Requestor refuses delivered goods?
How does a requisition work differently from an order?
How does ordering reference and make use of the internal storage?

==============