

# Requirements

---

Chapter 4 and 5

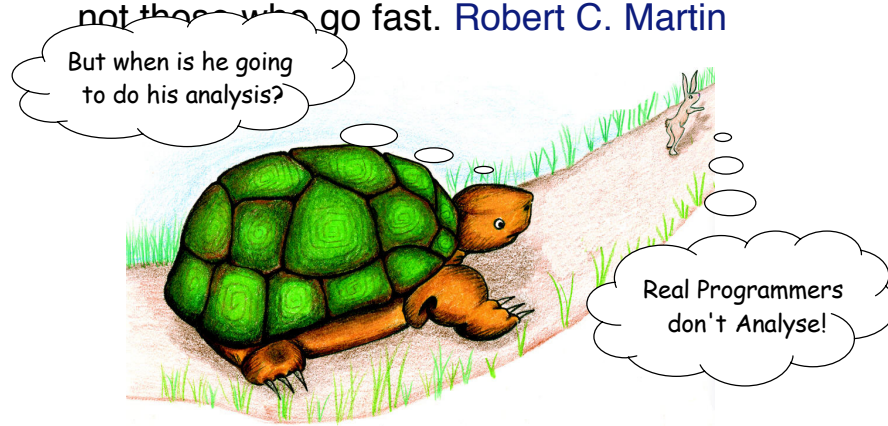
## Outline

---

- Requirements Analysis
- What are requirements
- Classic/Formal Requirements
- Use Cases
- Use Case Diagrams
- Other Issues
  - Architectural Vision
  - Staged Development
  - Rechecking the Requirements

## Requirements Analysis

- In software the race goes to those who go well, not those who go fast. Robert C. Martin



## What Are Requirements?

- Lots of different kinds of requirements, depending on context and environment
- Functional Requirements are the focus for this class
  - “What the system is for” / “What the system should do”
  - boundary placed on what is part of the system and what is external to the system
  - black-box - system from the outside
- Formed using **carefully constructed** English statements (with some supporting diagrams)
  - Classic or “Formal” Requirements
    - From the “system’s” viewpoint
  - Use Cases
    - From the “user’s” viewpoint

## What Are Requirements?

- Requirements are statements that would be unchanged even if the technology changed dramatically
- Contract between sponsor/stakeholders/user and developer
  - Every design element must be traceable to a formal requirement or use-case



## Requirements and Design



**Keep design out of the requirements!!!**

caveats:

- requirements should be technically feasible
- requirements could be “based on” or “inspired by” a technology
  - e.g. multi-player **internet** games



## Classic Requirements

- Definition:
  - A set of atomic statements about what the system
    - must do
    - should do
    - might do
  
- Example version 1:
  - The system must take a facial image, in any of the common image formats such as, today, JPEG, ..., TIFF, produce a numerically-based profile and find other pictures, from selected databases, that are likely to be of the same person

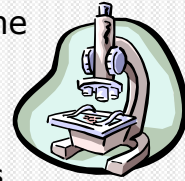
## Classic Requirements: Statement Style



- Atomic
  - Single “irreducible” concept per sentence/requirement
    - no ‘and’ or other conjunctions
  
- Example version 2:
  - The system must register the location of, or record the content of, images in any of the common image formats; today including such formats as JPEG, ..., TIFF
  - The system must take a selected facial image, and express the characteristics of the image as a numerically-based profile that will serve to identify that face among all possible faces
  - The system must take a facial image numeric profile and find all likely images of the same face from selected image sources

## Quantifying Requirements

- add **well researched** numbers to the requirements (e.g. numeric boundary constraints)
- time consuming
  - only do for most important requirements
- Example- version 3:
  - The system must register the location of indicated images using something at least as concise and precise as a URL, or record the content of the images in any of the common formats... to any resolution tending to be encountered, which today would vary from 75dp to 1200 dpi. Formats need not be restricted to raster formats; vector formats may also be employed.



## Classic Requirements: Statement Style

- Numbered
  - e.g. R1, R2, etc.
  - commonly in order of importance
  - important when referring to a requirement in the design document - for traceability
    - useful if have tool to automatically update requirement references in the design document
    - allows requirements to be inserted / re-ordered on next iteration of the “spiral”

# Classic Requirements: Advantages



- promotes thinking at a high level
  - gives good overview of the system
  - provides starting point to “pull” use cases from
- captures requirements that are not focused on user interactions with the system
  - Things that a system **cannot do**
  - Things system must do, but are implicit in the user interaction with the system
- concise
  - combinatorial (factorial) number of scenarios



# Case Study: ALICE

- The system would, at a minimum:
  - assist with placing and recording bookings, helping ensure optimal table utilization, balancing the avoidance of double bookings with the need to take one or two “floating” bookings;
  - take table orders, optionally splitting order printouts to departments (starter, main, desert, etc.; accept event information, such as “table away”, in order that, for example, tables still awaiting their main courses can be intuitively depicted;
  - produce bills;
  - assist with menu production and usage history;
  - store recipes for some dishes;
- The system might also:
  - maintain stock level information,
  - especially for the more expensive items.



<http://www.bestprices.com/cgi-bin/vlink/075992743921BT.html>

## Example: Critical (must)

1. Register the details of a booking request.
2. Allocate a given booking to a given table, advising of any unsuitable or impossible characteristics it discerns such as, but not limited to, a table that is too small, too big, not free, or not non-smoking.
3. Locate a table of suitable capacity for a given booking (or a table that is minimally bigger than the party size), where the table has the required characteristics – for example, but not limited to, smoking/non-smoking or quiet – and where the table is free at the time in question.
4. Note updates regarding table availability to reflect such things as “walk-ins” (diners who have not booked).
5. Register changes to booking details advising of any consequential unsuitability or impossibility.
6. Flexibly present the details (at least including the contact name) of bookings expected within a selected time period, including now, for one or more selected tables.
7. Register the arrival of a party who have booked thus updating the table to an occupied condition.

## Example: Great to have (should)

24. Understand (and use in matching bookings to tables) a set of booking characteristics that is user-configurable, and could include such things as smoking, non-smoking, quiet and good legroom.
25. When indicating and presenting to the user such things as tables, bookings and orders, use an intuitive and appropriate metaphor such as screen forms for bookings and orders, calendars for bookings, and two-dimensional approximated physical position for tables.
26. Register, update and recall the allocation of waiters to tables and changes thereto.
27. Report time to deliver a dish either by using recipe information or by using any kitchen or table preparation times registered for that dish.
28. Raise an alert when the likely number of servings remaining has been recorded for a dish, and a meal order requests servings in excess of that.

## Nice to have (could)

---

29. Calculate approximate stock depletion from meal orders placed.

## Change case (won't)

---

1. Notify stock items below minimum levels on request.
2. Notify stock items near expiry.
3. Register stock level corrections.
4. Register stock item information such as quantity-on-hand, normal unit, maximum level, minimum level, re-order level, typical re-order quantity.
5. Register stock acquisitions.
6. Accept the booking of a party to more than one table.
7. Support the analysis and correlation of takings, including but not limited to takings by table, time, date, season and menu.
8. Accept bookings for the entire restaurant.
9. Accept bookings for an entire floor of the restaurant.

## Break the following scenario down into "atomic", i.e. irreducible requirements.

"We would typically call up yesterday's menu. We would check with Chef to see if any particular stock items need using up via any particular dishes. I guess that maybe the system could help us with alerts as to any expensive stock approaching its use-by date. We would amend the menu to remove dishes not on offer today and to add the new dishes like the stock-using dishes and any other dishes that the manager or Chef have planned."

(You-asking questions) "Would you type up the details of each new dish?"

"Well I guess there might be totally new dishes; but mostly we would want to call up some kind of overall dish list, where the typical price, description, etc. was already entered. Sometimes we would have to amend the defaults – for expensive ingredients whose prices fluctuate wildly, for example."

(You-more questions) "And would you be changing yesterday's menu to become today's? Or would you be leaving yesterday's menu on record and creating today's from it?"

"Oh, we would definitely want to keep a record of previously used menus; we'd simply be using yesterday's as a typical starting point. I imagine there would be times, though, when we'd create a menu completely from scratch."

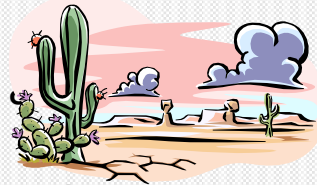
## Must/Should

- locate the menus used on a given date, and warn if the number of qualifying menus isn't exactly one.
- display [please see interface/look/feel requirements section] a selected menu.
- create a new menu by copying a selected, existing menu.
- note dish(es) selected from a selected menu.
- remove selected dish(es) from selected menu.
- locate any dishes including those that are not necessarily related to any menus.
- support the specification and creation of new dishes.
- register and record changes to the specification of an existing dish.
- register and record the addition of a selected dish to a selected menu.
- record the details of the menu used on a given day (including but not limited to the details of the dishes it offered).
- create a new and empty menu from scratch, ready to accept any dishes that will be selected for it.

## Disadvantages

---

- dry
- no sense of the dynamics of the system
  - humans tend to think in terms of “systems in action”
- can promote thinking level
  - can miss vital requirements



## Use Cases

---

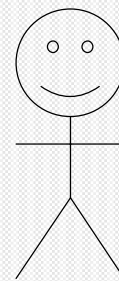
- use case is an example of the system-to-be “in action”
  - “action” as seen from “outside the system looking in”, not the internal workings of the system
- use cases are text, not diagrams
- use cases describe how the users interact with the system
  - but written in “third person”
  - an observing watching the users use the system
  - not users POV

## Use Case Scenarios

- factorial number of paths through system
  - each pathway through a system called a scenario
  - can't enumerate (let alone describe) all of them
- choose typical paths through system as 'main success' and then think about the alternates
  - remember use case scenarios are always about users interacting with the system
- set of related scenarios called a use case

## Actors

- Properties of 'Actors'
  - Have behaviour ,
  - Are outside of the system
  - interacts with the system
- Example
  - person
  - computer system
  - organization



## Actors: Three Types

- Primary
  - Main users of system
  - For whom the system was built to “serve”
- Supporting
  - Provides a service to the system
  - Often another computer system external to the system under discussion (e.g. Credit Card information, PayPal, , etc.)
- Offstage
  - has interest in use case but not directly involved
  - e.g. government tax department

## Descriptions

- scripts
  - used in simple use cases
  - two types
    - Brief
      - One paragraph “script”
      - very high level
    - casual
      - Multi paragraph “script”
      - much more detail than a brief description
- flows
  - used in fully dressed used cases
  - A step by step breakdown of the action of the system



VS



## A Use Case

---

- Brief format

Process Sale: A customer arrives at a checkout with items to purchase. The cashier uses the POS system to record each purchased item. The system presents a running total and line-item details. The customer enters payment information, which the system validates and records. The system updates inventory. The customer receives a receipt from the system and then leaves with the items.

- Casual format the same except has *Main Success Scenario* along with *Alternate Scenarios*

## A Use Case Template

---

**Use Case:** <Use Case Name>

**Primary Actor:** <has goals/needs satisfied by the UC>

**Goal:** <high-level description of use case purpose>

**Stakeholders List:** <list of anyone affected by the use case along with their goals>

**Initiating Event:** <the event(s) that must have happened before Use Case can be executed >

**Main Success Scenario:**

**Post Conditions:** <list of conditions that must be true when the Scenario is complete >

**Alternate Flows or Exceptions:**

**Use Cases Utilized:** < list of other use cases used>

**Scenario Notes:** < description of supporting actors, concurrency of actions, and any additional information such as requirements related to the UC.>

## Description Flows

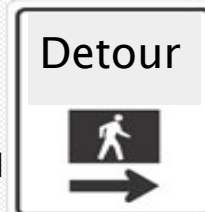
---

- basic flow
  - most typical scenario among the related scenarios of a use case:
  - also called the **main success scenario**
  - Should not have any branching
  - left to the extension section

## Extension Section or Alternate Flows

---

- A collection of exceptions and uncommon paths through the system
- Each alternate flow in section has a line label, title, and flow
  - line label should refer back to the line number(s) in the basic flow
    - letter is used after basic flow line number to uniquely identify the alternate flow
    - 3a, 3b, 2-4a, 7d, etc.
  - title describes condition that triggers alternate flow
  - flow itself indented and numbered (similar to the basic flow)



## Use Case Example: Fully Dressed

---

- **UC1:** Get paid for car accident
- **From Requirement:** R7
- **Importance:** Must
- **Stakeholders List:** Claimant, Insurance Company, Agent,
- **Primary Actor:** the claimant
- **Initiating Event:** Claimant has a car accident and wants to file an insurance claim
- **Basic Flow:**
  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

## Use Case Example: Fully Dressed

---

- **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 declines claim,
    - 2a2. Insurance Company notifies claimant,
    - 2a3. Insurance Company records all this,
    - 2a4. Insurance Company terminates proceedings.

## Use Case Example: Fully Dressed

---

- **Extensions:**

- 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 declines claim,

- 4a2. Insurance Company notifies claimant,

- 4a3. Insurance Company records all this,

- 4a4. Insurance Company terminates proceedings.

- 4b. Accident violates some minor policy guidelines:

- 4b1. Insurance Company begins negotiation with claimant as to degree of payment to be made.

## UC Components: Name and Reference Number

---

- All use cases must have names

- Should be:

- Short

- Be action based

- start with a verb

- don't use weasel words such as "handling"

- Should number use cases

- e.g. UC1, UC2, etc.

- commonly in order of creation or order of importance

- importance better, but harder if not automated

- important for use case references the in design documents - for traceability

## Description Flow Style

---

- Sentence Structure
  - <time or sequence factor> <actor> <action> <constraints>
  - Thus, typical sentences might look like:
    - Any time after dark, clerk gets quote and he may cancel the sale
    - At end of month, she sends a credit memo to all customers whose credit is larger than a certain value
- Each action should be numbered and start on a new line
  - keeps the narrative clear
  - improves traceability from requirements to design or test
  - allows specific line references needed in the Extensions section

## Description Flow Style

---

- Terse Good
  - do not use excess verbiage
  - delete “noise” words
- Focus on intent not user interface
  - yes: Administrator identifies self
  - no: Administrator enters ID and password in dialog box (see picture 3)
- Blackbox style
  - yes: The system records the sale
  - no: The system generates a SQL INSERT statement for the sale
- Actor and Actor-Goal perspective

## Use Case Calling Another Use Case

- used if a few use cases use the same scenario(s)
  - common scenario(s) become its own use case
  - called from the various use cases that it has been factored from
- underline called use case's name



## System Boundary & Primary Actors

- what is inside and outside the system depends on what system is under consideration



Customer

Checkout Service



Cashier

POS System

## Change Cases

---

- AKA Speculative Requirements
- Change cases tests
  - contrasting design alternatives
  - malleability of the design
    - how much would have to change in the design to add them
- Often drawn from the "won't" list of the must/should/might/won't categories

## How to find Use Cases

---

1. Choose the System Boundary
2. Find Primary Actors and Goals
3. Define Use Cases

## Find Actors and Goals

---

- Who starts and stops the system?
- Who does system administration?
- Who does user and security management?
- Does the system respond to time events? (Time might be an actor then)
- Does the system respond to events in another system?
- Who evaluates the system activity
- Is there a monitoring process that restarts the system?

## Tasks, Goals and Use Cases

---

- Actors have goals and will use your software as a tool to accomplish them.
- The achievement of the goals produces a result that the actor values
  - rather than ask 'what do you do' which gets you a description of what already exists
  - ask what are your goals and what are the measurable results
- Use case names often can be a verb associated with the goal

## Finding 'good' use cases

- big enough to form a useful picture of the purpose of an important part the system
- small enough that you suspect it can be implemented using half a dozen “objects”
- Boss Test (importance test)
  - logging in as use case?
  - tell boss “today I logged in to the system” as test
- Size test
  - Not a single action
  - should be complex enough so basic flow should have 5 to 15 steps

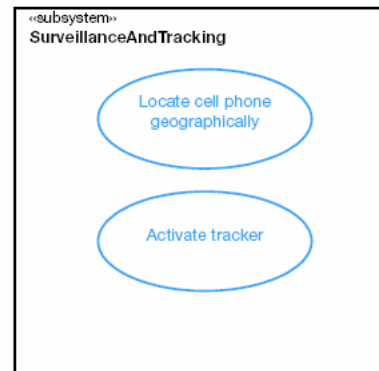


## Use Case Diagrams

- a summary of the use cases and actors in the system with relationships
- not even remotely complete
  - just a diagrammatic listing of use cases and actors
  - real work done in use cases themselves
  - can have use cases without use case diagram and be OK
- Use case diagram is not a flow diagram
  - not a flow chart, not a data flow
  - what can happen, not the order it happens in

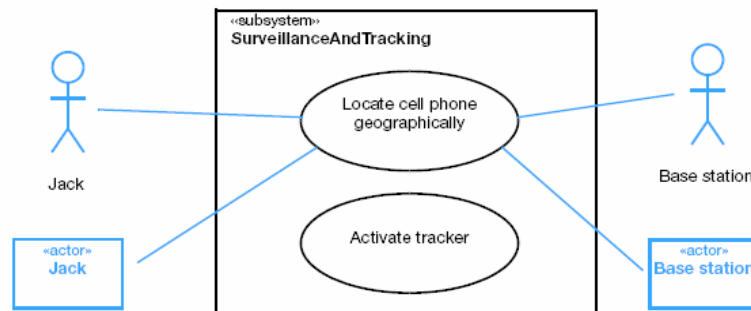
# Use Case Diagrams: The bits

**Figure C.115**  
Use case



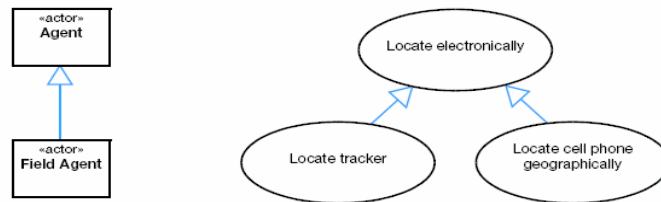
# Interactors

**Figure C.116**  
Interactors

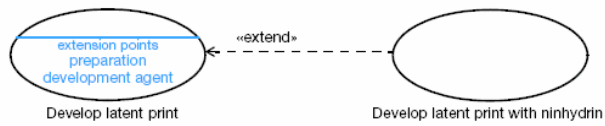


# Relationships

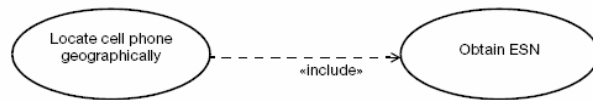
**Figure C.117**  
Generalization relationship



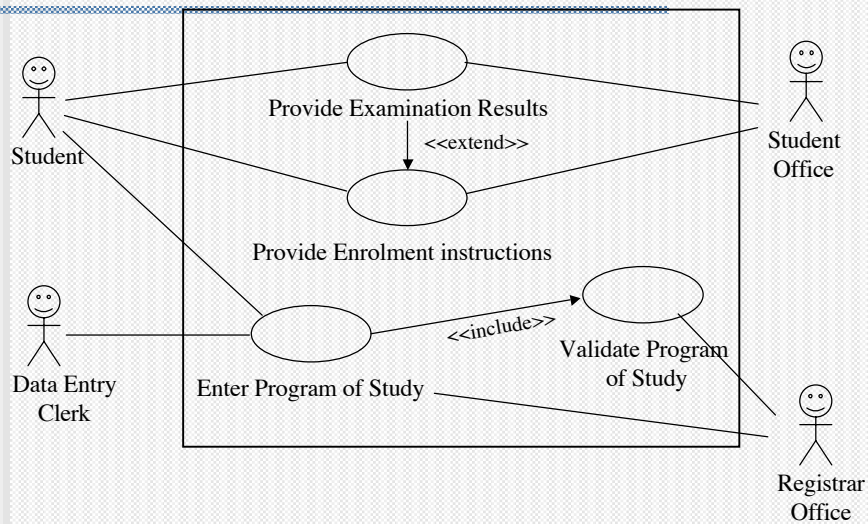
**Figure C.118**  
Extend relationship



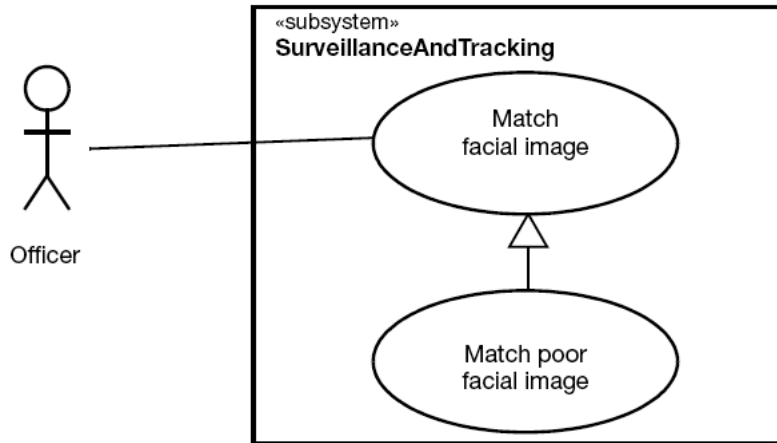
**Figure C.119**  
Include relationship



# Use Case Diagrams: Example



## Use Case Diagrams: Generalization



## Other Inputs to Analysis

- The Subject Matter
- Existing Systems
  - always check to see if the problem has been solved and how
- The Architectural Vision
  - from a system architecture viewpoint. Frequently the hardware choices have already been made, does that affect your analysis/design?
- Mock ups
  - can provoke fresh insights
  - should always be thrown away
  - make them obviously non-functional
- Analysis Patterns



## Checking Requirements

- Requirements do evolve, in spite of us- good to check that the initial requirements still apply
- Analysis (Subject Matter Model), Design (Object Type Model and Technical Model) can uncover missing or contradictory requirements
- Even if get it right, outside world may change during development
- Requirements coverage- is the system actually doing what is required of it... hard to do
  - Caveat: to trace all aspects of the system after completion (called requirements coverage) very very difficult and time consuming