Requirements Engineering

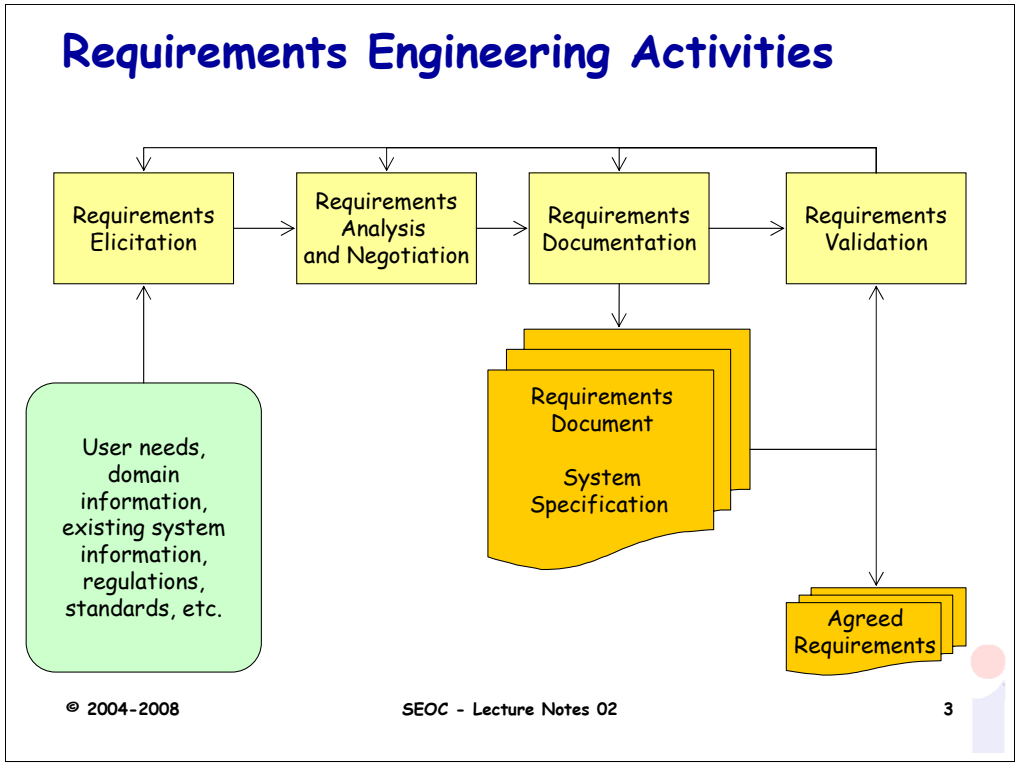
Massimo Felici

IF-3.46 0131 650 5899

mfelici@inf.ed.ac.uk

10 Top Reasons for Not Doing Requirements

- We don't need requirements, we're using objects, java, web...
- The users don't know what they want
- We already know what the users want
- Who cares what the users want?
- We don't have time to do requirements
- It's too hard to do requirements
- My boss frowns when I write requirements
- The problem is too complex to write requirements
- It's easier to change the system later than to do the requirements up front
- We have already started writing code, and we don't want to spoil it



Main activities involved in Software Requirements engineering:

- **Elicitation**: Identify sources; Elicit requirements
- Analysis and Negotiation: Classify requirements; Model; Top-level architecture; Allocate requirements to components; Negotiate requirements
- **Documentation**: Requirements Definition Doc; Software Requirements Specification; Document Standards; Document Quality
- **Validation**: Reviews; Prototypes; Modeling; Test definition
- **Management**: Traceability; Attributes; Change/Evolution

The pattern, sequence and interaction of these activities is orchestrated by a Requirements Engineering Process.

Readings

- I. Sommerville. Integrated Requirements Engineering: A Tutorial. IEEE Software, January/February 2005, pp. 16-23.

Suggested Readings

- I. Sommerville, P. Sawyer. Requirements Engineering: A Good Practice Guide. John Wiley & Sons, 1997.
- G. Kotonya, I. Sommerville. Requirements Engineering: Processes and techniques. John Wiley & Sons, 1998.
- I. Sommerville. Software Engineering, Eighth Edition, Addison-Wesley 2007.
  - Chapter 6 on Software Requirements.
  - Chapter 7 on Requirements Engineering Processes.

Requirements Elicitation Activities

- **Application domain understanding**
  - Application domain knowledge is knowledge of the general area where the system is applied
- **Problem understanding**
  - The details of the specific customer problem where the system will be applied must be understood
- **Business understanding**
  - You must understand how systems interact and contribute to overall business goals
- **Understanding the needs and constraints of system stakeholders**
  - You must understand, in detail, the specific needs of people who require system support in their work

Requirements Elicitation Techniques

- **Interviews with stakeholders**
  - Close/Open (Structured/Unstructured), Facilitated Meetings (e.g., professional group work)
- **Scenarios**
  - Elicit the "usual" flow of work
  - Are stories which explain how a system might be used
  - Expose possible system interactions and reveal system facilities which may be required
- **Prototypes**
  - mock-up using paper, diagrams or software
- **Observations**
  - Observing "real world" work

Some requirements elicitation techniques find grounds in **Ethnography** - a technique from the social sciences. *Note that actual work processes often differ from formal prescribed processes.*

Requirements Analysis

- Discovers **problems**, **incompleteness** and **inconsistencies** in the elicited requirements
- A problem checklist may be used to support analysis
- **A Problem Checklist**
  - Premature design
  - Combined requirements
  - Unnecessary requirements
  - Requirements ambiguity
  - Requirements realism
  - Requirements testability

Requirements Analysis involves: Classification, Conceptual Modeling, Architectural Design and Requirements Allocation and Requirements Negotiation. Requirements Analysis deals with large volume of requirements information, detects and resolves conflicts, scopes the system and defines interfaces with the environment, translates system requirements into software requirements and provides feedback to the stakeholders (in order to resolve conflicts through the negotiation process).

A Problem Checklist: Premature design, Combined requirements, Unnecessary requirements, Use of non-standard hardware, Conformance with business goals, Requirements ambiguity, Requirements realism, Requirements testability.

Non-functional Requirements

- **Non-functional requirements** (e.g., safety, security, usability, reliability, etc.) define the overall qualities or attributes of the resulting system
- **Constraints** on the product being developed and the development process
- **Warnings: unclear distinction between non-functional and functional requirements**

Readings

- J. Boegh, S. De Panfilis, B. Kitchenham, A. Pasquini. A Method for Software Quality Planning, Control, and Evaluation. IEEE Software, March/April 1999, pp. 69-77.

Other Activities

- **Constructing specifications**
  - System requirements definition: customer facing, at system level
  - Software Requirements Specification: developer facing, at software level
- **Requirements validation**
  - define the acceptance test with stakeholders
- **Requirements Management**
  - Manage requirements and maintain **traceability**
  - Requirements **change** because the environment changes and there is a need to **evolve**

There are four main types of traceability links with respect to their process relationships to requirements:

Forward from requirements. Responsibility for requirements achievement must be assigned to system components, such that accountability is established and the impact of requirements change can be evaluated.

Backward to requirements. Compliance of the system with requirements must be verified, and *gold-plating* (designs for which no requirements exist) must be avoided.

Forward to requirements. Changes in stakeholder needs, as well as in technical assumptions, may require a radical reassessment of requirements relevance.

Backward from requirements. The contribution structures underlying requirements are crucial in validating requirements, especially in highly political settings.

Suggested Readings

- M. Jarke. Requirements Tracing. Communications of the ACM, Vol. 41, No. 12, December 1998.

How to organize requirements?

- **Software Requirements Specification (SRS)**
  - The SRS document is a structured documents that containing the identified requirements
- The **VOLERE Template** identifies the following SRS main parts:
  - **PROJECT DRIVERS** (e.g., The Purpose of the Product, Stakeholders, etc.)
  - **PROJECT CONSTRAINTS** (e.g., Costs)
  - **FUNCTIONAL REQUIREMENTS**
  - **NON-FUNCTIONAL REQUIREMENTS** (e.g., Usability, Performance, Operational, Maintainability, Portability, Safety, Reliability, Security, Cultural, etc.)
  - **PROJECT ISSUES** (e.g., Open Issues, Risks, Evolution, etc.)

You may want to use the VOLERE template (tailored for your purposes) as support for your practical work. The VOLERE requirements shell provides a guide for writing requirements.

Readings

•J. Robertson, S. Robertson. VOLERE: Requirements Specification Template. Edition 10.1, Atlantic Systems Guild.

Suggested Readings

•S. Robertson, J. Robertson. Mastering the Requirements Process. Addison-Wesley, 1999.

Requirements Engineering Practices

Examples of Requirements Engineering practices are:

- **Define a standard document structure**
  - For example, tailor a standard requirements specification template to your needs
- **Identify requirements uniquely**
  - For example, number each requirements specified in the requirements documentation

Suggested Readings

•I. Sommerville, P. Sawyer. Requirements Engineering: A Good Practice Guide. John Wiley & Sons, 1997.

Readings

- **Requirements Specification Template**
  - J. Robertson, S. Robertson. VOLERE: Requirements Specification Template. Edition 10.1, Atlantic Systems Guild.
- I. Sommerville. Integrated Requirements Engineering: A Tutorial. IEEE Software, January/February 2005, pp. 16-23.
- J. Boegh, S. De Panfilis, B. Kitchenham, A. Pasquini. A Method for Software Quality Planning, Control, and Evaluation. IEEE Software, March/April 1999, pp. 69-77.

Suggested Readings

- I. Sommerville, P. Sawyer. Requirements Engineering: A Good Practice Guide. John Wiley & Sons, 1997.
- G. Kotonya, I. Sommerville. Requirements Engineering: Processes and techniques. John Wiley & Sons, 1998.
- M. Jarke. Requirements Tracing. Communications of the ACM, Vol. 41, No. 12, December 1998.
- S. Robertson, J. Robertson. Mastering the Requirements Process. Addison-Wesley, 1999.
- I. Sommerville. Software Engineering, Eighth Edition, Addison-Wesley 2007.
  - Chapter 6 on Software Requirements
  - Chapter 7 on Requirements Engineering Processes

Summary

- **Requirements engineering**
  - Involves diverse activities
  - Supports the construction of quality systems
- **Issues** are very wide ranging
  - Poor requirements lead to very poor systems
  - Negotiating agreement between all the stakeholders is hard
- In some application areas it may be possible to use a more formal notation to capture some aspects of the system (e.g., control systems, compilers, …)

VolBank - Volunteer Bank

VolBank: Requirements

1. To develop a system that will handle the registration of volunteers and the depositing of their time.
2. To handle recording of opportunities for voluntary activity.
3. To match volunteers with people or organizations that need their skills.
4. To generate reports and statistics on volunteers, opportunities an time deposited.

For each volunteer, the System has:

- To record the details of volunteers, contact details, skills and needs
- To record the time that each volunteer deposits in the system
- To transfer from the web-server details of volunteers and the time they are depositing.

For each organization, the system has:

1. To record details of member voluntary organizations
2. To record the needs of voluntary organizations
3. To record the needs of individuals (including volunteers) for help.

The system has:

1. To match volunteer with local opportunities
2. To match local opportunity with a team of volunteers
3. To record matches between volunteers and opportunities
4. To notify volunteers of a match
5. To notify organizations of a match
6. To Record if agreement is reached from a particular match.

VolBank: Elicitation

- **Goals** (why the system is being developed)
  - An high level goal is to increase the amount of volunteer effort utilized by needy individuals and organizations
  - Possible requirements in measurement and monitoring
- **Domain Knowledge**
  - Some specific requirements, e.g., Safety and Security
- **Stakeholders**
  - volunteers, organizations, system administrators, needy people, operator, maintenance, manager
- **Operational Environment**
  - Probably constrained by software and hardware in the office
- **Organizational Environment**
  - legal issues of keeping personal data, safety issues in "matching"

Elicitation aims to identify (potential sources of) requirements.

VolBank: Examples of requirements

- **Volunteer** identifies:
  1. The need for security/assurance in contacting organizations, …
- **Management** identifies:
  1. The number of hours volunteered per month above a given baseline as the key metric
- **Operator** identifies:
  1. The need to change details when people move home
  2. The need to manage disputes when a volunteer is unreliable, or does bad work

VolBank: Analysis and Classification

- **Functional Requirements**
  - The system allows a volunteer to be added to the register of volunteers. The following data will be recorded:...
- **Non-functional Requirements**
  - The system ensures confidentiality of personal data and will not release it to a third party
  - The system ensures the safety of all participants

VolBank: A Failed Match Scenario

- **Goal**: to handle failure of a match
- **Context**: the volunteer and organization have been matched and a date for a preliminary meeting established
- **Resources**: time for volunteer and organization
- **Actors**: volunteer, operator, organization
- **Episodes**:
  - The volunteer arrives sees the job to be done and decides (s)he cannot do it
  - Organization contacts operator to cancel the match and reorganize
- **Exceptions**: volunteer fails to show up

VolBank: Conceptual Modeling

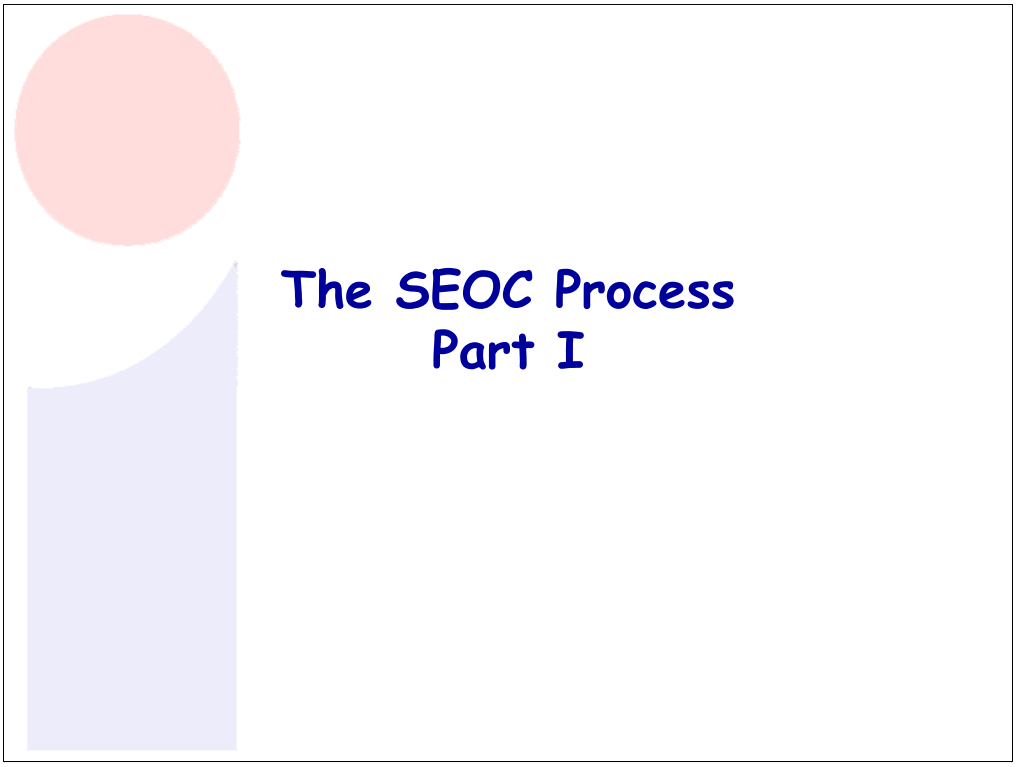
- Process of requirements engineering is usually guided by a requirements method
- Requirement methods are systematic ways of producing system models
- System models important bridges between the analysis and the design process
- Begin to identify classes of object and their associations:
  - volunteer, contact details, match, skills, organization, needs, etc.
- Start to consider some high level model of the overall workflow for the process using modeling tools

VolBank: Design and Allocation

- How do we allocate requirements?
  - The system shall ensure the safety of all participants?
- Further analysis to identify principal threats:
  - Safety of the volunteer from hazards at the work site
  - Safety of the organizations from hazards of poor or inadequate work
  - Safety of people from volunteers with behavioural problems
  - …
- Design might allow us to allocate:
  - 1 to an information sheet
  - 2 to a rating component and procedures on allocating work
  - 3 to external police register
  - …

VolBank: Negotiation

- **Safety** and **Privacy** requirements
  - may be **inconsistent** or **conflicting**
  - need to modify one or both
  - **Privacy**: only authorized releases for safety checks will be permitted and there is a procedure for feeding back to the individual if a check fails.
- Some requirements may be achievable but only at great **effort**
  - Attempt to downscale
  - **Prioritize**
  - It may be too much effort to implement a fault reporting system in the first release of the system

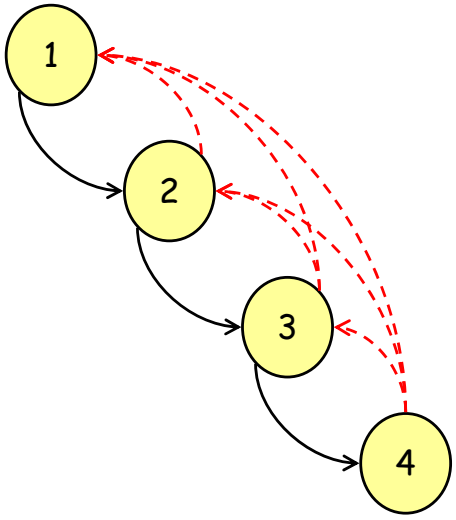

The SEOC Process
Part I

The SEOC process – Part I

1. Gathering Requirements
   - Writing a Requirements Specification Document (e.g., see the VOLERE template)
2. Capturing functional requirements into Use Cases
   - Describe use cases by a Use Case Template
3. Modelling a preliminary system design into Class Diagrams
4. Validating your design by CRC cards

The SEOC process – Part I

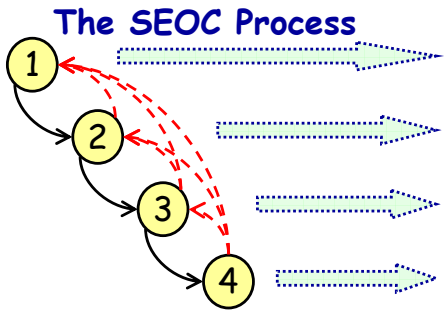
1. Requirements
2. Use Cases
3. Class Diagrams
4. Validation



The SEOC Project Deliverable 1

1. Requirements Specification
2. Use Case Model
3. Class Model
4. Validation of Class Model
5. Deliverable Assessment

SEOC Activity Deliverables



Deliverable 1

- Requirements Specification
- Use Case Model
- Class Model
- Validation of Class Model
- Deliverable Assessment

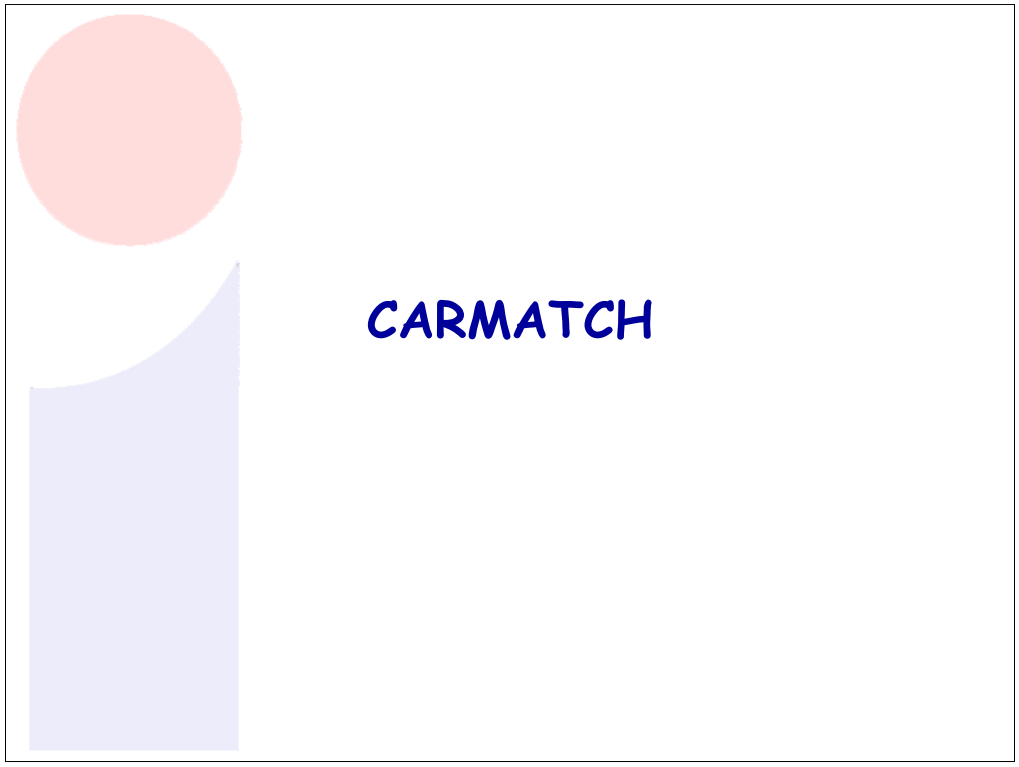
Deliverable 1 Assessment

Deliverable 1

- Requirements Specification
- Use Case Model
- Class Model
- Validation of Class Model

Part 3 - Deliverable Marking Scheme

Deliverable Marking Scheme		
Deliverable Part	**Questions**	**Marks**
Requirements	**Q1.** Did you organise/collect the system requirements by using a Requirements Specification template (e.g., Volere)? Assess the quality of your Software Requirements Specification (SRS) document.	[/ 5]
Marks Limit: [20/100]	**Q2.** Did you distinguish different types of requirements (e.g., functional or non-functional)? Assess how your SRS identifies different types of requirements.	[/ 5]
	Q3. Do you believe you got most of the system requirements (requirements completeness)? Assess the extent to which you have elicited and gathered requirements from the main sources.	[/ 5]
	Q4. Have you identified/resolved conflicting requirements (requirements correctness)? Assess the extent to which you have resolved conflicting requirements among different types (e.g., functional and non-functional) or across teams.	[/ 5]
Use Cases	**Q5.** Did you graphically represent the functional requirements by Use Cases? Assess to which extent your use case diagram captures main system functionalities and actors.	[/ 10]
Marks Limit: [30/100]	**Q6.** Did you refine the use cases by generalization, include or extend relationships? Assess to which extent you have refined and structured use cases.	[/ 10]
	Q7. Did you use a template for describing use cases? Assess to which extent you have clarified and described use case information (completeness and correctness).	[/ 10]
Class Diagrams	**Q8.** Does your class diagram identify the main classes of the system? Assess to which extent your class diagram realizes system use cases.	[/ 10]
Marks Limit: [30/100]	**Q9.** Did you specify Attributes and Operations for each class? Assess the completeness of class specification.	[/ 10]
	Q10. Did you identify relationships (i.e., Dependency, Association, Aggregation, Composition and Inheritance or Generalization) between classes? Assess the object orientation quality of your class diagram.	[/ 10]
CRC Cards	**Q11.** Did you construct CRC cards for your system design? Assess the completeness and correctness of CRC cards.	[/ 10]
Marks Limit: [20/100]	**Q12.** Did you verify your Class Diagrams? Did you play any use case with the CRC Cards in order to verify your class diagram? Assess the quality and the coverage of your requirements and design verification by CRC cards.	[/ 10]
Deliverable Mark		[/100]


CARMATCH

CARMATCH Background

- CARMATCH is a franchising company that is being set up to promote car sharing
- Organizational goal: reduce carbon emissions
- CARMATCH seeks to promote car sharing
  - Matching potential car sharers
- CARMATCH consists of a three layer structure: (non-for-profit trust) global operation; national central operating company; local franchises
- In some countries, it offers insurances
- Main Profits: membership fees, consultancies, insurance commissions
- CARMATCH needs (has the requirements for) a computer system that can be used by its franchisees

CARMATCH Requirements

1. To develop a system that will hold information about members of the CARMATCH scheme
2. To match members up with other members as car sharers
3. To record insurance sales
4. To record details of potential and actual consultancy in the area of operation
5. The system must be capable of future expansion to incorporate information about toll and road-pricing and equipment sold to and installed for members

CARMATCH Requirements Specification

The ..... System
Requirements Specification
Version ...

Table of Contents

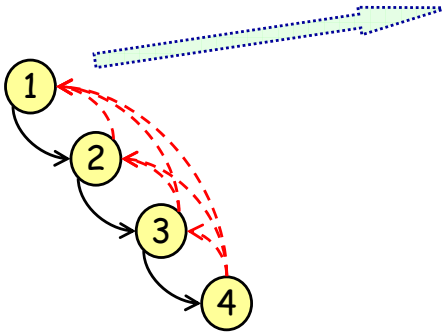
Specification prepared by ..... Date .....

- **Project Drivers**
  - CARMATCH background
  - CARMATCH organization, Local governments, EU?, Locan franchises, car sharers, etc.
  - System Users?
- **Project Constraints**
  - Budget, Deadlines, Laws, etc.?
- **Functional Requirements**

1. To hold information about members
   1. 1
   2. 2
   3. 3
2. To match car sharers
3. To record insurance sales
4. To record details consultancies

- **Non-functional Requirements**
- **Project Issues**

Requirements Specification



CARMATCH Requirements Specification

The ____ System
Requirements Specification
Version ...

Table of Contents
PROJECT DRIVERS
1. The Purpose of the Project
2. The Stakeholders and other Stakeholders
3. Users of the Product
PROJECT CONSTRAINTS
4. Mandated Constraints
5. Naming Conventions and Definitions
6. Relevant Facts and Assumptions
FUNCTIONAL REQUIREMENTS
7. The Scope of the Work
8. The Scope of the Product
9. Functional and Data Requirements
NON-FUNCTIONAL REQUIREMENTS
10. Look and Feel Requirements
11. Usability and Humanity Requirements
12. Performance Requirements
13. Operational Requirements
14. Maintainability and Support Requirements
15. Security Requirements
16. Cultural and Political Requirements
17. Legal Requirements
PROJECT ISSUES
18. Open Issues
19. Off-the-Shelf Solutions
20. New Problems
21. Tasks
22. Cutover
23. Risks
24. Costs
25. User Documentation and Training
26. Waiting Room
27. Ideas for Solutions

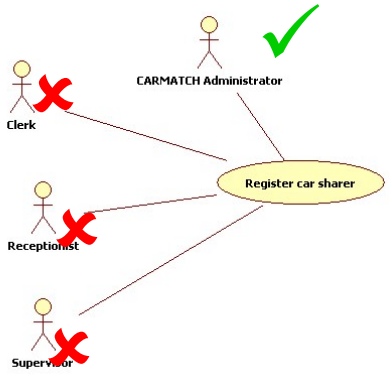
Specification prepared by: ____ Date: ____

- **Project Drivers**
  - CARMATCH background
  - CARMATCH organization, Local governments, EU, Lacan franchises, car sharers, etc.
  - System Users?
- **Project Constraints**
  - Budget, Deadlines, Laws, etc.?
- **Functional Requirements**

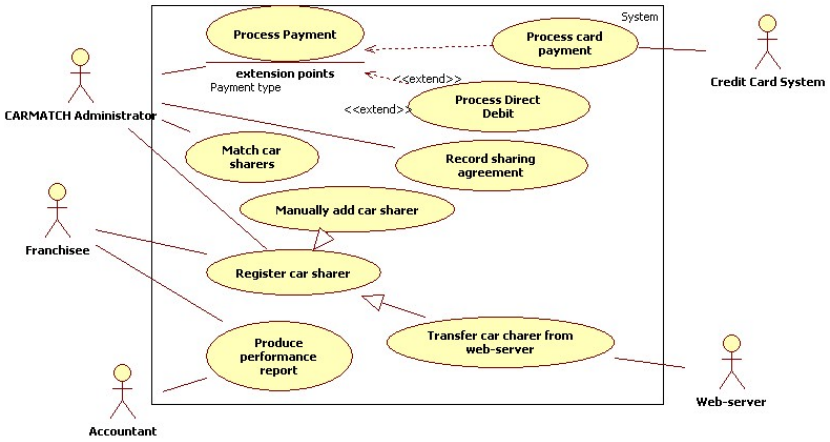
1. To hold information about members
   1. 1
   2. 2
   3. 3
2. To match car sharers
3. To record insurance sales
4. To record details consultancies

- **Non-functional Requirements**
- **Project Issues**

CARMATCH Actors and Use Cases



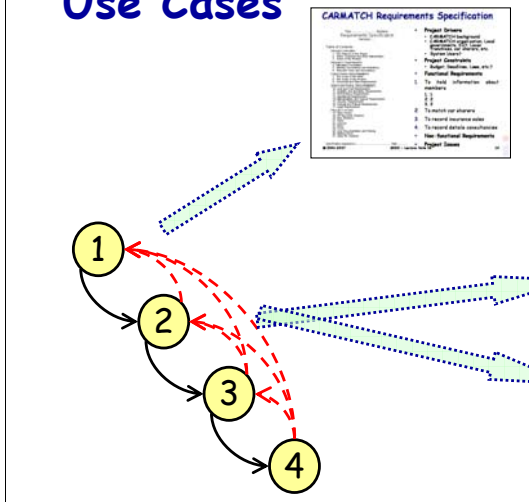
CARMATCH System Use cases

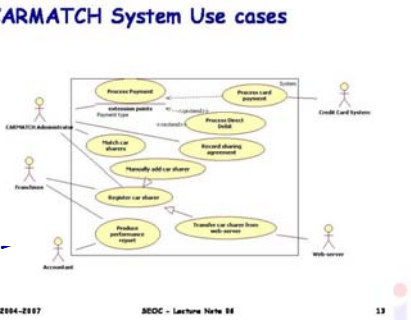


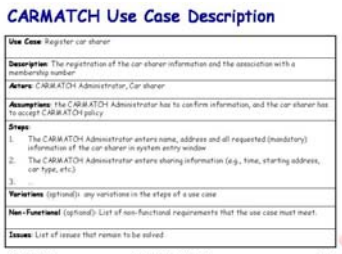
CARMATCH Use Case Description

Use Case: Register car sharer
Description: The registration of the car sharer information and the association with a membership number
Actors: CARMATCH Administrator, Car sharer
Assumptions: the CARMATCH Administrator has to confirm information, and the car sharer has to accept CARMATCH policy
Steps: 1. The CARMATCH Administrator enters name, address and all requested (mandatory) information of the car sharer in system entry window 2. The CARMATCH Administrator enters sharing information (e.g., time, starting address, car type, etc.) 3. …
Variations (optional): any variations in the steps of a use case
Non-Functional (optional): List of non-functional requirements that the use case must meet.
Issues: List of issues that remain to be solved

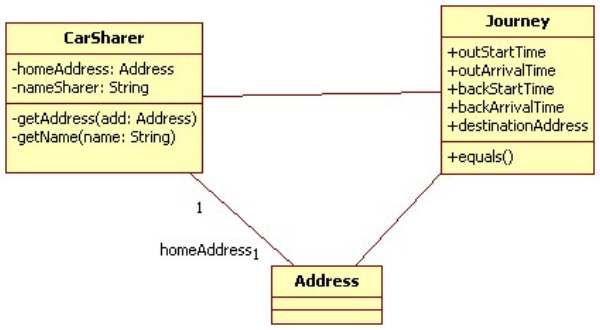
Use Cases


CARMATCH Requirements Specification
1
2
3
4

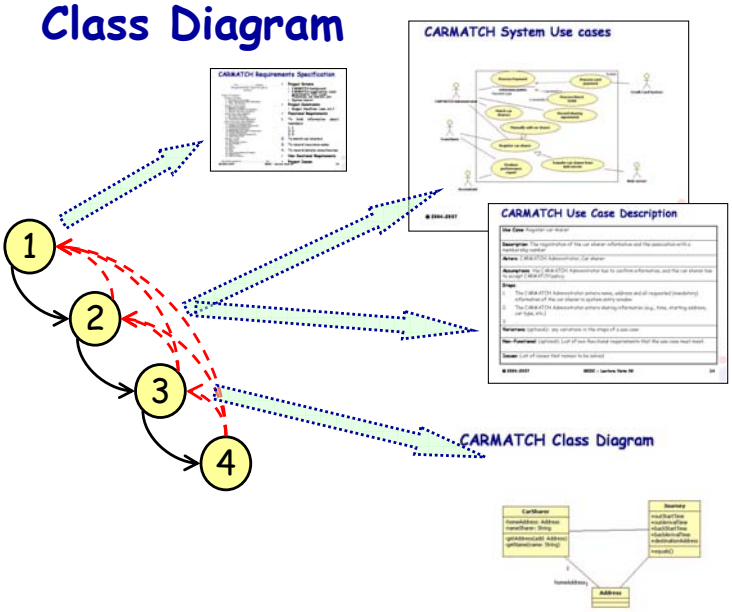

CARMATCH System Use cases
Process Payment
Process card payment
Credit Card System
CARMATCH Administrator
Process Direct Debit
Match car sharers
Record sharing agreement
Manually add car sharer
Franchisee
Register car sharer
Produce performance report
Transfer car sharer from web-server
Web-server
Accountant
© 2004-2007
SEOC - Lecture Note 08
13


CARMATCH Use Case Description
Use Case: Register car sharer
Description: The registration of the car sharer information and the association with a membership number
Actors: CARMATCH Administrator, Car sharer
Assumptions: the CARMATCH Administrator has to confirm information, and the car sharer has to accept CARMATCH policy
Steps:
1. The CARMATCH Administrator enters name, address and all requested (mandatory) information of the car sharer in system entry window
2. The CARMATCH Administrator enters sharing information (e.g., time, starting address, car type, etc.)
3.
Variations (optional): any variations in the steps of a use case
Non-Functional (optional): List of non-functional requirements that the use case must meet.
Issues: List of issues that remain to be solved
© 2004-2007
SEOC - Lecture Note 08
14

CARMATCH Class Diagram



Class Diagram



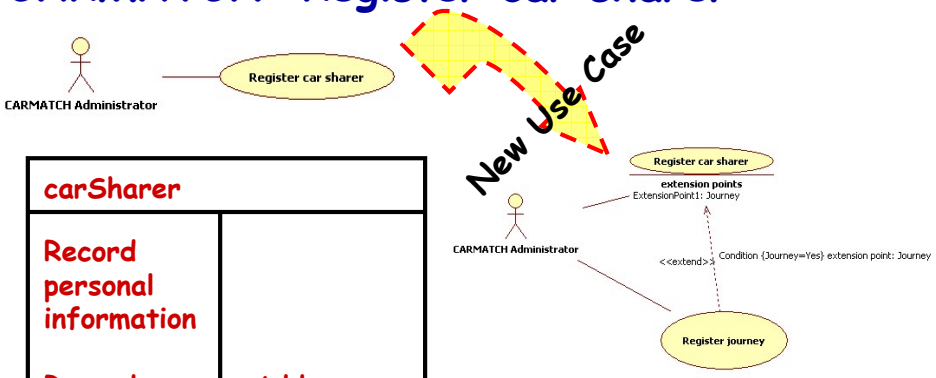
CARMATCH Validation

carSharer	
Record personal information	
Record Address	Address
Record Journeys	Journey

Journey	
Record journey Times	
Record journey Addresses	Address
Find journey matches	

Address	
Record Address	

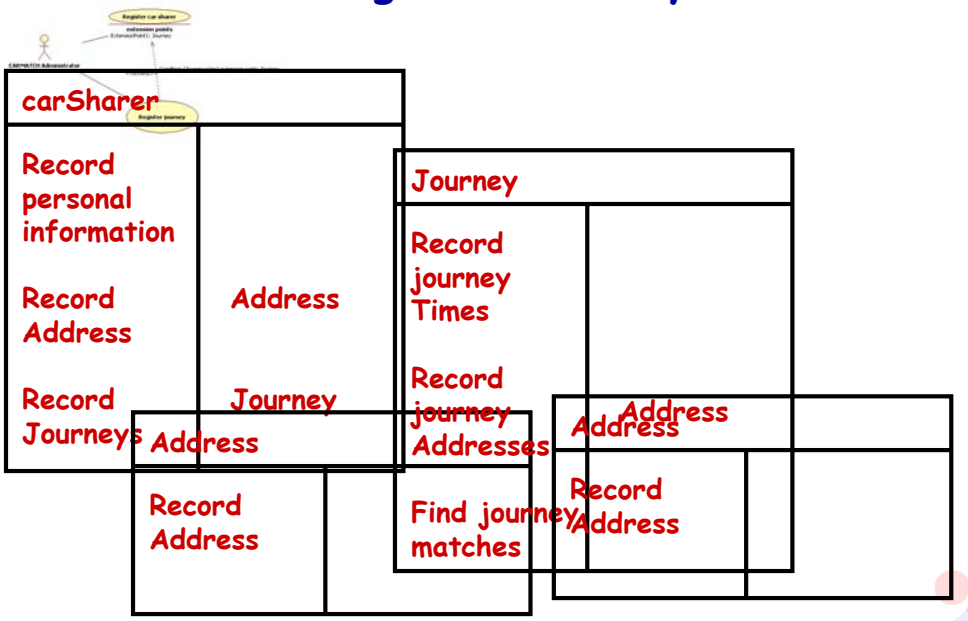
CARMATCH: Register car sharer



carSharer	
Record personal information	
Record Address	Address
Record Journeys	Journey

Address	
Record Address	

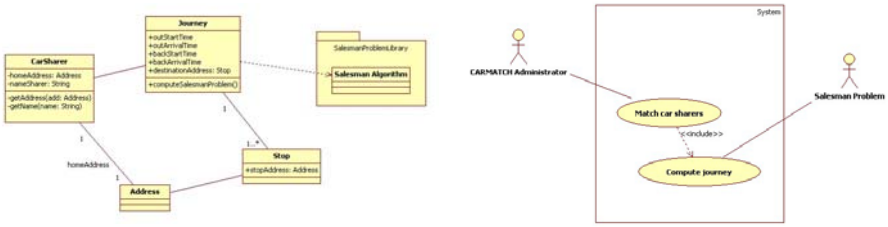
CARMATCH: Register Journey



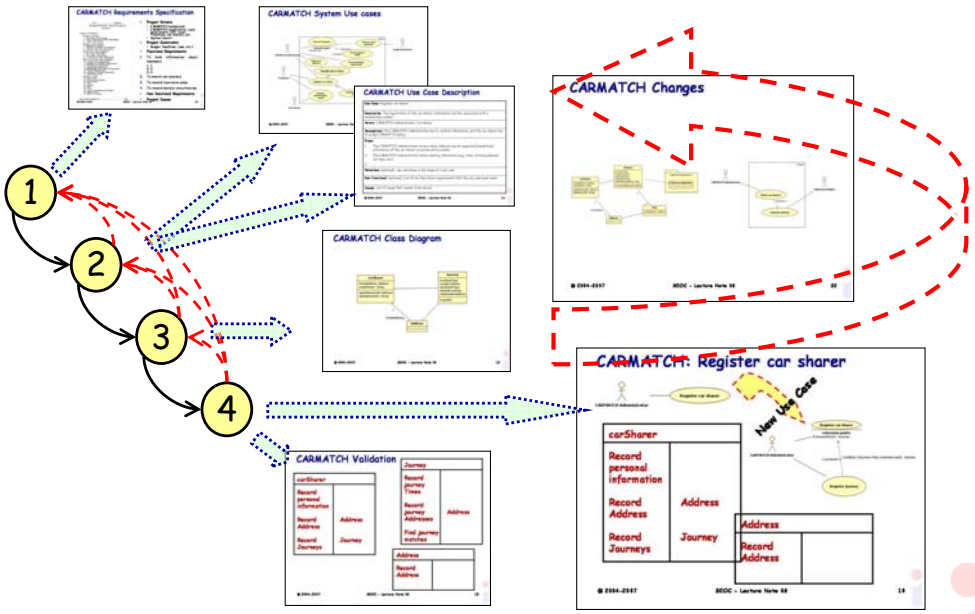
CARMATCH new requirements

- Efficiency: maximize the combination of journeys by combining multiple stops (i.e., journeys)
- Note that it is a non-functional requirements
- Are there any implications? How does it affect your preliminary design?

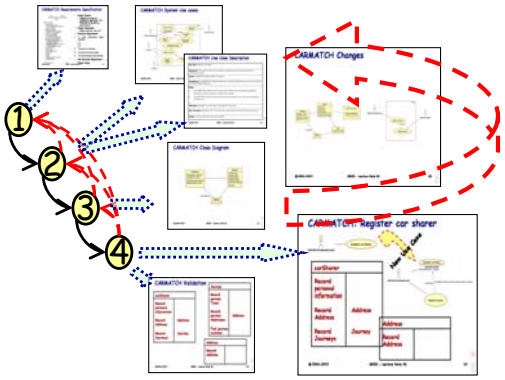
CARMATCH Changes



CARMATCH validation



Assessment



Part 3 - Deliverable Marking Scheme

Deliverable Marking Scheme		
Deliverable Part	**Questions**	**Marks**
Requirements	**Q1.** Did you organise/collect the system requirements by using a Requirements Specification template (e.g., Volere)? Assess the quality of your Software Requirements Specification (SRS) document.	[/ 5]
Marks Limit: [20/100]	**Q2.** Did you distinguish different types of requirements (e.g., functional or non-functional)? Assess how your SRS identifies different types of requirements.	[/ 5]
	Q3. Do you believe you got most of the system requirements (requirements completeness)? Assess the extent to which you have elicited and gathered requirements from the main sources.	[/ 5]
	Q4. Have you identified/resolved conflicting requirements (requirements correctness)? Assess the extent to which you have resolved conflicting requirements among different types (e.g., functional and non-functional) or across teams.	[/ 5]
Use Cases	**Q5.** Did you graphically represent the functional requirements by Use Cases? Assess to which extent your use case diagram captures main system functionalities and actors.	[/ 10]
Marks Limit: [30/100]	**Q6.** Did you refine the use cases by generalization, include or extend relationships? Assess to which extent you have refined and structured use cases.	[/ 10]
	Q7. Did you use a template for describing use cases? Assess to which extent you have clarified and described use case information (completeness and correctness).	[/ 10]
Class Diagrams	**Q8.** Does your class diagram identify the main classes of the system? Assess to which extent your class diagram realizes system use cases.	[/ 10]
Marks Limit: [30/100]	**Q9.** Did you specify Attributes and Operations for each class? Assess the completeness of class specification.	[/ 10]
	Q10. Did you identify relationships (i.e., Dependency, Association, Aggregation, Composition and Inheritance or Generalization) between classes? Assess the object orientation quality of your class diagram.	[/ 10]
CRC Cards	**Q11.** Did you construct CRC cards for your system design? Assess the completeness and correctness of CRC cards.	[/ 10]
Marks Limit: [20/100]	**Q12.** Did you verify your Class Diagrams? Did you play any use case with the CRC Cards in order to verify your class diagram? Assess the quality and the coverage of your requirements and design verification by CRC cards.	[/ 10]
Deliverable Mark		[/100]