LOYOLA UNIVERSITY CHICAGO

COMP 330
Software Engineering

Professor Honig

What is a requirement?

- **It may range from a high-level abstract statement of a service or of a system constraint to a detailed mathematical functional specification**
- **This is inevitable as requirements may serve a dual function**
  - May be the basis for a bid for a contract - therefore must be open to interpretation
  - May be the basis for the contract itself - therefore must be defined in detail
  - Both these statements may be called requirements

Capturing the requirements

- **Requirement: a feature of the system or a description of something the system is capable of doing in order to fulfill the system's purpose**
- **Three kinds of requirements:**
  - those that absolutely must be met
  - those that are highly desirable but not necessary
  - those that are possible but could be eliminated

Requirements documents

- **Requirements definition: complete listing of what the customer expects the system to do**
- **Requirements specification: restates the definition in technical terms so that the designer can start on the design**
- **Configuration management: supports direct correspondence between the two documents**

Requirements documentation

- **Requirements definition document: what the customer wants**
  - general purpose
  - background and objectives of system
  - description of customer-suggested approach
  - detailed characteristics
  - operational environment
- **Requirements specification document: what the designers need to know**

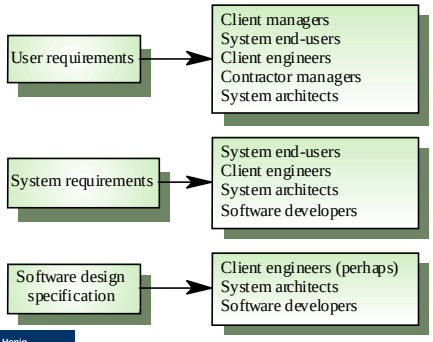
Configuration management

- **Set of procedures that track**
  - requirements that define what the system should do
  - design modules that are generated from requirements
  - program code that implements the design
  - tests that verify the functionality of the system
  - documents that describe the system

Dr. Honig's Requirements Rules

- **The minimum you need for acceptable requirements**
  1. A numbered set of requirements
  2. Each individual requirement small and specific
  3. Identify Functional and Non Functional Requirements
  4. Identify Mandatory and Optional Requirements

Requirements readers



Traditional Mismatch…a problem

How developers see users	*How users see developers*
Users don't know what they want.	Developers don't understand operational needs.
Users can't articulate what they want.	Developers place too much emphasis on technicalities.
Users have too many needs that are politically motivated.	Developers try to tell us how to do our jobs.
Users want everything right now.	Developers can't translate clearly-stated needs into a successful system.
Users can't prioritize needs.	Developers say no all the time.
Users refuse to take responsibility for the system.	Developers are always over budget.
Users are unable to provide a usable statement of needs.	Developers are always late.
Users are not committed to system development projects.	Developers ask users for time and effort, even to the detriment of the users' important primary duties.
Users are unwilling to compromise.	Developers set unrealistic standards for requirements definition.
Users can't remain on schedule.	Developers are unable to respond quickly to legitimately changing needs.

Functional vs. non-functional requirements

- **Functional: describes an interaction between the system and its environment**
- **Examples:**
  - System shall communicate with external system X.
  - What conditions must be met for a message to be sent
- **Non-functional: describes a restriction or constraint that limits our choices for constructing a solution**
- **Examples:**
  - Paychecks distributed no more than 4 hours after initial data are read.
  - System limits access to senior managers.

Types of requirements

- **Physical environment**
- **Interfaces**
- **Users and human factors**
- **Functionality**
- **Documentation**
- **Data**
- **Resources**
- **Security**
- **Quality assurance**

Characteristics of requirements

- **Are they correct?**
- **Are they consistent?**
- **Are they complete?**
- **Are they realistic?**
- **Does each describe something the customer needs?**
- **Are they verifiable?**
- **Are they traceable?**

Problems with NL specification

- **Ambiguity**
  - The readers and writers of the requirement must interpret the same words in the same way. NL is naturally ambiguous so this is very difficult
- **Over-flexibility**
  - The same thing may be said in a number of different ways in the specification
- **Lack of modularisation**
  - NL structures are inadequate to structure system requirements

Static descriptions of requirements

- **Indirect reference**
  - Example: k equations in n unknowns
- **Recurrence relations**
  - Example: F(0)=1; F(1)=1; F(n+1)=F(n)+F(n-1)
- **Axiomatic definition**
- **Expression as a language**
  - Example: Backus-Naur form


```
<condition>          ::=   <bool-term> | <bool-term> or <condition>
<bool-term>          ::=   <bool-factor> | <bool-factor> and <bool-
      term>
<bool-factor>        ::=   <expr> <relop> <expr> | (<condition>)
<relop>              ::=   < | ≤ | = | ≥ | > | < >
<expr>               ::=   <term> | <expr> <addop> <term> | <addop>
      <expr>
<term>               ::=   <factor> | <term> <mpyop> <factor>
<factor>             ::=   <scaled-expr> | <primary>
<scaled-expr>        ::=   (<expr>) <scale> | <number> <scale>
<primary>            ::=   (<expr>) <regname> | <number> | <func>
      (<expr>)
<number>             ::=   <integer> | <integer>. | .<integer> |
      <integer>.<integer>
<regname>            ::=   $ <regchar> | <regname> <regchar>
<integer>            ::=   <digit> | <digit> <integer>
<regchar>            ::=   <digit> | <letter> | <underscore>
<addop>              ::=   + | -
<digit>              ::=   0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9
<func>               ::=   abs | trunc
<letter>             ::=   A | a | B | b | C | c | D | d | E | e |
      . . . | Y | y | Z | z
<mpyop>              ::=   * | / | mod
<scale>              ::=   c | d | h | i | l | P | p | q | t | v
<underscore>         ::=   _ (ASCII character 95)
```

Data abstraction

Semester record
- *Semester type*
- Semester date
- Grade-point average
- Completed hours

Semester type
- (Fall, Spring, Summer)

Address information
- Telephone number
- Street address
- City
- State
- Postal code

Student record
- Name
- Student number
- *Address information*
- Number of semesters
- {Semester record}

Dynamic descriptions

- **Decision tables**

Table 4.1. Decision table.

	Rule 1	*Rule 2*	*Rule 3*	*Rule 4*	*Rule 5*
High standardized exam scores	T	F	F	F	F
High grades	-	T	F	F	F
Outside activities	-	-	T	F	F
Good recommendations	-	-	-	T	F
Send rejection letter			X	X	X
Send admission forms	X	X			

Dynamic descriptions (2)

- **Functional descriptions and transition diagrams**

$$f(Si, Cj) = Sk$$

Table 4.2. Transition table.

Current state	*Input*	*Next state*
S_1	0	S_2
S_1	1	S_1
S_2	0	S_2
S_2	1	S_1
S_3	0	S_1
S_3	1	S_3

Object-oriented specifications

- Each entity in the system is an object.
- A method or operation is an action that can be performed directly by the object or can happen to the object.
- Encapsulation: the methods form a protective boundary around an object.
- Class hierarchies of objects encourage inheritance.
- Polymorphism: same method for different objects, each with different behavior

Guidelines for writing requirements

- Invent a standard format and use it for all requirements
- Use language in a consistent way. Use shall for mandatory requirements, should for desirable requirements
- Use text highlighting to identify key parts of the requirement
- Avoid the use of computer jargon

Requirements document structure

- Introduction
- Glossary
- User requirements definition
- System architecture
- System requirements specification
- System models
- System evolution
- Appendices
- Index

Key points

- Requirements set out what the system should do and define constraints on its operation and implementation
- Functional requirements set out services the system should provide
- Non-functional requirements constrain the system being developed or the development process
- User requirements are high-level statements of what the system should do

Key points

- User requirements should be written in natural language, tables and diagrams
- System requirements are intended to communicate the functions that the system should provide
- System requirements may be written in structured natural language, a PDL or in a formal language
- A software requirements document is an agreed statement of the system requirements

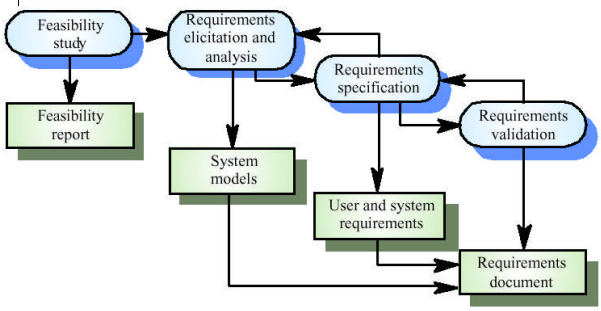
Requirements Engineering Processes

Processes used to discover, analyse and validate system requirements

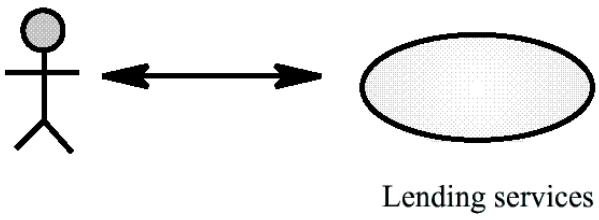
Objectives

- To describe the principal requirements engineering activities
- To introduce techniques for requirements elicitation and analysis
- To describe requirements validation
- To discuss the role of requirements management in support of other requirements engineering processes

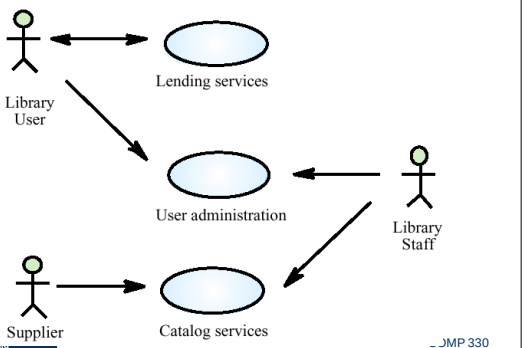
The requirements engineering process



Lending use-case



Library use-cases



Catalogue management