Software Engineering COMP 201

Lecturer: **Sebastian Coope**

Ashton Building, Room G.18

E-mail: ***coopes@liverpool.ac.uk***

COMP 201 web-page:

http://www.csc.liv.ac.uk/~coopes/comp201

Lecture 5 – Software Requirements

Software Requirements

Descriptions and specifications of a system

Last time:

- Introduce the concepts of **user and system requirements**
- Described **functional / non-functional requirements**

This lecture:

- To explain **two techniques** for describing system requirements
- To explain **how software requirements may be organised** in a requirements document

System Requirements

System Requirements are more detailed specifications of user requirements:

- They serve as a basis for designing the system;
- may be used as part of the system contract;
- may be expressed using **system models** (will be discussed more later).

Requirements and Design

- In principle, requirements should state ***what*** the system should do and the design should describe ***how*** it does this
- In practice, requirements and design are inseparable
  - A system architecture may be designed to structure the requirements
  - The system may inter-operate with other systems that generate design requirements
  - The use of a specific design may be a domain requirement

User Requirements → System Requirements→ Software requirements→ Software Design

- The system will have a secure login (User requirement)
- The system must provide a secure mechanism to allow a user to deal with the issue of forgotten or compromised passwords (User requirements)
- The system will have a secure login which complies with NIST Special Publication 800-63-1 authentication guidelines (System Requirement)
- The system must use only allow 3 attempts and login before locking the user out (Software Requirement)
- There is a class called Login, which will generically handle the login attempt regardless of the underlying authentication mechanism?

Problems with Natural Language Specification

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

An Example

- Imagine the following example of an informal specification from a critical system [1] :
  - "The message must be triplicated. The three copies must be forwarded through three different physical channels. The receiver accepts the message on the basis of a two-out-of-three voting policy."
- Questions: Can you identify any ambiguities in this specification?
- We will later see some other ways with documenting this example by a Petri net (see Lecture 9).

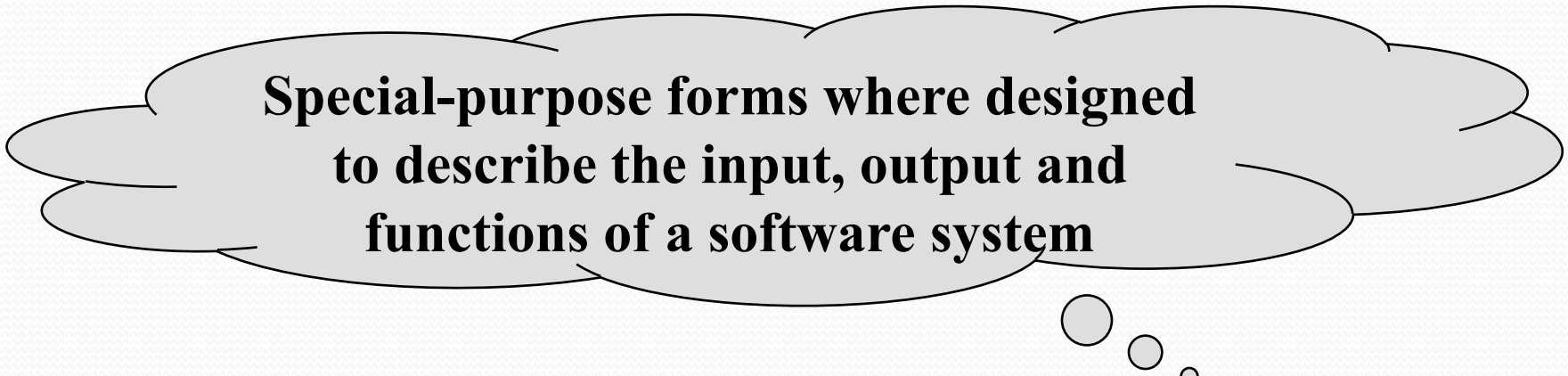
[1] - C. Ghezzi, M. Jazayeri, D. Mandrioli, "Fundamentals of Software Engineering", Prentice Hall, Second Edition, page 196 - 198

Alternatives to NL Specification

Notation	Description
Structured natural language	This approach depends on defining standard forms or templates to express the requirements specification.
Design description languages	This approach uses a language like a programming language but with more abstract features to specify the requirements by defining an operational model of the system.
Graphical notations	A graphical language, supplemented by text annotations is used to define the functional requirements for the system. An early example of such a graphical language was SADT (Ross, 1977; Schoman and Ross, 1977). More recently, use-case descriptions (Jacobsen, Christerson et al., 1993) have been used.
Mathematical specifications	These are notations based on mathematical concepts such as finite-state machines or sets. These unambiguous specifications reduce the arguments between customer and contractor about system functionality. However, most customers don't understand formal specifications and are reluctant to accept it as a system contract.

Structured Language Specifications

- **A limited form of natural language** may be used to express requirements
- **This removes some of the problems** resulting from ambiguity and flexibility and imposes a degree of uniformity on a specification



- Often best supported using a forms-based approach

Forms

- Define the information required
- Constrain its format
- Keeps the information in a defined structure
- Filter out extra information that might cause confusion
- Makes it possible to read specification quickly
- Supports tasks like system testing

Form-Based Specifications

- Definition of the function or entity
- Description of inputs and where they come from
- Description of outputs and where they go to
- Indication of other entities required
- Pre and post conditions (if appropriate)
- The side effects (if any)

Form-Based Specification Example

Insulin Pump/Control Software/SRS/3.3.2

Function Compute insulin dose: Safe sugar level

Description Computes the dose of insulin to be delivered when the current measured sugar level is in the safe zone between 3 and 7 units.

Inputs Current sugar reading (r2), the previous two readings (r0 and r1)

Source Current sugar reading from sensor. Other readings from memory.

Outputs CompDose Š the dose in insulin to be delivered

Destination Main control loop

Action: CompDose is zero if the sugar level is stable or falling or if the level is increasing but the rate of increase is decreasing. If the level is increasing and the rate of increase is increasing, then CompDose is computed by dividing the difference between the current sugar level and the previous level by 4 and rounding the result. If the result, is rounded to zero then CompDose is set to the minimum dose that can be delivered.

Requires Two previous readings so that the rate of change of sugar level can be computed.

Pre-condition The insulin reservoir contains at least the maximum allowed single dose of insulin..

Post-condition r0 is replaced by r1 then r1 is replaced by r2

Side-effects None

Forms and incompleteness

- Forms help to check for incompleteness
- Example.. You could add the following
  - Does function have implications for data protection act?
  - Is the function compliant with 35.240.80: IT applications in health care technology?
  - Is this function time constrained and if those what are the constraints?
  - What other modules does this function need to perform it's task?

Tabular Specification

- Tabular Specification is used to supplement natural language.
- It is particularly useful when you have to define a number of possible alternative courses of action
- This can be thought of as a series of “if statements” to determine the action to be taken upon a certain criteria being met.

Tabular Specification Example

Condition	Action
Sugar level falling (r2 < r1)	CompDose = 0
Sugar level stable (r2 = r1)	CompDose = 0
Sugar level increasing and rate of increase decreasing ((r2-r1)<(r1-r0))	CompDose = 0
Sugar level increasing and rate of increase stable or increasing. ((r2-r1) □ (r1-r0))	CompDose = round ((r2-r1)/4) If rounded result = 0 then CompDose = MinimumDose

PDL-Based Requirements Definition

- Program Design Language - Requirements may be defined operationally using a language like a programming language but with more flexibility of expression
- Most appropriate in the following situations:
  - Where an operation is specified as a sequence of actions and the order is important
  - When hardware and software interfaces have to be specified
- Disadvantages include:
  - The PDL may not be sufficiently expressive to define domain concepts
  - The specification will be taken as a *design* rather than a *specification (lead to non optimal solution)*

Part of an ATM Specification

set try_count to 0

Do while PIN not equal to stored PIN

Get PIN from customer

If PIN doesn't equal stored PIN then increment try_count

If try_count equals to maximum tries retain card and quit transaction with error message

End Do

Interface Specification

- Most systems must operate with existing systems and **the operating interfaces** must be precisely specified as part of the requirements
- **Three types of interface** may have to be defined
  - **Procedural interfaces (calling methods)**
  - **Data structures that are exchanged (XML schema)**
  - **Data representations (UNICODE, ASCII etc.)**
- Formal notations are an effective technique for interface specification but their specialised nature means they are difficult to understand without special training.

Example - Interface Description

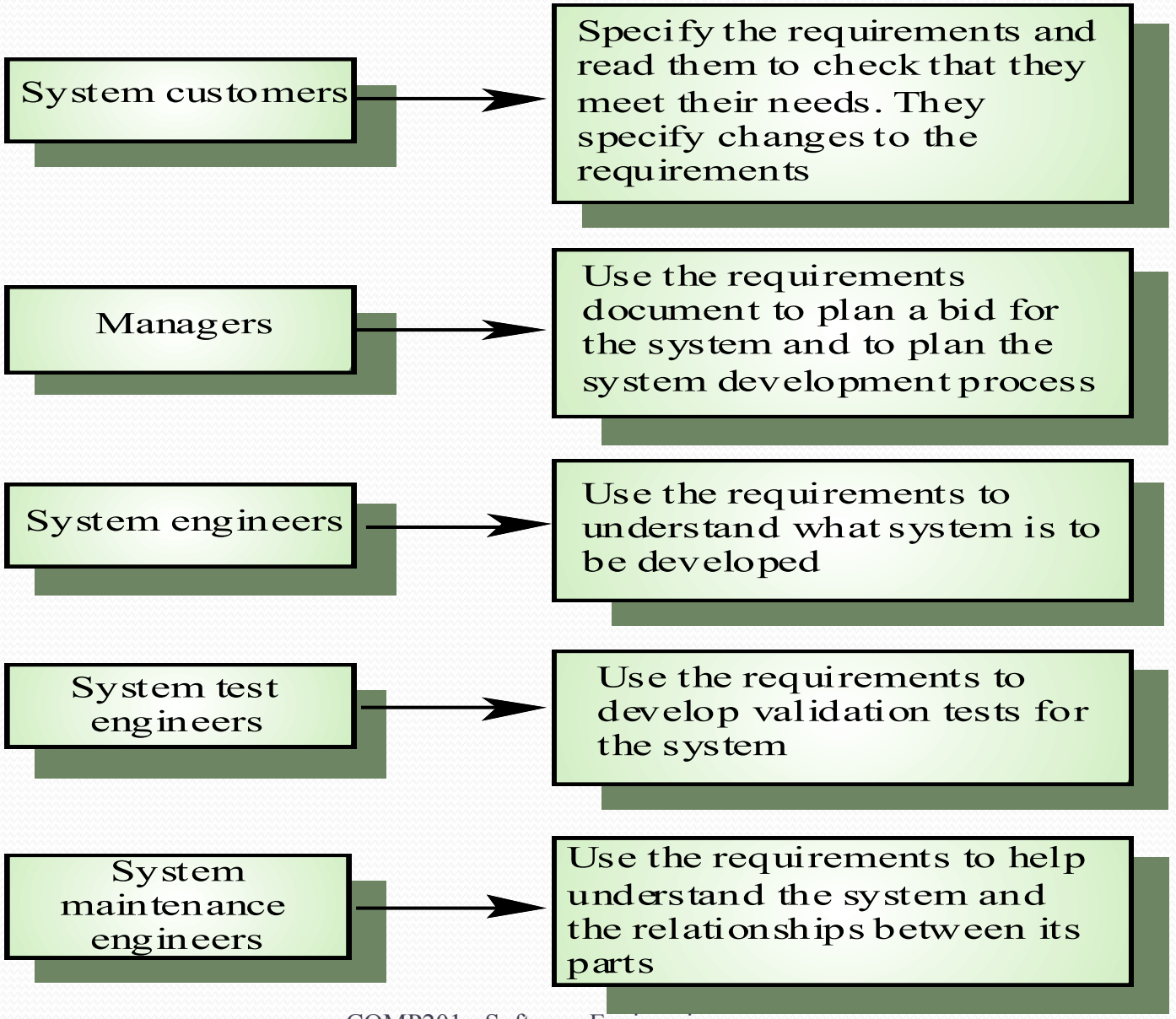
```
interface PrintServer {

// defines an abstract printer server
// requires: interface Printer, interface PrintDoc
// provides: initialize, print, displayPrintQueue, cancelPrintJob, switchPrinter

   void initialize ( Printer p ) ;
   void print ( Printer p, PrintDoc d ) ;
   void displayPrintQueue ( Printer p ) ;
   void cancelPrintJob (Printer p, PrintDoc d) ;
   void switchPrinter (Printer p1, Printer p2, PrintDoc d) ;
} //PrintServer
```

Note this is supported in Java allowing you to use the design to constrain the code

The Requirements Document

- The software requirements document is the **official statement** of what is required of the system developers
- Should include both a definition and a specification of requirements
- It is NOT a design document. As far as possible, it should set of WHAT the system should do rather than HOW it should do it
- The requirements document has a diverse set of users as we see in the next slide

Users of a Requirements Document



Requirements Document

- Specify external system behaviour (what does it do?)
- Specify implementation constraints (what system it must run on, what programming language it must use)
- Easy to change
- Serve as reference tool for maintenance
- Record forethought about the life cycle of the system i.e. predict changes (how can it be expanded for more users)
- Characterise responses to unexpected events (e.g. what should it do if power is lost!)

Requirements Document

- The level of detail used within the requirements document depends on both the type of system and the development process being used.
- For an **evolutionary development** model the requirements may change many times. In the **waterfall model** however, it should be more complete since this has more impact on later stages of the software design process.
- If the (sub)-system will be developed by an external contractor or it is a critical system, more time needs to be taken on finalizing the requirements document.

Requirements Document Structure example

- Preface (including change history)
- Introduction
- Contents
- Glossary
- User requirements definition
- System architecture
- System requirements specification
- System models
- System evolution (we have 10,000 customers, what happens if we have 100,000,000)
- Appendices
- Index

Requirements Document Structure

- Preface
  - Define the expected readers of the document, the **version** history with a rationale for version changes and a summary of changes. Author list
- Introduction
  - Describes the need for the system and the functions provided as well as how it interacts with existing systems. Explain how the software fits into the business or strategic objectives of the organisation.
- Glossary
  - Define technical terms used in the document making no assumptions on the technical expertise of the reader.

Requirements Document Structure

- User requirements definition
  - Describe the services provided for the user and the non-functional requirements of the system using natural language, diagrams understandable by customers. Define any product and process standards.
- System architecture
  - High-level overview of the system architecture showing the distribution of functions across system modules.
- System requirements specification
  - Detailed description of the functional and non-functional requirements.

Requirements Document Structure

- System models
  - Define system models showing relationships between system components, the system and its environment (object, data-flow models etc.)
- System evolution
  - Describe anticipated changes to the system due to hardware evolution and changing user requirements etc.
- Appendices
  - Detailed specific information about the system being developed such as hardware and database descriptions.
- Index
  - Indices of the document including diagrams and functions.

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

Next Lecture

- We shall be looking in more detail at requirements engineering processes and way in which we can elicit, analyse, validate and maintain our requirements.