CS330 Software Engineering

Project Name: calcIp
Author: Ignazio Leonardo Calogero Sperandeo

Software Requirements Specification

Document

Version: 2 **Date: 04/02/2024**

Table of Contents

1 1. Introduction

Il committente ha richiesto un software che dato un ipv4 con la sua subnet mask in notazione CIDR, riceva a video:

- Network address in decimale e binario.
- Broadcast address in decimale e binario.
- L'ipv4 dato in input in binario.
- Gli host massimi che può avere quella rete.
- La wild card.
- La classe di indirizzo, se possibile identificarla.

Inoltre il committente desidera avere la possibilità di effettuare subnetting e supernetting dando in input solamente la nuova subnet mask, in notazione CIDR.

1.1 1.1 Purpose

N/A

1.2 1.2 Scope

N/A

1.3 1.3 Definitions, Acronyms, and Abbreviations.

- **CIDR:** Classless Inter Domain Routing.

1.4 1.4 References

N/A

1.5 1.5 Overview

N/A

2 2. The Overall Description

Dalla richiesta del committente si è potuto comprendere il fatto che vuole un software in grado di estrarre delle informazioni da un ipv4 con la sua corrispettiva subnet mask in notazione CIDR. Inoltre l'utente deve avere la possibilità di effettuare un subnetting o supernetting, dando come input solamente la nuova subnet mask. Nel caso del subnetting l'utente vuole avere tutte le informazioni richieste su tutte le possibili sottoreti possibili.

2.1 2.1 Product Perspective

N/A

1.1 2.1.1 System Interfaces

N/A

1.2

1.3 2.1.2 Interfaces

L'interfaccia del software è CLI.

1.4 2.1.3 Hardware Interfaces

1.5

1.6

1.7

1.8 2.1.4 Software Interfaces
N/A

1.9 2.1.5 Communications Interfaces
N/A

1.10 2.1.6 Memory Constraints
N/A

1.11 2.1.7 Operations
N/A

1.12 2.1.8 Site Adaptation Requirements
N/A

2.2 2.2 Product Functions
N/A

2.3 2.3 User Characteristics
Il software è rivolto ad un pubblico con una conoscenza base di una CLI di sistemi operativi basati su linux e Windows. La macchina su cui dovrà essere eseguito il software dovrà avere installata la versione di python 3..* versioni non comprese in questo range non assicurano il corretto funzionamento del software in quanto alcune entità native o librerie di python non sono supportate.*

2.4 2.4 Constraints
N/A

2.5 2.5 Assumptions and Dependencies
N/A

2.6 2.6 Apportioning of Requirements.
N/A

3 3. Specific Requirements

This section contains all the software requirements at a level of detail sufficient to enable designers to design a system to satisfy those requirements, and testers to test that the system satisfies those requirements. Throughout this section, every stated requirement should be externally perceivable by users, operators, or other external systems. These requirements should include at a minimum a description of every input (stimulus) into the system, every output (response) from the system and all functions performed by the system in response to an input or in support of an output. The following principles apply:

1. *Specific requirements should be stated with all the characteristics of a good SRS*
   - *correct*
   - *unambiguous*
   - *complete*
   - *consistent*

- *ranked for importance and/or stability*
- *verifiable*
- *modifiable*
- *traceable*

2 *Specific requirements should be cross-referenced to earlier documents that relate*
3 *All requirements should be uniquely identifiable (usually via numbering like 3.1.2.3)*
4 *Careful attention should be given to organizing the requirements to maximize readability (Several alternative organizations are given at end of document)*

Before examining specific ways of organizing the requirements it is helpful to understand the various items that comprise requirements as described in the following subclasses. This section reiterates section 2, but is for developers not the customer. The customer buys in with section 2, the designers use section 3 to design and build the actual application.

Remember this is not design. Do not require specific software packages, etc unless the customer specifically requires them. Avoid over-constraining your design. Use proper terminology:
The system shall… A required, must have feature
The system should… A desired feature, but may be deferred til later
The system may… An optional, nice-to-have feature that may never make it to implementation.

Each requirement should be uniquely identified for traceability. Usually, they are numbered 3.1, 3.1.1, 3.1.2.1 etc. Each requirement should also be testable. Avoid imprecise statements like, "The system shall be easy to use" Well no kidding, what does that mean? Avoid "motherhood and apple pie" type statements, "The system shall be developed using good software engineering practice"

Avoid examples, This is a specification, a designer should be able to read this spec and build the system without bothering the customer again. Don't say things like, "The system shall accept configuration information such as name and address." The designer doesn't know if that is the only two data elements or if there are 200. List every piece of information that is required so the designers can build the right UI and data tables.

3.1

3.2 3.1 External Interfaces

This contains a detailed description of all inputs into and outputs from the software system. It complements the interface descriptions in section 2 but does not repeat information there. Remember section 2 presents information oriented to the customer/user while section 3 is oriented to the developer.

It contains both content and format as follows:

- *Name of item*
- *Description of purpose*
- *Source of input or destination of output*
- *Valid range, accuracy and/or tolerance*
- *Units of measure*

- *Timing*
- *Relationships to other inputs/outputs*
- *Screen formats/organization*
- *Window formats/organization*
- *Data formats*
- *Command formats*
- *End messages*

3.3 3.2 Functions

Functional requirements define the fundamental actions that must take place in the software in accepting and processing the inputs and in processing and generating the outputs. These are generally listed as "shall" statements starting with "The system shall…

These include:

- *Validity checks on the inputs*
- *Exact sequence of operations*
- *Responses to abnormal situation, including*
    - *Overflow*
    - *Communication facilities*
    - *Error handling and recovery*
- *Effect of parameters*
- *Relationship of outputs to inputs, including*
    - *Input/Output sequences*
    - *Formulas for input to output conversion*

It may be appropriate to partition the functional requirements into sub-functions or sub-processes. This does not imply that the software design will also be partitioned that way.

3.4 3.3 Performance Requirements

This subsection specifies both the static and the dynamic numerical requirements placed on the software or on human interaction with the software, as a whole. Static numerical requirements may include:

(a) The number of terminals to be supported
(b) The number of simultaneous users to be supported
(c) Amount and type of information to be handled

Static numerical requirements are sometimes identified under a separate section entitled capacity.

Dynamic numerical requirements may include, for example, the numbers of transactions and tasks and the amount of data to be processed within certain time periods for both normal and peak workload conditions.

All of these requirements should be stated in measurable terms.

For example,

95% of the transactions shall be processed in less than 1 second

rather than,

An operator shall not have to wait for the transaction to complete.

(Note: Numerical limits applied to one specific function are normally specified as part of the processing subparagraph description of that function.)

3.5 3.4 Logical Database Requirements

This section specifies the logical requirements for any information that is to be placed into a database. This may include:

- *Types of information used by various functions*
- *Frequency of use*
- *Accessing capabilities*
- *Data entities and their relationships*
- *Integrity constraints*
- *Data retention requirements*

If the customer provided you with data models, those can be presented here. ER diagrams (or static class diagrams) can be useful here to show complex data relationships. Remember a diagram is worth a thousand words of confusing text.

3.6 3.5 Design Constraints

Specify design constraints that can be imposed by other standards, hardware limitations, etc.

6.1 3.5.1 Standards Compliance

Specify the requirements derived from existing standards or regulations. They might include:

(1) Report format
(2) Data naming
(3) Accounting procedures
(4) Audit Tracing

For example, this could specify the requirement for software to trace processing activity. Such traces are needed for some applications to meet minimum regulatory or financial standards. An audit trace requirement may, for example, state that all changes to a payroll database must be recorded in a trace file with before and after values.

3.7 3.6 Software System Attributes

There are a number of attributes of software that can serve as requirements. It is important that required attributes by specified so that their achievement can be objectively verified. The following items provide a partial list of examples. These are also known as non-functional requirements or quality attributes.

These are characteristics the system must possess, but that pervade (or cross-cut) the design. These requirements have to be testable just like the functional requirements. Its easy to start philosophizing here, but keep it specific.

7.1 3.6.1 Reliability

Test da eseguire:

CONVERSIONE IPv4 da decimale a binario:

Input: 192.168.1.0/24 Output: 11000000.1001000.0000001.00000000
Input: 210.47.206.46/19 Output: 11010010.00101111.11001110.00101110
Input: 410.5.0.1/-3 Output: INPUT NON VALIDO
Input: 192.167.1.78/-10 Output: INPUT NON VALIDO
Input: 192.170.3000.1/13 Output: INPUT NON VALIDO

CONVERSIONE network address da decimale a binario:

Input: 192.168.1.0/24 Output: 11000000.1001000.00000001.00000000
Input: 210.47.206.46/19 Output: 11010010.00101111.1100000.00000000
Input: 410.5.0.1/-3 Output: INPUT NON VALIDO
Input: 192.167.1.78/-10 Output: INPUT NON VALIDO
Input: 192.170.30.1/13 Output: 11000000.10101000.00000000.00000000

CONVERSIONE broadcast address da decimale a binario:

Input: 192.168.1.0/24 Output: 11000000.1001000.00000001.11111111
Input: 210.47.206.46/19 Output: 11010010.00101111.1111111.11111111
Input: 410.5.0.1/-3 Output: INPUT NON VALIDO
Input: 255.255.255.255/10 Output: 11111111.11111111.11111111.11111111
Input: 192.170.30.1/13 Output: 11000000.10101111.11111111.11111111

CONVERSIONE wild card da decimale a binario:

Input: 97.56.88.1/13 Output: 00000000.00000111.11111111.11111111
Input 45.78.1.67/24 Output: 00000000.00000000.00000000.11111111
Input: 76.55.120.4/33 Output: INPUT NON VALIDO
Input: 55.340.1.5/1 Output: INPUT NON VALIDO
Input: 192.168.1.0/20 Output: 00000000.00000000.00001111.11111111

TEST sulla classe dell'indirizzo:

Input: 192.168.1.0/24 Output: C
Input: 10.0.0.1/8 Output: A
Input: 173.6.56.88/16 Output: B
Input: 23.11.65.88/11 Output: None
Input: 55.123.200.1/31 Output: None

TEST affidabilità:

Input: 33 Output: None
Input: CIAO Output: None
Input: C.A.B.D Output: None

Input: CIAO.COME.STAI.BENE Output: None
Input: ipv4 Output: None

7.2 3.6.2 Availability

Specify the factors required to guarantee a defined availability level for the entire system such as checkpoint, recovery, and restart. This is somewhat related to reliability. Some systems run only infrequently on-demand (like MS Word). Some systems have to run 24/7 (like an e-commerce web site). The required availability will greatly impact the design. What are the requirements for system recovery from a failure? "The system shall allow users to restart the application after failure with the loss of at most 12 characters of input".

7.3 3.6.3 Security

Specify the factors that would protect the software from accidental or malicious access, use, modification, destruction, or disclosure. Specific requirements in this area could include the need to:

- *Utilize certain cryptographic techniques*
- *Keep specific log or history data sets*
- *Assign certain functions to different modules*
- *Restrict communications between some areas of the program*
- *Check data integrity for critical variables*

7.4

7.5 3.6.4 Maintainability

Specify attributes of software that relate to the ease of maintenance of the software itself. There may be some requirement for certain modularity, interfaces, complexity, etc. Requirements should not be placed here just because they are thought to be good design practices. If someone else will maintain the system

7.6 3.6.5 Portability

Specify attributes of software that relate to the ease of porting the software to other host machines and/or operating systems. This may include:

- *Percentage of components with host-dependent code*
- *Percentage of code that is host dependent*
- *Use of a proven portable language*
- *Use of a particular compiler or language subset*
- *Use of a particular operating system*

Once the relevant characteristics are selected, a subsection should be written for each, explaining the rationale for including this characteristic and how it will be tested and measured. A chart like this might be used to identify the key characteristics (rating them High or Medium), then identifying which are preferred when trading off design or implementation decisions (with the ID of the preferred one indicated in the chart to the

right). The chart below is optional (it can be confusing) and is for demonstrating tradeoff analysis between different non-functional requirements. H/M/L is the relative priority of that non-functional requirement.

ID	Characteristic	H/M/L	1	2	3	4	5	6	7	8	9	10	11	12
1	Correctness													
2	Efficiency													
3	Flexibility													
4	Integrity/Security													
5	Interoperability													
6	Maintainability													
7	Portability													
8	Reliability													
9	Reusability													
10	Testability													
11	Usability													
12	Availability													

Definitions of the quality characteristics not defined in the paragraphs above follow.

- *Correctness - extent to which program satisfies specifications, fulfills user's mission objectives*
- *Efficiency - amount of computing resources and code required to perform function*
- *Flexibility - effort needed to modify operational program*
- *Interoperability - effort needed to couple one system with another*
- *Reliability - extent to which program performs with required precision*
- *Reusability - extent to which it can be reused in another application*
- *Testability - effort needed to test to ensure performs as intended*
- *Usability - effort required to learn, operate, prepare input, and interpret output*

THE FOLLOWING (3.7) is not really a section, it is talking about how to organize requirements you write in section 3.2. At the end of this template there are a bunch of alternative organizations for section 3.2. Choose the ONE best for the system you are writing the requirements for.

3.8 3.7 Organizing the Specific Requirements

For anything but trivial systems the detailed requirements tend to be extensive. For this reason, it is recommended that careful consideration be given to organizing these in a manner optimal for understanding. There is no one optimal organization for all systems. Different classes of systems lend themselves to different organizations of requirements in section 3. Some of these organizations are described in the following subclasses.

8.1 3.7.1 System Mode

Some systems behave quite differently depending on the mode of operation. When organizing by mode there are two possible outlines. The choice depends on whether interfaces and performance are dependent on mode.

8.2 3.7.2 User Class

Some systems provide different sets of functions to different classes of users.

8.3

8.4 3.7.3 Objects

Objects are real-world entities that have a counterpart within the system. Associated with each object is a set of attributes and functions. These functions are also called services, methods, or processes. Note that sets of objects may share attributes and services. These are grouped together as classes.

8.5 3.7.4 Feature

A feature is an externally desired service by the system that may require a sequence of inputs to effect the desired result. Each feature is generally described in as sequence eof stimulus-response pairs.

8.6 3.7.5 Stimulus

Some systems can be best organized by describing their functions in terms of stimuli.

8.7 3. 7.6 Response

Some systems can be best organized by describing their functions in support of the generation of a response.

8.8 3.7.7 Functional Hierarchy

When none of the above organizational schemes prove helpful, the overall functionality can be organized into a hierarchy of functions organized by either common inputs, common outputs, or common internal data access. Data flow diagrams and data dictionaries can be use dot show the relationships between and among the functions and data.

3.9 3.8 Additional Comments

Whenever a new SRS is contemplated, more than one of the organizational techniques given in 3.7 may be appropriate. In such cases, organize the specific requirements for multiple hierarchies tailored to the specific needs of the system under specification.

Three are many notations, methods, and automated support tools available to aid in the documentation of requirements. For the most part, their usefulness is a function of organization. For example, when organizing by mode, finite state machines or state charts

may prove helpful; when organizing by object, object-oriented analysis may prove helpful; when organizing by feature, stimulus-response sequences may prove helpful; when organizing by functional hierarchy, data flow diagrams and data dictionaries may prove helpful.

In any of the outlines below, those sections called "Functional Requirement i" may be described in native language, in pseudocode, in a system definition language, or in four subsections titled: Introduction, Inputs, Processing, Outputs.

4 Change Management Process

Identify the change management process to be used to identify, log, evaluate, and update the SRS to reflect changes in project scope and requirements. How are you going to control changes to the requirements. Can the customer just call up and ask for something new? Does your team have to reach consensus? How do changes to requirements get submitted to the team? Formally in writing, email or phone call?

5 Document Approvals

Identify the approvers of the SRS document. Approver name, signature, and date should be used.

6 Supporting Information

The supporting information makes the SRS easier to use. It includes:

- *Table of Contents*
- *Index*
- *Appendices*

The Appendices are not always considered part of the actual requirements specification and are not always necessary. They may include:

(a) Sample I/O formats, descriptions of cost analysis studies, results of user surveys
(b) Supporting or background information that can help the readers of the SRS
(c) A description of the problems to be solved by the software
(d) Special packaging instructions for the code and the media to meet security, export, initial loading, or other requirements

When Appendices are included, the SRS should explicitly state whether or not the Appendices are to be considered part of the requirements.

Tables on the following pages provide alternate ways to structure section 3 on the specific requirements. You should pick the best one of these to organize section 3 requirements.

Outline for SRS Section 3
Organized by mode: Version 1

3. Specific Requirements
   3.1 External interface requirements
      1 User interfaces
      2 Hardware interfaces
      3 Software interfaces
      4 Communications interfaces
    2 Functional requirements
    3.2.1 Mode 1
      3.2.1.1 Functional requirement 1.1
      .....
      3.2.1.*n* Functional requirement 1.*n*
        2 Mode 2
      .....
    3.2.*m* Mode *m*
      3.2.*m*.1 Functional requirement *m*.1
      .....
      3.2.*m.n* Functional requirement *m.n*
   3.3 Performance Requirements
   3.4 Design Constraints
   3.5 Software system attributes
   3.6 Other requirements

Outline for SRS Section 3
Organized by mode: Version 2

3. Specific Requirements
  3.1 Functional Requirements
    1 Mode 1
    3.1.1.1 External interfaces
      3.1.1.1 User interfaces
      3.1.1.2 Hardware interfaces
      3.1.1.3 Software interfaces
      3.1.1.4 Communications interfaces
    3.1.1.2 Functional Requirement
      3.1.1.2.1 Functional requirement 1
      .....
      3.1.1.2.*n* Functional requirement *n*
    3.1.1.3 Performance
  3.1.2 Mode 2
    .....
  3.1.*m* Mode *m*
  2 Design constraints
  3 Software system attributes
  4 Other requirements

Outline for SRS Section 3
Organized by user class (i.e. different types of users ->System Adminstrators, Managers, Clerks, etc.)

3. Specific Requirements
   3.1 External interface requirements
      1 User interfaces
      2 Hardware interfaces
      3 Software interfaces
      4 Communications interfaces
   2 Functional requirements
   3.2.1 User class 1
   3.2.1.1 Functional requirement 1.1
   .....
   3.2.1.*n* Functional requirement 1.*n*
   2 User class 2
   .....

   3.2.*m* User class *m*
   3.2.*m*.1 Functional requirement *m*.1
   .....
   3.2.*m.n* Functional requirement *m.n*
   3.3 Performance Requirements
   3.4 Design Constraints
   3.5 Software system attributes
   3.6 Other requirements

Outline for SRS Section 3
Organized by object (Good if you did an object-oriented analysis as part of your requirements)

3 Specific Requirements
3.1 External interface requirements
1 User interfaces
2 Hardware interfaces
3 Software interfaces
4 Communications interfaces
2 Classes/Objects
3.2.1 Class/Object 1
3.2.1.1 Attributes (direct or inherited)
1 Attribute 1
.....
3.2.1.1.*n* Attribute *n*

2 Functions (services, methods, direct or inherited)
3.2.1.2.1 Functional requirement 1.1
.....
3.2.1.2.*m* Functional requirement 1.*m*
3.2.1.3 Messages (communications received or sent)
3.2.2 Class/Object 2
.....
3.2.*p* Class/Object *p*
3.3 Performance Requirements
3.4 Design Constraints
3.5 Software system attributes
3.6 Other requirements

Outline for SRS Section 3
Organized by feature (Good when there are clearly delimited feature sets.

3 Specific Requirements
3.1 External interface requirements
1 User interfaces
2 Hardware interfaces
3 Software interfaces
4 Communications interfaces
2 System features
3.2.1 System Feature 1
3.2.1.1 Introduction/Purpose of feature
3.2.1.2 Stimulus/Response sequence
3.2.1.3 Associated functional requirements
3.2.1.3.1 Functional requirement 1
.....
3.2.1.3.*n* Functional requirement *n*
3.2.2 System Feature 2
.....
3.2.*m* System Feature *m*
.....
3.3 Performance Requirements
3.4 Design Constraints
3.5 Software system attributes
3.6 Other requirements

Outline for SRS Section 3
Organized by stimulus (Good for event driven systems where the events form logical groupings)

3 Specific Requirements
3.1 External interface requirements
1 User interfaces
2 Hardware interfaces
3 Software interfaces
4 Communications interfaces
2 Functional requirements
3.2.1 Stimulus 1
3.2.1.1 Functional requirement 1.1
.....
3.2.1.*n* Functional requirement 1.*n*
3.2.2 Stimulus 2
.....
3.2.*m* Stimulus *m*
3.2.*m*.1 Functional requirement *m*.1
.....
3.2.*m.n* Functional requirement *m.n*
3.3 Performance Requirements
3.4 Design Constraints
3.5 Software system attributes
3.6 Other requirements

Outline for SRS Section 3
Organized by response (Good for event driven systems where the responses form logical groupings)

3 Specific Requirements
3.1 External interface requirements
1 User interfaces
2 Hardware interfaces
3 Software interfaces
4 Communications interfaces
2 Functional requirements
3.2.1 Response 1
3.2.1.1 Functional requirement 1.1
.....
3.2.1.*n* Functional requirement 1.*n*
3.2.2 Response 2
.....
3.2.*m* Response *m*
3.2.*m*.1 Functional requirement *m*.1
.....
3.2.*m.n* Functional requirement *m.n*
3.3 Performance Requirements
3.4 Design Constraints
3.5 Software system attributes
3.6 Other requirements

Outline for SRS Section 3
Organized by functional hierarchy (Good if you have done structured analysis as part of your design.)

3 Specific Requirements
3.1 External interface requirements
1 User interfaces
2 Hardware interfaces
3 Software interfaces
4 Communications interfaces
2 Functional requirements
3.2.1 Information flows
3.2.1.1 Data flow diagram 1
1 Data entities
2 Pertinent processes
3 Topology
3.2.1.2 Data flow diagram 2
1 Data entities
2 Pertinent processes
3 Topology
.....
3.2.1.*n* Data flow diagram *n*
3.2.1.*n*.1 Data entities
3.2.1.*n*.2 Pertinent processes
3.2.1.*n*.3 Topology
3.2.2 Process descriptions
1 Process 1
1 Input data entities
2 Algorithm or formula of process
3 Affected data entities
3.2.2.2 Process 2
3.2.2.2.1 Input data entities
3.2.2.2.2 Algorithm or formula of process
3.2.2.2.3 Affected data entities
…..
3.2.2.*m* Process *m*
3.2.2.*m*.1 Input data entities
3.2.2.*m*.2 Algorithm or formula of process
3.2.2.*m*.3 Affected data entities
3.2.3 Data construct specifications
3.2.3.1 Construct 1
3.2.3.1.1 Record type
3.2.3.1.2 Constituent fields
3.2.3.2 Construct 2
3.2.3.2.1 Record type
3.2.3.2.2 Constituent fields
…..
3.2.3.*p* Construct *p*
3.2.3.*p*.1 Record type

Outline for SRS Section 3
Showing multiple organizations (Can't decide? Then glob it all together)

3 Specific Requirements
3.1 External interface requirements
1 User interfaces
2 Hardware interfaces
3 Software interfaces
4 Communications interfaces
2 Functional requirements
3.2.1 User class 1
3.2.1.1 Feature 1.1
3.2.1.1.1 Introduction/Purpose of feature
3.2.1.1.2 Stimulus/Response sequence
3.2.1.1.3 Associated functional requirements
3.2.1.2 Feature 1.2
3.2.1.2.1 Introduction/Purpose of feature
3.2.1.2.2 Stimulus/Response sequence
3.2.1.2.3 Associated functional requirements
…..
3.2.1.*m* Feature 1.*m*
3.2.1.*m*.1 Introduction/Purpose of feature
3.2.1.*m*.2 Stimulus/Response sequence
3.2.1.*m*.3 Associated functional requirements
3.2.2 User class 2
.....
3.2.*n* User class *n*
.....
3.3 Performance Requirements
3.4 Design Constraints
3.5 Software system attributes
3.6 Other requirements

Outline for SRS Section 3
Organized by Use Case (Good when following UML development)

3. Specific Requirements
  3.1 External Actor Descriptions
    3.1.1 Human Actors
    3.1.2 Hardware Actors
    3.1.3 Software System Actors
  3.2 Use Case Descriptions
    3.2.1 Use Case 1
    3.2.2 Use Case 2

    3.2.n Use Case n
  3.3 Performance Requirements
  3.4 Design Constraints
  3.5 Software system attributes
  3.6 Other requirements