

PART 1: Overview & Software Requirements Specification

N.B. 1: You must use all the following notations while describing the requirements in Part 1:

- 1) Natural Language
- 2) Structured Natural Language (Forms “Templates” / Tabular “Tables” / etc.)
- 3) Graphical Notations (Diagrams such as simple Use-Case Diagrams, Activity Diagrams, etc.)
- 4) Mathematical Specification (using mathematical formulas or sets)

N.B. 2: When describing the requirements in Part 1, you must include an explanation (rationale) of why a requirement is necessary whenever possible.

1) Introduction:

- a) Purpose.
- b) Project Scope.
- c) Glossary and Abbreviations (*for any technical or non-technical terms*).
- d) List of the System Stakeholders.
- e) References.

2) Functional Requirements:

- a) User Requirements Specification.
- b) System Requirements Specification.
- c) Requirements' Priorities (*using the MoSCoW Scheme or any other Scheme you may select*).

3) Non-functional requirements:

- a) The General Types/Categories of Non-Functional Requirements (*that you will follow; you may select from the categories presented in the lecture*).
- b) Non-functional requirements Specification (*including the category/type of each*).
- c) The fit criteria for every Non-Functional Requirement (*Testable Non-Functional Requirements*).
- d) How would the above-mentioned Non-Functional Requirements affect the System's overall Architecture?

4) Design & Implementation Constraints.

5) System Evolution:

- a) Anticipated changes.
- b) How should any anticipated changes in the future (*due to hardware evolution, changing user needs, and so on*) affect the system design?

6) What are the requirements discovery approaches that you'll rely on? (*Give detailed examples*)

7) What requirements validation techniques will you employ/use? (*Give detailed examples*)

PART 2: System Design & Models

8) Functional Diagrams:

- a) **Use-Case Diagram(s) including all the system Use Cases** *(the diagram should include any required inclusions/extensions between use cases and any required generalisations between actors).*
- b) **Detailed Use-Cases Description** *(for every Use-Case, the description should include the Use-Case ID and name, Goal, Initiator, Pre-condition(s), Post-condition(s), Main Success Scenario, and any Alternative or Unsuccessful Scenarios).*
- c) **Package Diagram grouping relevant Use Cases into Packages.**

9) Structural & Behavioural Diagrams:

- a) **System Architecture** *(including applied Architectural Pattern(s), i.e. What patterns have you used and why?).*
- b) **Activity Diagrams** *(for every significant business process in the system, at least 6 diagrams)*
- c) **Based on the Activity Diagrams, the List of User Interfaces required for the System and the corresponding users of each interface.**
- d) **Class Diagram 1: An initial version based on the requirements and Use-Case/Activity diagrams** *(including classes, initial attributes, and basic operations).*
- e) **Sequence Diagram(s)** *(for every Use Case.)*
- f) **System Sequence Diagrams (SSDs)** *(at least 6 diagrams for every significant business process in the system).*
- g) **Collaboration/Communication Diagram(s)** *(including all the messages mentioned in the sequence diagrams).*
- h) **Which strategy (or strategies) did you use to implement the use cases: One Central Class, Actor Class, or Use-Case class? Please explain your choice and the potential advantages/disadvantages of your design.**
- i) **Class Diagram 2: An intermediate version based on the interaction diagrams** *(including all classes, attributes, and operations mentioned in the interaction diagrams).*
- j) **Three Design Patterns Applied** *(including the description for each design pattern, what problem it solves, and how it affects your system's design).*
- k) **Class Diagram 3: The final version, after applying the design pattern(s) and any other modifications** *(the Class Diagram should include Associations, Self-associations (Recursive Associations), Multiplicities, Roles, Inheritance relationships, Polymorphism, Aggregation(s), and/or Composition(s), Qualified association(s), Association classes, and Interface class(es)).*
- l) **[1 Bonus Mark] Define a design smell, highlight a design smell in your design, and suggest how it can be avoided.**
- m) **[1 Bonus Mark] Read about Class Structuring Criteria and classify all the classes in your class diagram into one of the different categories of application classes. The general categories are an Entity Class, a Boundary Class, a Control Class, or an Application Logic Class. Some of those categories are further categorised.**
- n) **Did you rely on Forks or Cascades in your interaction diagrams? Give an example of your choice and mention its advantage(s)/disadvantage(s).**
- o) **Package Diagram grouping relevant Classes into Packages.**
- p) **Object Diagrams** *(including object diagrams that illustrate the preconditions and the post-conditions for every significant function).*
- q) **Database Specification (ERD, Tables.)**

PART 3: Development Phase (Implementation Details)

- 10) Create and document a Front-End Design for all Functions (HTML, Bootstrap).**
- 11) Document an Implementation based on the abovementioned Requirements & Design**
(it should include the following modules, in addition, of course, to modules specific to your projects):
 - a) User Role Management Module.**
 - b) User manipulation Module** *(Login, Add / Delete / Update / Search, List).*
 - c) Controlling Resources Module** *(Rooms, Orders, Products, etc.).*
 - d) Reservation and Rescheduling Module.**
 - e) Generating Reports Module** *(PDFs, ... etc.).*
 - f) Sending Emails or Notifications Module.**
 - g) File Uploaders.**

PART 4: Complexity & Testing

- 12) Are there pairs of Software Quality Factors that are not independent in your system? Give an example.**
- 13) Calculate the LOC and CCM (Cyclomatic Complexity Metric) for the main functions in your system.**
- 14) For the classes in your system, calculate all the following OO Complexity Metrics**
(mention exactly the equation you've used):
 - a) WMC (Weighted Methods per Class)**
 - b) DIT (Depth of Inheritance Tree)**
 - c) NOC (Number of Children)**
 - d) CBO (Coupling Between Objects)**
 - e) RFC (Response for Class)**
 - f) LCOM (Lack of Cohesion of Methods)**
- 15) Considering White-Box Testing, generate a Unit-Testing Test Report for at least 6 main functions in your system. For each function, consider path testing by determining a set of test cases (the value of the function's parameters) such that each path through the function is executed at least once.**
- 16) Considering Black-Box Testing, generate a Functionality System-Testing Test Report for at least 6 main functions in your system. For each function, consider boundary testing by determining a set of test cases (the value of the function's parameters) from the extreme ends or boundaries between partitions of the input values.**