TABLE OF CONTENTS

1. INTRODUCTION

The introduction should contain the following information:

1.1 Purpose and Scope

This section provides a brief description of the proposed systems, and you should describe the system in narrative form using non-technical terms.

1.2 Problem Statement

State the problem to be solved. What are the research questions? Why are you doing this work and what significance does it have in the relevant literature? Even if your project is applied (as opposed to research-oriented), you are building a system because a problem, requiring a solution in the form of a computer program, exists.

1.3 Solution Statement

State your solution to the problem.

1.4 Contribution

State how your solution builds upon and extends current technology.

1.5 Glossary

Supply a glossary of all terms and abbreviations used in this document. If the glossary is several pages in length, it may be included as an appendix.

2. LITERATURE REVIEW

The literature review is an extremely important component of the senior project to ensure that students are making research a vital part of their decision making.

A literature review (also expressed as "a review of the literature") is an overview of previous research and the important aspect of the project topic. It identifies, describes and analyzes both 1) related research that has already been done, and 2) the state of knowledge about the topic.

The research process often begins with a question that the student(s) would like to answer. In order to identify what other research has addressed this question and to find out what is already known about it, the student will conduct a literature review. This entails examining scholarly books and journal articles, and sometimes additional resources such as conference proceedings and dissertations, to learn about previous research related to the question. Students want to be able to identify what is already known about the question and to build upon existing knowledge. Familiarity with previous research also helps students design their own study. Once this literature review foundation is developed, a student decides how she or he will study the subject, designs a research method or methods, collects and analyzes the data, and reflects on what has been learned.

2.1 The Objectives of a Literature Review

Students should try to accomplish the following four important objectives in preparing a literature review:

1. The review should provide a thorough overview of previous research on the topic. This should be a helpful review for readers who are already familiar with the topic and an essential background for readers who are new to the topic. The review should provide a clear sense about how the author's current research fits into the broader, sociological understanding of the topic. When the reader completes reading of the literature review, she or he should be able to say, "I now know what previous research has learned about this topic."

2. The review should contain references to important previous studies related to the research question that are found in high quality sources such as scholarly books and journals. A good literature review conveys to readers that the author has been conscientious in examining previous research and that the author's research builds on what is already known. In this process, highly interested readers are also provided with a set of references that they may wish to read themselves.

3. The review should be succinct and well-organized.

4. The review should follow generally established stylistic guidelines. This conveys to readers that the author is familiar with scholarly publication style, and that can add legitimacy to the author's work. Plus, when the typical style is used, it is easier for readers to immediately follow the report's organization.

3. SOFTWARE REQUIREMENTS SPECIFICATION

NOTE: This section describes the SRS report in detail. You do not have to complete all the subheading in your report, but you need to clearly explain the all requirements of your project.

3.1 Introduction

This chapter presents the software requirements specifications.
In general, it describes:

- Requirements of functionalities, performances, interfaces, environment …
- Tests principles and definitions of validation methods of requirements,
- The compliance of requirements to customer needs,
- The relative importance and precedence of requirements

3.2 The Overall Description

Describe the general factors that affect the product and its requirements. This section does not state specific requirements. Instead, it provides a background for those requirements, and makes them easier to understand. In a sense, this section tells the requirements in plain English for the consumption of the customer.

3.2.1 Product Perspective

Put the product into perspective with other related products. If the product is independent and totally self-contained, it should be so stated here. If the SRS defines a product that is a component of a larger system, as frequently occurs, then this subsection relates the requirements of the larger system to functionality of the software and identifies interfaces between that system and the software. If you are building a real system, compare its similarity and differences to other systems in the marketplace. If you are doing a research-oriented project, what related research compares to the system you are planning to build. A block diagram showing the major components of the larger system, interconnections, and external interfaces can be helpful. This is not a design or architecture picture. It is more to provide context, especially if your system will interact with external actors. The system you are building should be shown as a black box. Let the design document present the internals.

The following subsections describe how the software operates inside various constraints.

3.2.1.1 System Interfaces

List each system interface and identify the functionality of the software to accomplish the system requirement and the interface description to match the system. These are external systems that you have to interact with. For instance, if you are building a business application that interfaces with the existing employee payroll system, what is the API to that system that designer's will need to use?

3.2.1.2 Interfaces

Specify:

(1) The logical characteristics of each interface between the software product and its users.

(2) All the aspects of optimizing the interface with the person who must use the system

3.2.1.3 Hardware Interfaces

Specify the logical characteristics of each interface between the software product and the hardware components of the system. This includes configuration characteristics. It also covers such matters as what devices are to be supported, how they are to be supported and protocols. This is not a description of hardware requirements in the sense that "This program must run on a Mac with 64M of RAM". This section is for detailing the actual hardware devices your application will interact with and control. For instance, if you are controlling X10 type home devices, what is the interface to those devices? Designers should be able to look at this and know what hardware they need to worry about in the design. Many business type applications will have no hardware interfaces.

If none, just state "The system has no hardware interface requirements" If you just delete sections that are not applicable, then readers do not know if: a. This does not apply or b. You forgot to include the section in the first place.

3.2.1.4 Software Interfaces

Specify the use of other required software products and interfaces with other application systems. For each required software product, include:

(1) Name
(2) Mnemonic
(3) Specification number
(4) Version number
(5) Source

For each interface, provide:

(1) Discussion of the purpose of the interfacing software as related to this software product
(2) Definition of the interface in terms of message content and format

Here we document the APIs, versions of software that we do not have to write, but that our system has to use. For instance if your customer uses SQL Server 7 and you are required to use that, then you need to specify i.e. Microsoft SQL Server 7. The system must use SQL Server as its database component. Communication with the DB is through ODBC connections. The system must provide SQL data table definitions to be provided to the company DBA for setup.

A key point to remember is that you do NOT want to specify software here that you think would be good to use. This is only for **customer-specified systems** that you **have** to interact with. Choosing SQL Server 7 as a DB without a customer requirement is a Design choice, not a requirement. This is a subtle but important point to writing good requirements and not over-constraining the design.

3.2.1.5 Communications Interfaces

Specify the various interfaces to communications such as local network protocols, etc. These are protocols you will need to directly interact with. If you happen to use web services transparently to your application then do not list it here. If you are using a custom protocol to communicate between systems, then document that protocol here so designers know what to design. If it is a standard protocol, you can reference an existing document or RFC.

3.2.1.6 Memory Constraints

Specify any applicable characteristics and limits on primary and secondary memory. Don't just make up something here. If all the customer's machines have only 128K of RAM, then your target design has got to come in under 128K so there is an actual requirement. You could also cite market research here for shrink-wrap type applications "Focus groups have determined that our target market has between 256-512M of RAM, therefore the design footprint should not exceed 256M." If there are no memory constraints, so state.

3.2.1.7 Operations

Specify the normal and special operations required by the user such as:

(1) The various modes of operations in the user organization

(2) Periods of interactive operations and periods of unattended operations

(3) Data processing support functions

(4) Backup and recovery operations

(Note: This is sometimes specified as part of the User Interfaces section.) If you separate this from the UI stuff earlier, then cover business process type stuff that would impact the design. For instance, if the company brings all their systems down at midnight for data backup that might impact the design. These are all the work tasks that impact the design of an application, but which might not be located in software.

3.2.1.8 Site Adaptation Requirements

In this section:

(1) Define the requirements for any data or initialization sequences that are specific to a given site, mission, or operational mode

(2) Specify the site or mission-related features that should be modified to adapt the software to a particular installation

If any modifications to the customer's work area would be required by your system, then document that here. For instance, "A 100Kw backup generator and 10000 BTU air conditioning system must be installed at the user site prior to software installation".

This could also be software-specific like, “New data tables created for this system must be installed on the company’s existing DB server and populated prior to system activation.” Any equipment the customer would need to buy or any software setup that needs to be done so that your system will install and operate correctly should be documented here.

3.3 Product Functions

Provide a summary of the major functions that the software will perform. Sometimes the function summary that is necessary for this part can be taken directly from the section of the higher-level specification (if one exists) that allocates particular functions to the software product.

For clarity:

(1) The functions should be organized in a way that makes the list of functions understandable to the customer or to anyone else reading the document for the first time.

(2) Textual or graphic methods can be used to show the different functions and their relationships. Such a diagram is not intended to show a design of a product but simply shows the logical relationships among variables.

This section describes the functionality of the system in the language of the customer. What specifically does the system that will be designed have to do? Drawings are good, but remember this is a description of what the system needs to do, not how you are going to build it. (That comes in the design document).

3.4 User Characteristics

Describe those general characteristics of the intended users of the product including educational level, experience, and technical expertise. Do not state specific requirements but rather provide the reasons why certain specific requirements are later specified in section 3.8.

What is it about your potential user base that will impact the design? Their experience and comfort with technology will drive UI design. Other characteristics might actually influence internal design of the system.

3.5 Constraints

Provide a general description of any other items that will limit the developer's options. These can include:

(1) Regulatory policies
(2) Hardware limitations (for example, signal timing requirements)
(3) Interface to other applications
(4) Parallel operation
(5) Audit functions
(6) Control functions
(7) Higher-order language requirements
(8) Reliability requirements
(9) Criticality of the application
(10) Safety and security considerations

This section captures non-functional requirements in the customers language.

3.6 Assumptions and Dependencies

List each of the factors that affect the requirements stated in the SRS. These factors are not design constraints on the software but are, rather, any changes to them that can affect the requirements in the SRS. For example, an assumption might be that a specific operating system would be available on the hardware designated for the software product. If, in fact, the operating system were not available, the SRS would then have to change accordingly. This section is catch-all for everything else that might influence the design of the system and that did not fit in any of the categories above.

3.7 Apportioning of Requirements.

Identify requirements that may be delayed until future versions of the system. After you look at the project plan and hours available, you may realize that you just cannot get everything done. This section divides the requirements into different sections for development and delivery. Remember to check with the customer – they should prioritize the requirements and decide what does and does not get done. This can also be useful if you are using an iterative life cycle model to specify which requirements will map to

which interation.

3.8 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 3.2, but is for developers not the customer. The customer buys in with section 3.2, the designers use section 3.3 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.

3.8.1 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.

3.8.2 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.)

3.8.3 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.

3.8.4 Design Constraints

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

3.8.4.1 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.

3.8.5 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.

3.8.5.1 Reliability

Specify the factors required to establish the required reliability of the software system at time of delivery. If you have MTBF requirements, express them here. This doesn't refer to just having a program that does not crash. This has a specific engineering meaning.

3.8.5.2 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".

3.8.5.3 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

3.8.5.4 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

3.8.5.5 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.

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 section talks about how to organize requirements you write in section 3.8.2. At the end of this template there are a bunch of alternative organizations for section 3.8.2. Choose the ONE best for the system you are writing the requirements for.

3.8.6 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 3.8. Some of these organizations are described in the following subclasses.

3.8.6.1 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.

3.8.6.2 User Class

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

3.8.6.3 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.

3.8.6.4 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.

3.8.6.5 Stimulus

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

3.8.6.6 Response

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

3.8.6.7 Functional Hierarchy

When none of he 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.

4. SYSTEM DESIGN SPECIFICATION

4.1 Introduction

This chapter describes the system requirements, operating environment, system and subsystem architecture, files and database design, input formats, output layouts, human-machine interfaces, detailed design, processing logic, and external interfaces.

Mockup is the sample modeling file for any of the content. This example can change the content of modeling files, the model can be used in different ways again and again without losing the different types can occur. The most common use of mockups in software development is to create user interfaces that show the end user what the software will look like without having to build the software or the underlying functionality. A software mockup will thus look like the real thing, but will not do useful work beyond what the user sees.

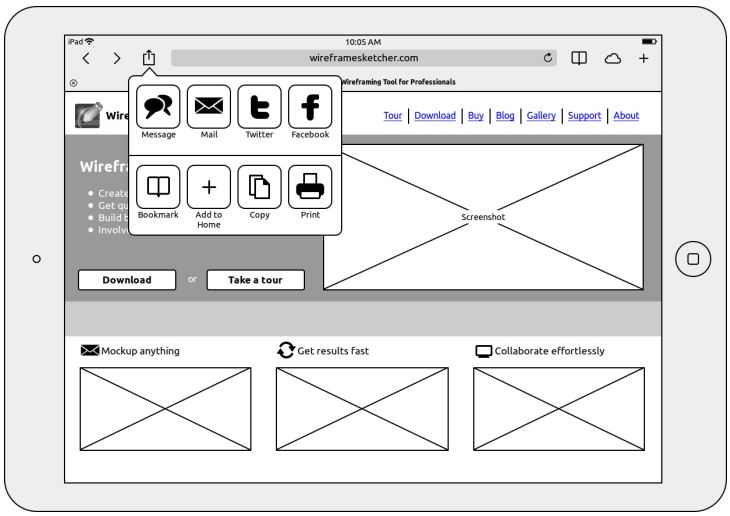

Figure 4.1. Mockup sample

4.2 SYSTEM ARCHITECTURE

In this section, describe the system and/or subsystem(s) architecture for the project. References to external entities should be minimal, as they will be described in detail in Section 4.5, External Interfaces.

4.2.1 System Software Architecture

In this section, describe the overall system software and organization. Include a list of

software modules (this could include functions, subroutines, or classes), computer languages, and programming computer-aided software engineering tools (with a brief description of the function of each item). Use structured organization diagrams/object-oriented diagrams that show the various segmentation levels down to the lowest level. All features on the diagrams should have reference numbers and names. Include a narrative that expands on and enhances the understanding of the functional breakdown. If appropriate, use subsections to address each module.

Note: The diagrams should map to the FRD data flow diagrams, providing the physical process and data flow related to the FRD logical process and data flow.

4.3 DATABASE DESIGN

Interact with the Database Administrator (DBA) when preparing this section. The section should reveal the final design of all database management system (DBMS) files. Additional information may add as required for the particular project. Provide a comprehensive data dictionary showing data element name, type, length, source, validation rules, maintenance (create, read, update, delete (CRUD) capability), data stores, outputs, aliases, and description. ER diagrams are commonly used in conjunction with a data flow diagram to display the contents of a data store. It helps to visualize how data is connected in a general way, and are particularly useful for constructing a relational database. This part should also reveal the ER diagram of the project's database. Can be included as an appendix.

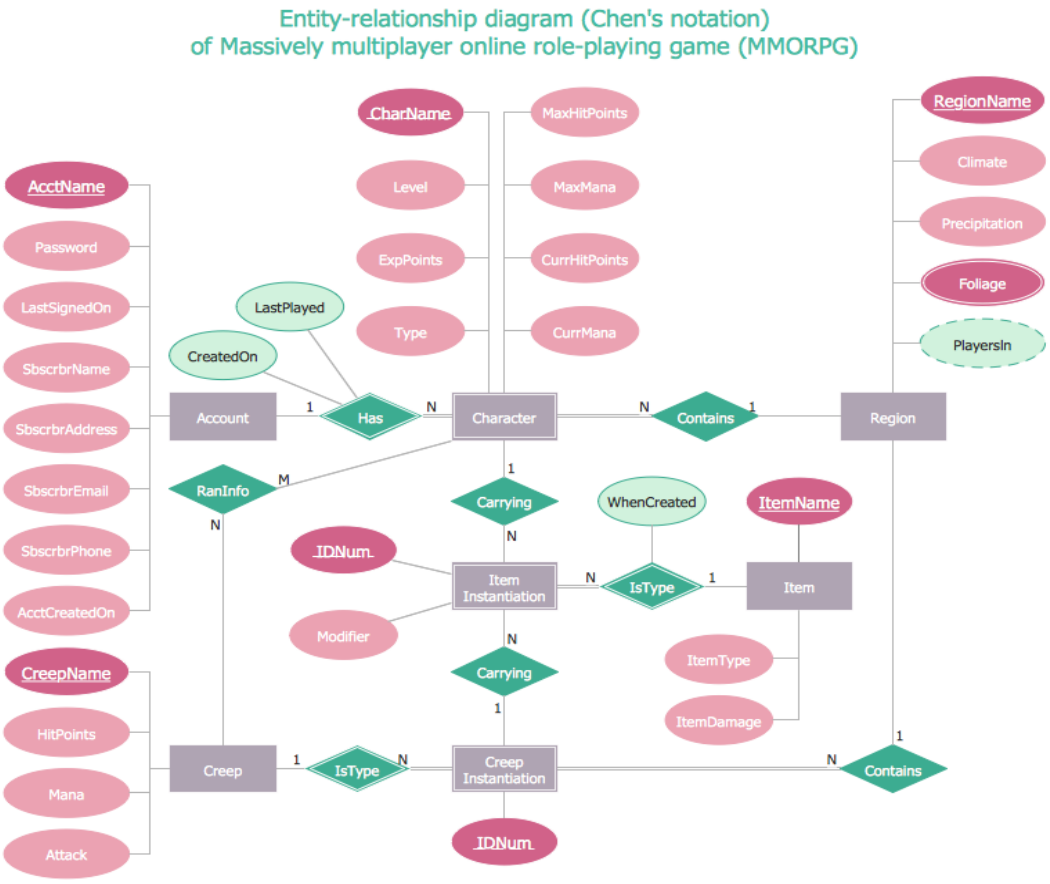

Figure 4.2. ER Diagram sample

4.4 HUMAN-MACHINE INTERFACE

This section provides the detailed design of the system and subsystem inputs and outputs relative to the user/operator. Any additional information may be added to this section and may be organized according to whatever structure best presents the operator input and output designs. Depending on the particular nature of the project, it may be appropriate to repeat these sections at both the subsystem and design module levels. Additional information may be added to the subsections if the suggested lists are inadequate to describe the project inputs and outputs.

Use case diagrams are usually referred to as behavior diagrams used to describe a set of actions (use cases) that some system or systems (subject) should or can perform in collaboration with one or more external users of the system (actors). Each use case should provide some observable and valuable result to the actors or other stakeholders of the system.

Use case diagrams are used to specify:

- (external) Requirements, required usages of a system under design or analysis (subject) - to capture what the system is supposed to do;
- The functionality offered by a subject – what the system can do;
- Requirements the specified subject poses on its environment - by defining how environment should interact with the subject so that it will be able to perform its services.

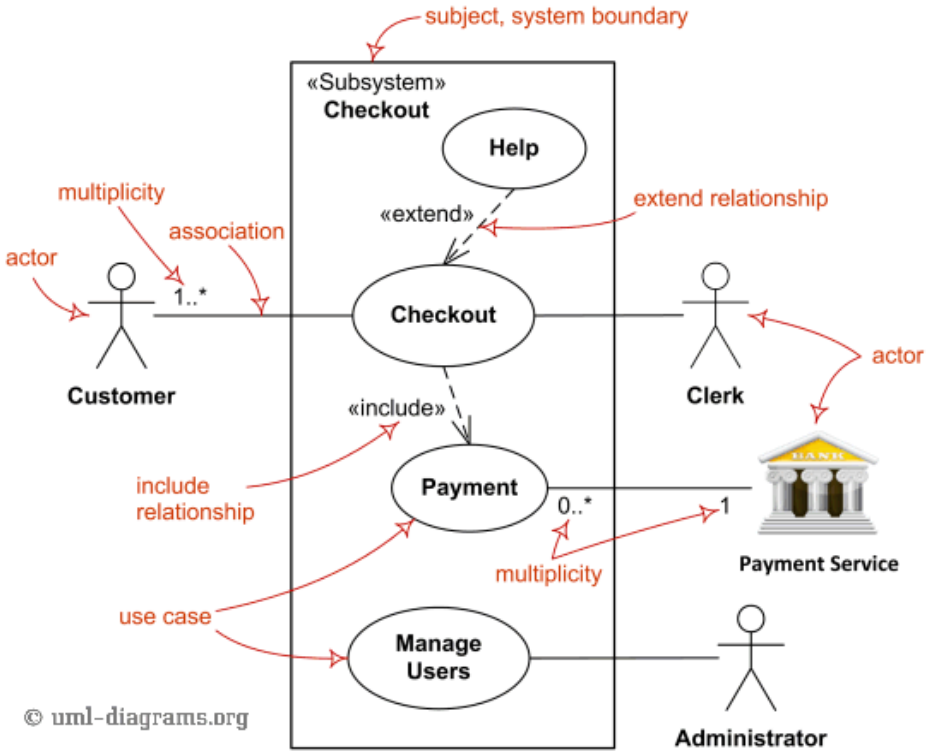

Figure 4.3. Use Case Diagram sample

4.4.1 Inputs

This section is a description of the input media used by the operator for providing information to the system; show a mapping to the high-level data flows described in chapter 1, System Overview. For example, data entry screens, optical character readers, bar scanners, etc. If appropriate, the input record types, file structures, and database structures provided in Section 4.3, File and Database Design, may be referenced. Include data element definitions, or refer to the data dictionary.

Provide the layout of all input data screens or graphical user interfaces (GUTs) (for example, windows). Provide a graphic representation of each interface. Define all data

elements associated with each screen or GUI, or reference the data dictionary.

This section should contain edit criteria for the data elements, including specific values, range of values, mandatory/optional, alphanumeric values, and length. Also address data entry controls to prevent edit bypassing.

Discuss the miscellaneous messages associated with operator inputs, including the following:

- Copies of form(s) if the input data are keyed or scanned for data entry from printed forms
- Description of any access restrictions or security considerations
- Each transaction name, code, and definition, if the system is a transaction-based processing system

4.4.2 Outputs

This section describes of the system output design relative to the user/operator; show a mapping to the high-level data flows described in the first chapter. System outputs include reports, data display screens and GUIs, query results, etc. The output files are described in Section 4.3 and may be referenced in this section. The following should be provided, if appropriate:

- Identification of codes and names for reports and data display screens
- Description of report and screen contents (provide a graphic representation of each layout and define all data elements associated with the layout or reference the data dictionary)
- Description of the purpose of the output, including identification of the primary users
- Report distribution requirements, if any (include frequency for periodic reports)
- Description of any access restrictions or security considerations

4.5 DETAILED DESIGN

This section provides the information needed for a system development team to actually build and integrate the hardware components, code and integrate the software modules, and interconnect the hardware and software segments into a functional product. Additionally, this section addresses the detailed procedures for combining separate COTS packages into a single system.

4.5.1 Software Detailed Design

A software module is the lowest level of design granularity in the system. Depending on the software development approach, there may be one or more modules per system. This section should provide enough detailed information about logic and data necessary to completely write source code for all modules in the system (and/or integrate COTS software programs).

If there are many modules or if the module documentation is extensive, place it in an appendix or reference a separate document. Add additional diagrams and information, if necessary, to describe each module, its functionality, and its hierarchy. Industry-standard module specification practices should be followed. Include the following information in the detailed module designs:

- A narrative description of each module, its function(s), the conditions under which it is used (called or scheduled for execution), its overall processing, logic, interfaces to other modules, interfaces to external systems, security requirements, etc.; explain any algorithms used by the module in detail
- For COTS packages, specify any call routines or bridging programs to integrate the package with the system and/or other COTS packages (for example, Dynamic Link Libraries)
- Data elements, record structures, and file structures associated with module input and output
- Graphical representation of the module processing, logic, flow of control, and algorithms, using an accepted diagramming approach (for example, structure charts, action diagrams, flowcharts, etc.)
- Data entry and data output graphics; define or reference associated data elements; if the project is large and complex or if the detailed module designs will be incorporated into a separate document, then it may be appropriate to repeat the screen information in this section
- Report layout

5. IMPLEMENTATION

At the implementation phase, the students begin to implement the project, showing end-to-end functionality. At the end of this phase, the following objectives must be reached:

- The implementation should show explicitly prove the developed model is operational and functional;
- All the proposed features or objectives must be implemented;
- The implementation must be robust;

This part must include the screenshot of the project with brief information. The important parts of the code may be mentioned. The codes must be at appendix.

6. RESULT & CONCLUSION

In the results area of the student's project, the student explains what happened during the process of development of the project. The student should include what s/he thought might happen and what s/he wanted to prove, as well as what actually happened. The student may use as much data as s/he can from investigations and document founded with charts or graphs whenever possible. The results section of the project should clearly explain to viewers what is learned during project and how that lined up with the project. The test result, outputs are must be included in this part.

The conclusion is where the student summarizes everything learned from the experiment and compare it to what is expected would happen. Start the conclusion by listing project and what that project was based on. Explain whether the results held up the project or disproved it, and then extrapolate on these founded to form an idea of where could take the project in the future. The student can mention changes can be made if s/he was to try your project again.

REFERENCES

APPENDIX A

Your appendix goes here if any available.

APPENDIX B

Your appendix goes here if any available.