EXCERCISE NO. 1

AIM: To prepare PROBLEM STATEMENT for any project.

REQUIREMENTS:

Hardware Interfaces

- Pentium(R) 4 CPU 2.26 GHz, 128 MB RAM
- Screen resolution of at least 800 x 600 required for proper and complete viewing of screens. Higher resolution would not be a problem.
- CD ROM Driver

Software Interfaces

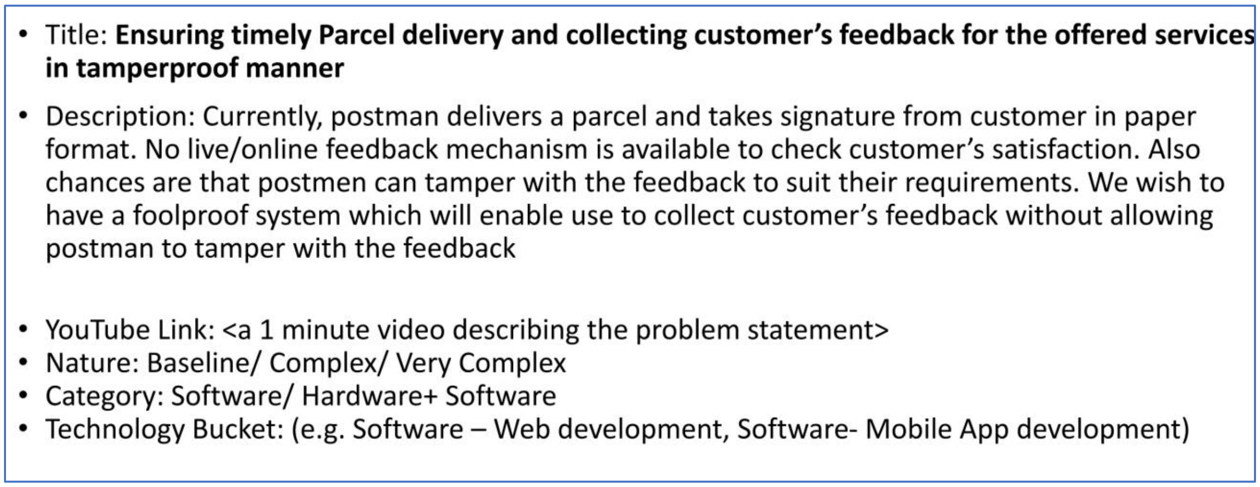
- Any window-based operating system (Windows 95/98/2000/XP/NT)
- WordPad or Microsoft Word

THEORY:

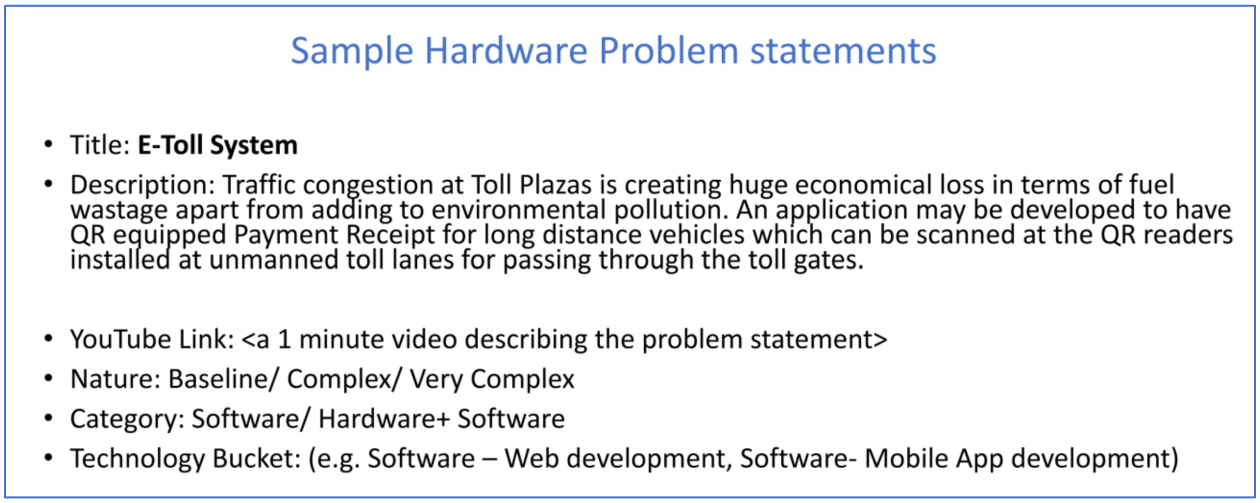
- The problem statement is the initial starting point for a project.
- It is basically a one to three page statement that everyone on the project agrees with that describes what will be done at a high level.
- The problem statement is intended for a broad audience and should be written in **non- technical terms**.
- It helps the non-technical and technical personnel communicate by providing a description of a problem. It doesn't describe the solution to the problem.
- The input to requirement engineering is the problem statement prepared by customer. It may give an overview of the existing system along with broad expectations from the new system.
- **The first phase** of requirements engineering begins with **requirements elicitation** i.e. gathering of information about requirements. Here, requirements are identified with the help of customer and existing system processes. So, from here begins the preparation of problem statement.

So, basically a problem statement describes **what** needs to be done without describing **how**.

Problem statements samples



- Title: **Ensuring timely Parcel delivery and collecting customer's feedback for the offered services in tamperproof manner**
- Description: Currently, postman delivers a parcel and takes signature from customer in paper format. No live/online feedback mechanism is available to check customer's satisfaction. Also chances are that postmen can tamper with the feedback to suit their requirements. We wish to have a foolproof system which will enable use to collect customer's feedback without allowing postman to tamper with the feedback
- YouTube Link: <a 1 minute video describing the problem statement>
- Nature: Baseline/ Complex/ Very Complex
- Category: Software/ Hardware+ Software
- Technology Bucket: (e.g. Software – Web development, Software- Mobile App development)



Sample Hardware Problem statements

- Title: **E-Toll System**
- Description: Traffic congestion at Toll Plazas is creating huge economical loss in terms of fuel wastage apart from adding to environmental pollution. An application may be developed to have QR equipped Payment Receipt for long distance vehicles which can be scanned at the QR readers installed at unmanned toll lanes for passing through the toll gates.
- YouTube Link: <a 1 minute video describing the problem statement>
- Nature: Baseline/ Complex/ Very Complex
- Category: Software/ Hardware+ Software
- Technology Bucket: (e.g. Software – Web development, Software- Mobile App development)

EXCERCISE NO. 2

Aim: Understanding an SRS.

Requirements:

Hardware Requirements:

 PC with 300 megahertz or higher processor clock speed recommended; 233 MHz minimum required.

 128 megabytes (MB) of RAM or higher recommended (64 MB minimum supported)

 1.5 gigabytes (GB) of available hard disk space

 CD ROM or DVD Drive

 Keyboard and Mouse(compatible pointing device).

Software Requirements:

Rational Rose, Windows XP

Theory:

- An SRS is basically an organization's understanding (in writing) of a customer or potential client's system requirements and dependencies at a particular point in time (usually) **prior to** any actual design or development work.
- It's a two-way insurance policy that assures that both the client and the organization understand the other's requirements from that perspective at a given point in time.
- The SRS document itself states in precise and explicit language those functions and capabilities a software system (i.e., a software application, an eCommerce Web site, and so on) must provide, as well as states any required constraints by which the system must abide.
- The SRS also functions as a blueprint for completing a project with as little cost growth as possible. The SRS is often referred to as the **"parent"** document because all subsequent project management documents, such as design specifications, statements of work, software architecture specifications, testing and validation plans, and documentation plans, are related to it.
- It's important to note that an SRS contains functional and nonfunctional requirements only; it doesn't offer design suggestions, possible solutions to technology or business issues, or any other information other than what the development team understands the customer's system requirements to be.

A well-designed, well-written SRS accomplishes four major goals:

- It provides feedback to the customer. An SRS is the customer's assurance that the development organization understands the issues or problems to be solved and the software behavior necessary to address those problems. Therefore, the SRS should be written in natural language (versus a formal language, explained later in this article), in an unambiguous manner that may also include charts, tables, data flow diagrams, decision tables, and so on.
- It decomposes the problem into component parts. The simple act of writing down software requirements in a well-designed format organizes information, places borders around the problem, solidifies ideas, and helps break down the problem into its component parts in an orderly fashion.
- It serves as an input to the design specification. As mentioned previously, the SRS serves as the parent document to subsequent documents, such as the software design specification and statement of work. Therefore, the SRS must contain sufficient detail in the functional system requirements so that a design solution can be devised.
- It serves as a product validation check. The SRS also serves as the parent document for testing and validation strategies that will be applied to the requirements for verification.

SRSs are typically developed during the first stages of "Requirements Development," which is the initial product development phase in which information is gathered about what requirements are needed and not. This information-gathering stage can include onsite visits, questionnaires, surveys, interviews, and perhaps a return-on-investment (ROI) analysis or needs analysis of the customer or client's current business environment. The actual specification, then, is written after the requirements have been gathered and analyzed.

SRS should address the following

The basic issues that the SRS shall address are the following:

a) **Functionality**. What is the software supposed to do?

b) **External interfaces**. How does the software interact with people, the system‘s hardware, other hardware, and other software?

c) **Performance**. What is the speed, availability, response time, recovery time of various software functions, etc.?

d) **Attributes**. What are the portability, correctness, maintainability, security, etc. considerations?

e) **Design constraints imposed on an implementation**. Are there any required standards in effect, implementation language, policies for database integrity, resource limits, operating environment(s) etc.?

Chracteristics of a good SRS

An SRS should be

a) Correct

b) Unambiguous

c) Complete

d) Consistent

e) Ranked for importance and/or stability

f) Verifiable

g) Modifiable

h) Traceable

Correct - This is like motherhood and apple pie. Of course, you want the specification to be correct. No one writes a specification that they know is incorrect. We like to say - "Correct and Ever Correcting." The discipline is keeping the specification up to date when you find things that are not correct.

Unambiguous - An SRS is unambiguous if, and only if, every requirement stated therein has only one interpretation. Again, easier said than done. Spending time on this area prior to releasing the SRS can be a waste of time. But as you find ambiguities - fix them.

Complete - A simple judge of this is that is should be all that is needed by the software designers to create the software.

Consistent - The SRS should be consistent within itself and consistent to its reference documents. If you call an input "Start and Stop" in one place, don't call it "Start/Stop" in another.

Ranked for Importance - Very often a new system has requirements that are really marketing wish lists. Some may not be achievable. It is useful provide this information in the SRS.

Verifiable - Don't put in requirements like - "It should provide the user a fast response." Another of my favorites is - "The system should never crash." Instead, provide a quantitative requirement like: "Every key stroke should provide a user response within 100 milliseconds."

Modifiable - Having the same requirement in more than one place may not be wrong - but tends to make the document not maintainable.

Traceable - Often, this is not important in a non-politicized environment. However, in most organizations, it is sometimes useful to connect the requirements in the SRS to a higher level document. Why do we need this requirement.

A sample of basic SRS Outline

1. Introduction

1.1 Purpose
1.2 Document conventions
1.3 Intended audience
1.4 Additional information
1.5 Contact information/SRS team members
1.6 References

2. Overall Description

2.1 Product perspective
2.2 Product functions
2.3 User classes and characteristics
2.4 Operating environment
2.5 User environment
2.6 Design/implementation constraints
2.7 Assumptions and dependencies

3. External Interface Requirements

3.1 User interfaces
3.2 Hardware interfaces
3.3 Software interfaces
3.4 Communication protocols and interfaces

4. System Features

4.1 System feature A
4.1.1 Description and priority
4.1.2 Action/result
4.1.3 Functional requirements
4.2 System feature B

5. Other Nonfunctional Requirements

5.1 Performance requirements
5.2 Safety requirements
5.3 Security requirements
5.4 Software quality attributes
5.5 Project documentation
5.6 User documentation

6. Other Requirements

Appendix A: Terminology/Glossary/Definitions list Appendix B: To be determined