SOFTWARE REQUIREMENTS ENGINEERING

LECTURE # 1

INTRODUCTION

Engr. Ali Javed

Instructor Information

- Course Instructor: Engr. Ali Javed

  Assistant Professor

  Department of Software Engineering

  U.E.T Taxila

  - ✓ Email: ali.javed@uettaxila.edu.pk
  - ✓ Website: http://web.uettaxila.edu.pk/uet/UETsub/perSites/mySite.asp?frmEmail=ali.javed@uettaxila.edu.pk
  - ✓ Contact No: +92-51-9047747
  - ✓ Office hours:
    - **Monday, 09:00 - 11:00, Office # 7 S.E.D**

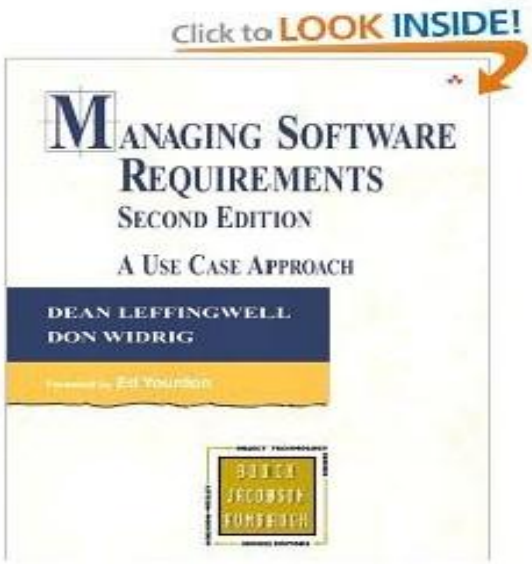
- Lab Instructor: Engr. Asra, Engr. Sobia

Course Information

- **Course Name: Software Requirements Engineering**
- **Course Code: SE-203**
- **Prerequisite Course: Introduction to Software Engineering**
- **CMS Link:** http://web.uettaxila.edu.pk/CMS/SP2013/seSREbs/

Reference Books

- Managing Software Requirements: A Use Case Approach, Second Edition By Dean Leffingwell, Don Widrig, Addison-Wesley



- Software Engineering 7th Edition By Roger Pressman

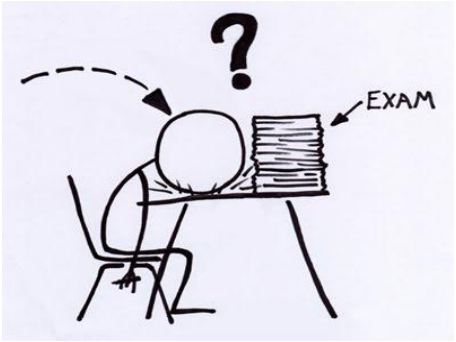
- Software Requirements, Second Edition by Karl E. Wiegers ISBN:0735618798, Microsoft Press

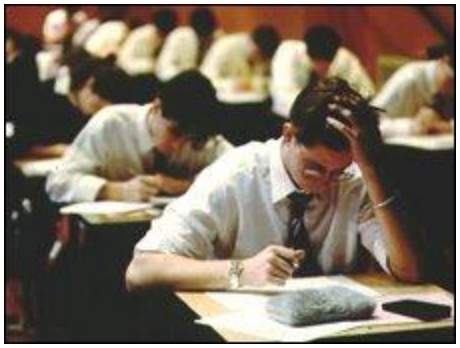
Grading Criteria

- Grading
  - ✓ Mid Term - 20%
  - ✓ End Term - 40%
  - ✓ Quiz - 10 %
  - ✓ Assignment - 10 %
  - ✓ Lab Sessions - 20 %

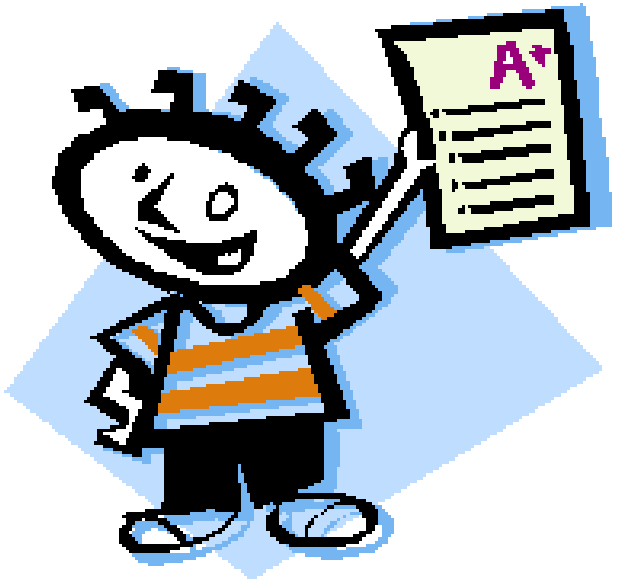
Quizzes and Assignments

- Quiz- 4 quizzes
  - 2 before mid term
  - 2 after mid term





- Assignment- 4 assignments
  - 2 before mid term
  - 2 after mid term



Presentation Outline

- What is Requirement?
- Software Requirements
- Sources of Requirements
- The Goal of Software Development
- Stakeholder's Environment
- Customer's Types
- Root Causes of Project Success and Failures
- The Frequency of Requirement Errors
- The High Cost of Requirement Errors
- Requirements Management
- Types of Software Applications
- Types of Software Requirements
- The Problem Domain
- The Solution Domain

Requirement

- Something required, something wanted or needed
  - Webster's dictionary

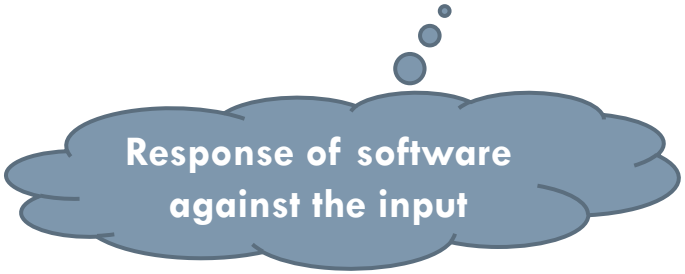
- There is a huge difference between *wanted* and *needed* and it should be kept in mind all the time

  - ✓ **Need**- something you have to have
  - ✓ **Want**- something you would like to have

Software Requirements

- A complete description of *what* the software system will do without describing *how* it will do it is represented by the software requirements

- Software requirements are complete specification of the desired external behavior of the software system to be built



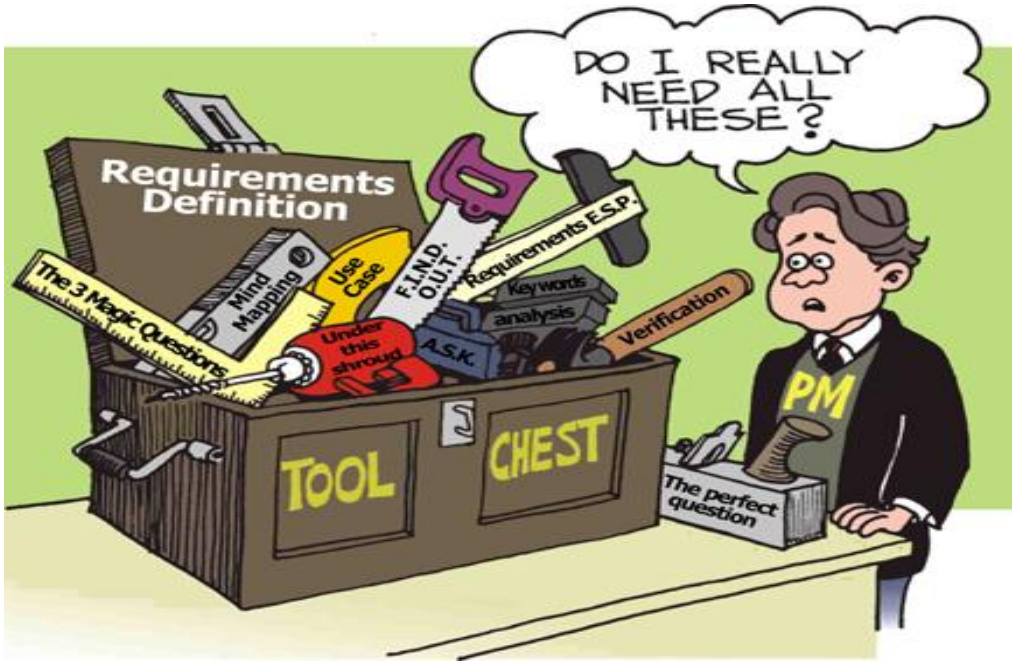
- Software requirements may be:
  - ✓ Abstract statements of services
  - ✓ Detailed mathematical functions
  - ✓ Part of the bid of contract
  - ✓ The contract itself
  - ✓ Part of the technical document, which describes a product

Requirement [7]

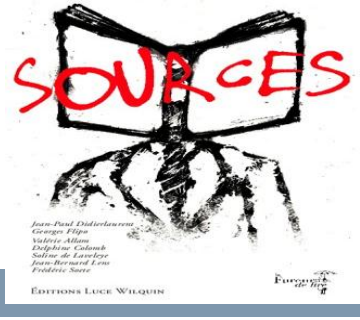
Can be constraint

functionality

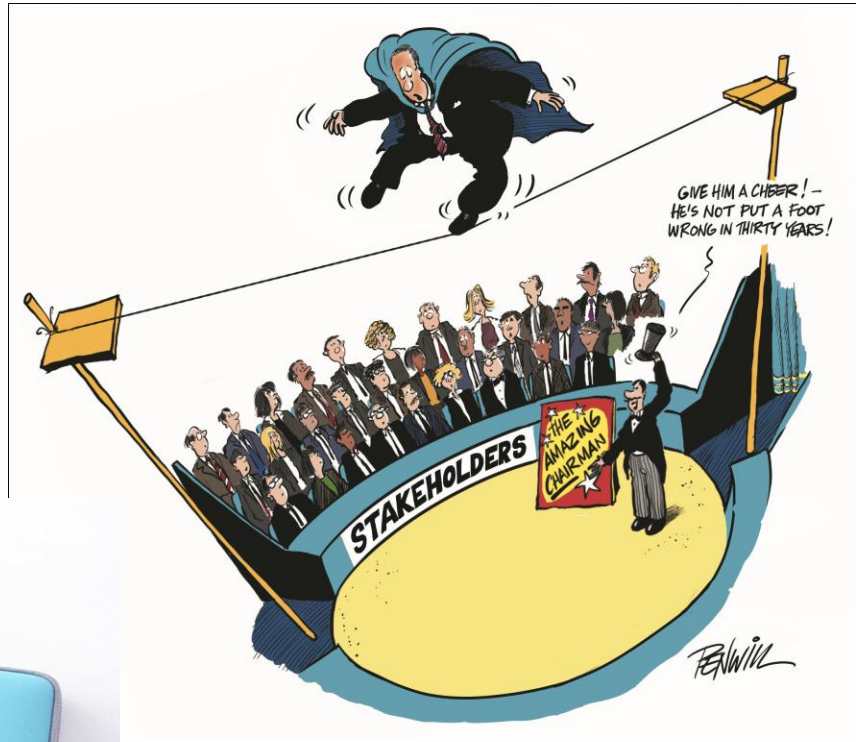
- A condition or capability that must be met or possessed by a system...to satisfy a contract, standard, specification, or other formally imposed document
  - IEEE Std 729



Sources of Requirement



- Stakeholders
  - ✓ People affected in some way by the system
- Documents



Sources of Requirement

- Existing system

- Application Domain

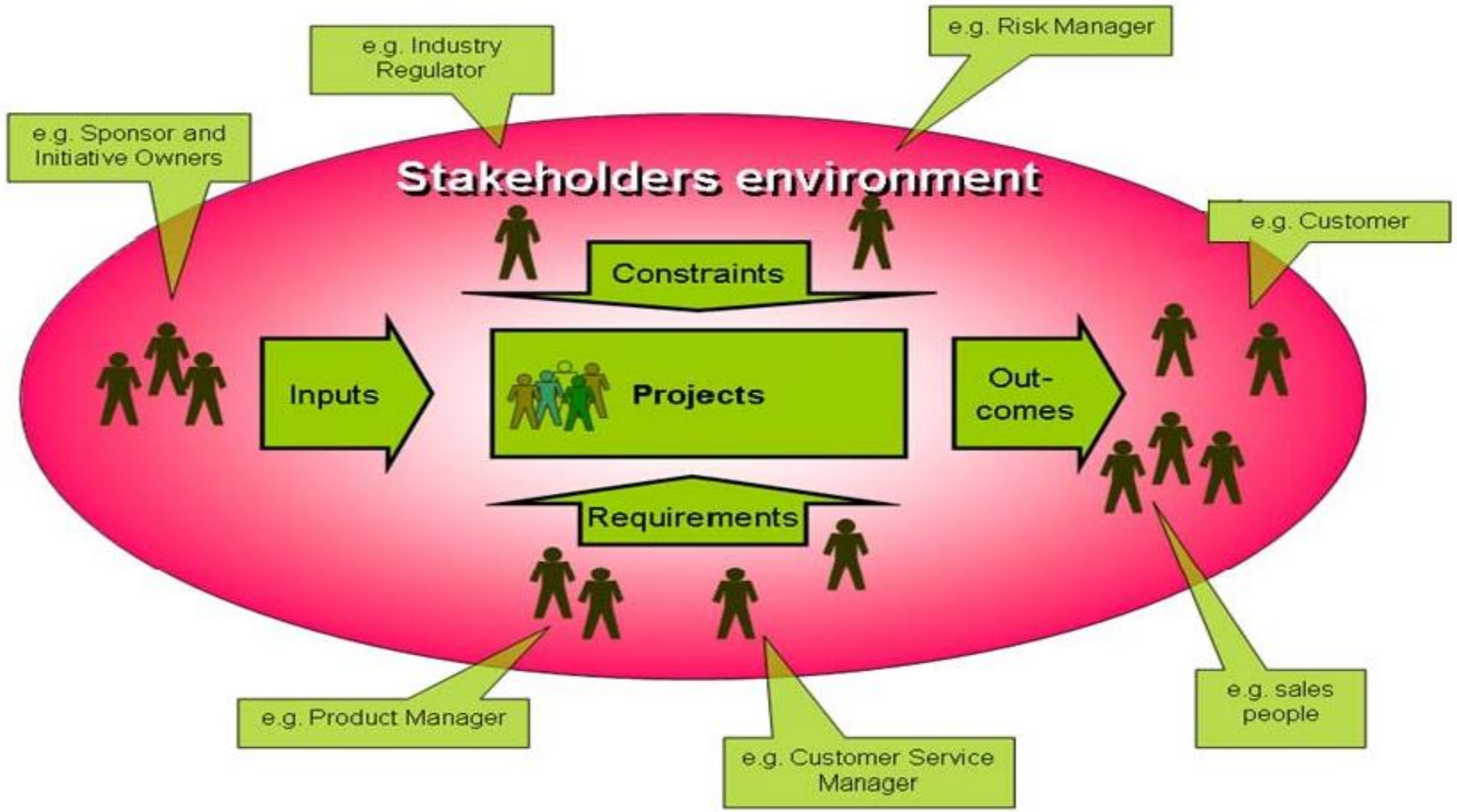
The Goal of Software Development [1]

successfully project=Quality,DOT+Requirement conformance

- The goal of software development is to develop quality software—on time and on budget—that meets customers' real needs.

- A software is good if it MEETS STAKEHOLDERS EXPECTATIONS:
  - ✓ it is (at least) correct, reliable, maintainable, user-friendly …
  - ✓ the total cost it incurs over all phases of its life cycle is minimal and within the budget

Context: Stakeholder's Environment [6]



Importance of Software Requirements[9]

The hardest single part of building a software system is deciding what to build….. No other part of the work so cripples the resulting system if done wrong. No other part is difficult to rectify later. (Fred Brooks)

The Root Causes of Project Success and Failure[1]

- The first step in resolving any problem is to understand the root causes.
- The 1994 Standish Group survey study noted the three most commonly cited factors that caused projects to be "challenged":

  - ✓ **Lack of user input: 13 percent of all projects**
  - ✓ **Incomplete requirements and specifications: 12 percent of all projects**
  - ✓ **Changing requirements and specifications: 12 percent of all projects**

”I am so busy!..

..to waste time with requirements”

The Root Causes of Project Success and Failure[1]

- Thereafter, the data diverges rapidly. Of course, your project could fail
  - ✓ because of an unrealistic schedule or time frame (4 percent of the projects cited this),
  - ✓ inadequate staffing and resources (6 percent),
  - ✓ inadequate technology skills (7 percent), or various other reasons.

- The survey shows that at least a third of development projects run into trouble for reasons that are directly related to requirements gathering, requirements documentation, and requirements management.

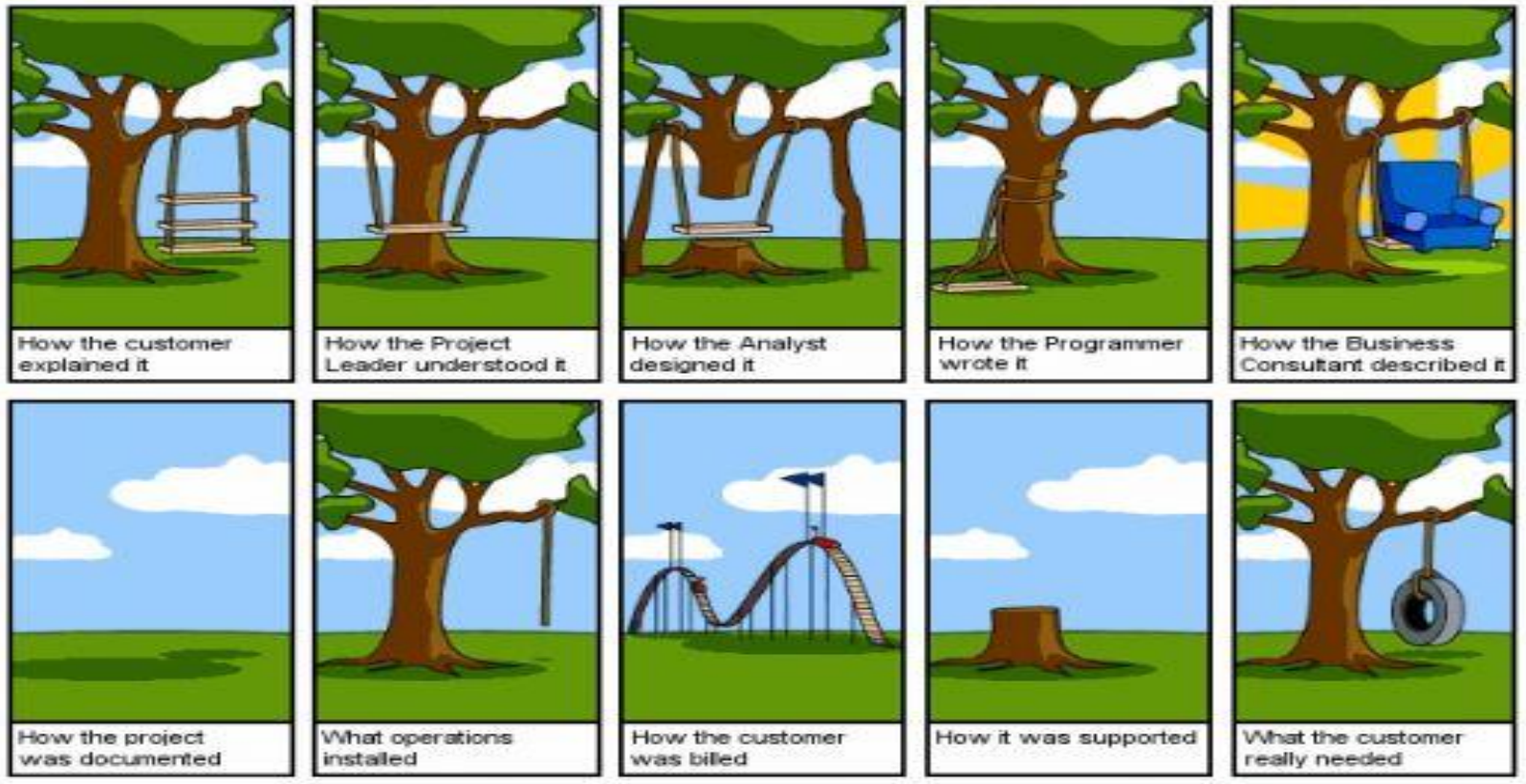
The Root Causes of Project Success and Failure[1]

- Although the majority of projects do seem to experience schedule/budget overruns, the Standish Group found that 9 percent of the projects in large companies were delivered on time and on budget; 16 percent of the projects in small companies enjoyed a similar success. That leads to an obvious question: **What were the primary "success factors" for those projects?**

- According to the Standish study, the three most important factors were
  - ✓ **User involvement: 16 percent of all successful projects**

The Root Causes of Project Success and Failure[1]

- ✓ **Executive management support: 14 percent of all successful projects**
- ✓ **Clear statement of requirements: 12 percent of all successful projects**



Frequency of Requirement Errors [1]

- Table 1. summarizes a 1994 study by Capers Jones that provides data regarding the likely number of "potential" defects in a development project and the typical "efficiency" with which a development organization removes those defects through various combinations of testing, inspections, and other strategies.

Table 1. Defect Summary

DefectOrigins	Defect Potentials	Removal Efficiency	Delivered Defects
Requirements	**1.00**	**77%**	**0.23**
Design	1.25	85%	0.19
Coding	1.75	95%	0.09
Documentation	0.60	80%	0.12
Bad fixes	0.40	70%	0.12
Total	5.00	85%	0.75

Requirements errors top the delivered defects and contribute approximately one third of the total delivered defects to the defect pile

High Cost of Requirement Errors [1]

- If a unit cost of one is assigned to the effort required to detect and repair an error during the coding stage ....

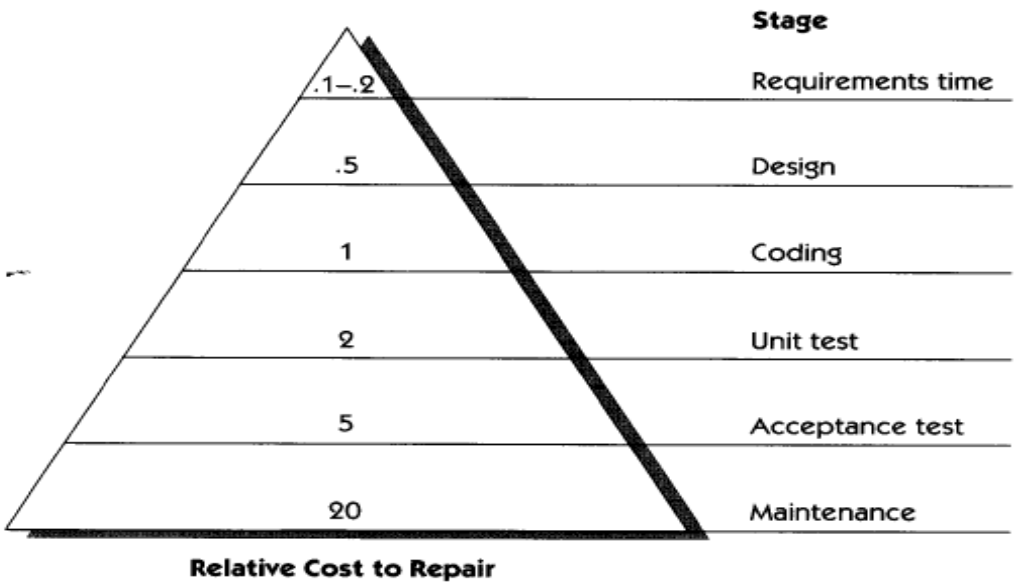


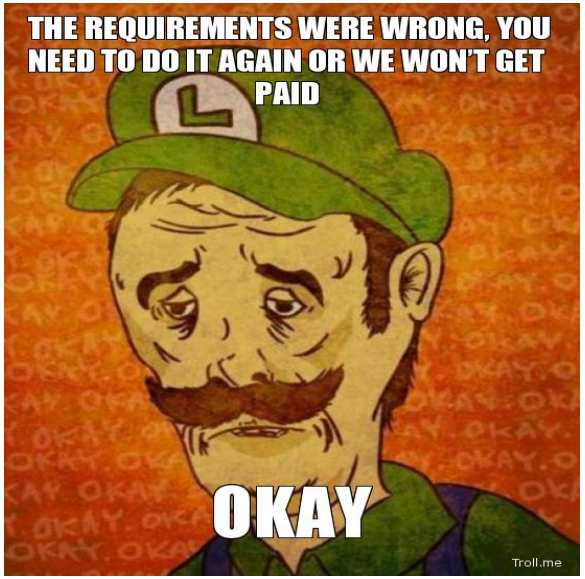
Figure 1–2 Relative cost to repair a defect at different lifecycle phases. (Data derived from Davis [1993].)

High Cost of Requirement Errors [1]

- The errors discovered during the design of a development project could fall into one of two categories:
  - ✓ errors that occurred when the development staff created a technical design from a correct set of requirements or
  - ✓ errors that should have been detected as requirements errors somewhat earlier in the process but that somehow "leaked" into the design phase of the project.

High Cost of Requirement Errors [1]

- It's the latter category of errors that turn out to be particularly expensive, for two reasons.
  - ✓ By the time the requirements-oriented error is discovered, the development group will have invested time and effort in building a design from those erroneous requirements. As a result, the design will probably have to be thrown away or reworked.
  - ✓ The true nature of the error may be disguised; everyone assumes that they're looking for design errors during the testing or inspection activities that take place during this phase, and considerable time and effort may be wasted until someone says, "Wait a minute! This isn't a design mistake after all; we've got the wrong requirements."



High Cost of Requirement Errors- Defect Leakage [1]

- One organization that has done the research on defect leakage is Hughes Aircraft. A study by Snyder and Shumate [1992] follows the leakage phenomenon for a large collection of projects Hughes has conducted over the past 15 years.

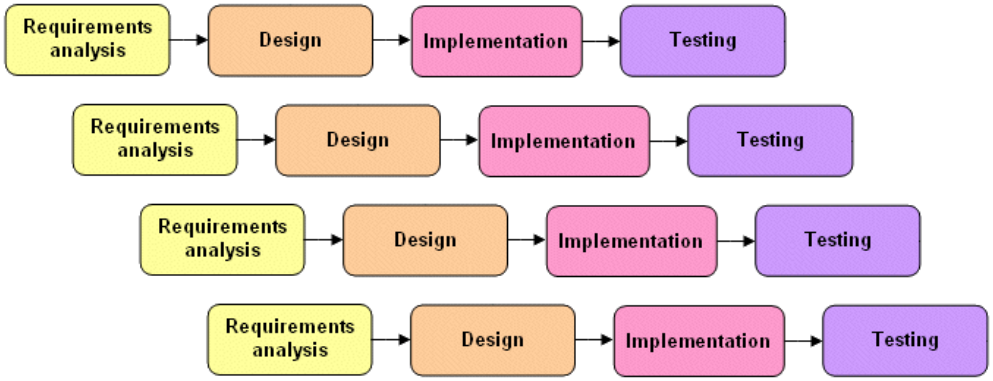
- The study indicates that

  - ✓ Defects discovered at Requirements Analysis phase-- 74 percent of the requirements-oriented defects
  - ✓ Defects discovered at preliminary, or high-level, design-- 4 percent of the requirements defects "leak" and that 7 percent leak further into detailed design.
  - ✓ The leakage continues throughout the lifecycle, and a total of 4 percent of the requirements errors aren't found until maintenance, when the system has been released to the customers and is likely in full-scale operation.

High Cost of Requirement Errors- Defect Leakage [1]

- whenever a defect is discovered in a software application, we're likely to experience the effect of 50–100 times cost.

  - ✓ Re-specification.
  - ✓ Redesign.
  - ✓ Recoding.
  - ✓ Retesting.
  - ✓ Change orders
  - ✓ Corrective action

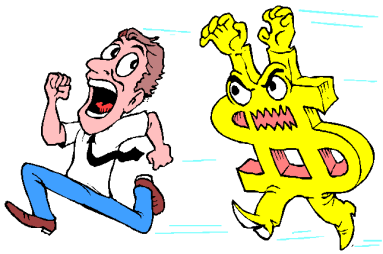


High Cost of Requirement Errors- Defect Leakage [1]

- Scrap

- Recall of defective versions ot sottware and associated manuals from users.

- Warranty costs.

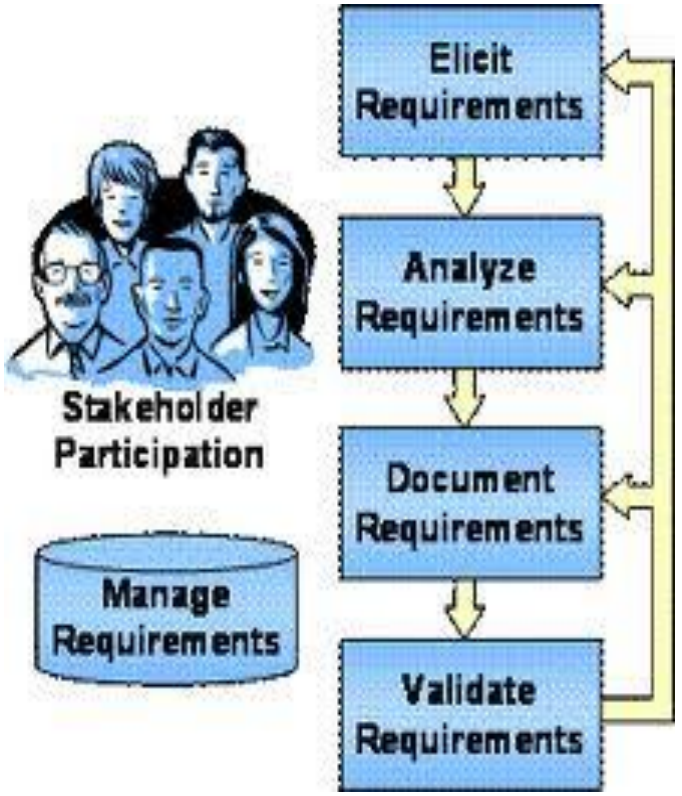


- Product liability

- Service costs for a company representative to visit a customer's field location to reinstall the new software.
- Documentation

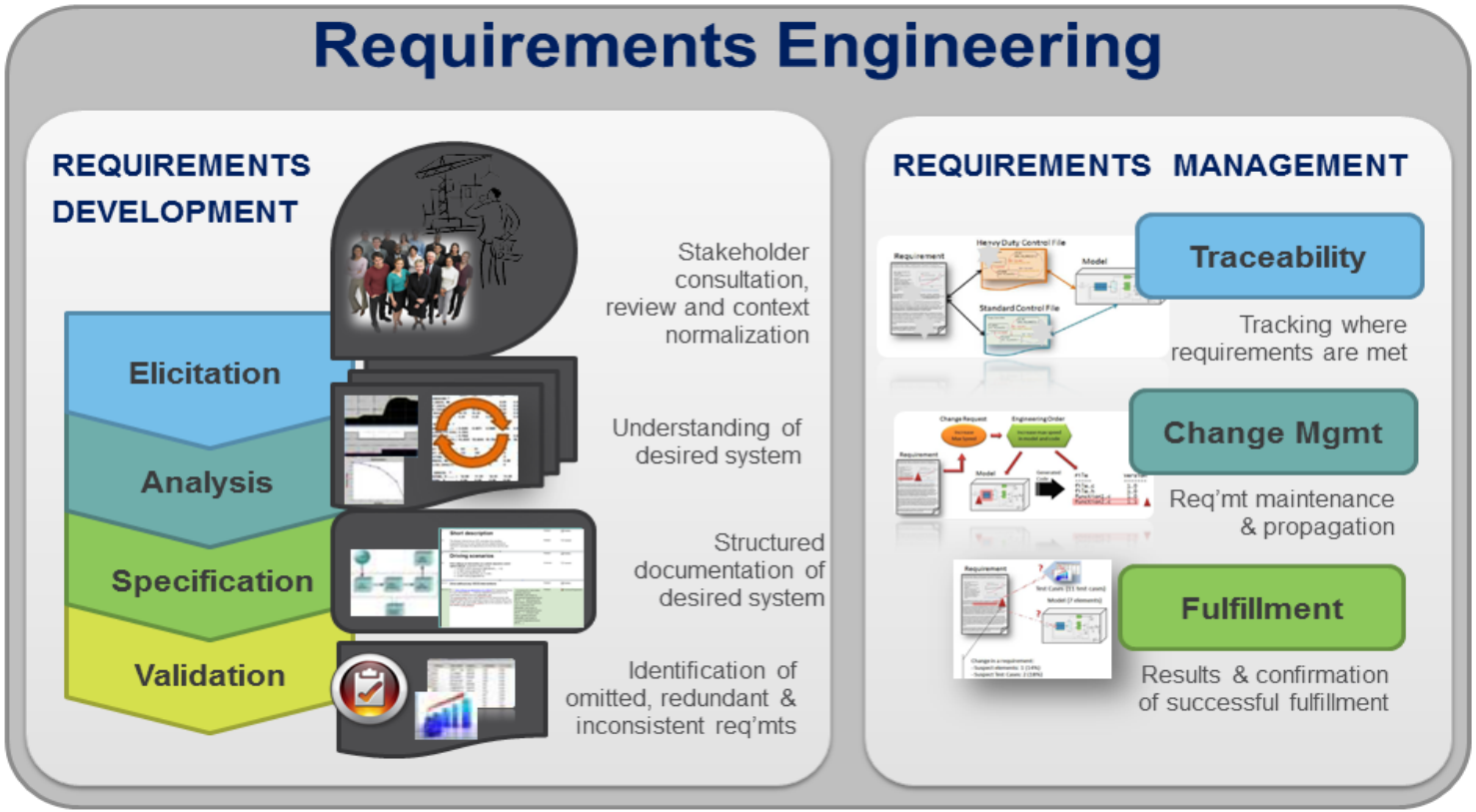
Requirements Engineering Process

- **Elicitation:** work with the customer on gathering requirements
- **Analysis:** process this information to understand it, classify in various categories, and relate the customer needs to possible software requirements
- **Specification:** Structure the customer input and derived requirements as written documents and diagrams
- **Validation:** you'll ask your customer to confirm that what you've written is accurate and complete and to correct errors.



Requirements Management [10]

- **Requirements management** is the process of documenting, analyzing, tracing, prioritizing and agreeing on requirements and then controlling change and communicating to relevant stakeholders.



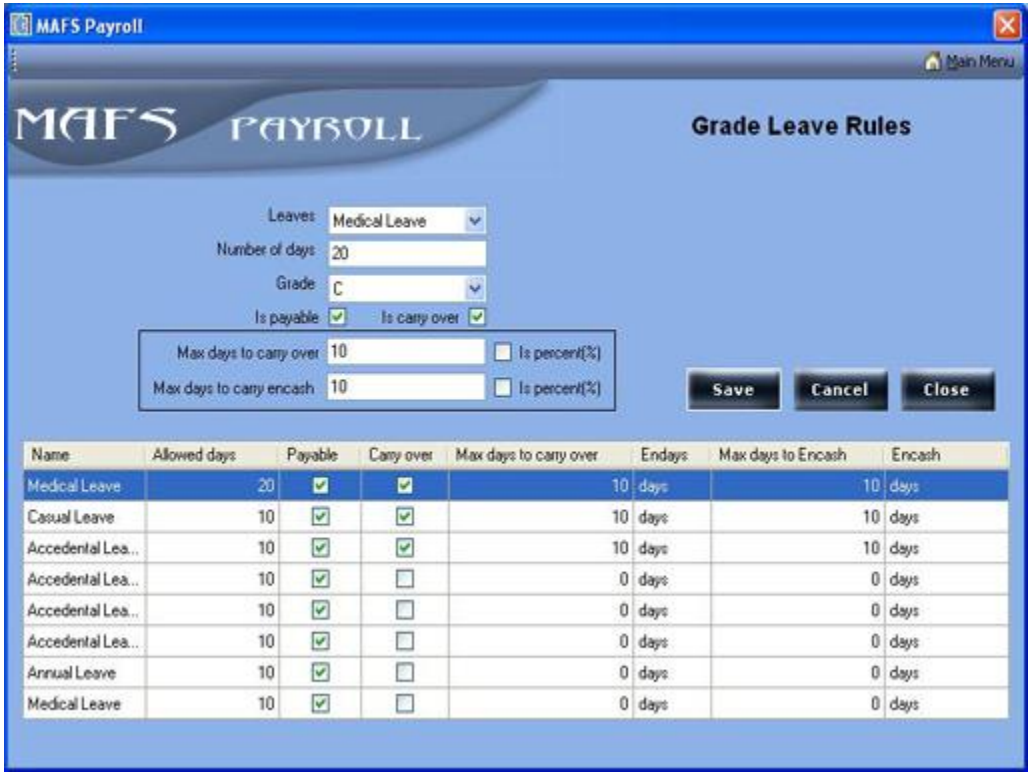
Types of Software Apps

- Information Systems
- Commercial Software Apps
- Embedded Systems

Types of Software Applications [1]

- **Information Systems**

  - ✓ Information systems and other applications developed for use within a company (such as the payroll system being used to calculate the take-home pay for our next paycheck). This category is the basis for the information system/information technology industry, or IS/IT.



Types of Software Applications [1]

- **Commercial Software Applications**
  - ✓ Software developed and sold as commercial products (such as the MS Office). Companies developing this type of software are often referred to as independent software vendors, or ISVs.

Types of Software Applications [1]

- **Embedded Software's**

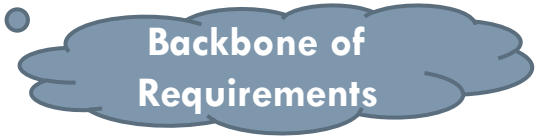
  - ✓ Software that runs on computers embedded in other devices, machines, or complex systems (such as those contained in the airplane, in cell phones, in washing machines, in ovens, the automobile and many more…….) This type of software's are known as software embedded-systems applications, or embedded applications.

Types of Software Requirements

- Functional requirements
- Non-functional requirements
- Domain requirements
- Inverse requirements
- Design and implementation constraints

Functional Requirements [2]

- In software engineering, a **functional requirement** defines a function of a software system or its component.



- A function is described as a set of inputs, the behavior, and outputs.
- Functional requirements may be calculations, technical details, data manipulation and processing and other specific functionality that define *what* a system is supposed to accomplish.
- Functional requirements are the statements and services that system should provide in two clearly described external behaviors
  - ✓ Reaction to particular input (e.g give some input to software and it produces a result)
  - ✓ Behavior in particular situations (e.g If I click on an exit button then system behaves in a particular way)

Functional Requirements [2]

- Functional Requirements can be stated from either static or dynamic perspective
  - ✓ The dynamic perspective describes the behavior of the system in terms of results
  - ✓ The static perspective describes the functions performed by each entity and the way each interacts with other entities and the environment
- Abnormal behavior is also documented as functional requirements in the form of exception handling
- Functional requirements should be complete and consistent
- Customers and developers usually focus all their attention on functional requirements

Non-Functional Requirements [3]

- In requirements engineering, a **non-functional requirement** is a requirement that specifies criteria that can be used to judge the operation of a system, rather than specific behaviors.

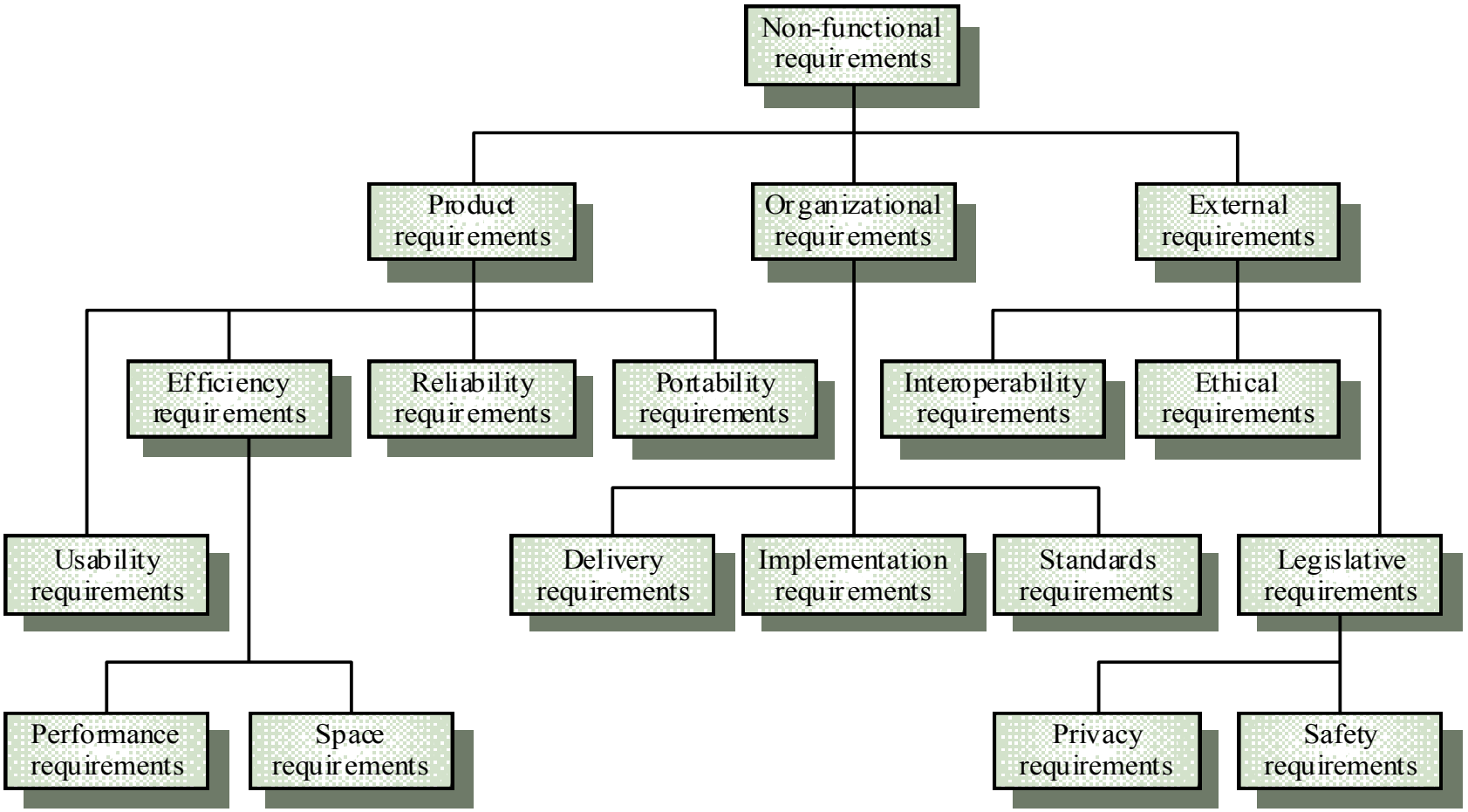
- Non-functional requirements are often called **qualities** of a system. Other terms for non-functional requirements are "constraints", "quality attributes", "quality goals", "quality of service requirements" and "non-behavioral requirements".

- Informally these are sometimes called the "ilities", from attributes like stability and portability.

Non-Functional Requirements

Types	Explanation
Product requirements:	specify that the delivered product *must behave* in a particular way e.g. *execution speed, reliability, etc.*
Organizational requirements:	are a consequence of *organizational policies* and procedures e.g. *process standards used, implementation requirements, etc.*
External requirements:	arise from *factors which are external* to the system and its development process e.g. *interoperability requirements, legislative requirements, etc.*

Non-Functional Requirements



Domain Requirements [8]

- Requirements that come from the application domain and reflect fundamental characteristics of that application domain
- These can be both the functional or non-functional requirements
- These requirements, sometimes, are not explicitly mentioned
- Their absence can cause significant dissatisfaction
- Domain requirements can impose strict constraints on solutions
- Domain-specific terminology can also cause confusion
  - ✓ Example:

    In a commission-based sales businesses, there is no concept of negative commission. However, if care is not taken novice developers can be tempted into developing systems, which calculate negative commission

Inverse Requirements [5]

- They explain what the system shall **not** do.
- Inverse requirements describe the constraints on the allowable behaviors
- Many people find it convenient to describe their needs in this manner
- These requirements indicate the indecisive nature of customers about certain aspects of a new software product
- Example:

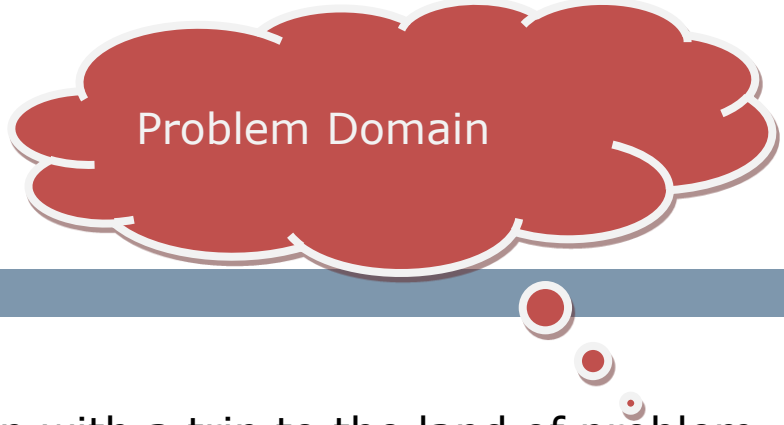
> **The system shall not use red color in the user interface, whenever it is asking for inputs from the end-user**

- Safety and security requirements are usually stated in this manner

Design and Implementation Constraints [5]

- They are development guidelines within which the designer must work
- These requirements can seriously limit design and implementation options
- Can also have impact on human resources
- Example:
  - ✓ The system shall be developed using the **Microsoft .Net platform**
  - ✓ The system shall be developed using **open source tools[4]** and shall run on **Linux operating system**
  - ✓ The System should be implemented using **c sharp language**
  - ✓ The software must fit into the memory of a 512Kbyte machine.

Problem Domain [1]



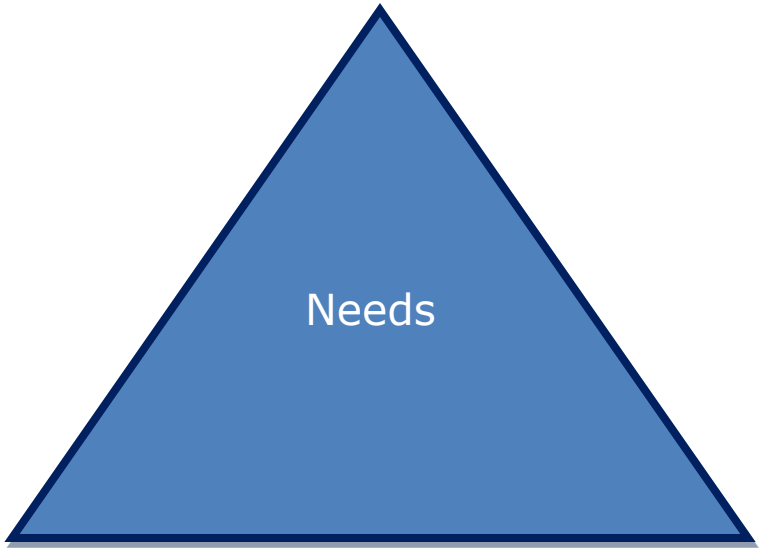
- Most successful requirements journeys begin with a trip to the land of problem.
- This problem domain is the home of real users and other stakeholders, people whose needs must be addressed in order for us to build the perfect system.
- This is the home of the people who need the rock or a new sales order entry system or a configuration management system good enough to blow the competition away.
- In all probability, these people are not like us. Their technical and economic backgrounds are different from ours.
- On rare occasions, they are just like us. They are programmers looking for a new tool or system developers who have asked you to develop a portion of the system.

Problem Domain [1]

- These users have business or technical problems that they need our help to solve.
- Therefore, it becomes our problem to understand their problems, in their culture and their language, and to build systems that meet their needs.
- Within the problem domain, we use a set of team skills to gain an understanding of the problem and the needs that must be filled to address this problem.

Stakeholder Needs [1]

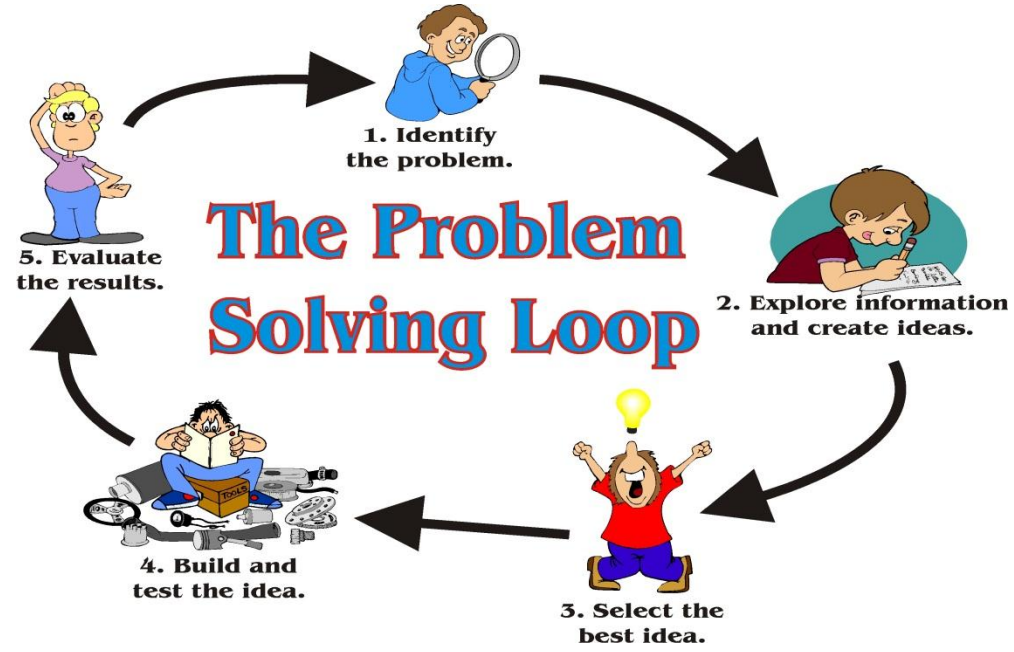
- Stakeholders are people who have a stake or interest in the project
- It is also our responsibility to understand the needs of users and other stakeholders whose lives will be affected by our solution.
- As we elicit those needs, we'll stack them in a little pile called stakeholder needs, which we represent as a pyramid.



Moving towards the solution domain [1]

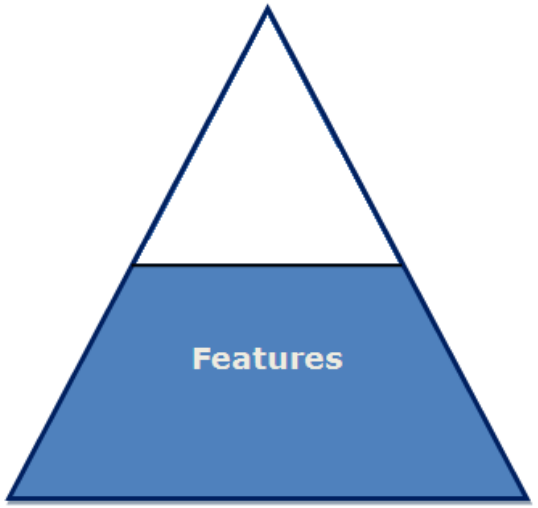
- We move to Solution domain from the problem domain in order to provide a solution to the problem at hand.

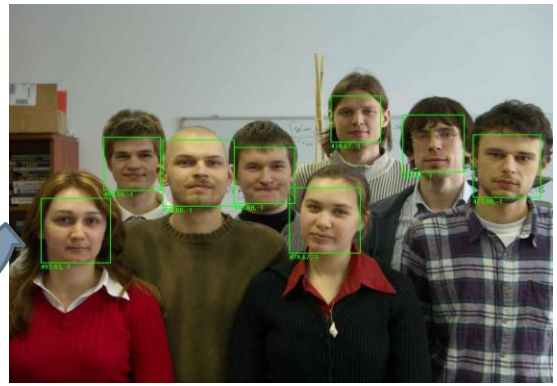
- In the solution space, we focus on defining a solution to the user's problem; this is the realm of computers, programming, operating systems, networks, and processing nodes. Here, we can apply the skills we have learned much more directly.



Moving towards the solution domain [1]

- **Features of the System**
  - ✓ a service provided by the system that fulfills one or more stakeholder needs.
- This list could consists of such items as the following.
  - ✓ "The car will have power windows."
  - ✓ "The cell provides pattern to lock screen"
  - ✓ " The Attendance of students will be enrolled recognition ."
  - ✓ These are simple descriptions, in the user's language, that we will use as labels to communicate with the user how our system addresses the problem.

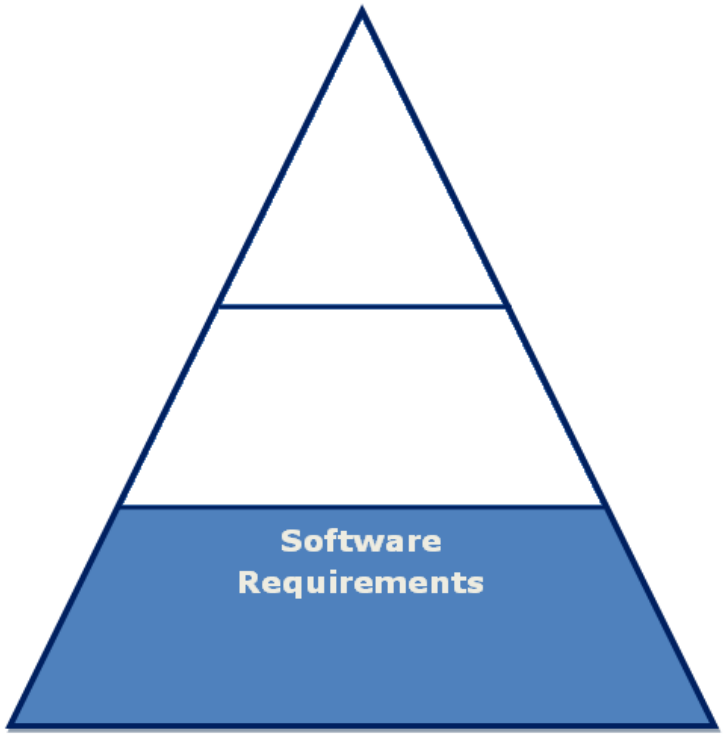




Moving towards the solution domain [1]

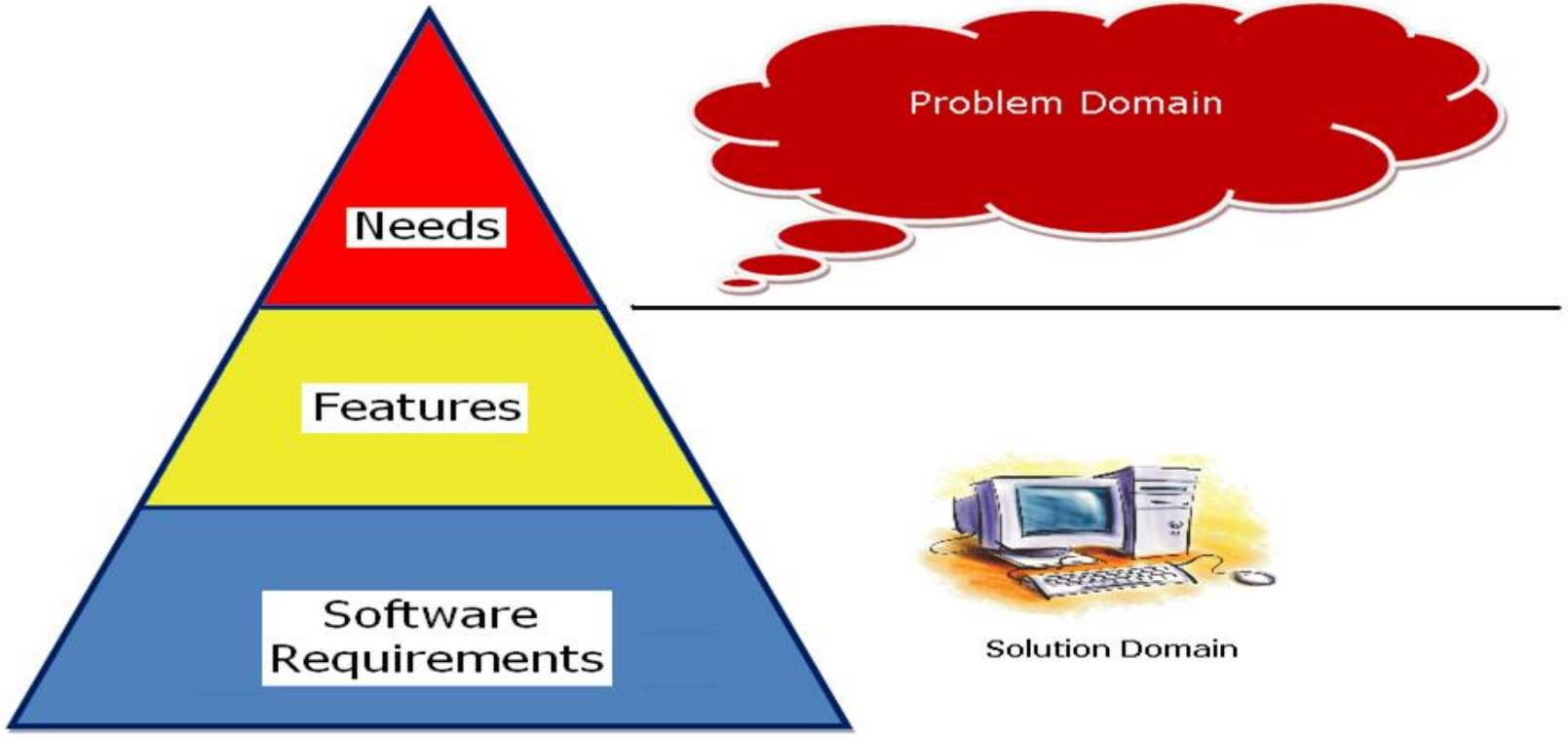
Software Requirements

- Once we have established the feature set and have gained agreement with the customer, we can move on to defining the more specific requirements we will need to impose on the solution.
- If we build a system that conforms to those requirements, we can be certain that the system we develop will deliver the features we promised. In turn, since the features address one or more stakeholder needs, we will have addressed those needs directly in the solution.
- These more specific requirements are the software requirements.



Overview of the Problem Domain and Solution Domain [1]

- We have moved from the problem domain, represented by the user needs we discovered, to a definition of a system that will constitute the solution domain, represented by the features of the system and the software requirements that will drive its design and implementation.



References

1. Managing Software Requirements: A Use Case Approach, Second Edition By Dean Leffingwell, Don Widrig, Addison- Wesley
2. http://en.wikipedia.org/wiki/Functional_requirement
3. http://en.wikipedia.org/wiki/Non-functional_requirement
4. http://www.webopedia.com/TERM/O/open_source_tools.html
5. http://www.sei.cmu.edu/reports/90cm019.pdf
6. http://finntrack.co.uk/learners/business_society.htm
7. http://en.wikipedia.org/wiki/Requirement
8. http://www.cs.st-andrews.ac.uk/~ifs/Books/SE9/Web/Requirements/DomainReq.html
9. http://faculty.salisbury.edu/~xswang/Research/Papers/SERelated/no-silver-bullet.pdf
10. http://en.wikipedia.org/wiki/Requirements_management

For any query Feel Free to ask