SOFTWARE ENGINEERING
(Week-2)

USAMA MUSHARAF

LECTURER (Department of Computer Science)

FAST-NUCES PESHAWAR

CONTENTS OF WEEK # 2

- Software Quality Engineering
  - Quality Engineering Economics
  - Quality Attributes
  - Quality Models
- Software Quality Engineering at Requirement Phase
  - Requirement Document
  - Requirement Engineering Related Activities
  - Requirement Traceability Matrix

SOFTWARE QUALITY ENGINEERING

WHAT IS SOFTWARE QUALITY ?

According to the IEEE

- Software quality is:
  1. *The degree to which a system, component, or process meets specified requirements.*
  2. *The degree to which a system, component, or process meets customer or user needs or expectations.*

WHAT IS SOFTWARE QUALITY?

- “Achieving high levels of user satisfaction, portability, maintainability, robustness, and fitness for use” by Dr. Barry Boehm.

- Quality means “conformance to user requirements” by Phil Crosby.

- Edwards Deming considers quality to be “striving for excellence” in reliability and functions by continuous improvement in the process of development, support by statistical analysis of the causes of failure.

WHAT IS SOFTWARE QUALITY?

- Watts Humphrey, of the SEI, tends to speak of quality as “achieving excellent levels of fitness for use, conformance to requirements, reliability and maintainability.”

- James Martin said that software quality means being on time, within budget and meeting user needs.

- Tom McCabe, the software complexity specialist, defines quality as “high level of user satisfaction and low defect levels, often associated with low complexity.

WHAT IS SOFTWARE QUALITY?

- John Musa of Bell Laboratories states that quality means combination of “low defect levels, adherence of software functions to users needs, and high reliability”

- Bill Perry, head of Quality Assurance Institute has defined quality as “high levels of user satisfaction and adherence to requirements”.

WHAT IS SOFTWARE QUALITY ASSURANCE?

According to the IEEE

Software quality assurance is:

1. *A planned and systematic pattern of all actions necessary to provide adequate confidence that an item or product conforms to established technical requirements.*
2. *A set of activities designed to evaluate the process by which the products are developed or manufactured. Contrast with: quality control.*

WHAT IS SOFTWARE QUALITY ASSURANCE?

According to D. Galin

Software quality assurance is:

> *“A systematic, planned set of actions necessary to provide adequate confidence that the software development process or the maintenance process of a software system product conforms to established functional technical requirements as well as with the managerial requirements of keeping the schedule and operating within the budgetary confines.”*

SOFTWARE QUALITY CHALLENGES

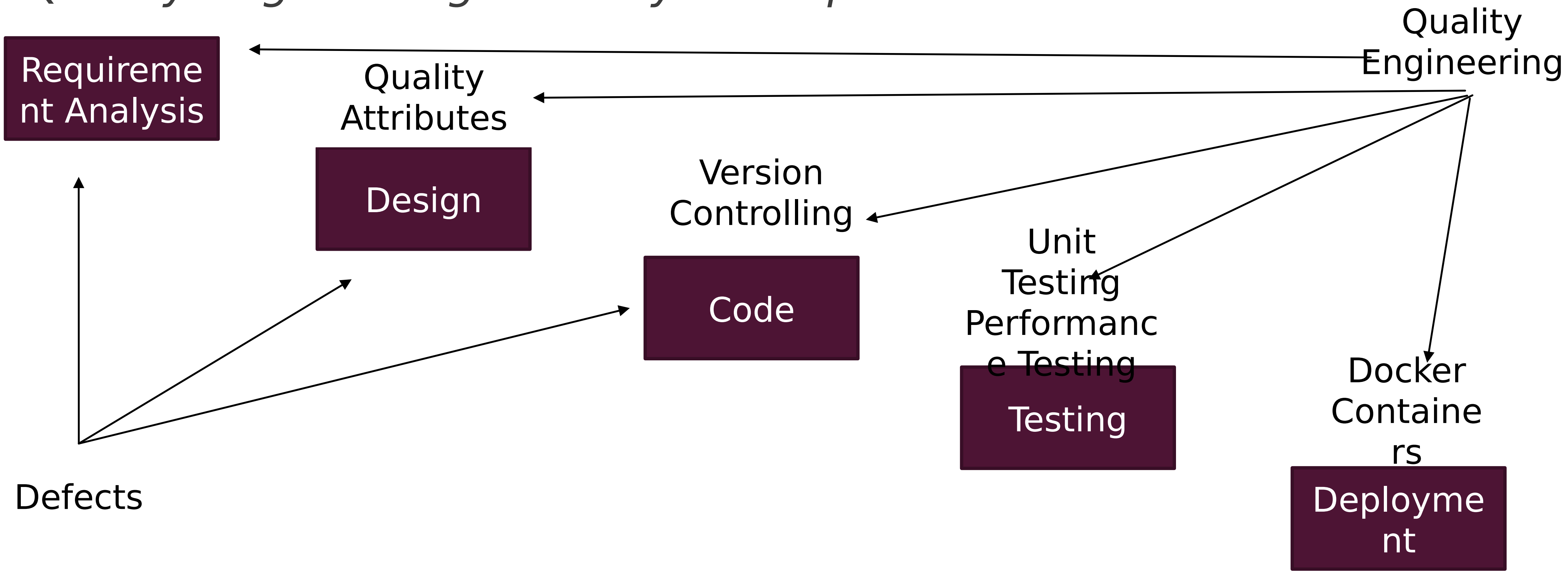
- Quality cannot be directly checked in the product; it must planned right from the beginning.

- The project must focus on the quality issues of the project from the beginning, ensuring that quality criteria are consistent with defined requirements.

- Quality means meeting requirements and meeting customer needs, which means a defect-free product from both the producer's and the customer's viewpoint.

SOFTWARE QUALITY ENGINEERING ROAD MAP

- *Quality Engineering at every SDLC phase:*



WHAT IS DEFECT?

- Defect is deviation from customer requirement.
- Mostly defects are found in the software after software is shipped to the customer at production site.

- Example:
- In online shopping, the option of searching a debit card for making payment is missing.

VERIFICATION

- Verification addresses the concern: "Are you building it right?"
- Ensures that the software system meets all the functionality.
- Verification takes place first and includes the checking for documentation, code, etc.
- Done by developers.
- It has static activities, as it includes collecting reviews, walkthroughs, and inspections to verify a software.
- It is an objective process and no subjective decision should be needed to verify a software.

VALIDATION

- Validation addresses the concern: "Are you building the right thing?"
- Ensures that the functionalities meet the intended behavior.
- Validation occurs after verification and mainly involves the checking of the overall product.
- Done by testers.
- It has dynamic activities, as it includes executing the software against the requirements.
- It is a subjective process and involves subjective decisions on how well a software works.

FUNCTION QUALITY COST (FQC)

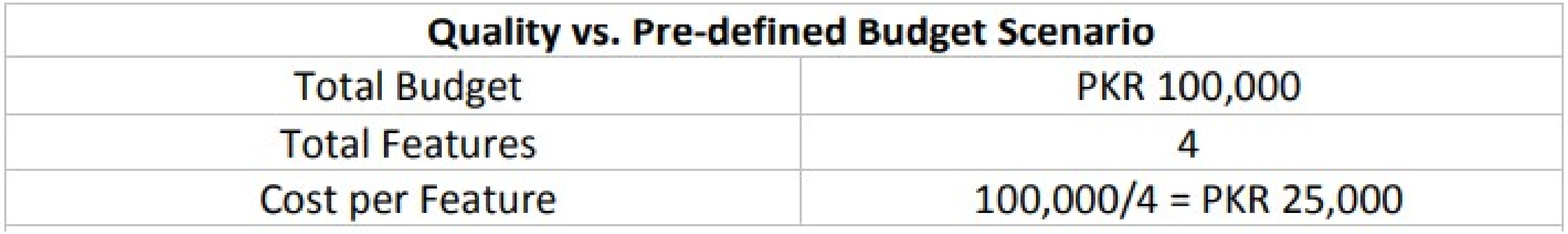
In most development projects, functionality and quality (QA precisely) are natural enemies.

Projects with open budgets are very rare, usually the budget is fixed and here the functionality and quality compete with each other in order to get a bigger share from budget.

- The Function-Quality-Cost comes out to be
  - Cost = AF + BQ
- Where A & B = Level of investment
- F = Features/Functions
- Q = Quality

FUNCTION QUALITY COST (FQC)

- It is very much clear that increasing feature in a closed-budget project will certainly decrease the budget share for quality of the product.
- The following example will elaborate the concept more clearly.



Quality vs. Pre-defined Budget Scenario	
Total Budget	PKR 100,000
Total Features	4
Cost per Feature	100,000/4 = PKR 25,000

(Dev + QA)

QUALITY ATTRIBUTES

- Quality attributes depend on each other.

- It is impossible to completely satisfy all quality attributes of a software system.

SOFTWARE QUALITY MODELS

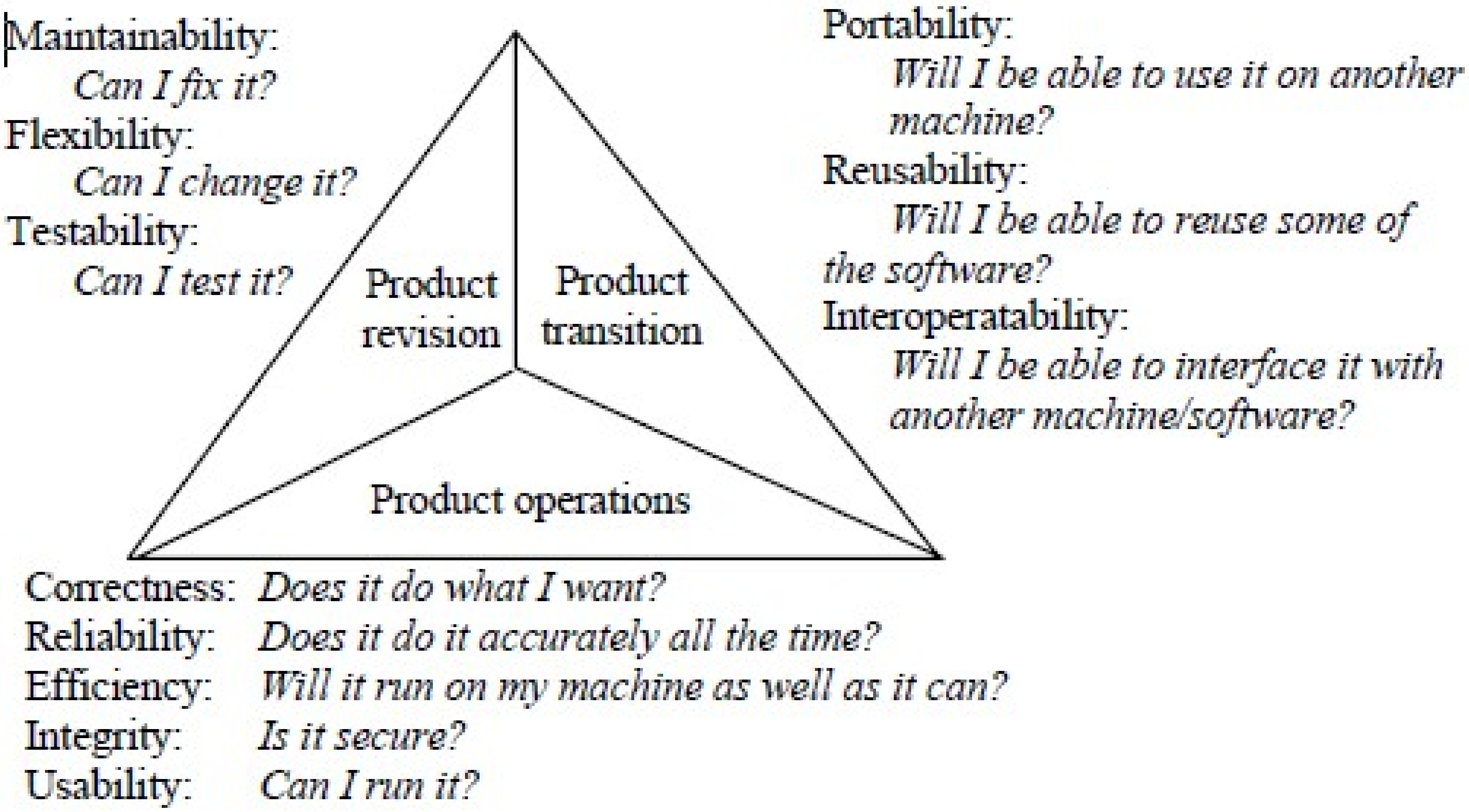
Quality is the excellence of the product or service.

- From a user’s point of view, quality is ‘fitness for purpose’.
- From the manufacturing point of view, the quality of a product is the conformance to specification.

HIERARCHICAL MODELS

McCall divided software quality attributes into 3 groups.

- Each group represents the quality with respect to one aspect of the software system while the attributes in the group contribute to that aspect.
- Each quality attribute is defined by a question so that the quality of the software system can be assessed by answering the question.



Maintainability:
Can I fix it?
Flexibility:
Can I change it?
Testability:
Can I test it?

Portability:
Will I be able to use it on another machine?
Reusability:
Will I be able to reuse some of the software?
Interoperatability:
Will I be able to interface it with another machine/software?

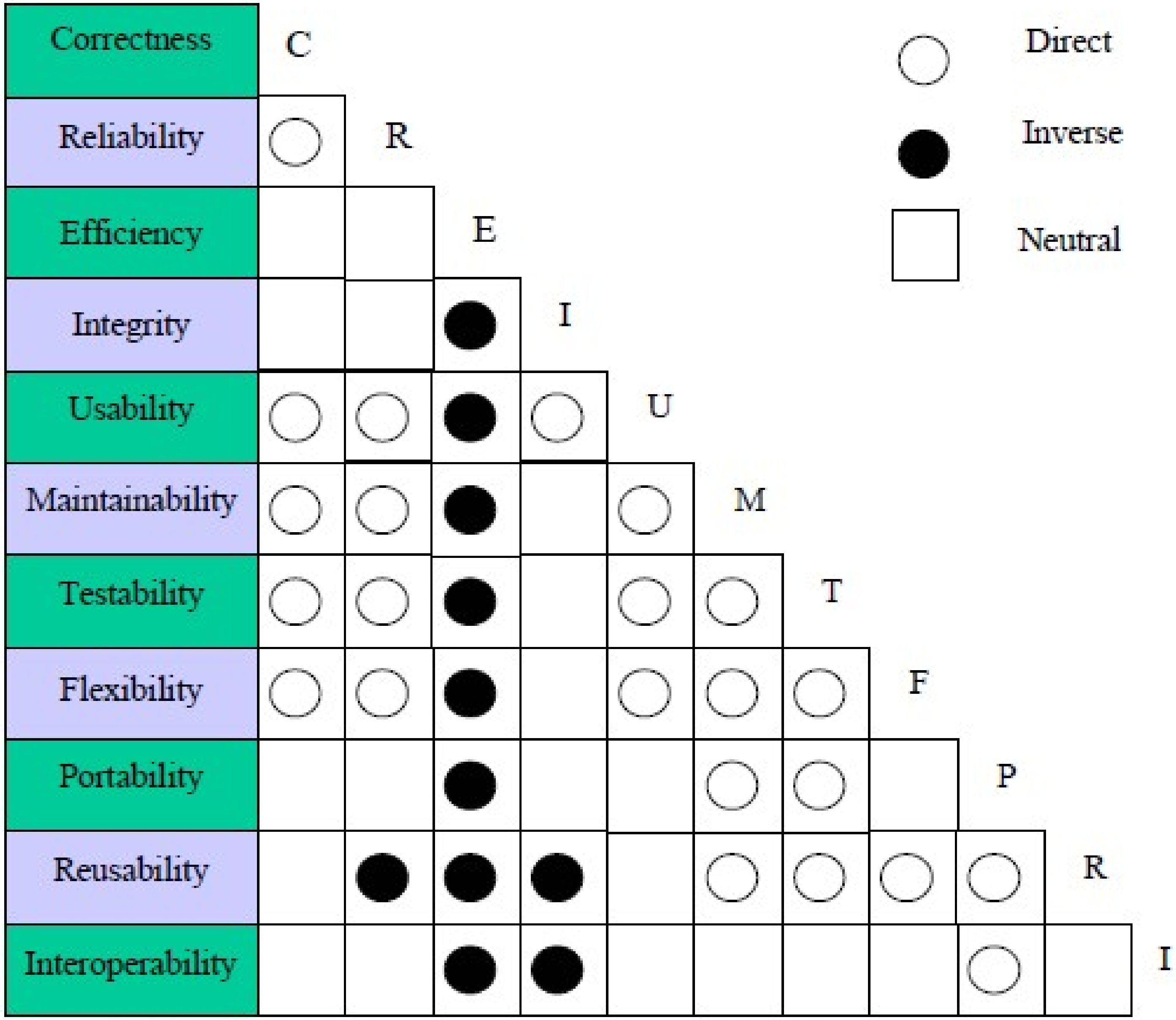
Correctness: *Does it do what I want?*
Reliability: *Does it do it accurately all the time?*
Efficiency: *Will it run on my machine as well as it can?*
Integrity: *Is it secure?*
Usability: *Can I run it?*

Figure 2.1 McCall's model of software quality

RELATIONAL MODELS

Perry’s model contains three types of relationship between the quality attributes.

- The direct relationship
- The inverse relationship
- The neutral relationship



	C	R	E	I	U	M	T	F	P	R	I
Correctness											
Reliability	○										
Efficiency											
Integrity			●								
Usability	○	○	●	○							
Maintainability	○	○	●		○						
Testability	○	○	●		○	○					
Flexibility	○	○	●		○	○	○				
Portability			●			○	○				
Reusability		●	●	●		○	○	○	○		
Interoperability			●	●					○		

○ Direct

● Inverse

□ Neutral

Figure 2.2 Perry's relational model of software quality

RELATIONSHIP B/W QUALITY ATTRIBUTES

- *Integrity vs. efficiency (inverse):*

  The control of data access will need additional code, leading to a longer runtime and more storage requirement.

- *Usability vs. efficiency (inverse):*

  Improvement of HCI will need more code and data, hence the system will be less efficient.

RELATIONSHIP B/W QUALITY ATTRIBUTES

- *Maintainability and testability vs. efficiency (inverse):*

  Compact and optimized code is not easy to maintain and test.

- *Flexibility, reusability vs. integrity (inverse):*

  Flexible data structures required for flexible and reusable software increase the data security problem.

- *Reusability vs. maintainability (direct):*

RELATIONSHIP B/W QUALITY ATTRIBUTES

- *Portability vs. reusability (direct):*

  Portable code is likely to be easily used in other environments. The code is likely well-structured and easier to be reused.

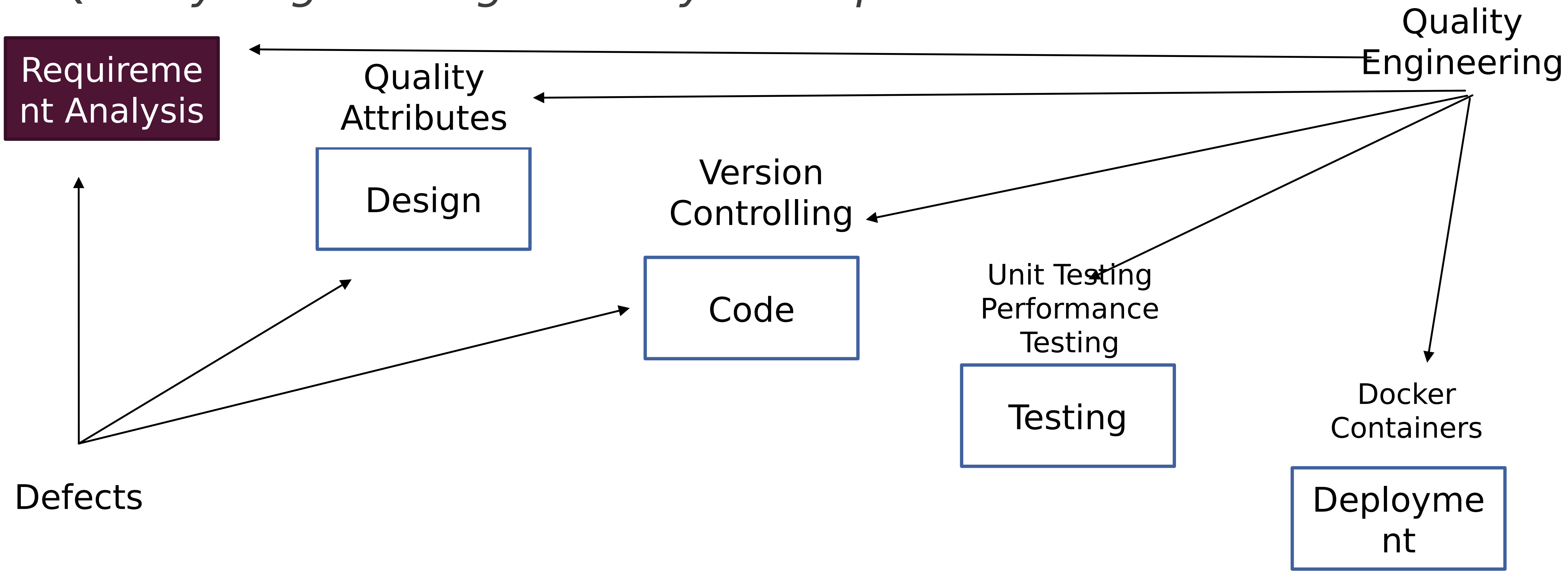
- *Correctness vs. efficiency (neutral):*

  The correctness of code has no relation with its efficiency. Correct code may be efficient or inefficient in operation.

SOFTWARE QUALITY ENGINEERING AT REQUIREMENT PHASE

SOFTWARE QUALITY ENGINEERING ROAD MAP

- *Quality Engineering at every SDLC phase:*



Requirement Analysis

Quality Attributes

Design

Version Controlling

Code

Unit Testing
Performance Testing

Testing

Docker Containers

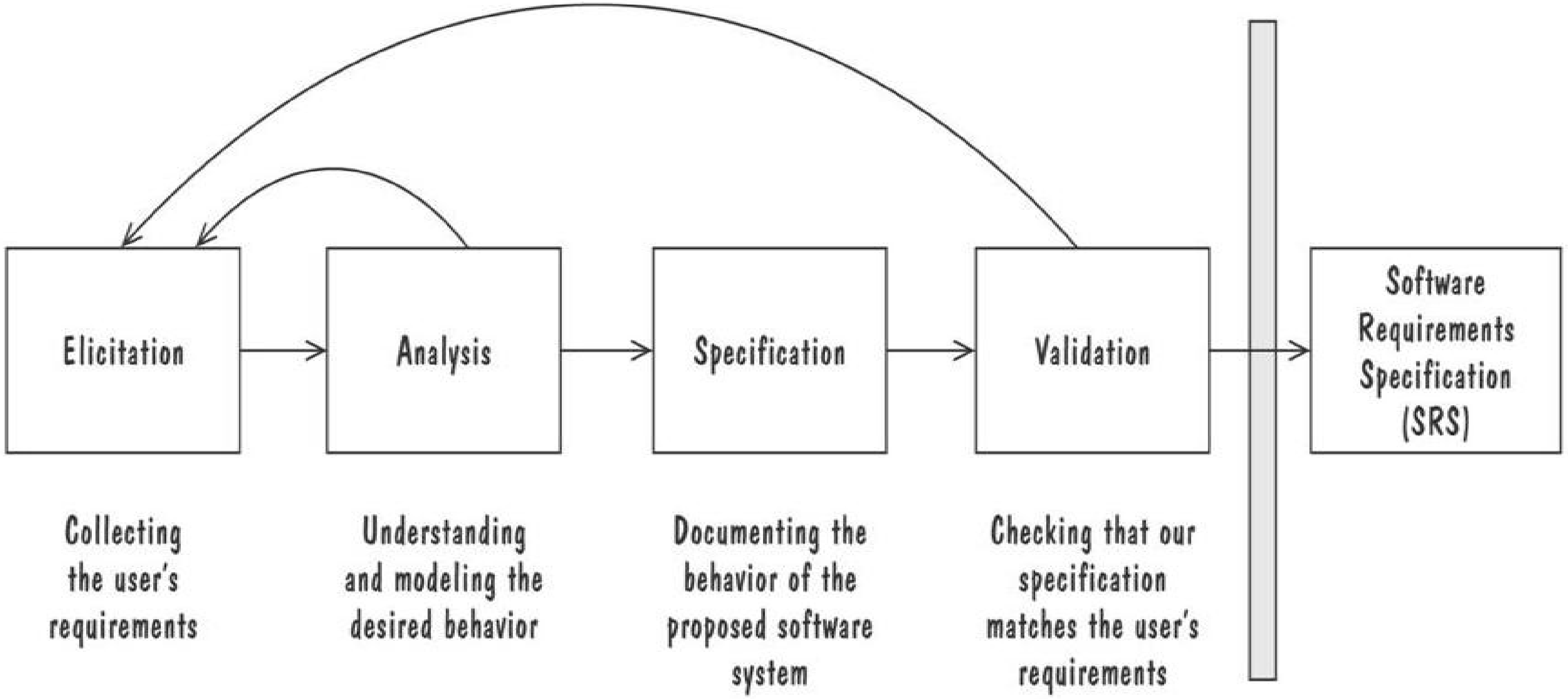
Deployment

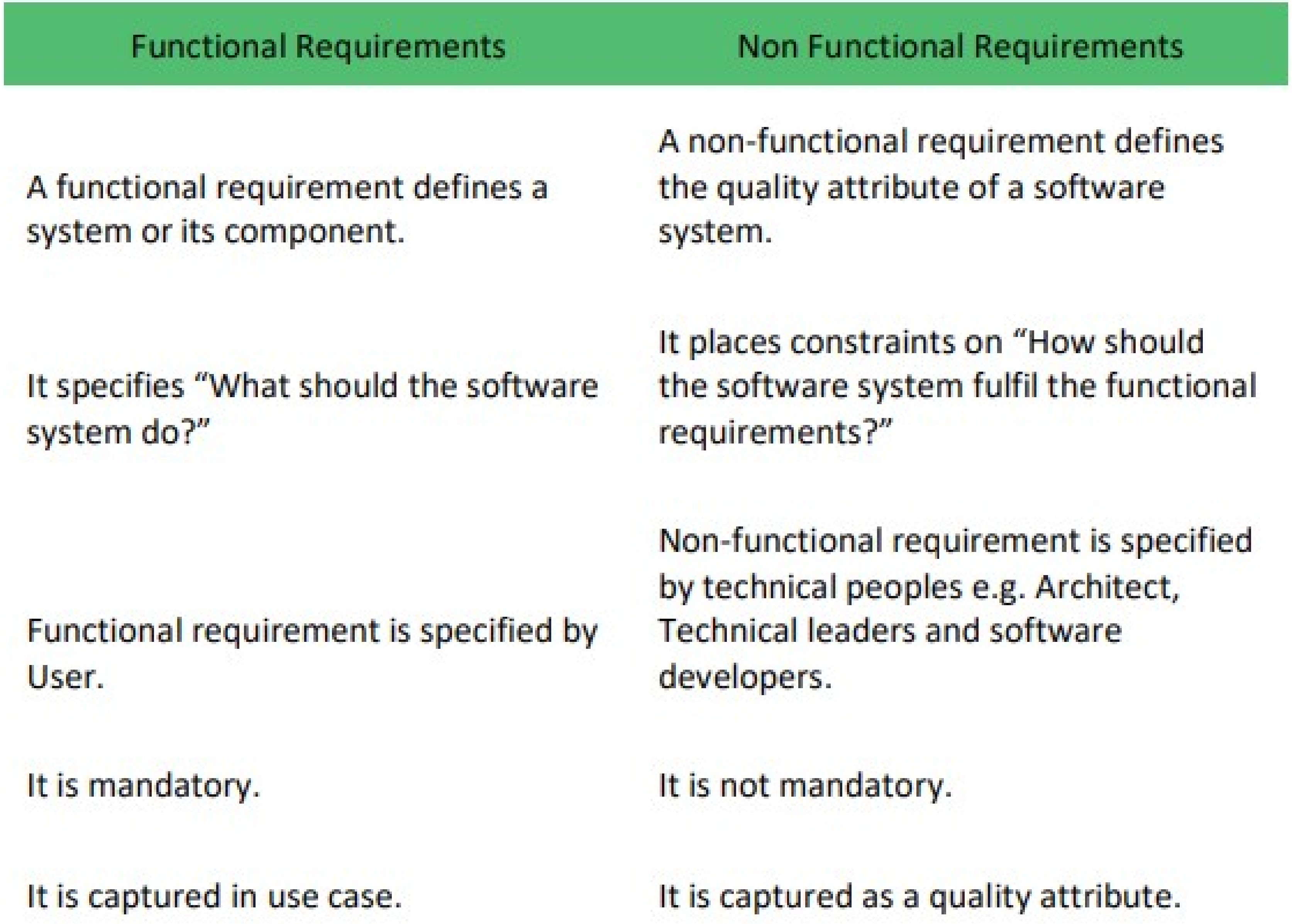
Quality Engineering

Defects

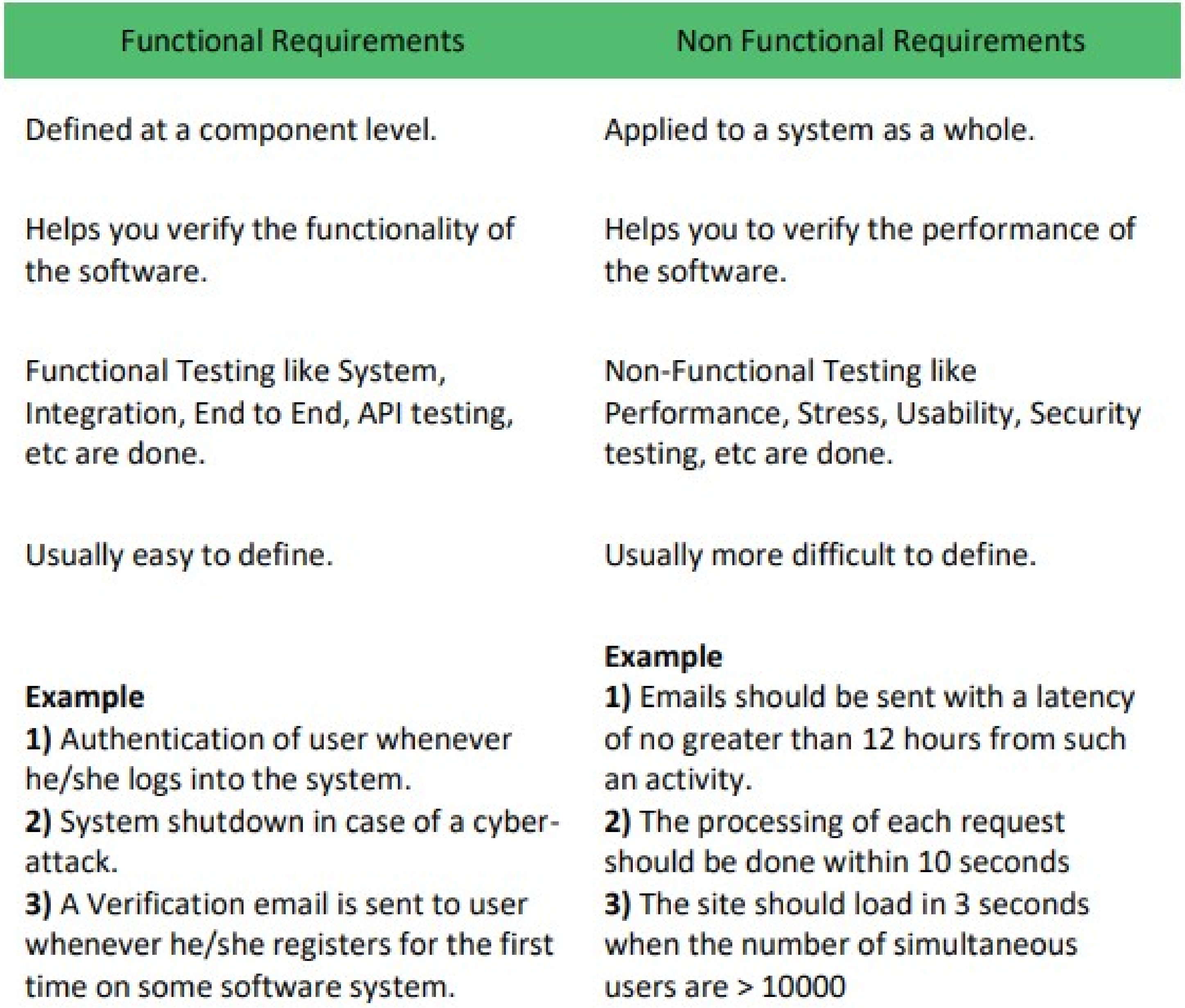
THE REQUIREMENTS PROCESS
(PROCESS FOR CAPTURING REQUIREMENTS)

- Performed by the req. analyst or system analyst
- The final outcome is a Software Requirements Specification (SRS) document


Elicitation
Analysis
Specification
Validation
Software Requirements Specification (SRS)
Collecting the user's requirements
Understanding and modeling the desired behavior
Documenting the behavior of the proposed software system
Checking that our specification matches the user's requirements



Functional Requirements	Non Functional Requirements
A functional requirement defines a system or its component.	A non-functional requirement defines the quality attribute of a software system.
It specifies "What should the software system do?"	It places constraints on "How should the software system fulfil the functional requirements?"
Functional requirement is specified by User.	Non-functional requirement is specified by technical peoples e.g. Architect, Technical leaders and software developers.
It is mandatory.	It is not mandatory.
It is captured in use case.	It is captured as a quality attribute.



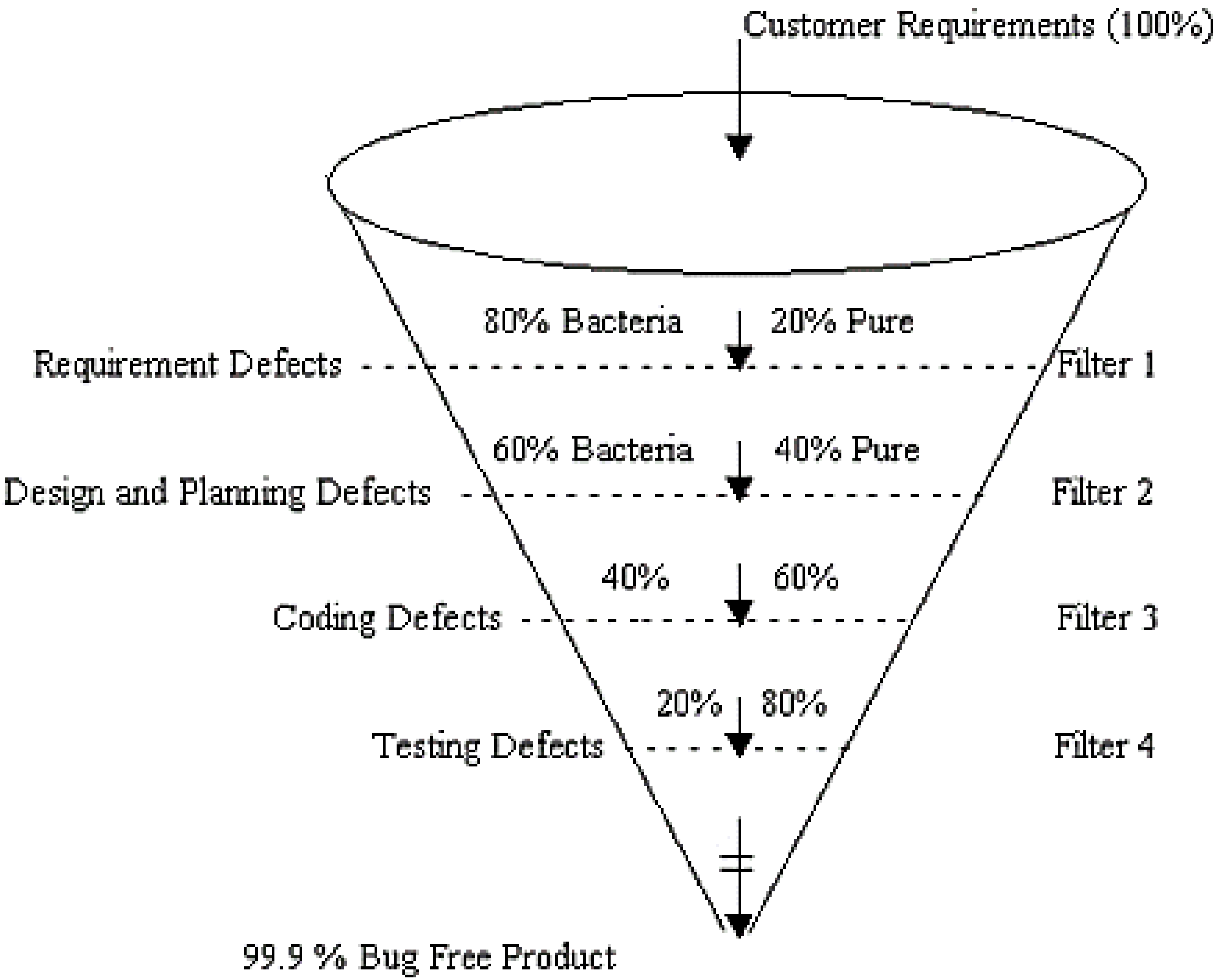
Functional Requirements	Non Functional Requirements
Defined at a component level.	Applied to a system as a whole.
Helps you verify the functionality of the software.	Helps you to verify the performance of the software.
Functional Testing like System, Integration, End to End, API testing, etc are done.	Non-Functional Testing like Performance, Stress, Usability, Security testing, etc are done.
Usually easy to define.	Usually more difficult to define.
Example **1)** Authentication of user whenever he/she logs into the system. **2)** System shutdown in case of a cyber-attack. **3)** A Verification email is sent to user whenever he/she registers for the first time on some software system.	**Example** **1)** Emails should be sent with a latency of no greater than 12 hours from such an activity. **2)** The processing of each request should be done within 10 seconds **3)** The site should load in 3 seconds when the number of simultaneous users are > 10000

WHAT IS DEFECT?

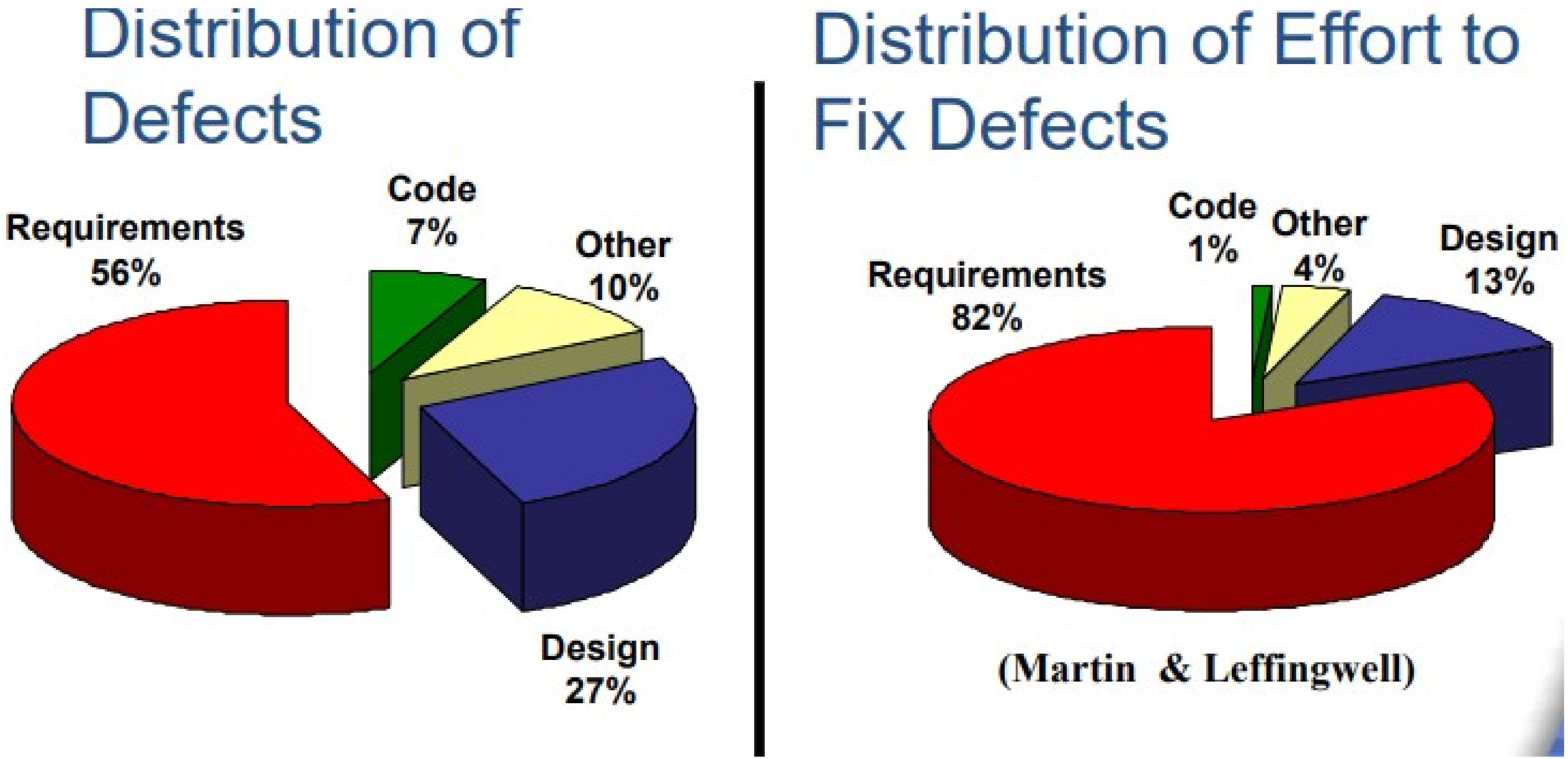
- Defect is deviation from customer requirement.
- Mostly defects are found in the software after software is shipped to the customer at production site.

- Example:
- In online shopping, the option of searching a debit card for making payment is missing.

DEFECTS


Customer Requirements (100%)
80% Bacteria
20% Pure
Requirement Defects
Filter 1
60% Bacteria
40% Pure
Design and Planning Defects
Filter 2
40%
60%
Coding Defects
Filter 3
20%
80%
Testing Defects
Filter 4
99.9 % Bug Free Product

DISTRIBUTION OF DEFECTS



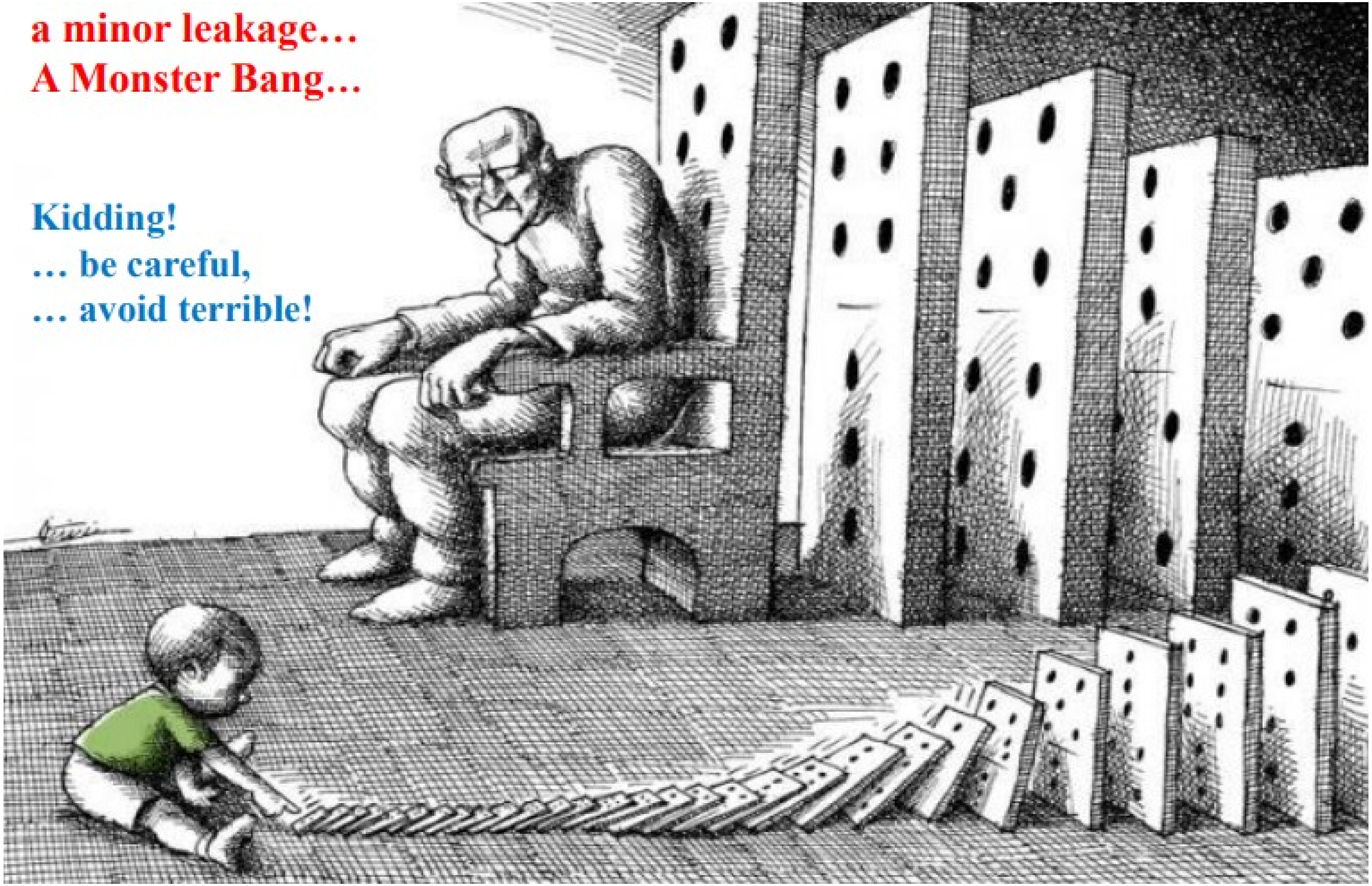
Distribution of Defects

Distribution of Effort to Fix Defects

(Martin & Leffingwell)

a minor leakage…
A Monster Bang…

Kidding!
… be careful,
… avoid terrible!



IMPACT OF REQUIREMENT DEFECTS

- Leakage into other phases
  - Design, Code, Implementation, Maintenance etc.
- Impact
  - re-requirement,
  - re-design,
  - re-code,
  - Re-testing,
  - re-implementation,
  - re-deployment, re-training .
  - increase cost, extra support & service cost.

WHAT ARE MISSING QUALITY REQUIREMENTS?

In a real-time scenario, more budgets mean more quality. This is both theoretically and practically true.

Putting in more money for quality of the software product will result in low probability of product failure and may save a lot of financial resources as the high quality product will be immune to threats.

Prioritize areas which are non-negotiable.

- For example, a software product with excellent User Interface (UI) but with no firewall for database security will face more threats. So adding a firewall to ensure Database is secure is more important than spending budget on cosmetic changes in UI.

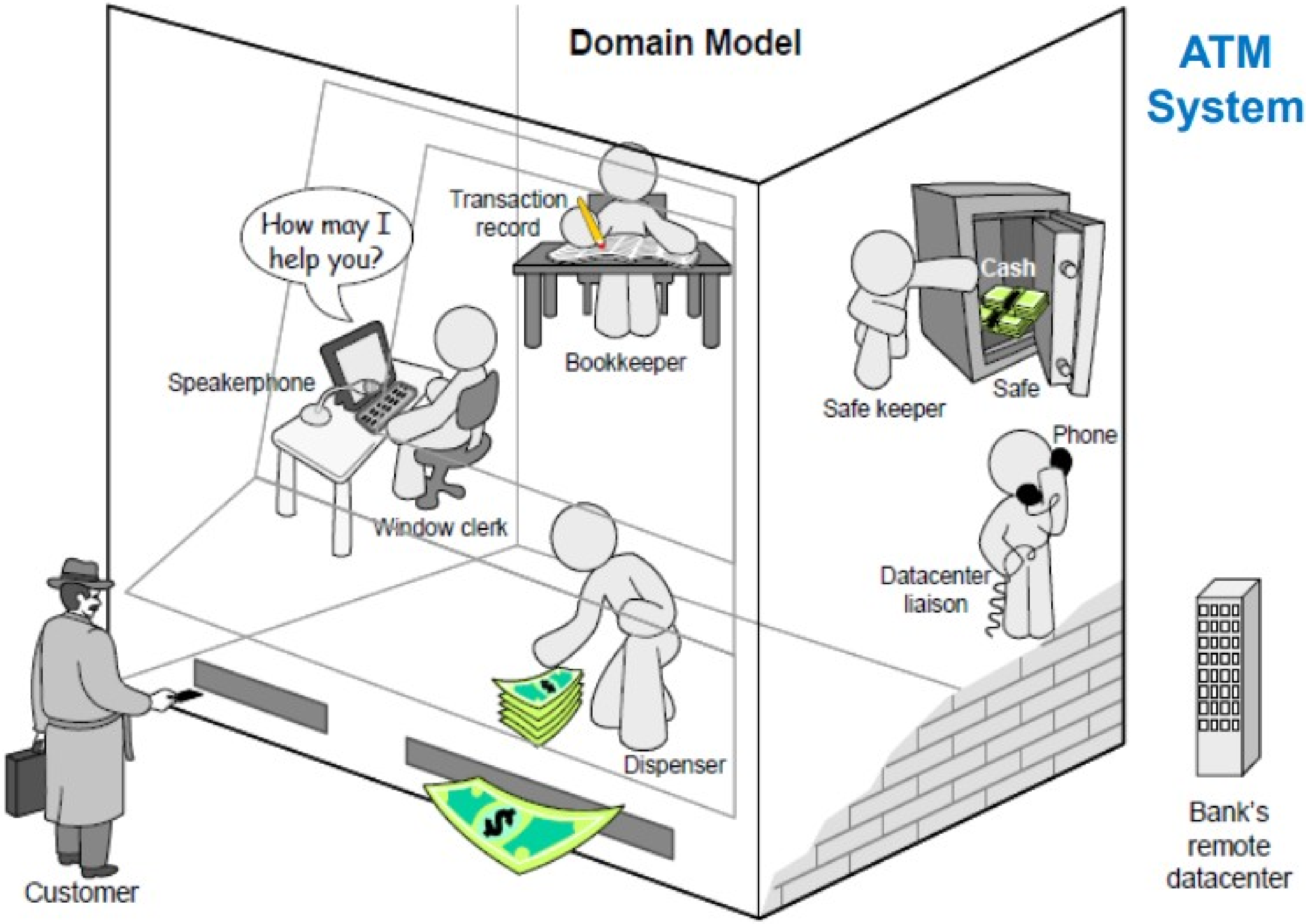
RE RELATED ACTIVITIES

1. **Functional Requirements**: describes what a software system should do
2. **Non-Functional:** place constraints on how the system will do so.
3. **Business Processes**: procedure or event with the purpose of reaching a goal
4. **Scope:** required tasks to accomplish, boundary of system
5. **Goals**: An observable and measurable end result having one or more objectives to be achieved within a more or less fixed timeframe
6. **Stakeholders:** person, group or organization that has interest or concern in an organization
7. **Sources:** someone or something that provides what is wanted or needed i.e. human, documents, context, situational factors, application types etc.
8. **Feasibility study:** a feasibility study is an analysis of the viability of an idea

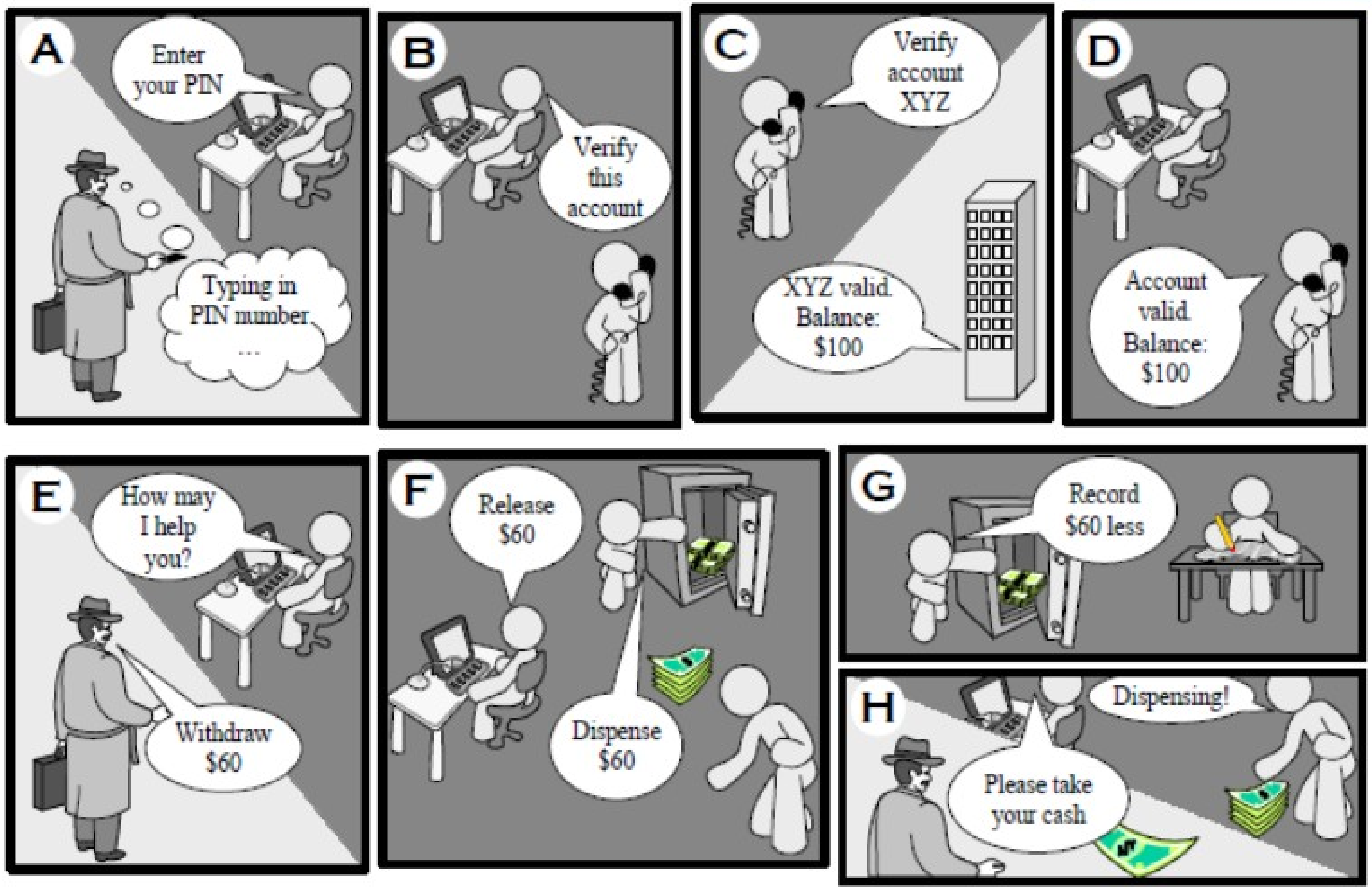
RE RELATED ACTIVITIES

9. **GUI**: Graphical User Interface
10. **Traceability:** concerned with documenting the relationships between different development artifacts (i.e. requirements and other artifacts).
11. **Measureable**: objectives should be measurable and achievable
12. **Domain**: area, business, discipline, field, realm, sphere
13. **Prototyping:** An easily modified and extensible model (representation, simulation or demonstration), partial or approximation of final product, useful for clarifying requirement.

ATM System



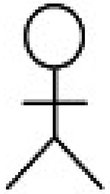
Imagined static structure of ATM shows internal components and their roles.



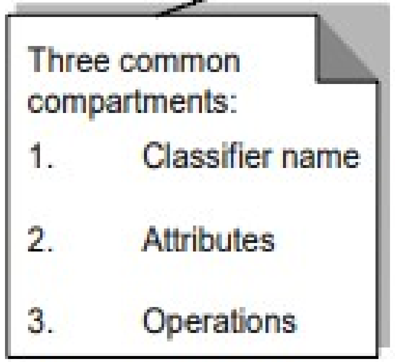
Dynamic interactions of the imagined components during task accomplishment.

ATM System

UML = Unified Modeling Language



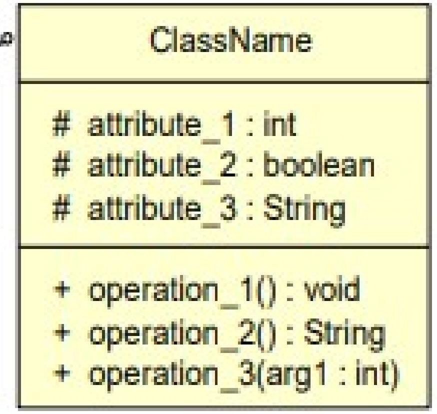
Actor



Three common compartments:

1. Classifier name
2. Attributes
3. Operations

Comment

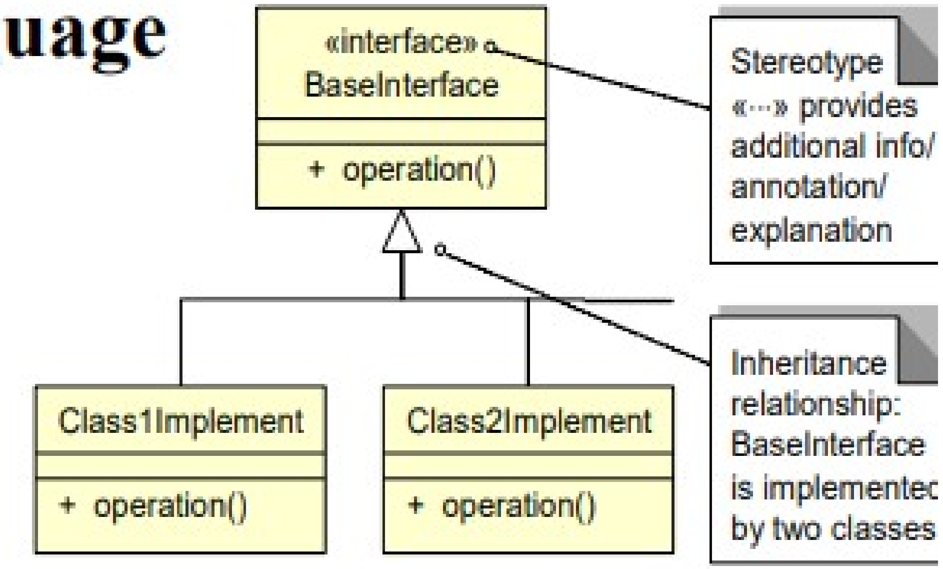


ClassName

attribute_1 : int
attribute_2 : boolean
attribute_3 : String

+ operation_1() : void
+ operation_2() : String
+ operation_3(arg1 : int)

Software Class



«interface»
BaseInterface

+ operation()

Class1Implement

+ operation()

Class2Implement

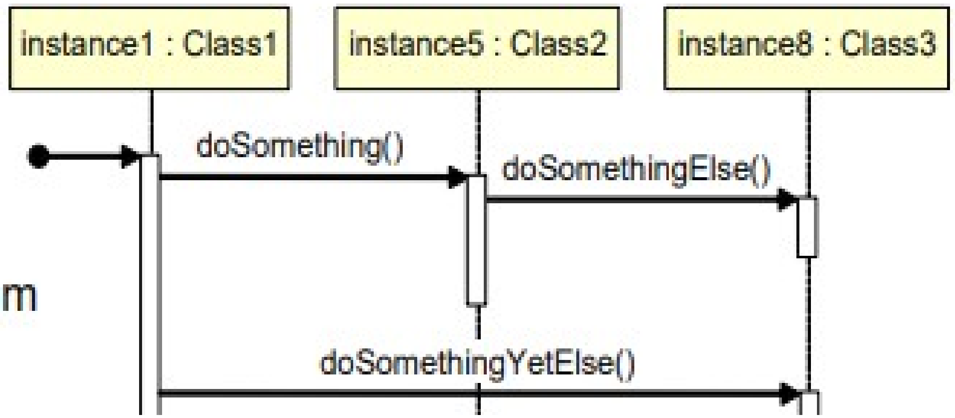
+ operation()

Stereotype «···» provides additional info/ annotation/ explanation

Inheritance relationship: BaseInterface is implementec by two classes

Software Interface Implementation

Interaction Diagram



instance1 : Class1

instance5 : Class2

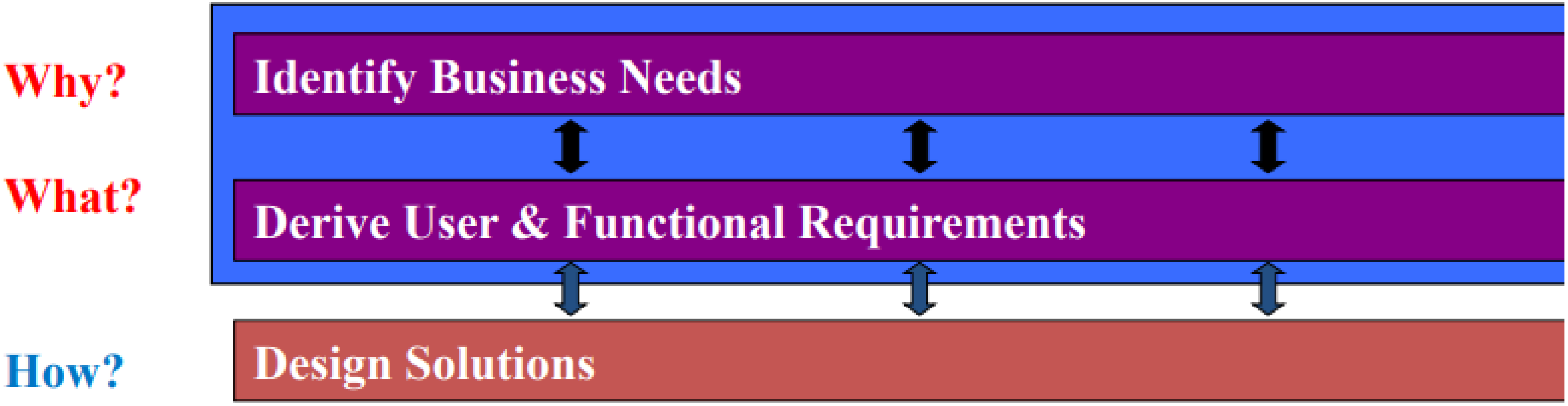
instance8 : Class3

doSomething()

doSomethingElse()

doSomethingYetElse()

RE Process and Related Activities: Why? What? How?



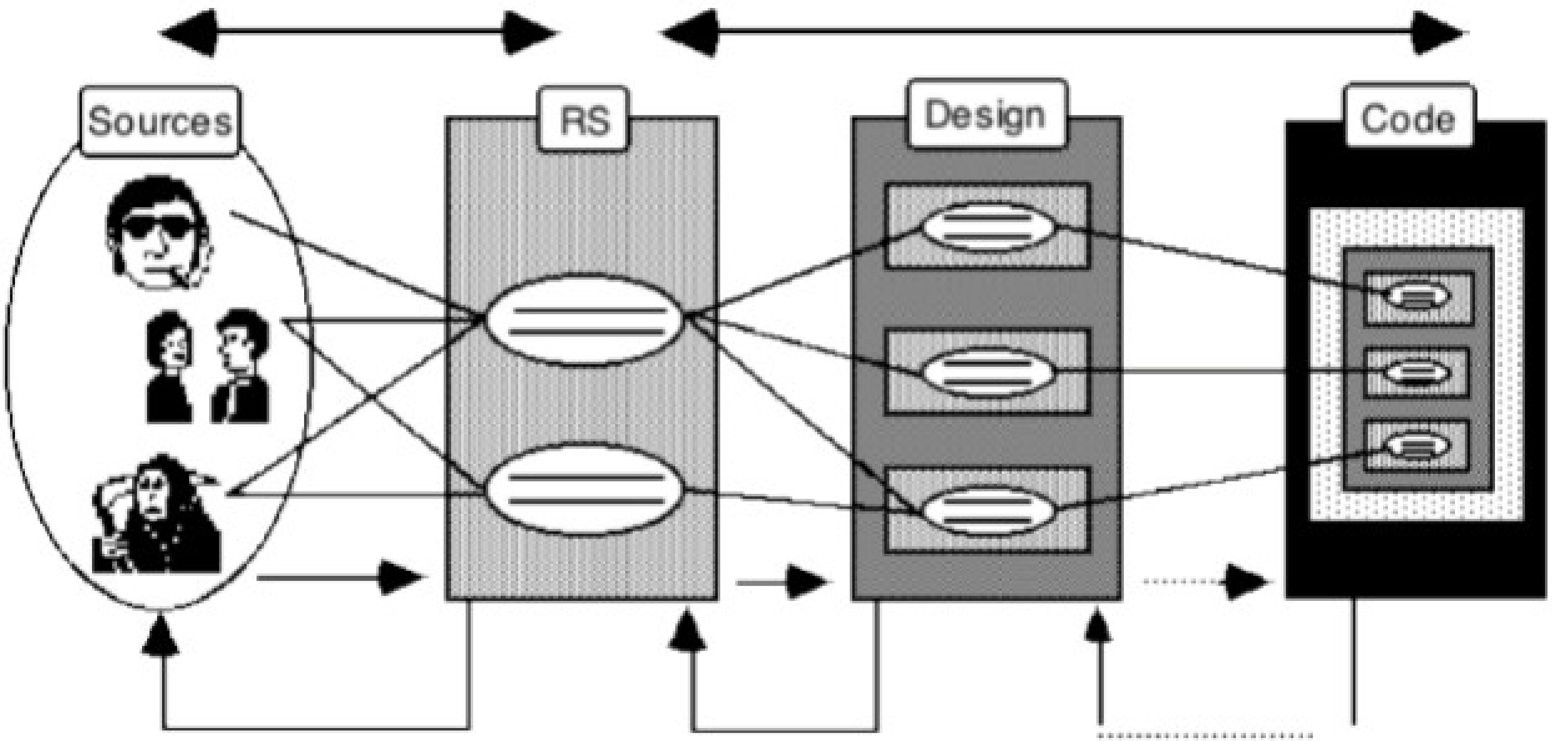
Why? Identify Business Needs

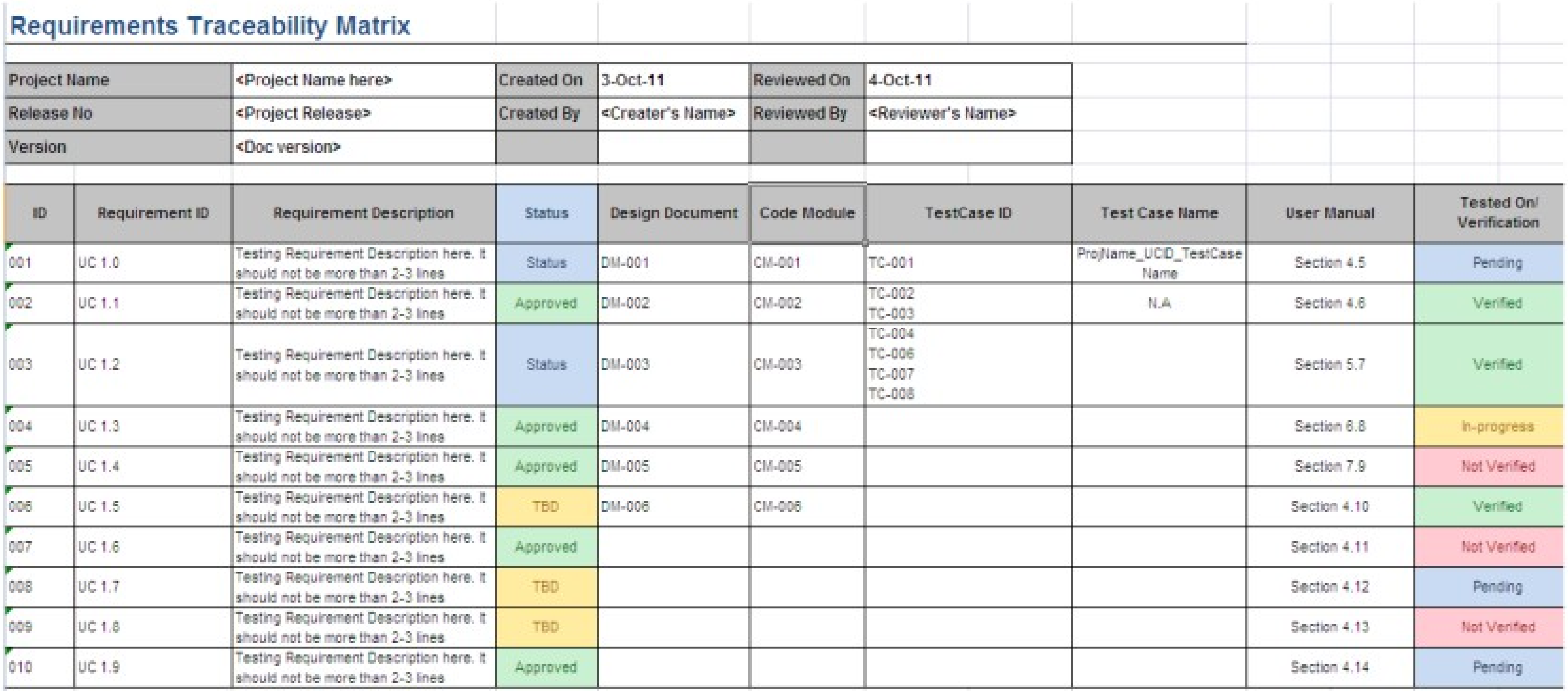
What? Derive User & Functional Requirements

How? Design Solutions

What , How, Why?

A Simplified Picture





Requirements Traceability Matrix

Project Name	<Project Name here>	Created On	3-Oct-11	Reviewed On	4-Oct-11
Release No	<Project Release>	Created By	<Creater's Name>	Reviewed By	<Reviewer's Name>
Version	<Doc version>				

ID	Requirement ID	Requirement Description	Status	Design Document	Code Module	TestCase ID	Test Case Name	User Manual	Tested On/ Verification
001	UC 1.0	Testing Requirement Description here. It should not be more than 2-3 lines	Status	DM-001	CM-001	TC-001	ProjName_UCID_TestCase Name	Section 4.5	Pending
002	UC 1.1	Testing Requirement Description here. It should not be more than 2-3 lines	Approved	DM-002	CM-002	TC-002 TC-003	N.A	Section 4.6	Verified
003	UC 1.2	Testing Requirement Description here. It should not be more than 2-3 lines	Status	DM-003	CM-003	TC-004 TC-006 TC-007 TC-008		Section 5.7	Verified
004	UC 1.3	Testing Requirement Description here. It should not be more than 2-3 lines	Approved	DM-004	CM-004			Section 6.8	In-progress
005	UC 1.4	Testing Requirement Description here. It should not be more than 2-3 lines	Approved	DM-005	CM-005			Section 7.9	Not Verified
006	UC 1.5	Testing Requirement Description here. It should not be more than 2-3 lines	TBD	DM-006	CM-006			Section 4.10	Verified
007	UC 1.6	Testing Requirement Description here. It should not be more than 2-3 lines	Approved					Section 4.11	Not Verified
008	UC 1.7	Testing Requirement Description here. It should not be more than 2-3 lines	TBD					Section 4.12	Pending
009	UC 1.8	Testing Requirement Description here. It should not be more than 2-3 lines	TBD					Section 4.13	Not Verified
010	UC 1.9	Testing Requirement Description here. It should not be more than 2-3 lines	Approved					Section 4.14	Pending

CASE STUDY

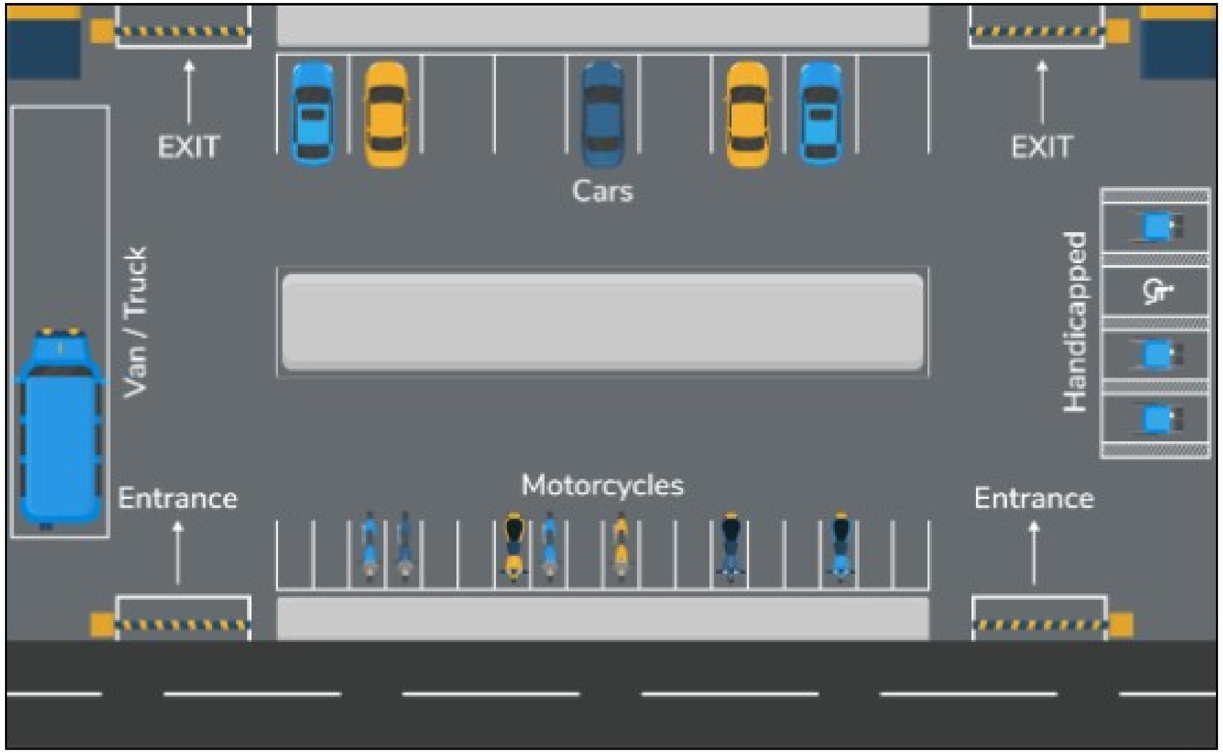
PARKING LOT

In a parking lot, there are a fixed number of parking spots available for different types of vehicles.

Each of these spots is charged according to the time the vehicle has been parked in the parking lot.

The parking time is tracked with a parking ticket issued to the vehicle at the entrance of the parking lot.

Once the vehicle is ready to exit, it can either pay at the automated exit panel or to the parking agent at the exit using a card or cash payment method.



INTERVIEW

An overview of some major expectations the interviewer will want an interviewee to discuss in more detail during the interview.

Payment:

1. How are customers able to pay at different exit points (i.e., either at the automated exit panel or to the parking agent) and by different methods (cash, credit, coupon)?
2. If there are multiple floors in the parking lot, how will the system keep track of the customer having already paid on a particular floor rather than at the exit?

INTERVIEW

Parking spot

1. What happens when a lot becomes full?
2. How can one keep track of the free parking spots on each floor if there are multiple floors in the parking lot?
3. If the parking spot of any vehicle type is booked, can a vehicle of another type park in the designated parking spot?

Pricing

4. How will pricing be handled? Should we accommodate having different rates for each hour? For example, customers will have to pay $4$4 for the first hour, $3.5$3.5 for the second and third hours, and $2.5$2.5 for all the subsequent hours.
5. Will the pricing be the same for the different vehicle types?

REQUIREMENTS COLLECTION

1. R1: The parking lot should have the capacity to park 40,000 vehicles.
2. R2: The four different types of parking spots are handicapped, compact, large, and motorcycle.
3. R3: The parking lot should have multiple entrance and exit points.
4. R4: Four types of vehicles should be allowed to park in the parking lot, which are as follows:
   - I. Car
   - II. Truck
   - III. Van
   - IV. Motorcycle

REQUIREMENTS COLLECTION

5. R5: The parking lot should have a display board that shows free parking spots for each parking spot type.
6. R6: The system should not allow more vehicles in the parking lot if the maximum capacity (40,000) is reached.
7. R7: If the parking lot is completely occupied, the system should show a message on the entrance and on the parking lot display board.
8. R8: Customers should be able to collect a parking ticket from the entrance and pay at the exit.
9. R9: The customer can pay for the ticket either with an automated exit panel or pay the parking agent at the exit.
10. R10: The payment should be calculated at an hourly rate.
11. R11: Payment can be made using either a credit/debit card or cash.

ACTORS

System

Our system is a "parking lot.“

Primary actors

Customer: This actor can park the vehicle in the allocated parking space according to the vehicle type and pay for the parking before exit.

Parking agent: The parking agent will assist the customer and perform all the tasks that a customer can do, such as paying the parking ticket on behalf of the customer.

ACTORS

Secondary actors

Admin: This can add, remove, or update a spot, agent, entry/exit panels, and view/update accounts.

System: This is responsible for giving details of parking spot availability and assigning a parking spot to a vehicle.

USE CASES

Admin

Add spot: To add a parking spot

Add agent: To add a new agent

Add/modify rate: To add/modify hourly rate

Add entry/exit panel: To add and update exit/entry panel at each entry/exit

Update account: To update account details and payment information

Login/Logout: To login/logout to/from agent or admin account

View account: To view account details like payment status or unpaid amount

USE CASES

Customer

Take ticket: To take a ticket at the entrance, that contains information regarding the vehicle and its entrance time

Scan ticket: To scan the ticket at the exit and get the parking fee

Pay ticket: To pay the parking fee at the exit panel via cash or a credit card

Cash: To pay the parking fee via cash

Credit card: To pay the parking fee via credit card

Park vehicle: To park the vehicle at the assigned destination

USE CASES

Parking agent

Update account: To update account details and payment information

Login/Logout: To log in/log out to/from the agent or admin account

View account: To view account details like payment status or unpaid amount

Take ticket: To take a ticket at the entrance, that contains information regarding the vehicle and its entrance time

Scan ticket: To scan the ticket at the exit and get the parking fee

Pay ticket: To pay the parking fee at the exit panel via cash or a credit card

Cash: To pay the parking fee via cash

Credit card: To pay the parking fee via credit card

Park vehicle: To park the vehicle at the assigned destination

USE CASES

System

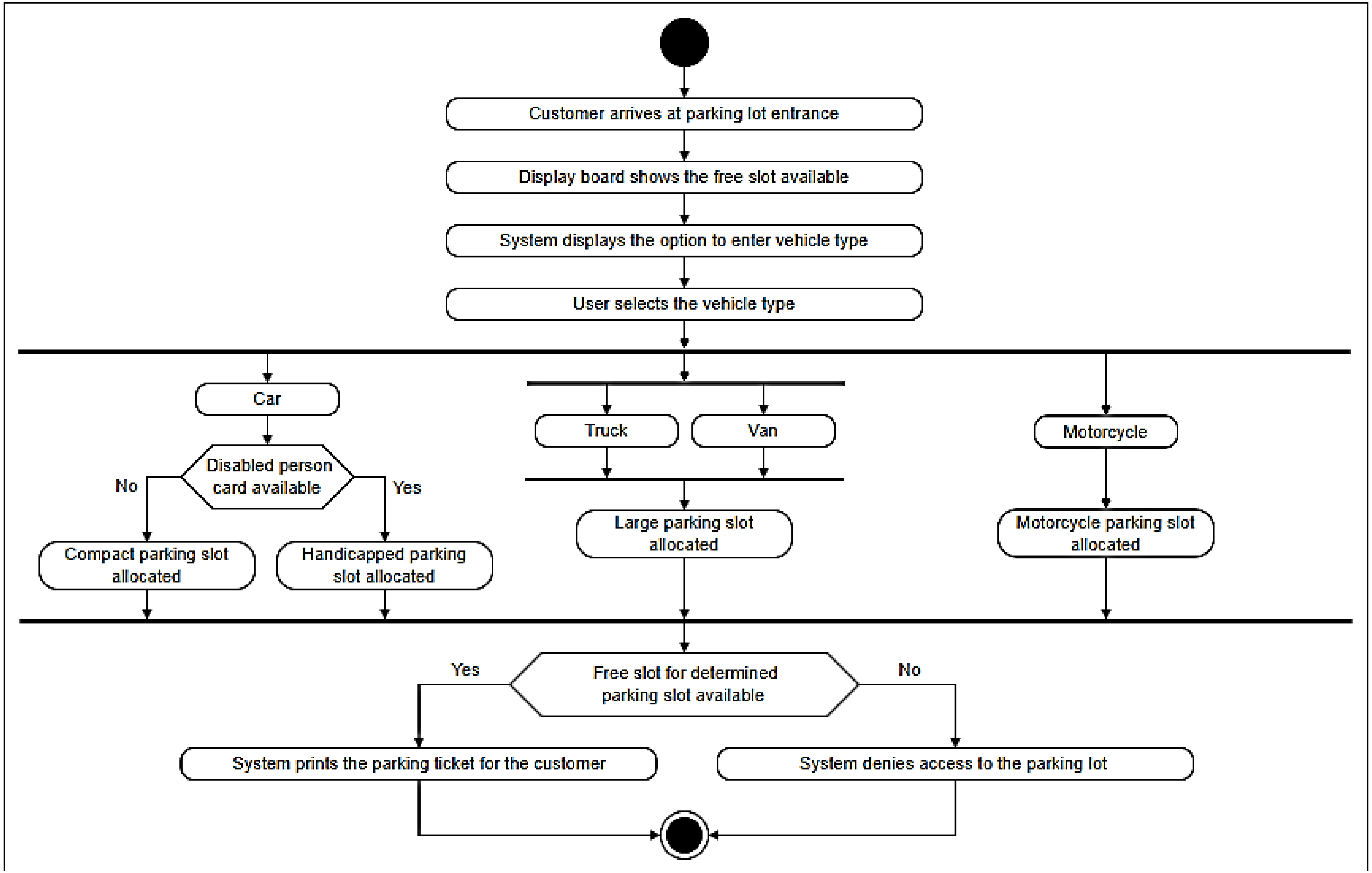
Assigning parking spots to vehicles: To check the vehicle type and associate a free spot according to it

Remove spot: To remove a parking spot if it is not available for parking

Show full: To display the status of the parking lot as full

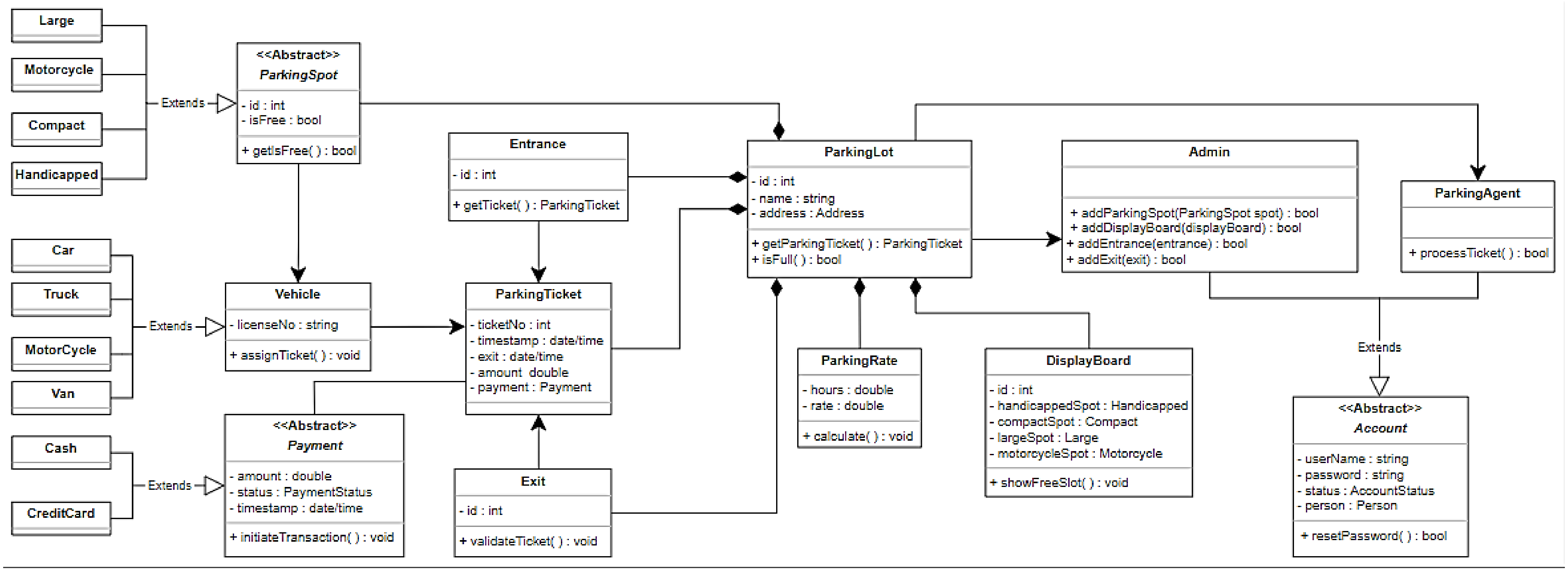
Show available: To show the details of available parking spots

Vehicle entering the parking lot



Customer arrives at parking lot entrance
Display board shows the free slot available
System displays the option to enter vehicle type
User selects the vehicle type
Car
Disabled person card available
No
Yes
Compact parking slot allocated
Handicapped parking slot allocated
Truck
Van
Large parking slot allocated
Motorcycle
Motorcycle parking slot allocated
Free slot for determined parking slot available
Yes
No
System prints the parking ticket for the customer
System denies access to the parking lot

Class diagram of the parking lot system



Large

Motorcycle

Compact

Handicapped

Extends

<<Abstract>>
ParkingSpot
- id : int
- isFree : bool
+ getIsFree( ) : bool

Entrance
- id : int
+ getTicket( ) : ParkingTicket

ParkingLot
- id : int
- name : string
- address : Address
+ getParkingTicket( ) : ParkingTicket
+ isFull( ) : bool

Admin
+ addParkingSpot(ParkingSpot spot) : bool
+ addDisplayBoard(displayBoard) : bool
+ addEntrance(entrance) : bool
+ addExit(exit) : bool

ParkingAgent
+ processTicket( ) : bool

Car

Truck

MotorCycle

Van

Extends

Vehicle
- licenseNo : string
+ assignTicket( ) : void

ParkingTicket
- ticketNo : int
- timestamp : date/time
- exit : date/time
- amount double
- payment : Payment

ParkingRate
- hours : double
- rate : double
+ calculate( ) : void

DisplayBoard
- id : int
- handicappedSpot : Handicapped
- compactSpot : Compact
- largeSpot : Large
- motorcycleSpot : Motorcycle
+ showFreeSlot( ) : void

Extends

<<Abstract>>
Account
- userName : string
- password : string
- status : AccountStatus
- person : Person
+ resetPassword( ) : bool

Cash

CreditCard

Extends

<<Abstract>>
Payment
- amount : double
- status : PaymentStatus
- timestamp : date/time
+ initiateTransaction( ) : void

Exit
- id : int
+ validateTicket( ) : void

HAVE A GOOD DAY!