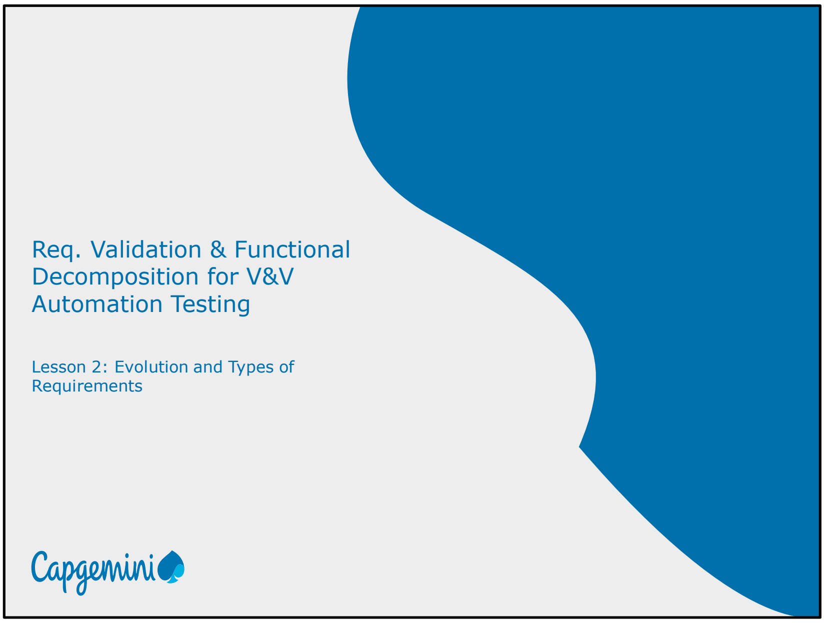


Req. Validation & Functional Decomposition for V&V Automation Testing

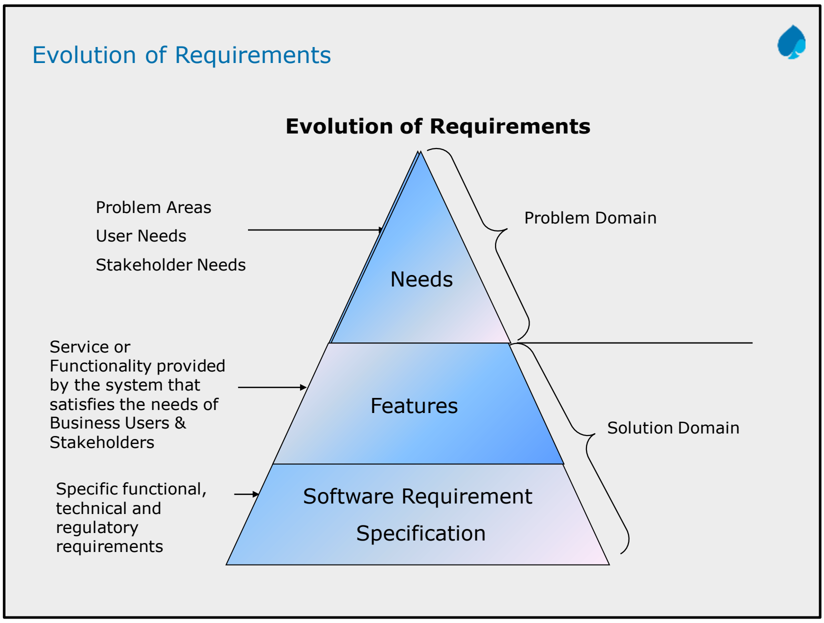
Lesson 2: Evolution and Types of Requirements

Capgemini

Lesson Objectives

To understand the following topics:

- Evolution of Requirements
- Who provides the Requirements?
- Types of Requirements
- Functional Vs Non-Functional Requirements
- Do not overlook the "Non-Functional Requirements"!
- Non Functional Requirements: FURPS +
- Other Non Functional Requirements: "+"
- What is a good software requirement?
- Summary
- Review Questions

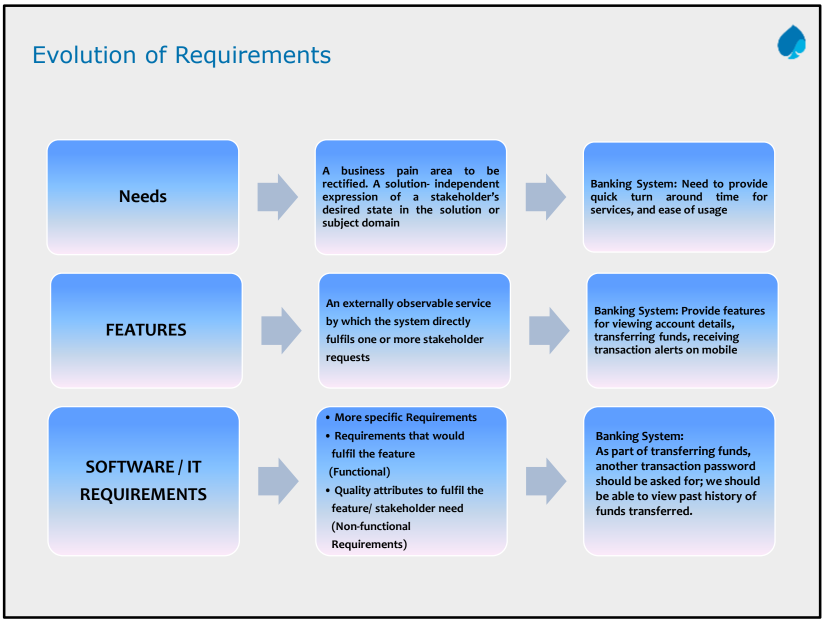


Evolution of Requirements

Evolution of Requirements
Problem Areas
User Needs
Stakeholder Needs
Needs
Problem Domain
Service or Functionality provided by the system that satisfies the needs of Business Users & Stakeholders
Features
Solution Domain
Specific functional, technical and regulatory requirements
Software Requirement Specification

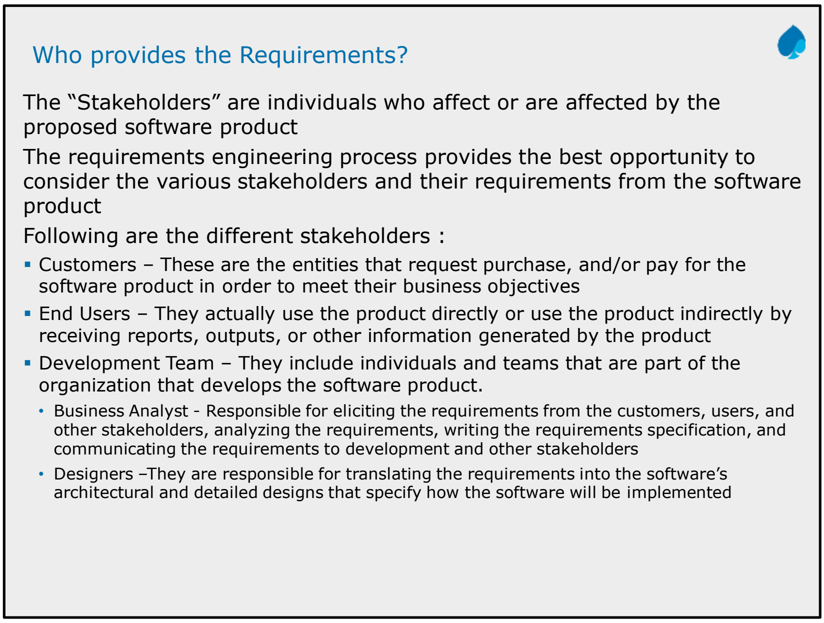
Evolution of Requirements:

Evolution involves the translation of stakeholder requests into a set of system features. These in turn are detailed into specifications for functional and nonfunctional requirements. Detailed specifications are translated into test procedures, design, and user documentation.



Evolution of Requirements

Needs	**A business pain area to be rectified. A solution- independent expression of a stakeholder's desired state in the solution or subject domain**	**Banking System: Need to provide quick turn around time for services, and ease of usage**
FEATURES	**An externally observable service by which the system directly fulfils one or more stakeholder requests**	**Banking System: Provide features for viewing account details, transferring funds, receiving transaction alerts on mobile**
SOFTWARE / IT REQUIREMENTS	• **More specific Requirements** • **Requirements that would fulfil the feature (Functional)** • **Quality attributes to fulfil the feature/ stakeholder need (Non-functional Requirements)**	**Banking System: As part of transferring funds, another transaction password should be asked for; we should be able to view past history of funds transferred.**



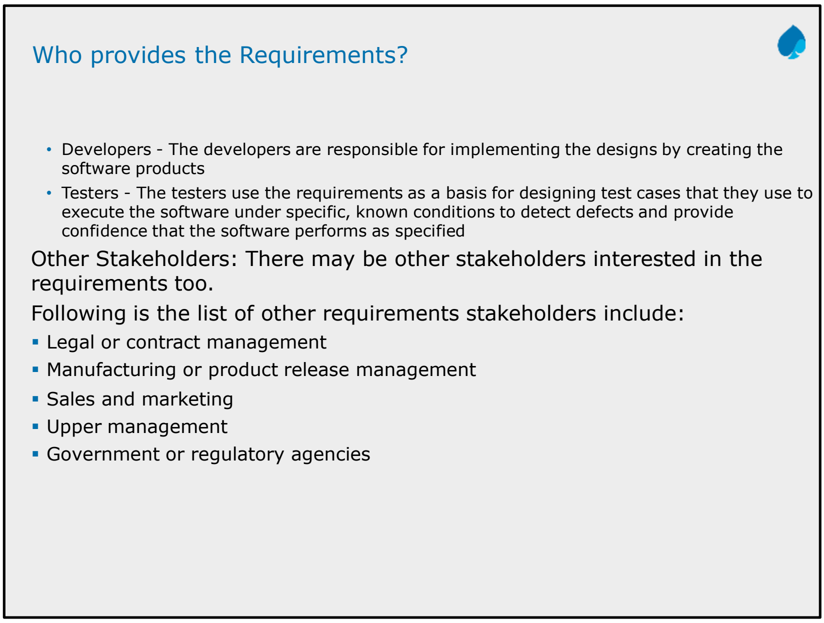
Who provides the Requirements?

The “Stakeholders” are individuals who affect or are affected by the proposed software product

The requirements engineering process provides the best opportunity to consider the various stakeholders and their requirements from the software product

Following are the different stakeholders :

- Customers – These are the entities that request purchase, and/or pay for the software product in order to meet their business objectives
- End Users – They actually use the product directly or use the product indirectly by receiving reports, outputs, or other information generated by the product
- Development Team – They include individuals and teams that are part of the organization that develops the software product.
  - Business Analyst - Responsible for eliciting the requirements from the customers, users, and other stakeholders, analyzing the requirements, writing the requirements specification, and communicating the requirements to development and other stakeholders
  - Designers –They are responsible for translating the requirements into the software’s architectural and detailed designs that specify how the software will be implemented



Who provides the Requirements?

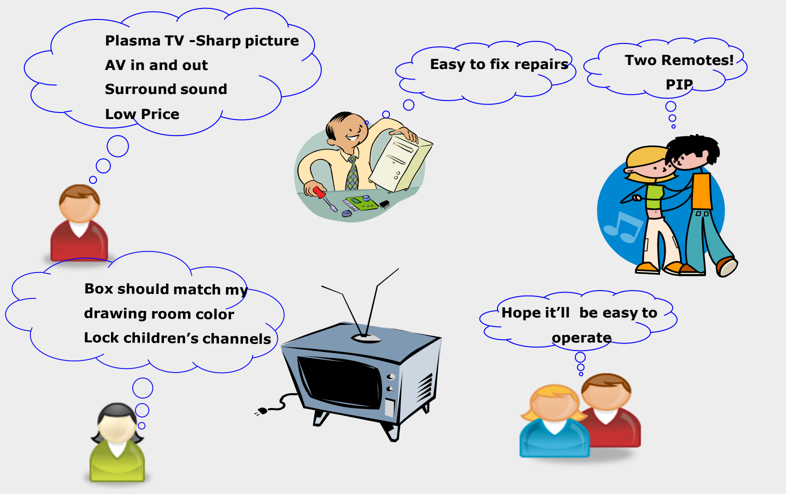
- Developers - The developers are responsible for implementing the designs by creating the software products
- Testers - The testers use the requirements as a basis for designing test cases that they use to execute the software under specific, known conditions to detect defects and provide confidence that the software performs as specified

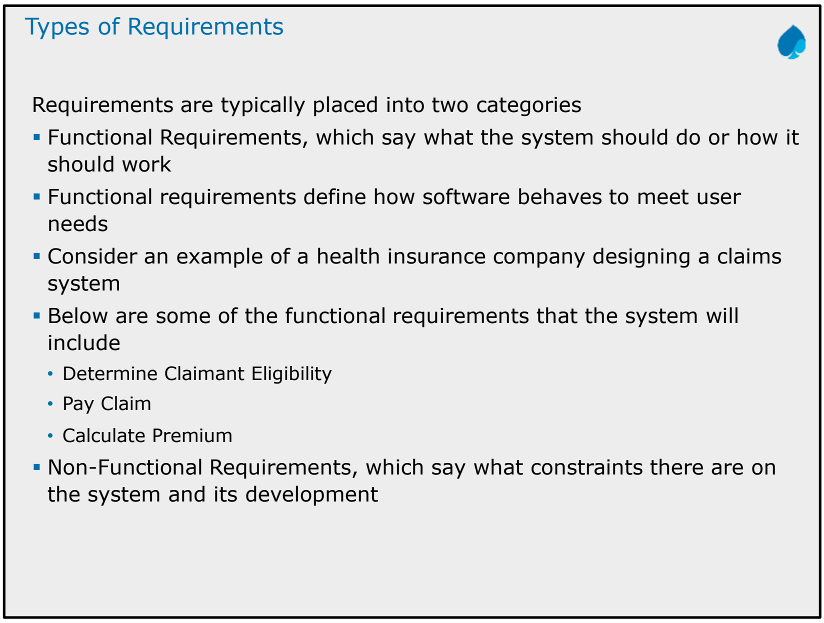
Other Stakeholders: There may be other stakeholders interested in the requirements too.

Following is the list of other requirements stakeholders include:

- Legal or contract management
- Manufacturing or product release management
- Sales and marketing
- Upper management
- Government or regulatory agencies

Different stakeholders….So different requirements





Types of Requirements

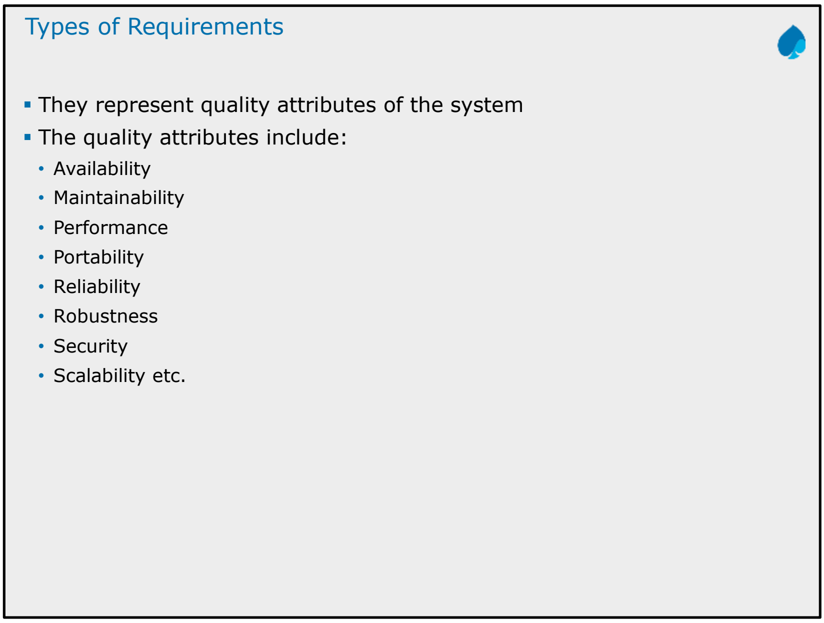
Requirements are typically placed into two categories

- Functional Requirements, which say what the system should do or how it should work
- Functional requirements define how software behaves to meet user needs
- Consider an example of a health insurance company designing a claims system
- Below are some of the functional requirements that the system will include
  - Determine Claimant Eligibility
  - Pay Claim
  - Calculate Premium
- Non-Functional Requirements, which say what constraints there are on the system and its development

Types of Requirements

1. Functional Requirements - Functional requirements capture the intended behaviour of the system. Functional requirements explain what has to be done by identifying the necessary task, action or activity that must be accomplished. It specifies actions that a system must be able to perform. Functional requirements thus specify the input and output behaviour of a systems.

2. Non Functional Requirements - Non functional requirements are the requirements that specify the criteria that can be used to determine the operation of a system, rather than a specific behaviour. Non functional Requirements specify the qualities that the product must possess. These are things such as security, compatibility, performance requirements etc.



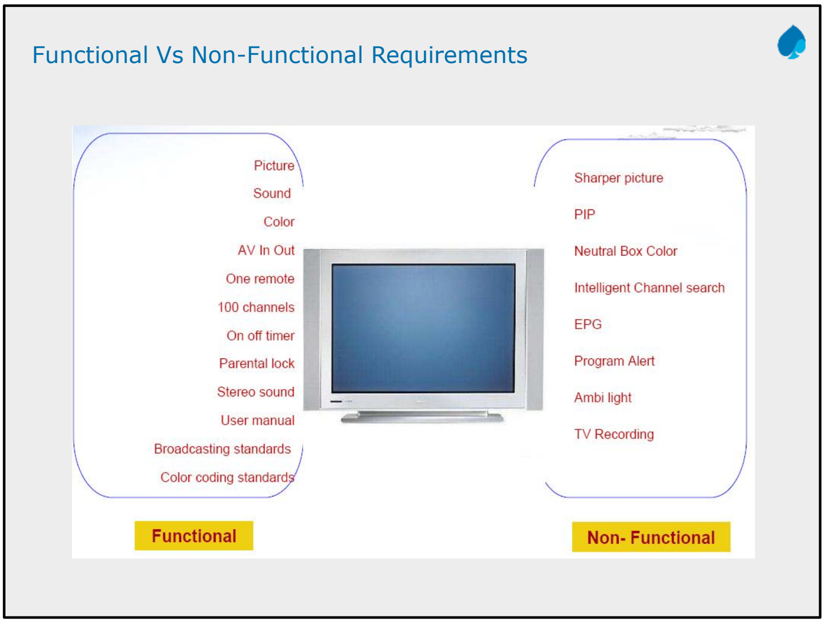
Types of Requirements

- They represent quality attributes of the system
- The quality attributes include:
  - Availability
  - Maintainability
  - Performance
  - Portability
  - Reliability
  - Robustness
  - Security
  - Scalability etc.

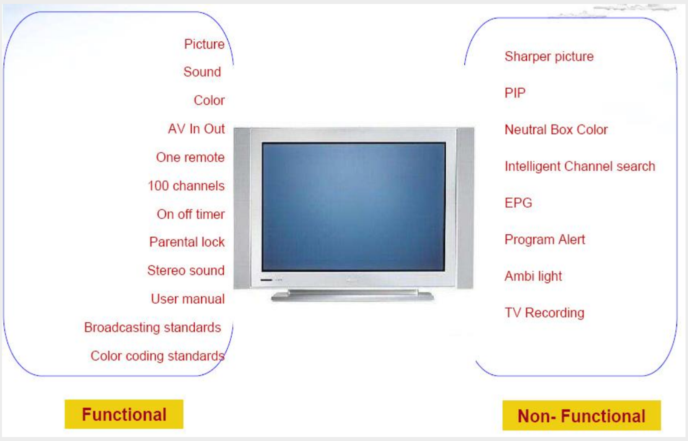
Types of Requirements

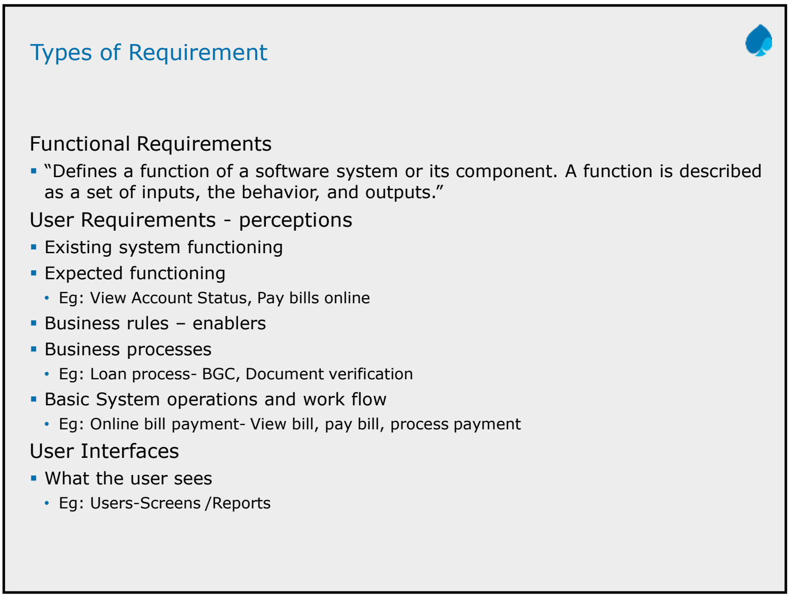
1. Functional Requirements - Functional requirements capture the intended behaviour of the system. Functional requirements explain what has to be done by identifying the necessary task, action or activity that must be accomplished. It specifies actions that a system must be able to perform. Functional requirements thus specify the input and output behaviour of a systems.

2. Non Functional Requirements - Non functional requirements are the requirements that specify the criteria that can be used to determine the operation of a system, rather than a specific behaviour. Non functional Requirements specify the qualities that the product must possess. These are things such as security, compatibility, performance requirements etc.



Functional Vs Non-Functional Requirements





Types of Requirement

Functional Requirements

- “Defines a function of a software system or its component. A function is described as a set of inputs, the behavior, and outputs.”

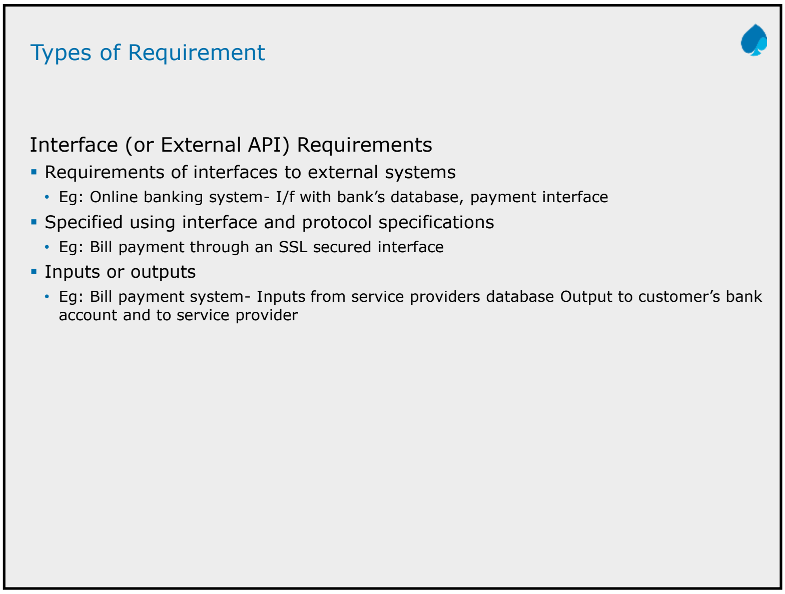
User Requirements - perceptions

- Existing system functioning
- Expected functioning
  - Eg: View Account Status, Pay bills online
- Business rules – enablers
- Business processes
  - Eg: Loan process- BGC, Document verification
- Basic System operations and work flow
  - Eg: Online bill payment- View bill, pay bill, process payment

User Interfaces

- What the user sees
  - Eg: Users-Screens /Reports

A functional requirement defines a function of a software-system or its component. A function is described as a set of inputs, the behaviour, and outputs (see also software). Functional requirements may be calculations, technical details, data manipulation and processing and other specific functionality that show how a use case is to be fulfilled.



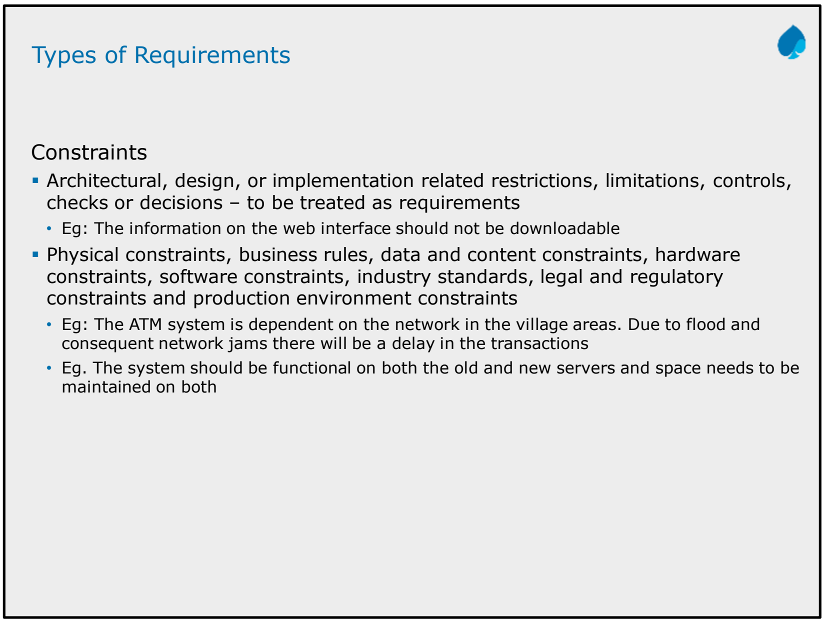
Types of Requirement

Interface (or External API) Requirements

- Requirements of interfaces to external systems
  - Eg: Online banking system- I/f with bank’s database, payment interface
- Specified using interface and protocol specifications
  - Eg: Bill payment through an SSL secured interface
- Inputs or outputs
  - Eg: Bill payment system- Inputs from service providers database Output to customer’s bank account and to service provider

The aggregate of means by which the users/ systems interact with the users/ systems-a particular machine, device, computer program or other complex tools.

The user interface provides means of:
Input, allowing the users to manipulate a system
Output, allowing the system to produce the effects of the users' manipulation.

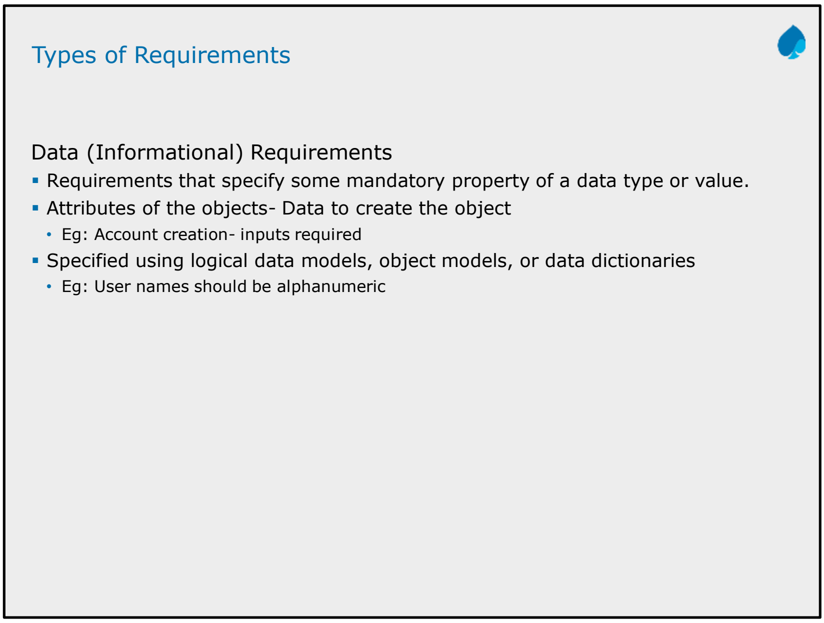


Types of Requirements

Constraints

- Architectural, design, or implementation related restrictions, limitations, controls, checks or decisions – to be treated as requirements
  - Eg: The information on the web interface should not be downloadable
- Physical constraints, business rules, data and content constraints, hardware constraints, software constraints, industry standards, legal and regulatory constraints and production environment constraints
  - Eg: The ATM system is dependent on the network in the village areas. Due to flood and consequent network jams there will be a delay in the transactions
  - Eg. The system should be functional on both the old and new servers and space needs to be maintained on both

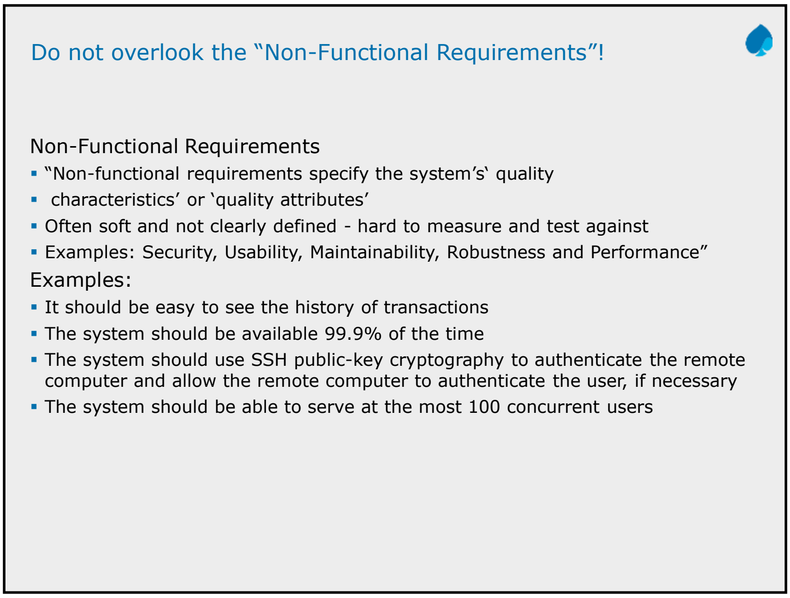
Project constraints have the potential to severely impact the ability to successfully provide a solution. Sometimes a constraint may look like an iceberg. On the surface, there may not appear to be much to it, but on close examination the iceberg may have a large potential impact. Maybe it will not be enough to sink the ship, but it may be enough to knock you way off course!



Types of Requirements

Data (Informational) Requirements

- Requirements that specify some mandatory property of a data type or value.
- Attributes of the objects- Data to create the object
  - Eg: Account creation- inputs required
- Specified using logical data models, object models, or data dictionaries
  - Eg: User names should be alphanumeric



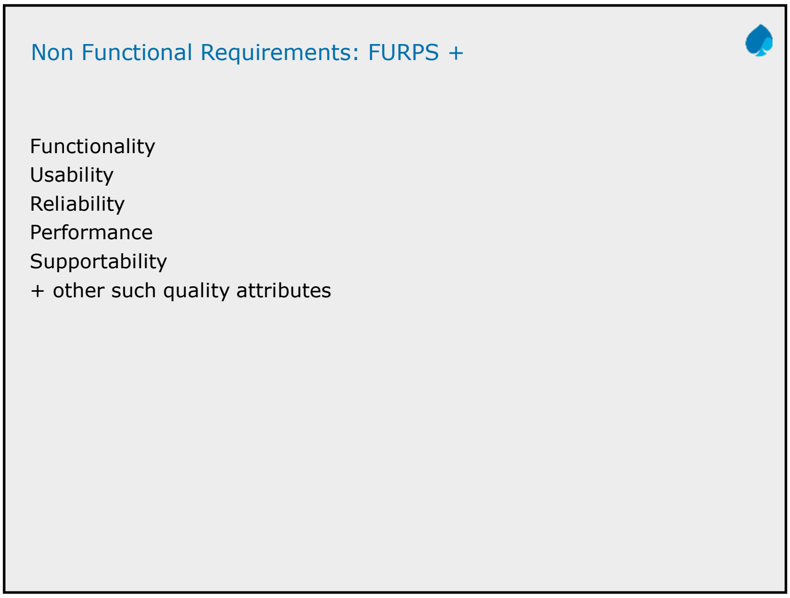
Do not overlook the “Non-Functional Requirements”!

Non-Functional Requirements

- “Non-functional requirements specify the system’s‘ quality
- characteristics’ or ‘quality attributes’
- Often soft and not clearly defined - hard to measure and test against
- Examples: Security, Usability, Maintainability, Robustness and Performance”

Examples:

- It should be easy to see the history of transactions
- The system should be available 99.9% of the time
- The system should use SSH public-key cryptography to authenticate the remote computer and allow the remote computer to authenticate the user, if necessary
- The system should be able to serve at the most 100 concurrent users



Non Functional Requirements: FURPS +

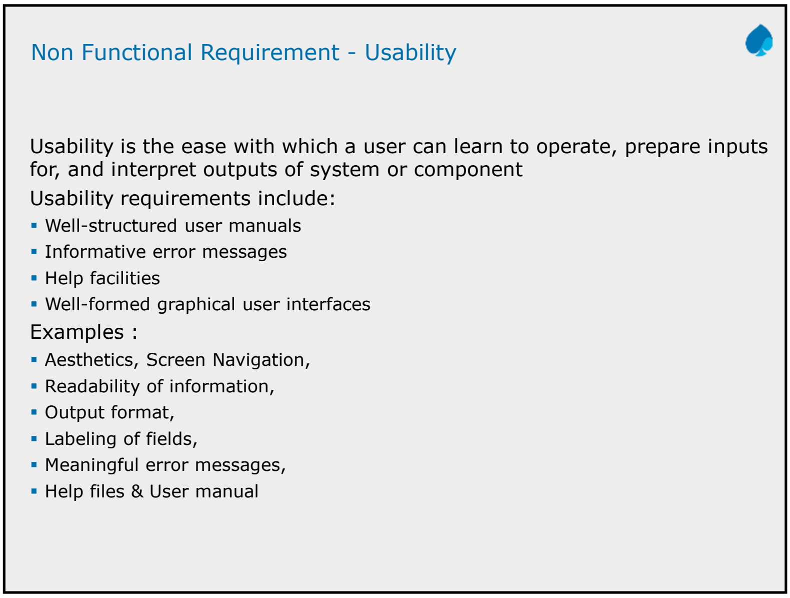
Functionality
Usability
Reliability
Performance
Supportability
+ other such quality attributes

Non Functional Requirements: FURPS +

1. **Functionality -** The degree to which the software satisfies stated needs as indicated by the following sub attributes: suitability, accuracy, interoperability, compliance, and security.
2. **Usability -** The degree to which the software is easy to use as indicated by the following aspects: understandability, learnability, operability.
3. **Reliability -** The amount of time that the software is available for use as indicated by the following aspects: maturity, fault tolerance, recoverability.
4. **Performance -** is concerned with characteristics such as throughput, response time, recovery time, start-up time, and shutdown time.
5. **Supportability -** is concerned with characteristics such as testability, adaptability, maintainability, compatibility, configurability, installability, scalability, and localizability.

The "+" in FURPS+ also helps us to remember concerns such as:

1. **Design requirements -** Design constraints for designing system, for example, if you specify that a relational database is required, that's a design constraint.
2. **Implementation requirements -** Specifies or constrains the coding or construction of a system. Examples might include required standards, implementation languages, and resource limits.
3. **Interface requirements -** Specifies an external item with which a system must interact, or constraints on formats or other factors used within such an interaction.
4. **Physical requirements -** Specifies a physical constraint imposed on the system , shape, size, or weight.



Non Functional Requirement - Usability

Usability is the ease with which a user can learn to operate, prepare inputs for, and interpret outputs of system or component

Usability requirements include:

- Well-structured user manuals
- Informative error messages
- Help facilities
- Well-formed graphical user interfaces

Examples :

- Aesthetics, Screen Navigation,
- Readability of information,
- Output format,
- Labeling of fields,
- Meaningful error messages,
- Help files & User manual

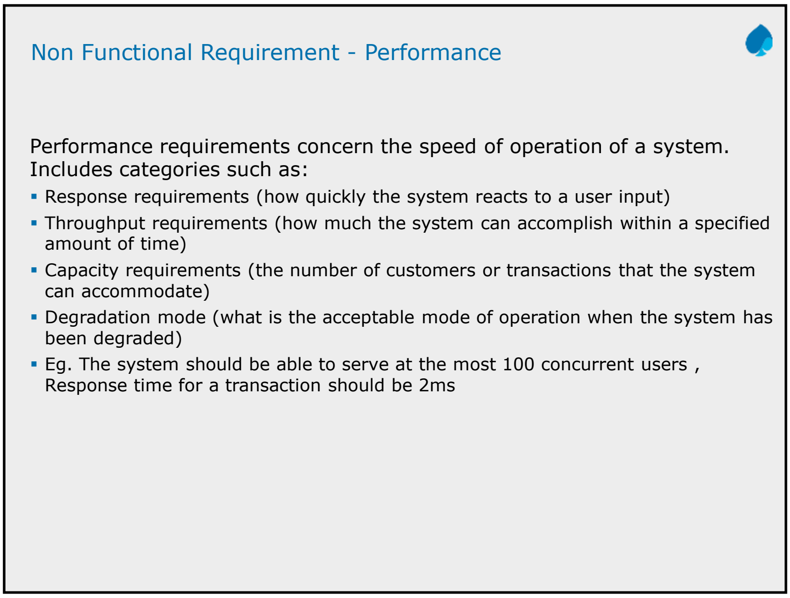
Non Functional Requirement - Reliability

Degree to which the system must behave in a user-acceptable fashion

Constraints on the run-time behaviour of the system

It includes the following:

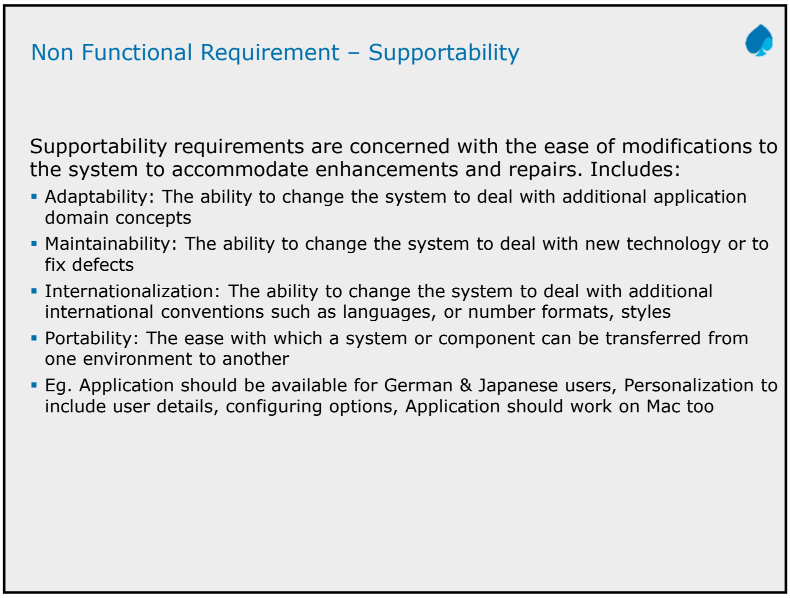
- Availability: Is the system available for service when requested by end users
- Failure rate: How often does the system fail to deliver the service as expected by end-users
- Accuracy: What precision is required in systems that produce numerical outputs
- Eg: System uptime should be 99.9%, Exception handling (no hanging!!)



Non Functional Requirement - Performance

Performance requirements concern the speed of operation of a system. Includes categories such as:

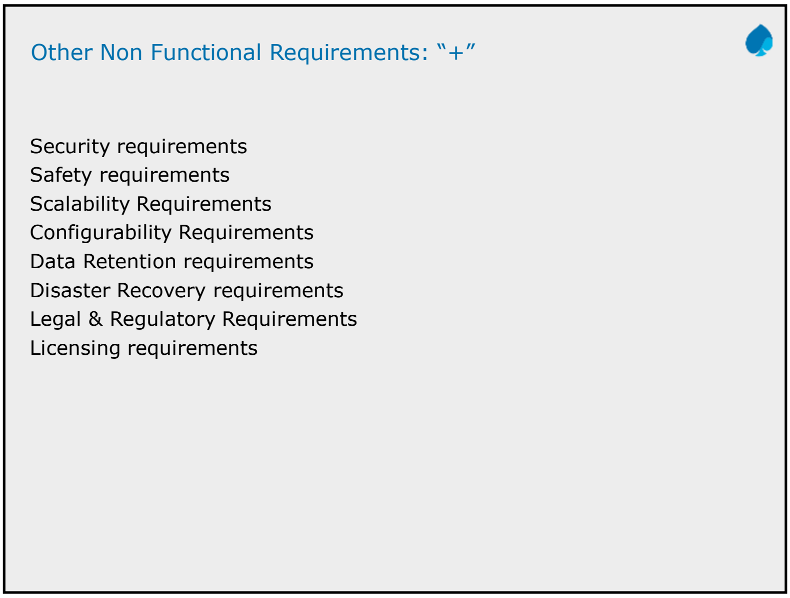
- Response requirements (how quickly the system reacts to a user input)
- Throughput requirements (how much the system can accomplish within a specified amount of time)
- Capacity requirements (the number of customers or transactions that the system can accommodate)
- Degradation mode (what is the acceptable mode of operation when the system has been degraded)
- Eg. The system should be able to serve at the most 100 concurrent users , Response time for a transaction should be 2ms



Non Functional Requirement – Supportability

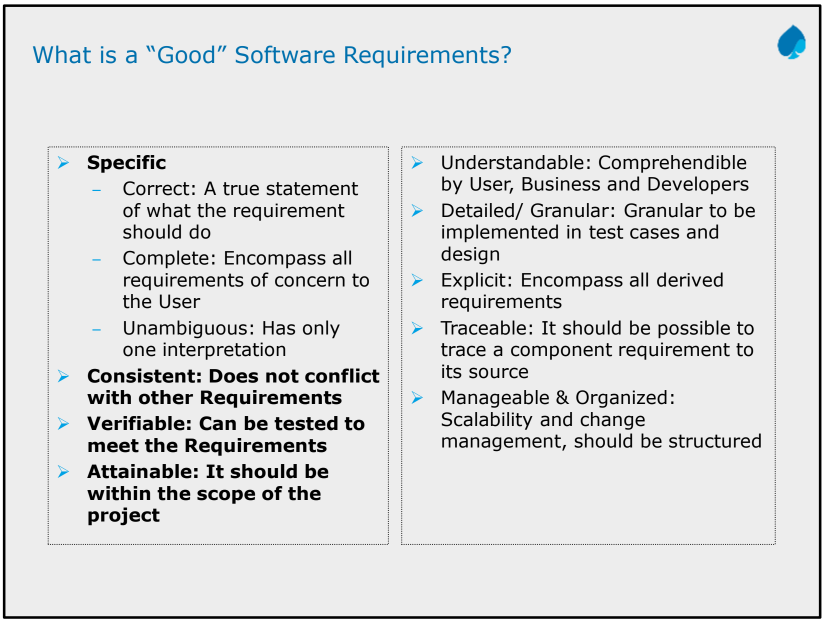
Supportability requirements are concerned with the ease of modifications to the system to accommodate enhancements and repairs. Includes:

- Adaptability: The ability to change the system to deal with additional application domain concepts
- Maintainability: The ability to change the system to deal with new technology or to fix defects
- Internationalization: The ability to change the system to deal with additional international conventions such as languages, or number formats, styles
- Portability: The ease with which a system or component can be transferred from one environment to another
- Eg. Application should be available for German & Japanese users, Personalization to include user details, configuring options, Application should work on Mac too



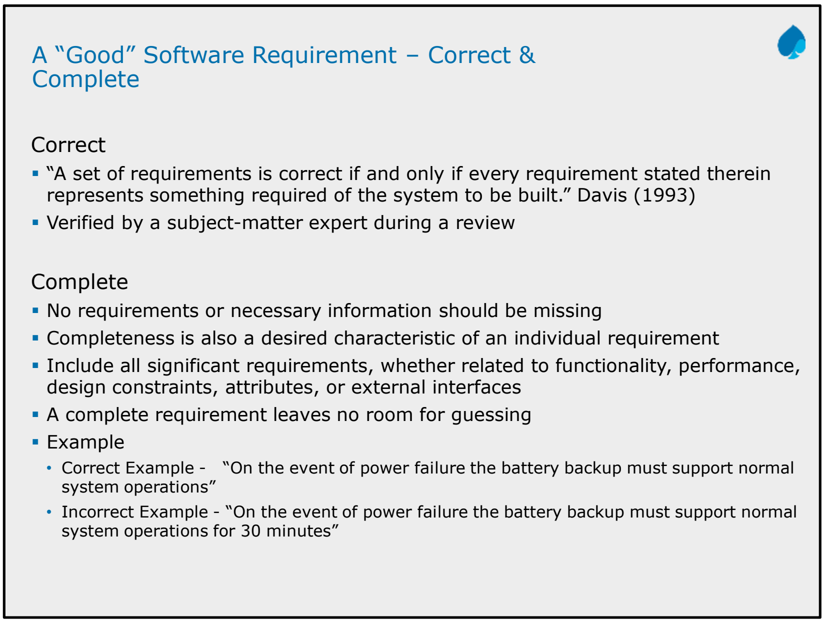
Other Non Functional Requirements: “+”

Security requirements
Safety requirements
Scalability Requirements
Configurability Requirements
Data Retention requirements
Disaster Recovery requirements
Legal & Regulatory Requirements
Licensing requirements



What is a "Good" Software Requirements?

- **Specific**
  - Correct: A true statement of what the requirement should do
  - Complete: Encompass all requirements of concern to the User
  - Unambiguous: Has only one interpretation
- **Consistent: Does not conflict with other Requirements**
- **Verifiable: Can be tested to meet the Requirements**
- **Attainable: It should be within the scope of the project**
- Understandable: Comprehendible by User, Business and Developers
- Detailed/ Granular: Granular to be implemented in test cases and design
- Explicit: Encompass all derived requirements
- Traceable: It should be possible to trace a component requirement to its source
- Manageable & Organized: Scalability and change management, should be structured



A “Good” Software Requirement – Correct & Complete

Correct

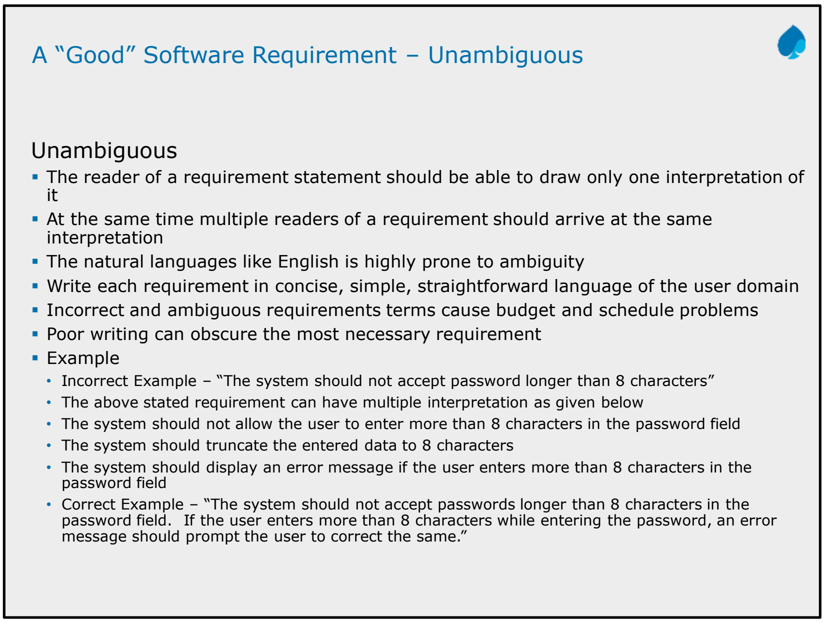
- “A set of requirements is correct if and only if every requirement stated therein represents something required of the system to be built.” Davis (1993)
- Verified by a subject-matter expert during a review

Complete

- No requirements or necessary information should be missing
- Completeness is also a desired characteristic of an individual requirement
- Include all significant requirements, whether related to functionality, performance, design constraints, attributes, or external interfaces
- A complete requirement leaves no room for guessing
- Example
  - Correct Example - “On the event of power failure the battery backup must support normal system operations”
  - Incorrect Example - “On the event of power failure the battery backup must support normal system operations for 30 minutes”

Characteristics of “GOOD” Requirements

1. **Correct :** Does every requirement state something required of the system? “A set of requirements is correct if and only if every requirement stated therein represents something required of the system to be built.” *Davis (1993)* It is not possible to determine if a requirement is correct simply by reading the requirement. The correctness is verified by a subject-matter expert during a review.
2. **Complete :** Does the set of requirements include all significant requirements, whether related to functionality, performance, design constraints, attributes, or external interfaces? Have the expected ranges of input values in all possible scenarios been identified and addressed? Have responses been included to both valid and invalid input values? Do all figures, tables, and diagrams include full labels, references, and definitions of all terms and units of measure?

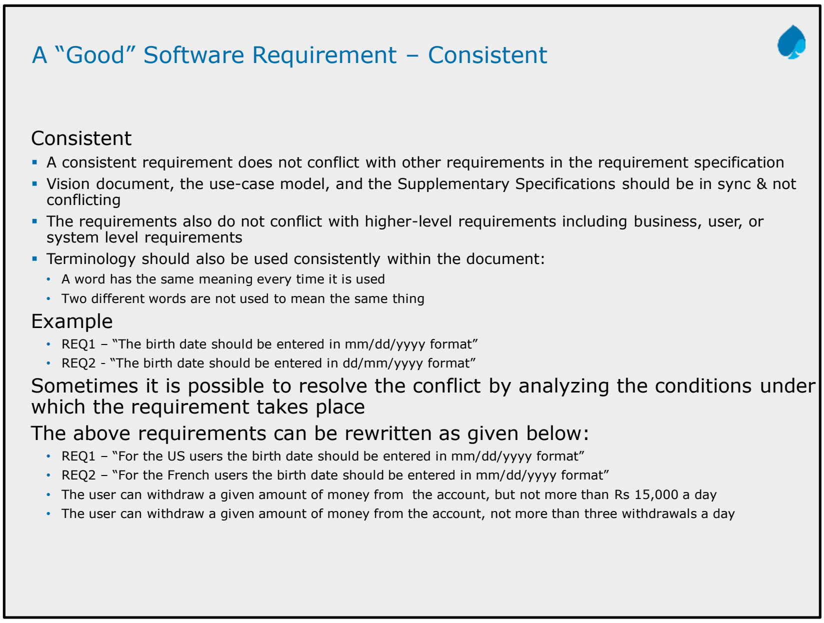


A "Good" Software Requirement – Unambiguous

Unambiguous

- The reader of a requirement statement should be able to draw only one interpretation of it
- At the same time multiple readers of a requirement should arrive at the same interpretation
- The natural languages like English is highly prone to ambiguity
- Write each requirement in concise, simple, straightforward language of the user domain
- Incorrect and ambiguous requirements terms cause budget and schedule problems
- Poor writing can obscure the most necessary requirement
- Example
  - Incorrect Example – "The system should not accept password longer than 8 characters"
  - The above stated requirement can have multiple interpretation as given below
  - The system should not allow the user to enter more than 8 characters in the password field
  - The system should truncate the entered data to 8 characters
  - The system should display an error message if the user enters more than 8 characters in the password field
  - Correct Example – "The system should not accept passwords longer than 8 characters in the password field. If the user enters more than 8 characters while entering the password, an error message should prompt the user to correct the same."

3. **Unambiguous*:*** Each requirement statement should have only one interpretation, and each requirement should be specified in a coherent, easy-to-understand manner.



A "Good" Software Requirement – Consistent

Consistent

- A consistent requirement does not conflict with other requirements in the requirement specification
- Vision document, the use-case model, and the Supplementary Specifications should be in sync & not conflicting
- The requirements also do not conflict with higher-level requirements including business, user, or system level requirements
- Terminology should also be used consistently within the document:
  - A word has the same meaning every time it is used
  - Two different words are not used to mean the same thing

Example

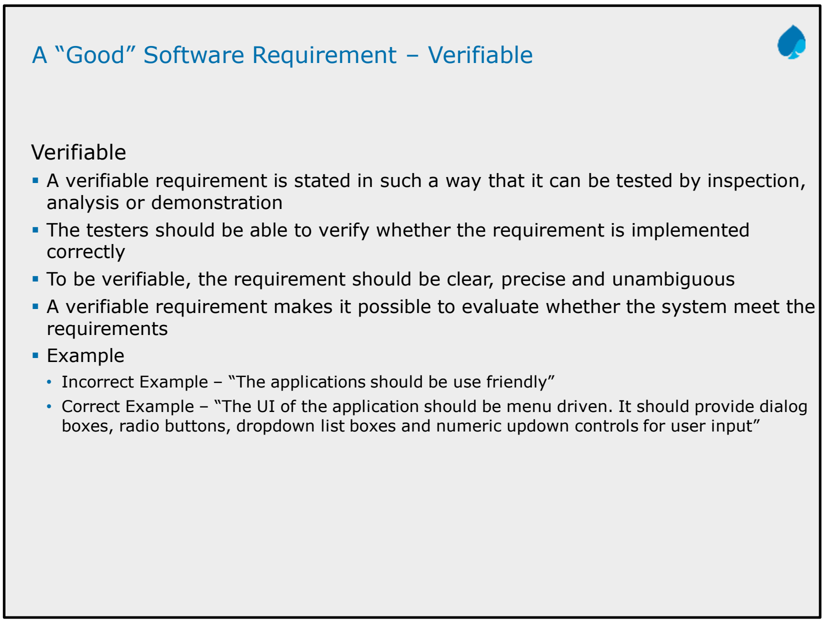
- REQ1 – "The birth date should be entered in mm/dd/yyyy format"
- REQ2 - "The birth date should be entered in dd/mm/yyyy format"

Sometimes it is possible to resolve the conflict by analyzing the conditions under which the requirement takes place

The above requirements can be rewritten as given below:

- REQ1 – "For the US users the birth date should be entered in mm/dd/yyyy format"
- REQ2 – "For the French users the birth date should be entered in mm/dd/yyyy format"
- The user can withdraw a given amount of money from the account, but not more than Rs 15,000 a day
- The user can withdraw a given amount of money from the account, not more than three withdrawals a day

4. **Consistent :** Is it internally consistent, with no subset of individual requirements described which are in conflict? (Examples: Vision document, the use-case model, and the Supplementary Specifications.)

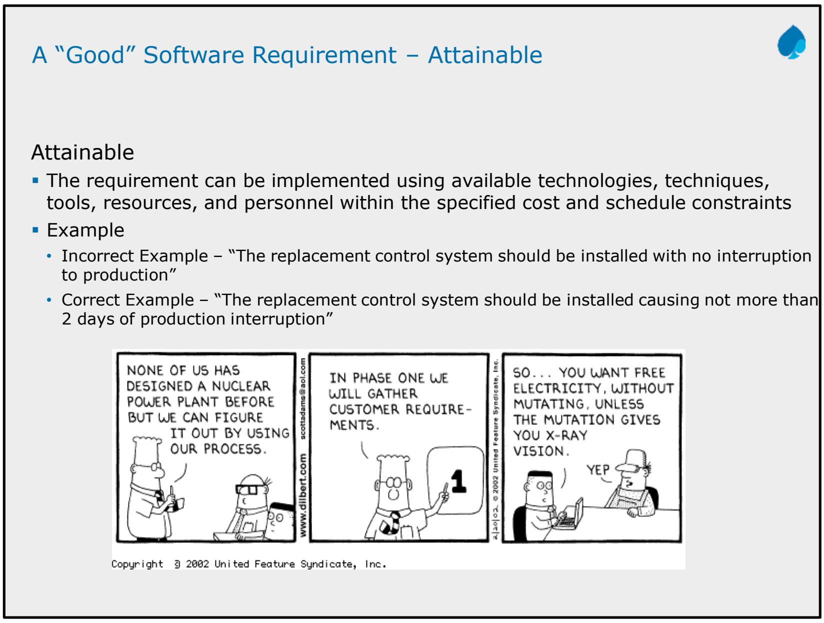


A "Good" Software Requirement – Verifiable

Verifiable

- A verifiable requirement is stated in such a way that it can be tested by inspection, analysis or demonstration
- The testers should be able to verify whether the requirement is implemented correctly
- To be verifiable, the requirement should be clear, precise and unambiguous
- A verifiable requirement makes it possible to evaluate whether the system meet the requirements
- Example
  - Incorrect Example – "The applications should be use friendly"
  - Correct Example – "The UI of the application should be menu driven. It should provide dialog boxes, radio buttons, dropdown list boxes and numeric updown controls for user input"

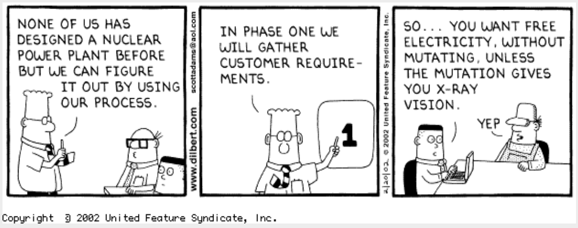
5. **Verifiable :** Is every requirement stated verifiable? Is there some finite cost-effective process with which a person or machine can check that the software product meets the requirement?



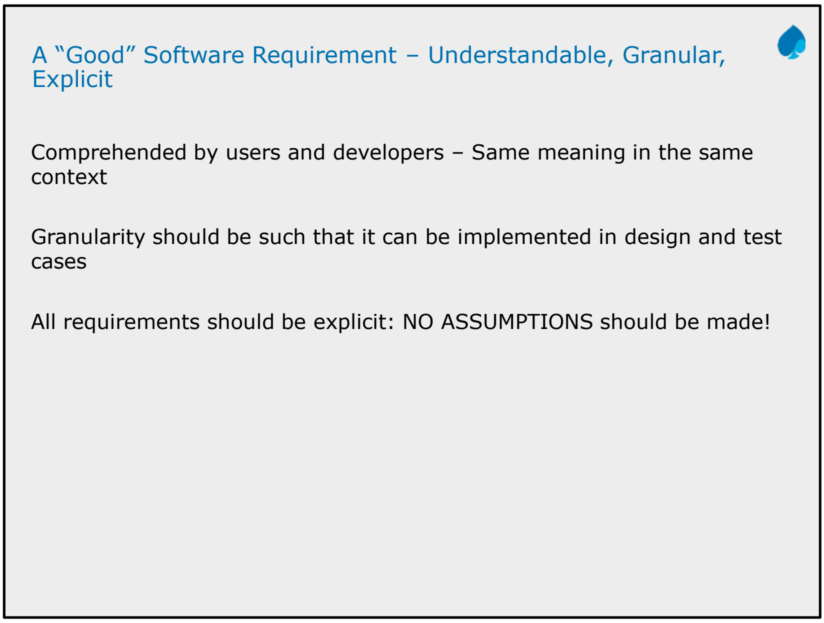
A “Good” Software Requirement – Attainable

Attainable

- The requirement can be implemented using available technologies, techniques, tools, resources, and personnel within the specified cost and schedule constraints
- Example
  - Incorrect Example – “The replacement control system should be installed with no interruption to production”
  - Correct Example – “The replacement control system should be installed causing not more than 2 days of production interruption”


NONE OF US HAS DESIGNED A NUCLEAR POWER PLANT BEFORE BUT WE CAN FIGURE IT OUT BY USING OUR PROCESS.
IN PHASE ONE WE WILL GATHER CUSTOMER REQUIREMENTS.
1
SO... YOU WANT FREE ELECTRICITY, WITHOUT MUTATING, UNLESS THE MUTATION GIVES YOU X-RAY VISION.
YEP
www.dilbert.com
scottadams@aol.com
© 2002 United Feature Syndicate, Inc.
Copyright © 2002 United Feature Syndicate, Inc.

6. **Attainable :** The requirement should be doable within existing constraints such as time, money, and available resources. An attainable requirement can be:
   - Met using existing technology
   - Achieved within the budget
   - Met within the schedule
   - Is something the organization has the necessary skills to utilize



A “Good” Software Requirement – Understandable, Granular, Explicit

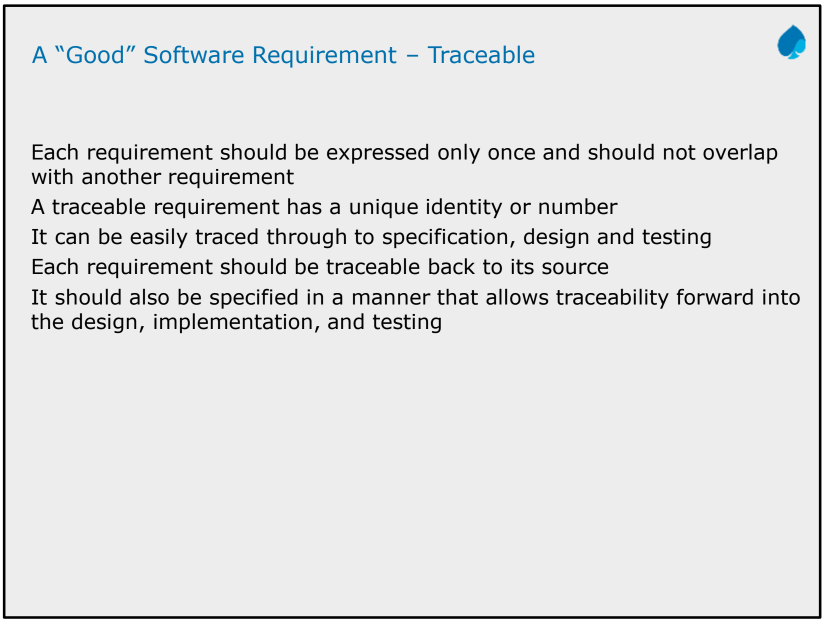
Comprehended by users and developers – Same meaning in the same context

Granularity should be such that it can be implemented in design and test cases

All requirements should be explicit: NO ASSUMPTIONS should be made!

7. **Understandable**: Comprehended by users and developers the same way, with the same intent.

Too often, requirements (especially derived requirements) are specified by developers who use their technical jargon that is not understandable to other stakeholders, especially customers, users, and managers. But individual requirements should be oriented around the needs of the customers and users if they are to be understandable and validable :
Is each requirement phrased in the language of the customer and user organizations?
Does each requirement avoid the technical jargon of the development organization?



A “Good” Software Requirement – Traceable

Each requirement should be expressed only once and should not overlap with another requirement

A traceable requirement has a unique identity or number

It can be easily traced through to specification, design and testing

Each requirement should be traceable back to its source

It should also be specified in a manner that allows traceability forward into the design, implementation, and testing

8. Traceable :

a. Does each requirement have a clear identifier?

b. Is it distinguishable from non-essential statements in the requirements set?

c. Is the origin of each requirement clear?

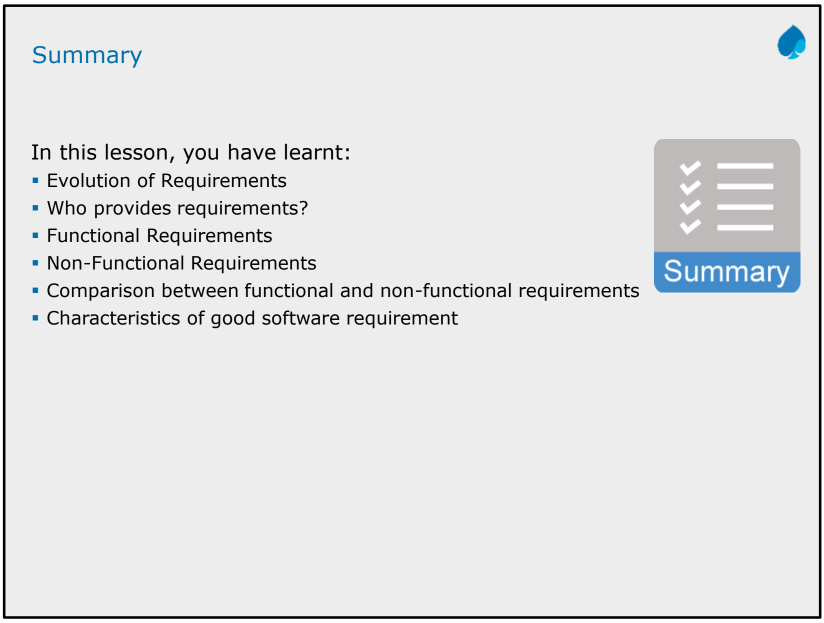
d. Is backward traceability maintained by explicitly referencing earlier artifacts?

e. Is a reasonable amount of forward traceability maintained to artifacts spawned by the requirements set? For example, test cases.

A "Good" Software Requirement – Manageable & Organized

Keep sentences short, numbered and specific
Avoid Paragraphs
Define terms in a glossary
Document in a hierarchical, structured form
Avoid redundancies

SR 1 – The User can access the most recent invoice through the email notification or the website
SR 1.1 – The User will be provided a link on the hope page to view 'Most Recent Invoice'
SRS 1.1.1 - On clicking the link will be taken directly to the invoice generation page
SR 1.2 – The User can also access the link from the email notification received by the User to the preferred email ID on or after the Invoice Generation date
SR1.2.1 - While clicking the link via email the User will have to login to the system, and on successful login will be taken directly to the invoice generation page



Summary

In this lesson, you have learnt:

- Evolution of Requirements
- Who provides requirements?
- Functional Requirements
- Non-Functional Requirements
- Comparison between functional and non-functional requirements
- Characteristics of good software requirement

Summary

Answers:

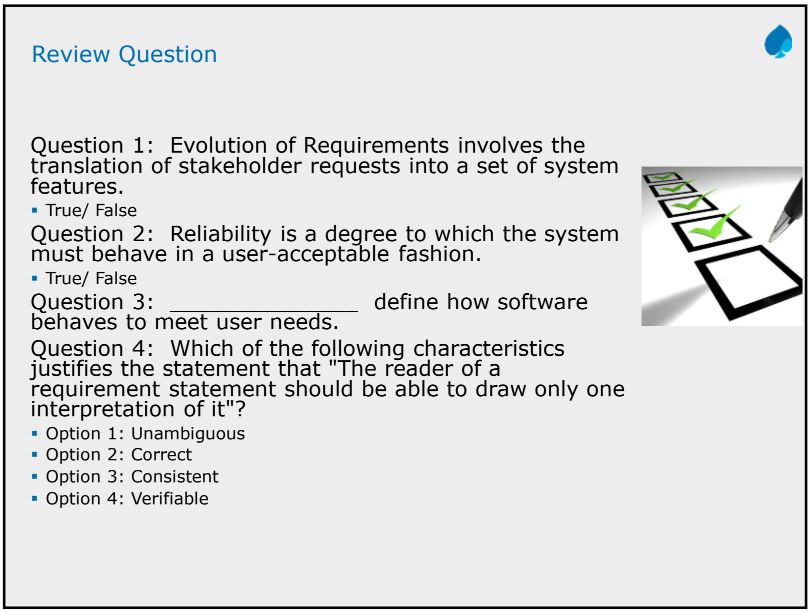
Question1:
True

Question2:
True

Question3:
Functional Requirements

Question4:
Unambiguous

Question5:
Internationalization



Review Question

Question 1: Evolution of Requirements involves the translation of stakeholder requests into a set of system features.

- True/ False

Question 2: Reliability is a degree to which the system must behave in a user-acceptable fashion.

- True/ False

Question 3: _______________ define how software behaves to meet user needs.

Question 4: Which of the following characteristics justifies the statement that "The reader of a requirement statement should be able to draw only one interpretation of it"?

- Option 1: Unambiguous
- Option 2: Correct
- Option 3: Consistent
- Option 4: Verifiable

Review Question

Question 5: Which of the following non functional requirements can ensure that the application should be available for German & Japanese users?

- Option 1: Internationalization
- Option 2: Security
- Option 3: Portability
- Option 4: Adaptability