Requirements Engineering

Education & Research

Confidential Information

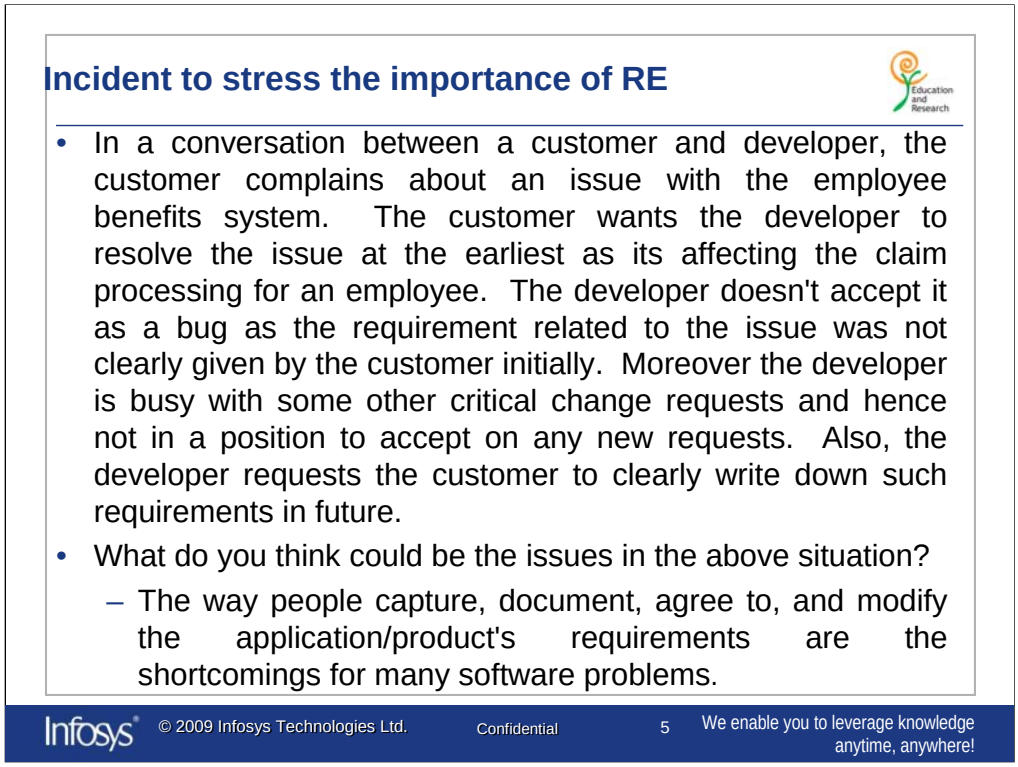
- *This Document is confidential to Infosys Technologies Limited. This document contains information and data that Infosys considers confidential and proprietary ("Confidential Information").*
- *Confidential Information includes, but is not limited to, the following:*
  - *Corporate and Infrastructure information about Infosys;*
  - *Infosys' project management and quality processes;*
  - *Project experiences provided included as illustrative case studies.*
- *Any disclosure of Confidential Information to, or use of it by a third party, will be damaging to Infosys.*
- *Ownership of all Infosys Confidential Information, no matter in what media it resides, remains with Infosys.*
- *Confidential information in this document shall not be disclosed, duplicated or used – in whole or in part – for any purpose without specific written permission of an authorized representative of Infosys.*

Course Objective

- Understand the Importance of Requirements Engineering
- Understand the need for Systematic Approach to Requirements Gathering
- Understand the characteristics of good requirements
- To know the components of requirements
- Understand Influx BPE Lifecycle

Session Plan

- Importance of Requirements Engineering
- Need for Systematic Approach to Requirements Gathering
- Characteristics of good requirements
- Components of requirements
- Requirements Engineering framework
- Influx BPE Lifecycle - Overview



Below is the actual conversation between the customer and developer.

Hi John? This is Richard from HR Employee Relations team. We're having a problem with the Employee Compensation and Benefits system your team has developed for us. An employee just had a Life Event where she wants to add her child for medical benefits. And the system is not accepting the child details. Can you resolve this at the earliest?

"Is she married and had a child, then what is the problem?" asked John.

"No, actually she adopted the child without being married" replied Richard. "There seems to be a problem accepting child without marriage".

"I don't see any document or mentioned anywhere about this possibility when we were discussing about the system" John said.

Richard said, "I figured you knew that people could legally adopt child". Anyway, we have to get this rectified out by this Friday, or the impacted employee will not be able claim for medical benefits for the child. Can you fix the bug by then?"

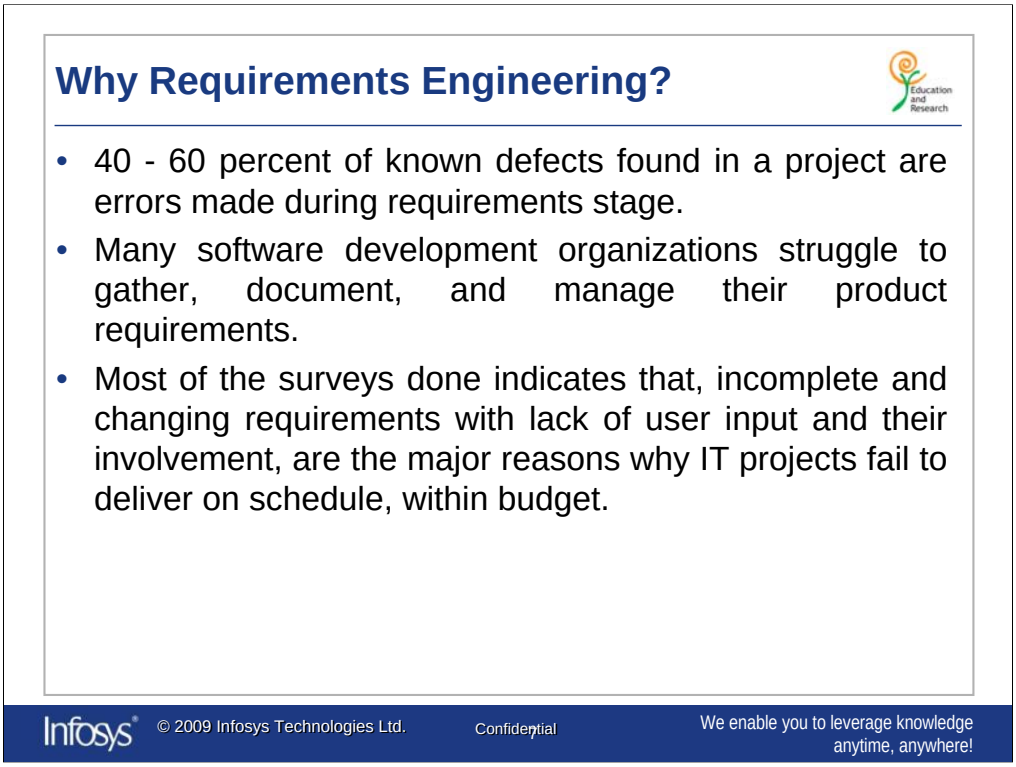
"It's not a bug! I never knew you needed this capability. I'm busy on the new Professional Development system. I think I have few other prioritized change requests for the Employee Compensation and benefits system here, too. I can look at a fix later, but not within a month. Next time, Please have these requirements documented and signed off earlier."

The above scenario can be a common scenario if requirements/needs are not gathered completely. This leads to the necessity of a process, so Requirements Engineering is the solution.

Requirements Engineering

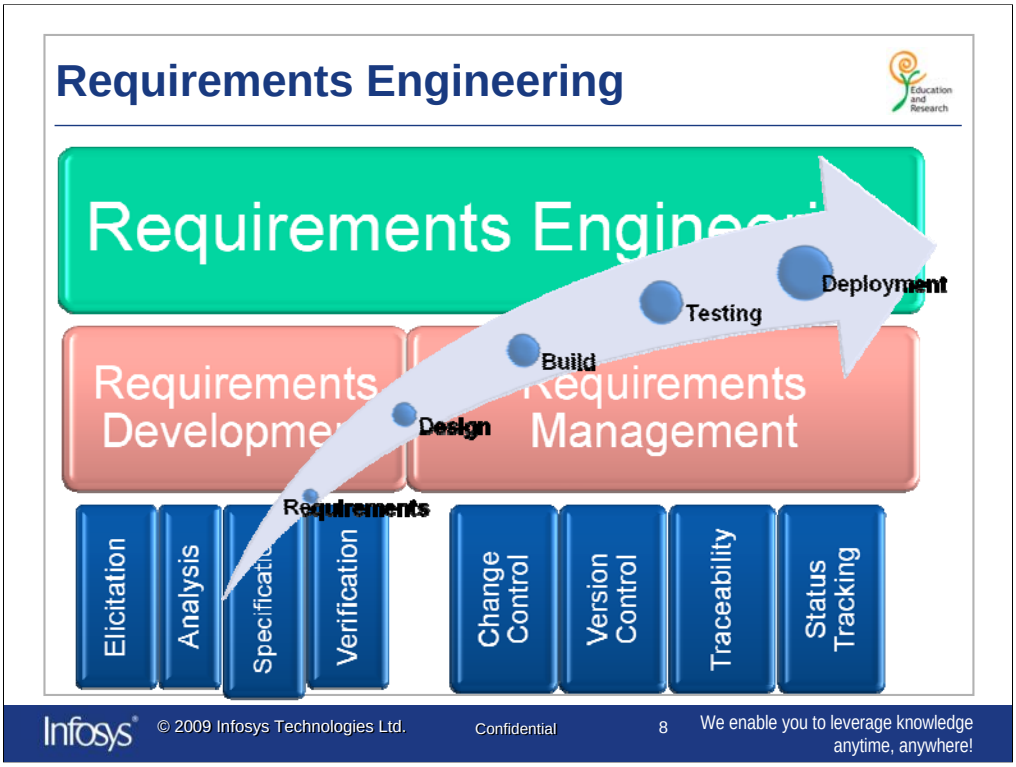
"*Requirements Engineering involves all life-cycle activities devoted to identification of user requirements, analysis of the requirements to derive additional requirements, documentation of the requirements as a specification, and validation of the documented requirements against user needs, as well as processes that support these activities***."**

- Department of Defense. *Software Technology Strategy.* Dec, 1991



We, as an industry, cannot bear the consequences of ineffective requirements engineering. The cost of inaccurate, misunderstood, and not signed off requirements affects all of us in terms of time, money and opportunities lost. The result will be of total chaos in terms of frustration, confusion, mistrust, higher cost, lack of quality, overtime, over budget, a general lack of understanding and incapability to handle issues.

The interests of all the known stakeholders in a software system, coincide more in requirement engineering phase than in other phases. This leads to exciting products, happy developers and delighted customer, if requirements are handled well, else it can become the source of frustration, friction and misunderstanding that can undermine product's business value and quality. As requirements is the base for both software development and project management activities, all stakeholders should follow an effective requirements engineering process.



Requirements Engineering constitutes both Requirements Development and Requirements Management.

<u>Requirements Development</u>:

• ***Requirements Elicitation and Analysis*** –

• Requirements discovery – Process of stakeholders interaction of the system in gathering their requirements leading to realization of lot of documentation and knowledge of the system.

•Requirements Classification and Organization – this activity helps in formulating unstructured requirements, group these requirements and arrange them into coherent groups or clusters.

• Prioritization and negotiation – Where multiple stakeholders are involved, requirements will conflict. This activity is concerned with prioritizing requirements, finding and resolving the conflicts through negotiation.

• Requirements documentation – These requirements are documented to go through another iteration. Formal or informal requirement documents may be produced.

•***Requirements specifications***

- It is the final work product produced by the requirements engineer. It serves as the foundation for subsequent Software Engineering activities. It describes the function and performance of a computer-based (software) system and the constraints that will govern its development.

Contd.,

•***Verification/Validation of the requirements from all the stakeholders.***

-Following checks should be carried out for requirement validation.

- Validity Checks – Validate by different stakeholders to come to the same conclusion of the functionality.

- Consistency Checks – That there should not be any conflicting or contradictory statements.

- Completeness Checks – include all requirements defining all functions and constraints intended by the system user.

- Realism Checks – Can it really be implemented taking into account of the budget and time.

- Verifiability – meaning should be able to write a set of tests that can demonstrate the intended function and output.

We can use different validation techniques such as

•Requirement reviews

•Prototyping

•Test-case generation.

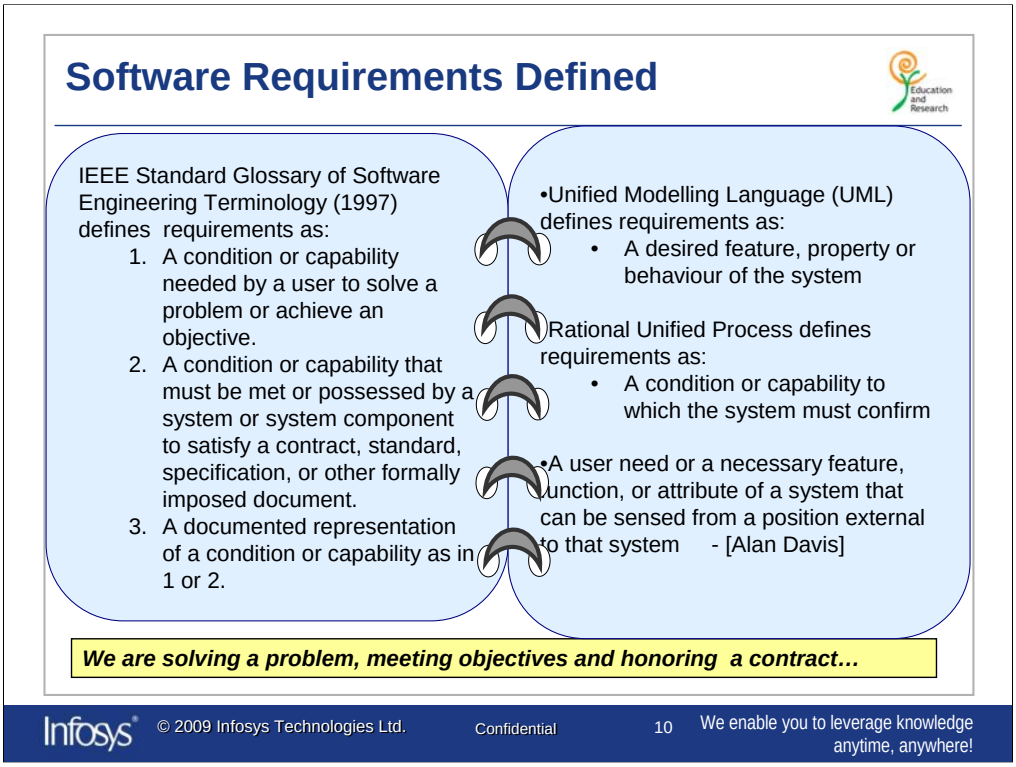
Requirements Management:

• Change control of the base-lined Requirements

• Version controlled

• Traceability from requirements to the actual product/module

• Status tracking to the finished deliverable/product.



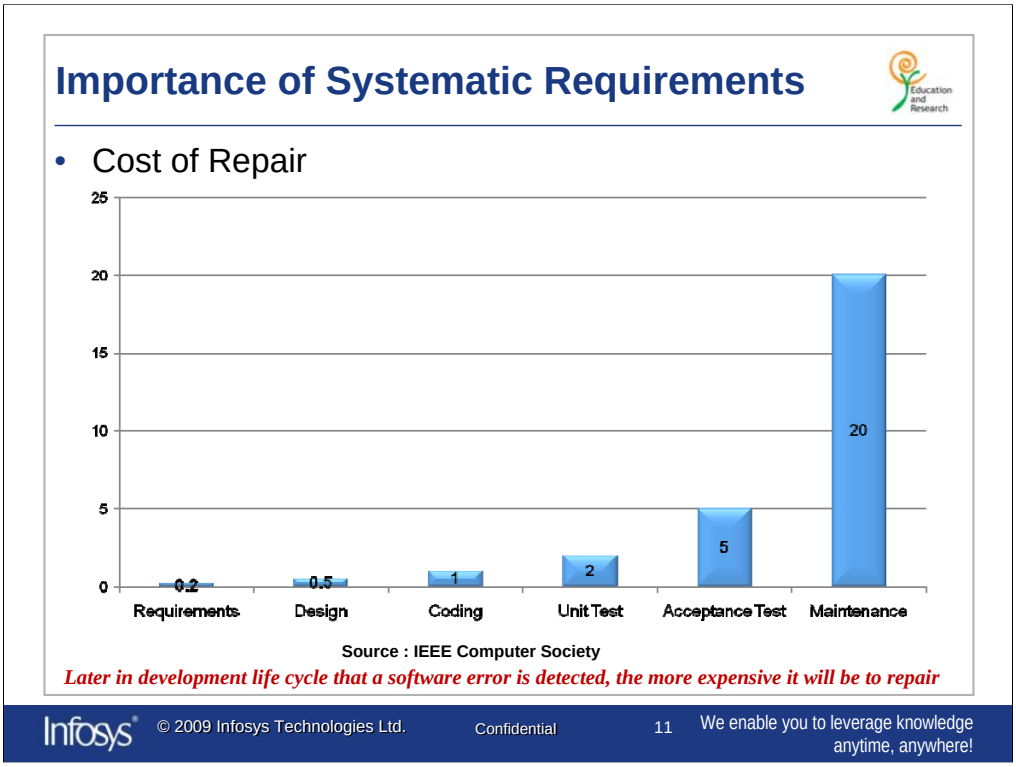
What are requirements?

Requirements are captured at the early stage of any software/system development as to what should be implemented. It can be constraints or standards to follow on the system's development. Therefore a requirement might describe:

- •A user-level facility (e.g. identity the customer search based on the phone call's ANI (Automatic Number Identification) information)
- •A very general system property (e.g. Medical records information is never made available without authorization)
- •A very specific constraint on the system (e.g. poll the Switchboard every millisecond for availability of a free line to route the call)
- •A contract between the customer and the development team on what customer intends to have as a product/solution. The resulting product should have the functionality and attributes that is asked for.

Software Requirements Definitions

Several definitions exists for the term "requirements" shows that there are no clear definition. The real requirements are in people's minds. Any documented form of the requirements is just a model or representation (Lawrence 1998). Hence all project stakeholders need to ensure a common understanding of the terms used to define these requirements.



Cost of Repair

A software error detected is expensive to repair, the later in the development life cycle.

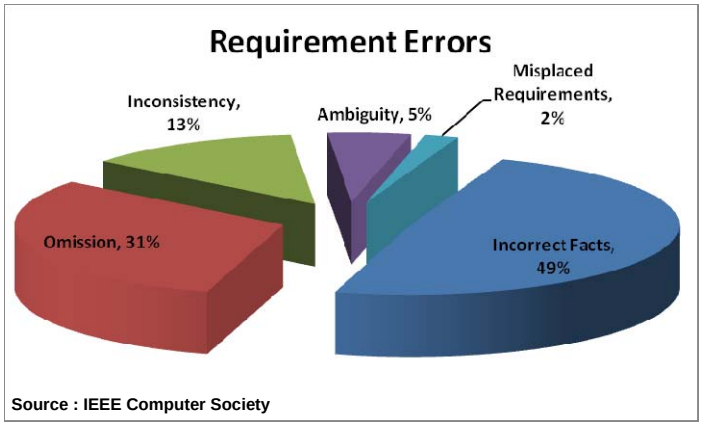
The Standish Group has found that even though projects are being delivered on time and within budget, the statistics for delivering requirements and meeting customer expectations are decreasing significantly. Let's see some statistics to stress the importance of effective Requirements Engineering based on the studies across the IT industry.

Supporting experimental evidence – In the early 1970s, three companies, GTE, TRW and IBM performed independent studies of this phenomenon. Although each of the three companies was apparently unaware of the other two, they all reached roughly the same result as given in the above slide.

If we assign unit cost for an effort to detect and repair an error during unit-test stage, it has been found that the cost to detect and repair an error during the requirements stage was between tenth and fifth than during maintenance would be ten times to detect and repair. All together that implies as much as a hundred to one ratio between the requirements and maintenance stages.

Importance of Systematic Requirements

- Typical Requirements Errors



Source : IEEE Computer Society

Typical Requirements Errors

Errors made in requirements specifications are typically incorrect facts, omissions, inconsistencies, and ambiguities.

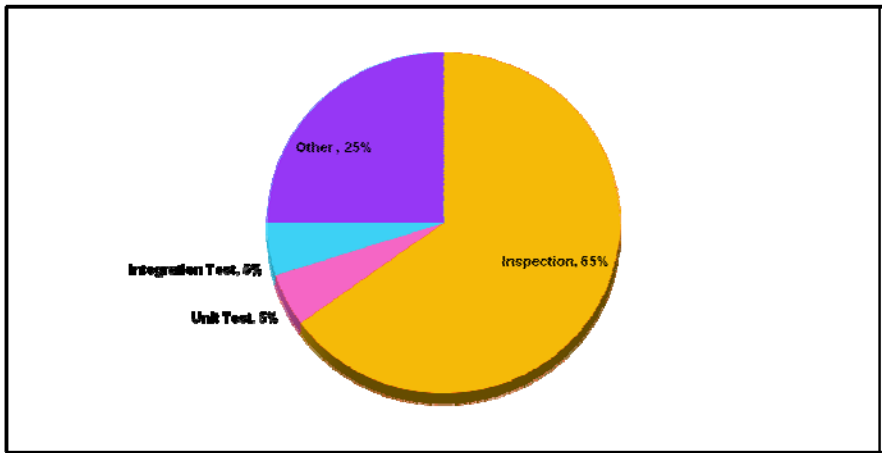
Supporting Experimental evidence – The Naval research Laboratory has been performing ongoing research in software development techniques since the mid-1970s. They are using the Navy A-7E aircraft's operational flight program as a realistic test case to demonstrate feasibility of new concepts. Following observations were made during the requirements phase. The data shows that 77% of all requirements errors on the A-7E project were non-clerical. The above slide gives the distribution of these non-clerical errors by category or error: 49% incorrect fact, 31% omission, 13% inconsistency, 5% ambiguity, and 2% misplacement(1981 IEEE Computer Society Press).

As part of Defect Prevention activity at different milestones in the project, we identify such root causes of errors and come up with corrective action.

Now we might conclude that, there are many requirement errors that are done, and these errors are un-noticed till detected, and we know what kind they are, but the reason for their latency may be due to the fact that errors are too difficult to detect, which leads us to the following hypothesis that "Requirements errors can be detected".

Importance of Systematic Requirements

- Requirement Errors can be detected



Need to spend more time analysing non-executable forms of s/w (Req, Design docs). Computer can easily find errors in programs.

Requirement errors can be detected

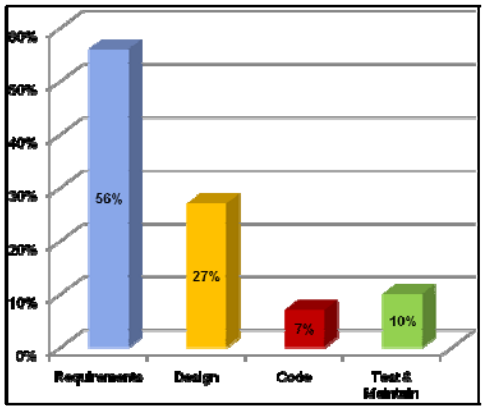
Supporting Experimental evidence –

Data from Bruggere's data show that the most effective way of finding an error in software is to inspect it. Around 65% of requirements error can be found by inspecting as given in the slide above. Basili and Weiss's data shows that 33% of requirements errors in the A-7E specification were detected by manual review.

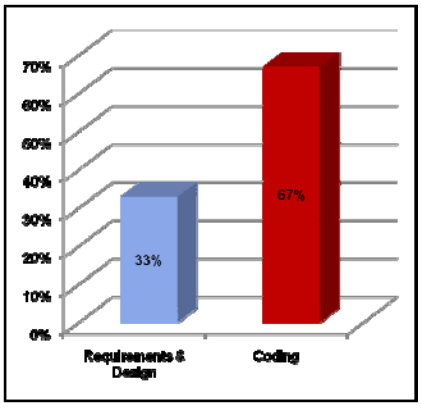
The probable errors in requirements are significant:

- The final software may be unsatisfactory to the real needs of the user.
- Ambiguous requirements cause friction between customers and developers, wasting resources leading to lawsuits.
- Impossible to test the software for its intended requirements or purpose.
- Building wrong system leading to waste of time and money.

Distribution of Errors in Project Lifecycle



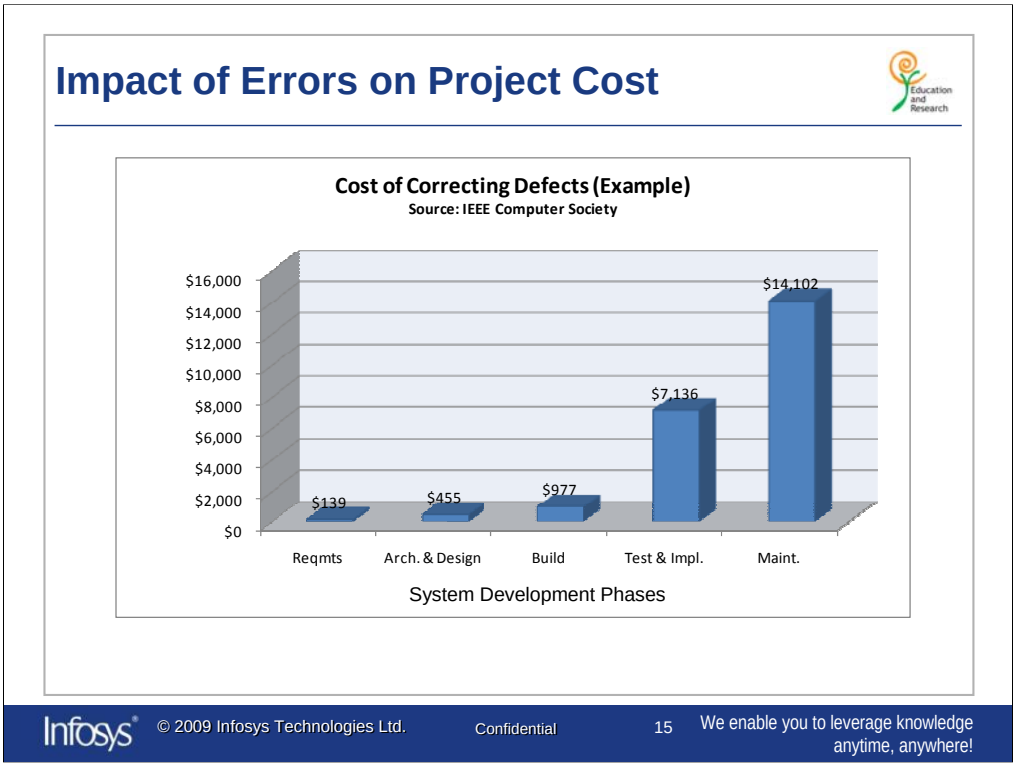
Source: 'Writing Testable Requirements' by Dick Bender



Source: Infosys Pride PCB data – Data given is for Type 4(Large size) development projects

The above slide shows the distribution of errors in project life cycle based on the study by Dick Bender (in his book "Writing testable requirements"). This shows that most of the errors are made at the requirements phase and these gets passed on to other phases in the project life cycle. It depicts the importance of detection and elimination of faults during the requirements phase of the software development. With early detection, the project team can mitigate faults and risks to the project as discussed in the previous slide.

Latest PCB data from Pride depicts that the major Error detection happens during "Build" or during "Requirements & Design", this only show that developers of the project are identifying the defects but enter the data into the IPM+. So it makes it more important that the Project Lead/Manager has to be very cautious as to which category the team entering data against.

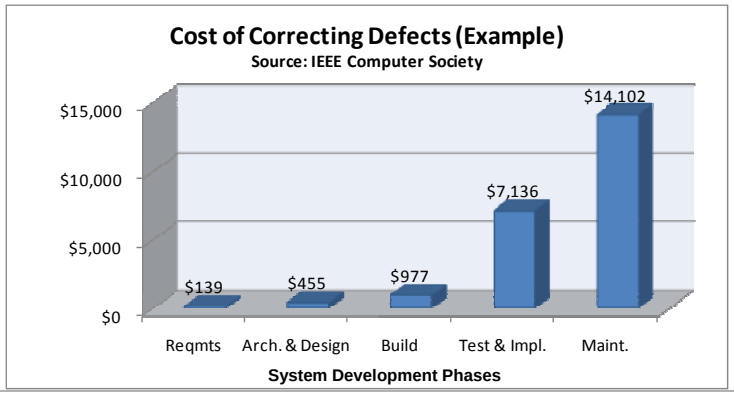


The above figure gives a picture of the "Cost of correcting defects" at various stages of project life cycle (**Source: B.Boehm and V.Basili, "Software Defect reduction Top 10 list", IEEE Computer Society**). This industry average is used as a baseline for arriving at cost savings.

Addressing defects earlier during the requirements and design phase ensures lower total cost of correcting defects. As defects are detected in later phases such as testing and maintenance the cost for remediation is exponentially higher.

Exercise

- If on an average just 20 requirements defects per project get passed on to implementation phase how much would these errors cost for a company running 40 projects a year ? **$ 5,708,800**

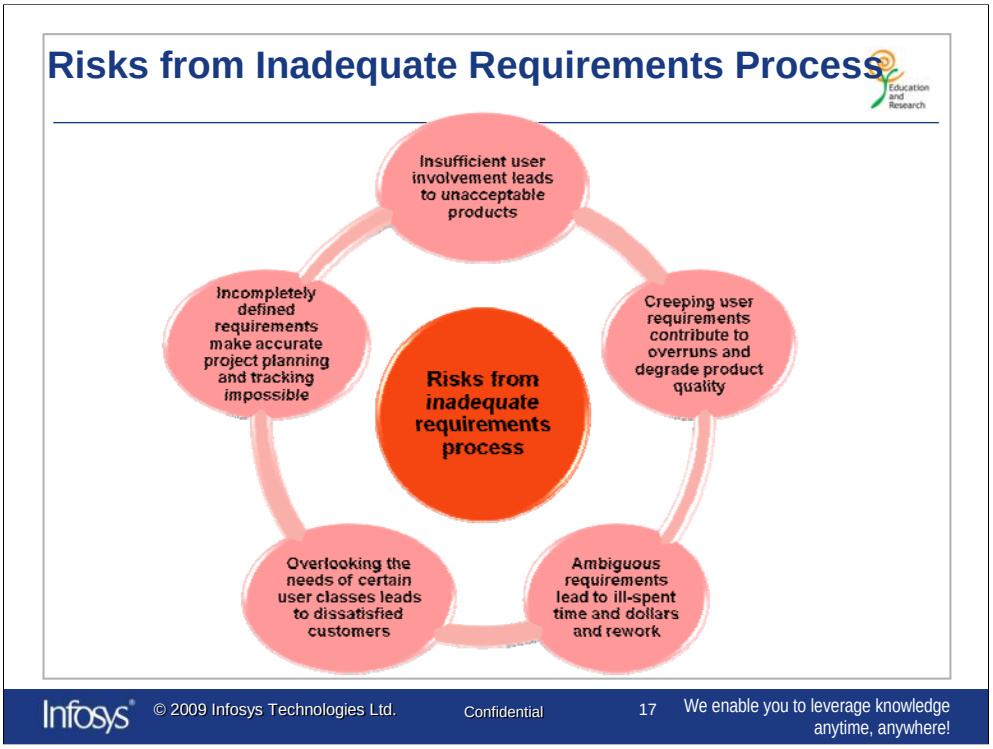


Using the statistics on the “Cost of correcting defects” as seen in the previous page, let's do a simple exercise as given in above figure. The $ given against each bar in the bar chart is $/Defect Corrected.

Answer:

20 requirements reaching Test & Implementation for 1 project, Let’s say A = 20 x 7,136

So for 40 projects 20 reqts reaching Test & Impl is B = A x 40 = 5,708,800 USD



Risks from Inadequate Requirements Processes

Project teams that neglect the requirements processes do so at their own peril. If requirements engineering practices are not followed then the project success stands to pose a risk, where "success" is defined as product which satisfies users expectations of functionality and quality at agreed on cost and timeliness. A few of these requirements risks are discussed below.

1. Insufficient user involvement leads to unacceptable products

The pharmaceutical customer produces drugs for infants. If any particle in the drug is below or above expected limit, then the system should raise a deviation. Root cause and corrective action has to be analyzed for the same. The customer gives this requirement to the business analyst (BA). BA stresses on meeting the users to get more details and observe the tasks performed by the users. But the customer disagrees to this as it's difficult to get the user's time and also that he had already given the requirements.

Gathering requirements and guaranteeing their quality is very important that needs to be sensitized to customers.

In one case, where the requirement is for building a system related to Stock Exchange. The BA has to go the trader's desk to observe the activities performed by the user and he cannot ask any questions to him. Once he interrupted and the user commented that he had made him loose some dollars.

In such cases gaining direct contact of those people who use the product is not easy, and usually customer surrogates are not a recommended alternative to get the requirements what other users really want. There is no substitute for involving some representative users directly with the development team early in the project and keeping them engaged throughout the development phase.

2.Scope creep contributes to product quality degrade and overruns

During development phase, requirements continue to evolve leading to scope creep, which in turn exceed the budget and their planned release dates. Not understanding project's requirements will lead to unrealistic size and complexity, adding risks and degrading developer's productivity, more over if requirements adds in (scope creep) leads to even larger problem. The problem lies with new changes to requirements and how developers react to those changes or addition of requirements.

To keep scope creep under control, have a clear statement of understanding of the project's vision, the scope, realistic objectives its limitations and the criteria for success. Use this statement as the reference frame against which all proposed new features or requirements changes are evaluated. A well-defined change control process that include impact analysis of each proposed change will make the stakeholders to an educated decision regarding what changes to accepts and their associated costs, time, needed resources, or feature trade-offs.

Slowly the architecture might crumble with changes seeping into the product being developed. Patch work will degrade the maintenance and understandability of the programs. Introduction of extra code will be the reason for violating design principles for a strong cohesion and loose coupling. If you can identify early on those features that are likely to involve more changes over time, you can develop a robust architecture for those portions of the system, to better accommodate the evolution. Quality degradation can be prevented if changes can be done through design than through code patches.

3. Ambiguous requirements lead to ill-spent time, dollars and rework

If multiple readers arrive at different conclusions or interpreted in more than one way by reading a requirement can be said as ambiguous.

Ambiguity of requirements leads developers write a solution for a wrong problem and get testers test a different behavior built by developers. The unavoidable outcome of ambiguity is rework. Rework can cost near to 40 percent of total development, and 70 to 85 percent of the total revisions can be attributed to requirement errors (Leffingwell 1997).

4. Neglecting the needs of certain users/stakeholders leads to dissatisfied users/clients

Take the example of Payana system in Infosys. It's a travel requisition system used globally. This is used by people at offshore and at onsite. There are certain features which the onsite/offshore users alone would need. Eliciting the requirements only from one user group here would result in dissatisfied customer.

Most products are used by different set of people, who may use different kind of features, having different frequencies of use, or having different educational and experience levels. If you don't classify all these major types of user, the user classes for the product early, some user class will be upset with what is delivered.

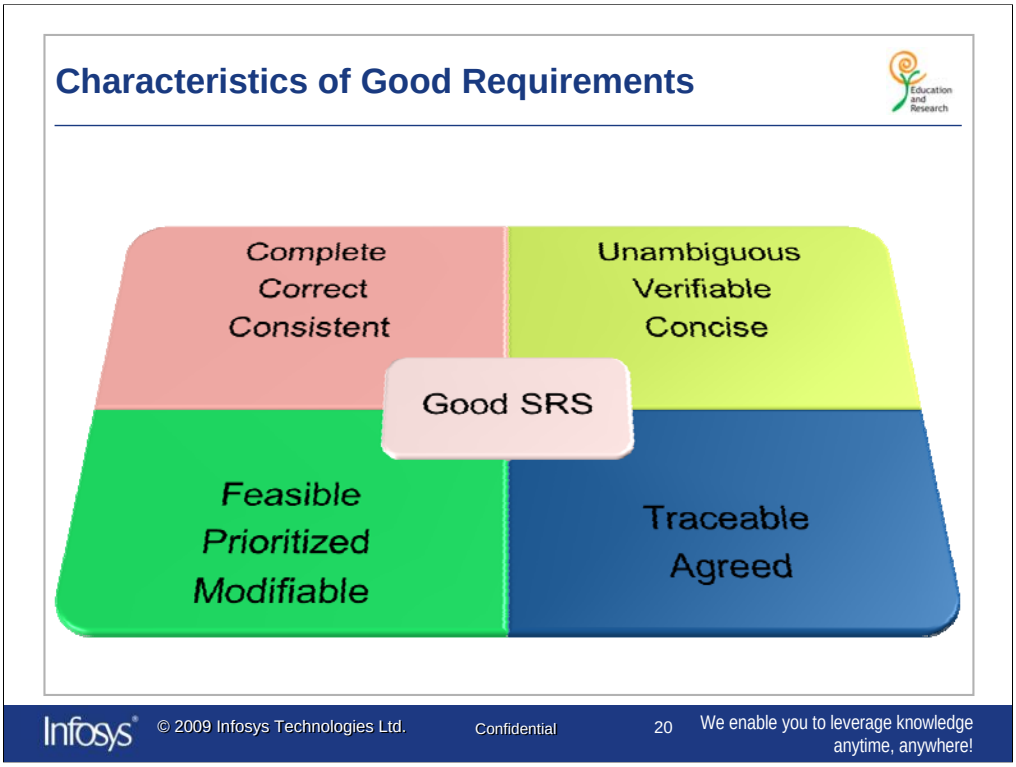
5. Incompletely defined requirements make accurate project planning and tracking impossible

Consider the scenario where the customer asks for estimates in lift/pantry. We sometimes just jump and give an estimate, feeling it's a test for our capability. There will be instance where estimates would be questioned in status meeting, if we have exceeded the figure we had given informally.

Requirements which are poorly understood lead to highly optimistic estimates, which will come back to bite us when the inevitable overruns occur. The top five reasons reported for poor sw estimation all related to Requirements Engineering:

- Frequently changing requirement.
- Missing requirement or misplaced
- Lack of communication with users to understand their requirements
- Requirements specification lacking clarity.
- Lack of requirement analysis.

Premature estimates based on limited information or limited thinking can easily be off by a factor or two or more. We should be trying to set achievable expectations and to consistently meet them.



The characteristics that should be exhibited by a good requirement statement or SRS is explained below.

1.Complete - A requirement should completely describe what functionality to be delivered. It should contain all the information a developer needs to design and execute the functionality.

Let us take an example of Infosys Resource procurement system. The process for RPS is - we raise a RPS for a new hardware in which we give details about the hardware, approver details. CCD forwards the same to the purchase department for approval of the cost. Once the order is placed, vendor ships it. When just said this, does it suggest that the requirements have been gathered for the system? What happens when approver doesn't approve of this. It is incomplete!

2.Correct - A requirement or SRS is correct only if the requirement stated is really required for the system to be built.

If the system must respond to validate boarding ticket in < 2 seconds, but the SRS states that "the system shall respond to validate boarding pass when it passes through the machine" leads to incorrect requirement. Only users can verify the correctness of user requirements, giving a strong a reason to include reviews of each requirements from users.

3.Consistent - An SRS or a requirement is consistent if there is no conflict with other prior documents, such as a system requirements specification or a statement of work, and no subset of requirements stated therein conflicts.

For example, the below sentences are inconsistent.

“The light shall be lit when and only when the button is pressed.

When the button is released, the light shall become lit”

Differences among requirements must be removed before the actual development can proceed.

4.Unambiguous - A requirement or SRS is unambiguous if the requirement can be interpreted or understood has only one meaning. Imagine if a requirement or SRS gives different meaning or interpreted from five different people in five different ways. If and only if it can give one interpretation of that requirement then you can call that as Unambiguous. . In particular, using natural language invites ambiguity because natural language is inherently ambiguous.

Let us take the example of “Air traffic controller problem”, for up to 10 aircraft a green color display will be used. Otherwise a blue colored display will be used. The ambiguity lies in the phrase “for up to 10”, does it mean “including or excluding the 10^{th}” aircraft. Requirements analysts capturing and writing the SRS might think it as trivial, but such as these ambiguity leads to devastating useless final products and dissatisfied clients.

5.Concise - A short and precise requirements exhibiting high quality requirement is better than elaborate explanation leading to confusion.

6.Feasible - A requirement should be implementable within the limitations and capabilities of the system and its environment. A member from the software engineering team and the technical person should be there with the marketing or the requirements analyst through out the elicitation process, to give a reality check on what can and cannot be done technically, and what can be done only at excessive cost.

7.Verifiable - A requirement is verifiable if a person or machine or test can check that the software product meets the requirement. Inconsistent, infeasible or ambiguous requirements are not verifiable.

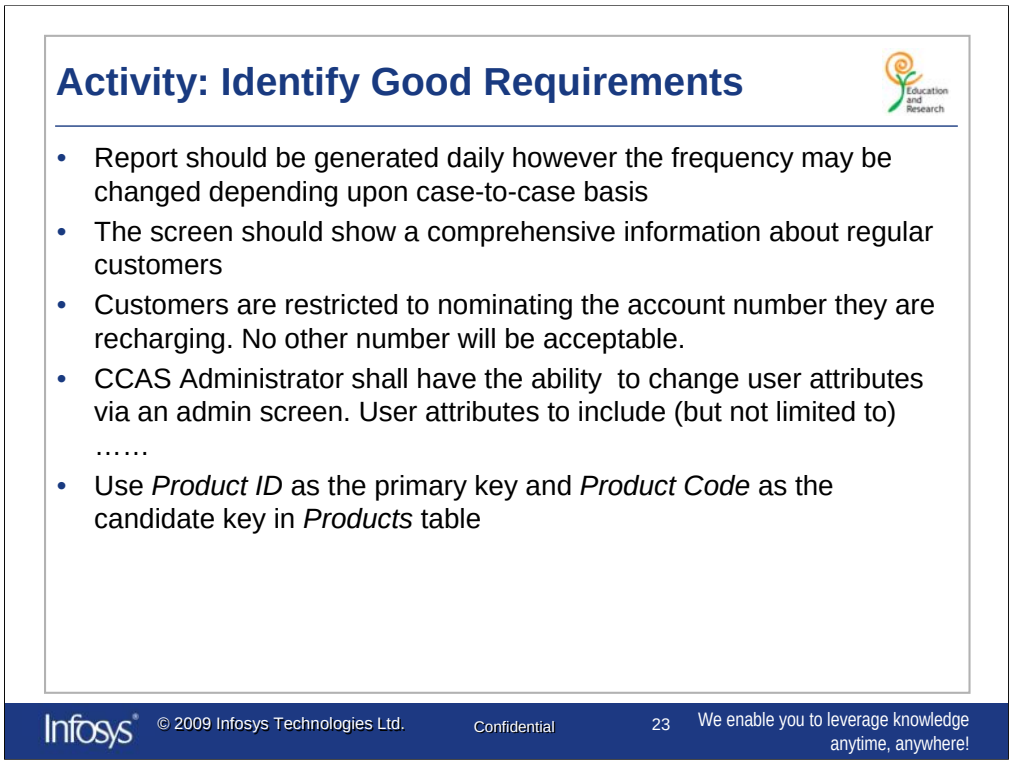
8.Prioritized - Prioritize which of the requirements, features or use cases are important for a particular product release. If all requirements becomes important then the Project Manager will have a tough time in adding new requirements, project parameters such as budget, schedule and scope will be tough to manage.

9. **Modifiable -** An SRS is modifiable if its structure and style are such that any necessary changes to the requirements can be made easily, completely, and consistently. Modifiability implies that there exists a table of contents, an index, and cross-references where necessary. Thus, if a requirement must be modified later, we can check and easily locate the section of the SRS that has to be modified.

   For example, if we want to change the required maximum response of a dial tone in a telephone switching system from 5 seconds to 3 seconds, we would look in the index under “dial tone” to locate all the references to dial tone in the document in order to make the necessary changes.

10. **Traceable -** The SRS document should be able to link each requirement from its origin to its implementation of that requirement, tracing to design documents, source files, the test cases written. Traceable requirements are distinctively labeled and written in a structured and fine-grained way. A requirements traceability matrix can relate to the design or test components to individual requirements

11. **Agreed -** Last but not the least, every stakeholder should agree on the requirement.



Now that we had see the attributes of the good requirements, let's try to do some exercise on identifying the good and bad requirements from the above slide.

1. Report should be generated daily however the frequency may be changed depending upon case-to-case basis.

In this case the requirement is not clear. What is the frequency to be considered?

2. The screen should show a comprehensive information about regular customers

This cannot be tested. What's meant by 'comprehensive'? How do you determine if a customer is a regular customer? Hence not verifiable.

3. Customers are restricted to nominating the account number they are recharging. No other number will be acceptable.

The requirement is clear here and hence a good one.

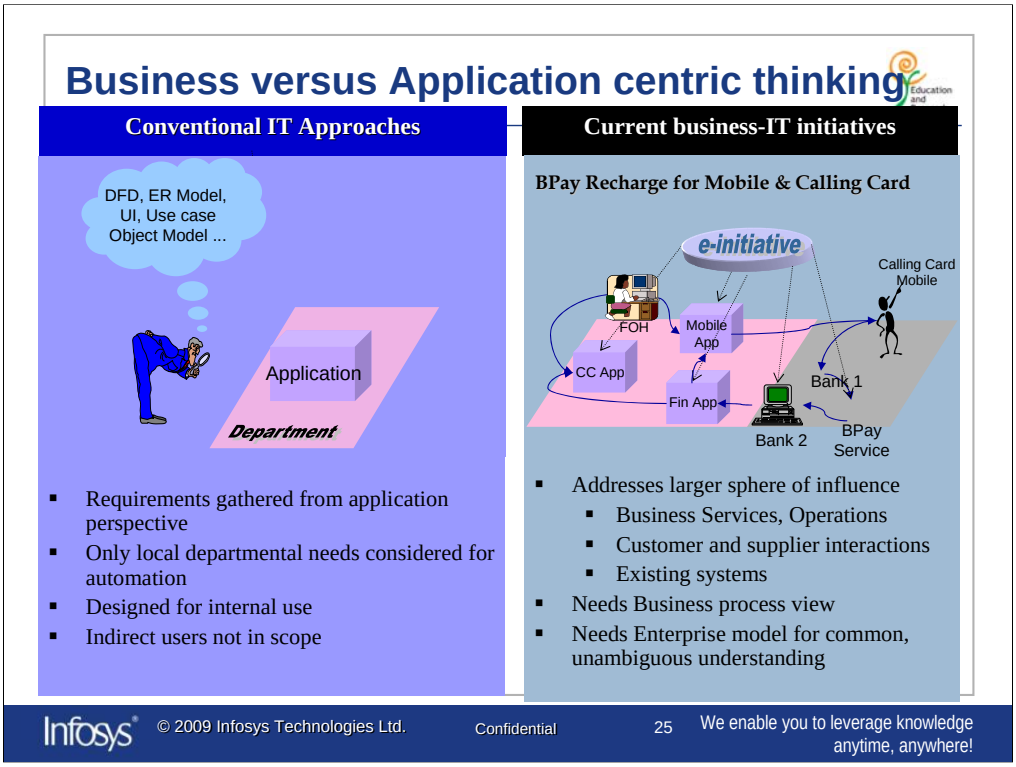
4. CCAS Administrator shall have the ability to change user attributes via an admin screen. User attributes to include (but not limited to) ……

Give a complete list of known user attributes. Cannot disclaim saying 'but not limited to …'. The requirement is not complete.

5. Use Product ID as the primary key and Product Code as the candidate key in Products table

It talks about design. Requirements should state about 'what' and not 'how'.

Components of Requirements



The above picture gives a comparison between conventional IT approaches to the current business IT initiative in Requirements gathering.

A project experience for illustration – The developer was asked to include a field in the export file extracted from a System A. This exported file is then passed on to other downstream systems. The developer just changed the logic to include the new field and did not consider the downstream systems that will be impacted. On the day of install, all the import processes of the downstream systems, which loads the extracted file from System A failed. The obvious reason is – Impact on the downstream systems are not analysed by the developer.

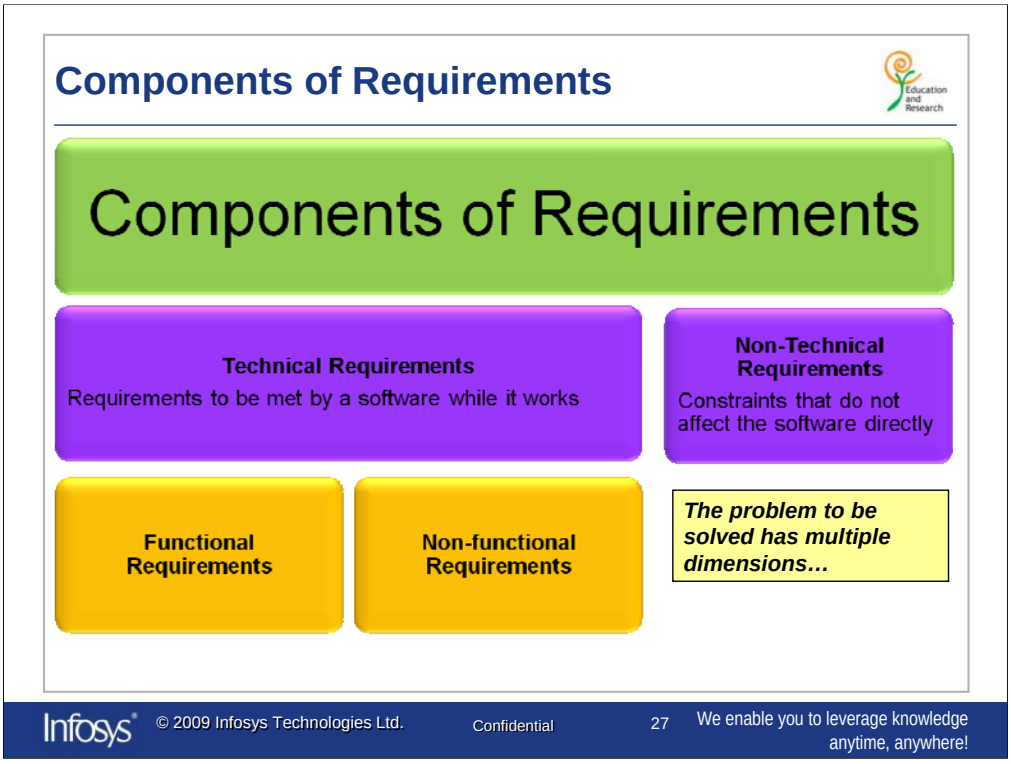
The conventional IT approach looks at the requirements mainly from the application perspectives. ER models ,DFDs, etc. were the primary focus for which requirements were gathered. Designers knew nothing about the business, and users weren't comfortable with the computer terminologies. The Gap existed in development process.

The conventional method of requirements gathering, doesn't consider all the stakeholders impacted. Not meeting the business needs leads to high risks. Not suitable when users are not clear about requirements. This results in user dissatisfaction and losing credibility.

As a complete solution is not identified, there will be frequent or uncontrolled changes in requirements at later phases in project lifecycle. If we identify requirements later in the project lifecycle we start incorporating changes and start fixing the bugs, then the cost of repair is very high as shown in the graph from earlier slides.

This approach is not suitable when users are not clear about their requirements. Right question to the right people need to be mapped, which calls for a good elicitation technique.

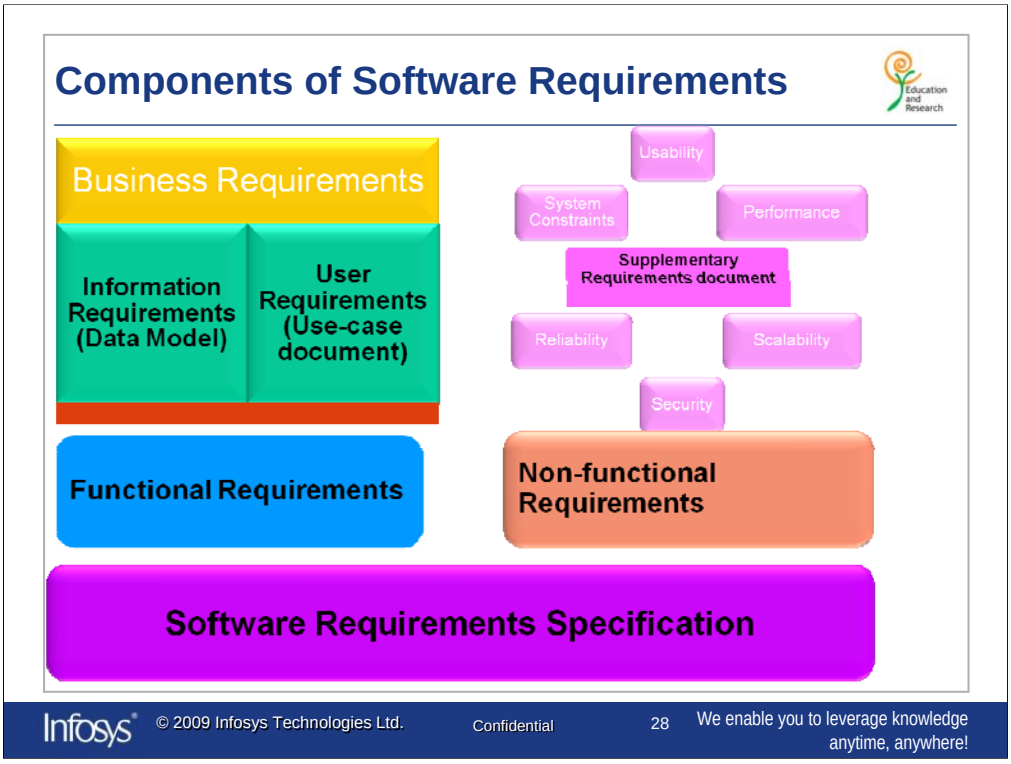
But in today's scenario, methodologies have evolved stressing the need to understand and have a holistic picture about the customer's business. For example, take influx methodology, the business modelling approach allows a structured way of capturing requirements like understanding who is who in the organization, its various locations, various activities, the people who performs it etc. Thus, it gives a full understanding of systems, process and the customer's business interactions. For example - In Infosys, we have Domain Certifications to mainly understand the business domain that we work, which would help in the elicitation process. Every developer would one day reach a level where they need to interface with the customers. So understanding the business is more important for effective RE.



Requirements may be technical or non-technical.

Requirements to be met by a software while it works is a **technical requirement**. In turn, it can be classified as functional requirement, non-functional requirement which will be discussed in detail in next slide.

A **non technical requirement** are those constraints that do not affect the software directly. Example the schedule of the project, costing of the project etc. These doesn't have direct impact on the software but still puts the restriction on the software that we build.



Software requirements include three distinct levels—business requirements, user requirements, and functional requirements. In addition, every system has an assortment of nonfunctional requirements. The above figure illustrates a way to think about these diverse types of requirements.

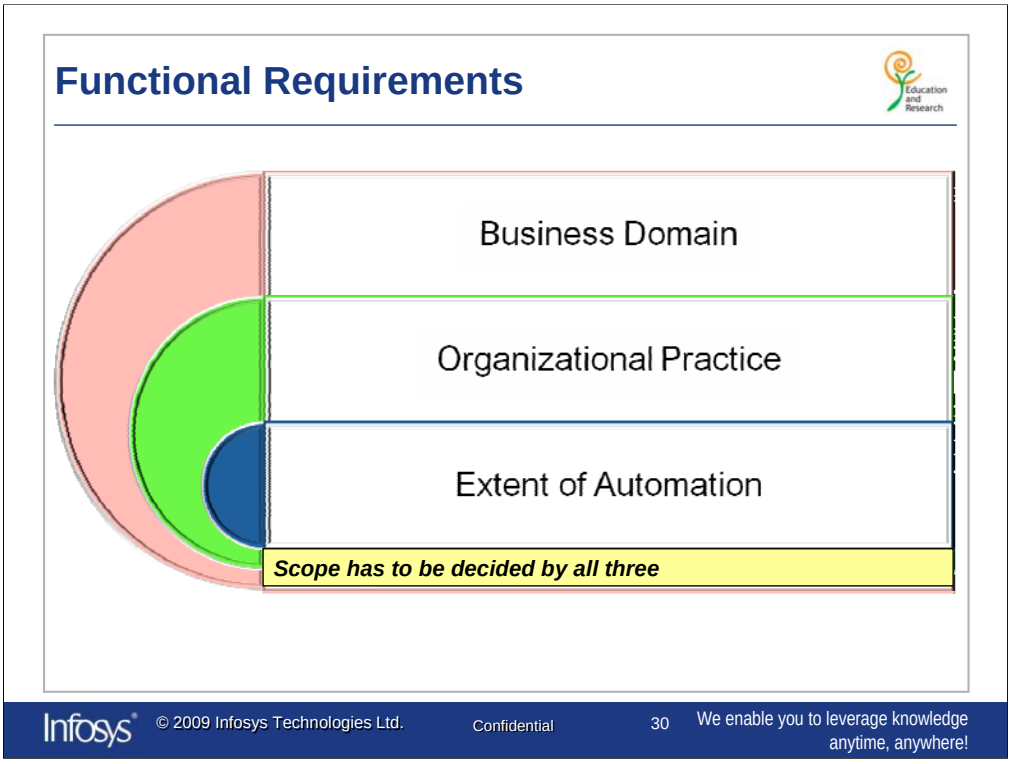
Business requirements characterizes the main objectives of the organization or client who have requested the system at a very high-level. These requirements usually comes from the project sponsor or managers to the actual users or an acquiring client, or from marketing or product visionary departments. These requirements are recorded in the vision or scope document of what the organization wants or hopes to accomplish.

User requirements describes the functionalities the users need to perform. Use case is a way of representing user requirements. User requirements are the ability of the system to help users what they intend to do.

Information requirements describe the way data/information is stored within organization. This is accomplished by data model. This takes care of analyzing the data that is needed for input, how it gets processed and the data that goes as an output.

Functional requirements indicates the functionality that a software enables the users to complete their tasks in turn satisfying business requirement to be built by the developers.

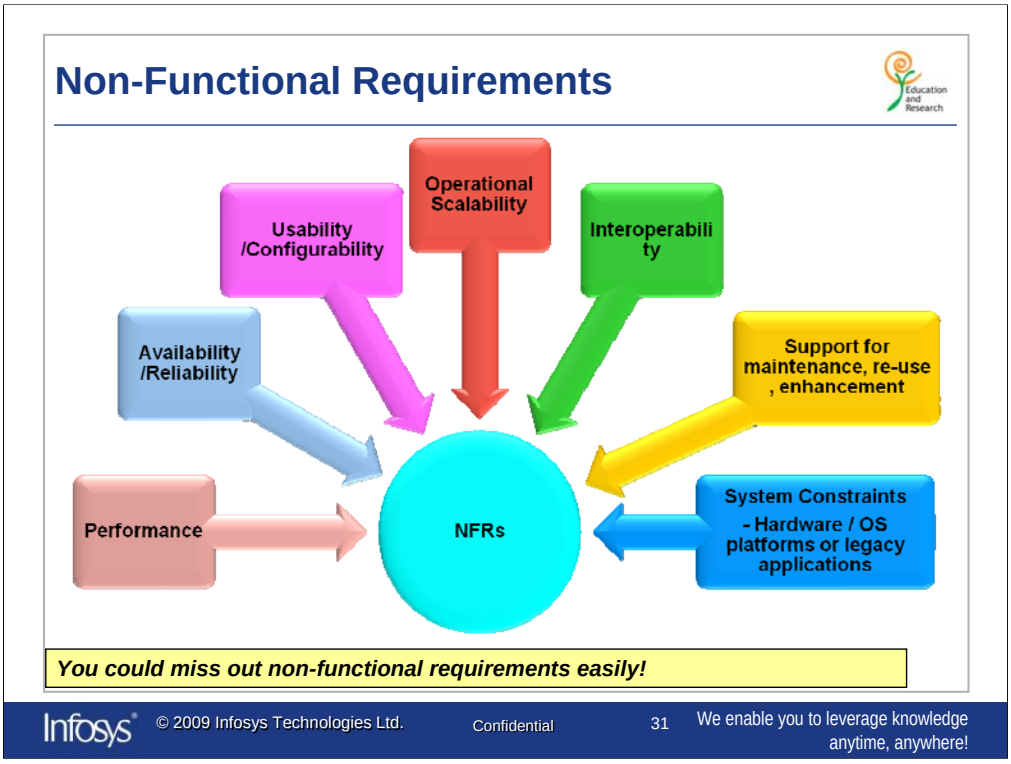
Functional requirements are documented in a **software requirements specification**(SRS), which describes as fully as necessary the expected behavior of the software system. In addition to the functional requirements, the SRS contains nonfunctional requirements. These include performance goals and descriptions of quality attributes. *Quality attributes* add to the product's functionality by elaborating product's significant characters in other different view which are important to all the stakeholders. These characteristics include usability, portability, integrity, efficiency, and robustness. Other nonfunctional requirements describe external interfaces between the system and the outside world, and design and implementation constraints. *Constraints* enforce the boundaries on the preferences offered to design and development of the products to the developer.



Scope of the functional Requirements is decided by the business domain, organization practices and rules and extent of automation intended.

The main aim of a software project is to automate a business process. Hence the scope of the functionality depends on the business domain for which the automation is needed. Let us take the example of “Infosys Claims Settlement System”. Is the entire claims settlement process automated? No. We still have to submit signed hard copy of the bills. So the functional requirement is limited to the extent of automation needed by the business.

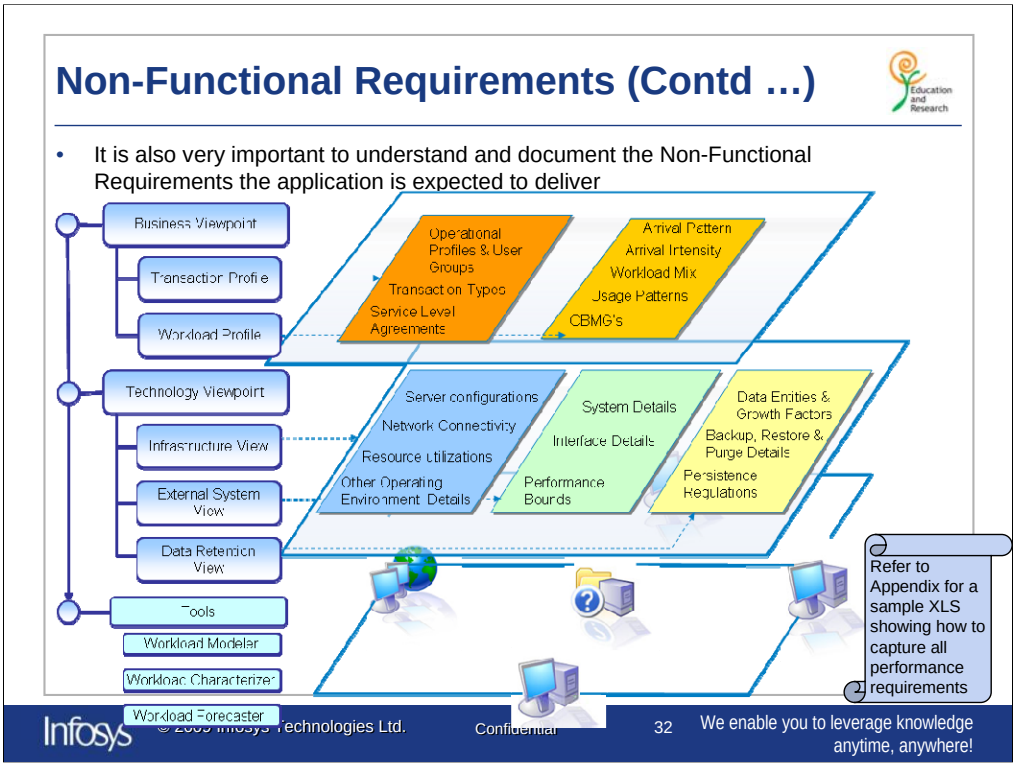
Organization practices and rules too decide the scope for functional requirements. For example, consider building a registration system like I-Lite for Infosys as well as Wipro. Though it’s the same system, the functionalities/features that are provided depend on the Organization practices and rules.



Non-functional requirements, as the name suggests, are those requirements which are not directly concerned with the specific functions delivered by the system.

Many non-functional requirements relate to the system as a whole rather than to individual functional requirements. While failure to meet an individual functional requirement may degrade the system, but failure to meet a non-functional system requirement may make the whole system unusable.

For example, if an aircraft system does not meet its reliability requirements, it will not be certified as safe for operation.



Its important to emphasize the fact that the Casper NFR Gathering template is to be used in the most appropriate manner keeping the original intent in mind viz. to capture those qualitative metrics that would be useful when developing or maintaining applications.

Therefore, in cases involving a fresh application development, it would be necessary to complete all the views and profiles that are specified in the Casper NFR Gathering methodology. However, in cases involving routine maintenance that might entail some degree of feature enhancement, it is left to the discretion of the application architect to take a call on the views and profiles that need to be addressed.

For example, if a new functionality is being added to an existing application, one needs to document the existing utilization of the system (Infrastructure View) and application workload (Workload & Transaction Profile) to assess the impact the delta increase in load (because of the new functionality) and take a call on the plausibility of hosting the enhanced application on the same infrastructure or recommend an upgrade. It would also be important to document the characteristics of the interface this new module can potentially share with external systems (External Systems View). However if this is not the case, updating or addressing this view might not be required.

The focus of the activities performed during this phase are to capture and analyze the qualitative attributes of the functional requirements. Each user would have different view on the performance e.g. response time requirement differs for different users.

The CASPER performance requirements elicitation technique is a systematic way for gathering performance requirements and thus minimizes ambiguity, omissions and people dependency. It defines two viewpoints to gather performance requirements from different users:

1. Business Viewpoint - This view gathers the performance requirements from business users perspectives using following profiles:

a. Transaction Profile captures the transaction types, operational profiles and user groups, service level agreements.

b. Workload Profile captures the arrival pattern, intensity, workload mix, usage pattern

SLAs differ for different type of transactions, workload mix. Workload Mix – One type of transaction might affect the performance of other transactions. There could be particular transaction which causes performance issues. Some database intensive operation might have more demand. So we need to know the right workload mix.

Usage pattern – Differs for different type of users. Based on user behavior, "Customer Behavior Model graph (CBMG)" can be created.

2. Technology Viewpoint - This is the technical users viewpoint. The technical users of the system like system administrators, DBAs are interviewed to collect the performance requirements using:

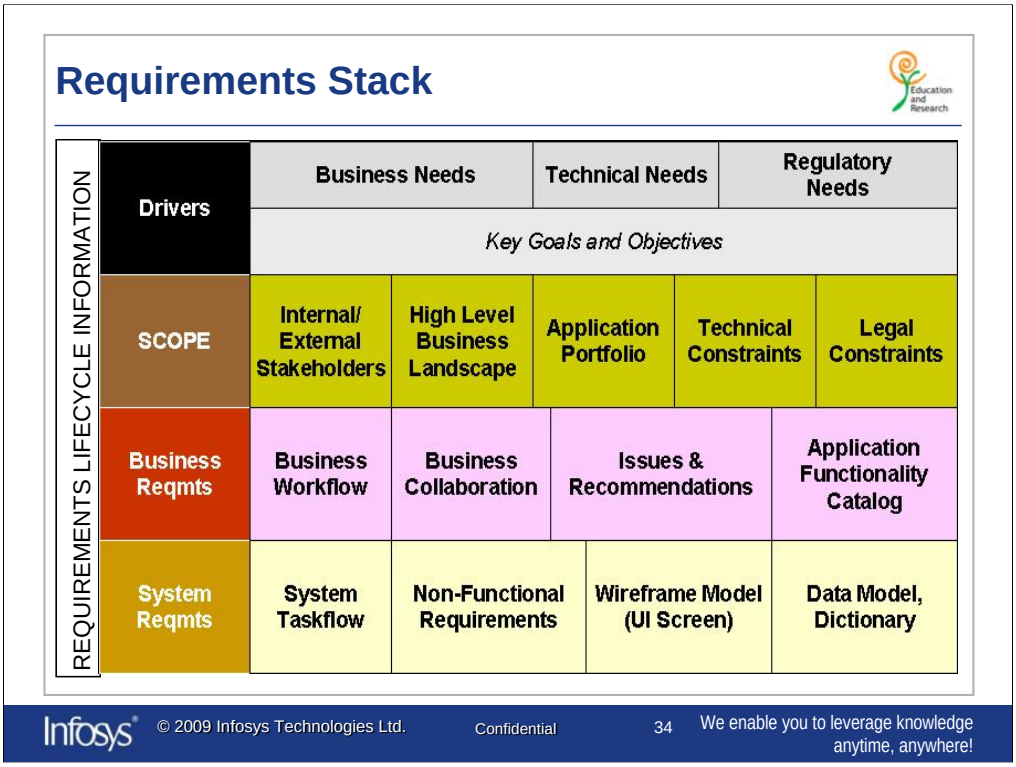
a. Infrastructure View gives the server configurations, network connectivity, resource utilizations and other operating environment details.

Resource Utilization – Upto what % of resource utilization the SLAs will be met. i.e. for upto 40% of CPU utilization, the response time is xx, throughput is yy. In BSE, if the resource utilization reaches 25%, they go for upgrading their resources. So resource utilization is a more critical data.

b. External System View gives the system details, interface details, performance bounds of the external systems.

Performance bounds has to be determined as defined with respect to the external systems. Performance issues in many cases has been due to issues with external system.

c. Data Retention View gives the data entities and their growth factors, backup, restore and purge details, data persistence regulations



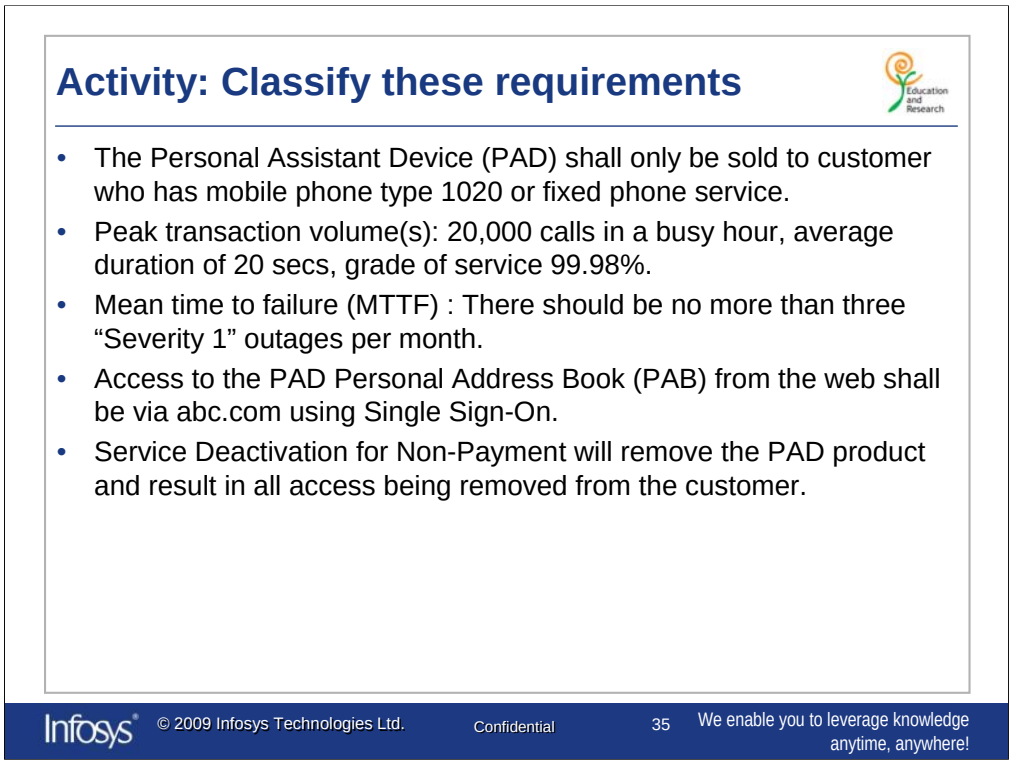
Let's decipher the "requirements stack" in the above diagram, the drivers which lead to requirements can come by the "Business needs" of a company or organization or a set of users to do their functional duties and responsibilities. It can be due to Technical needs or by the Regulatory Needs imposed on the organization. These drivers triggers to becomes the Key Goals and objectives.

Scope: The scope of the requirements can be from internal/external stakeholders, an enterprise level change where the product or application have to fit in, scope within where the application portfolio fits or may be due to technical or legal constraints.

Business Requirements: Could be due to change in workflow or collaboration with other applications or products. Issues and recommendations to be resolved.

System Requirements: So changes to System level, non-functional requirements to be taken care, should gather the Data model and any changes to the existing UI screen changes if any needs to be collected.

All the above requirements gathered would make it Lifecycle of requirements.



Now let's do some activity to classify the requirements as functional or non-functional.

1. The Personal Assistant Device (PAD) shall only be sold to customer who has mobile phone type 1020 or fixed phone service.

   This talks about the functionality of the system and hence a functional requirement.

2. Peak transaction volume(s): 20,000 calls in a busy hour, average duration of 20 secs, grade of service 99.98%.

   This is on the performance measures and hence a non-functional requirement.

3. Mean time to failure (MTTF) : There should be no more than three "Severity 1" outages per month.

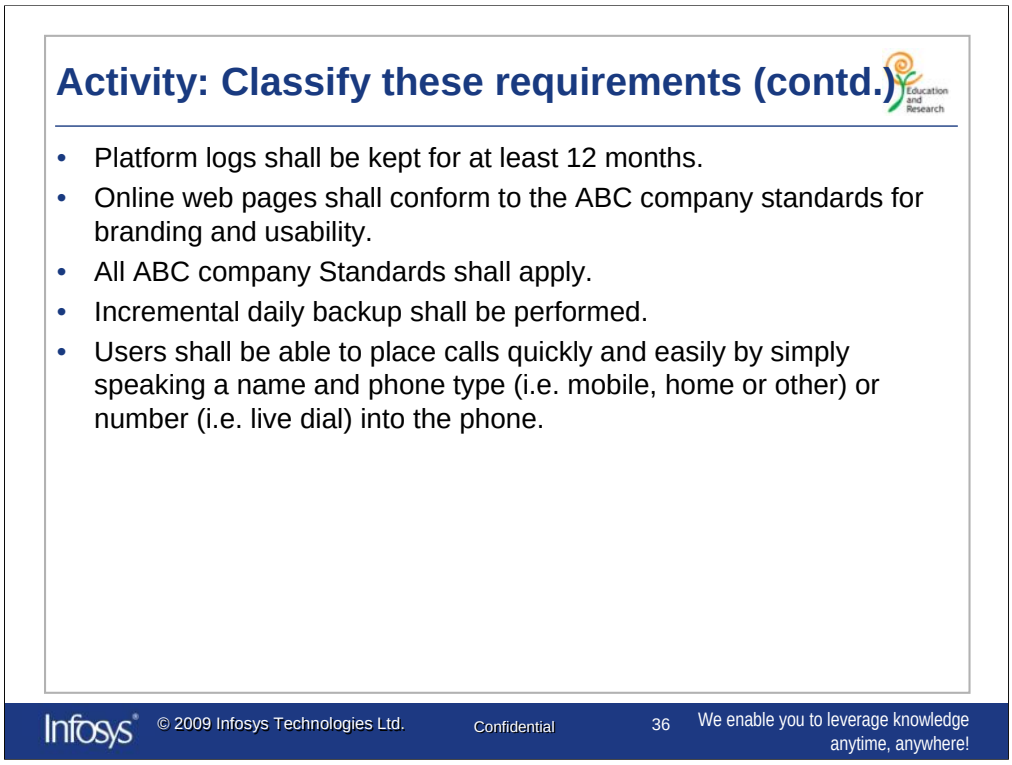
   This is about the reliability factor and hence a non-functional requirement.

4. Access to the PAD Personal Address Book (PAB) from the web shall be via abc.com using Single Sign-On.

   This is about the security feature and hence a non-functional requirement.

5. Service Deactivation for Non-Payment will remove the PAD product and result in all access being removed from the customer.

   This talks about the functionality of the system and hence a functional requirement.



6. Platform logs shall be kept for at least 12 months.

This is on compliance and hence a non-functional requirement. But if we have Audit department as one user class and if the above requirement is given by them, then it becomes a functional requirement.

7. Online web pages shall conform to the ABC company standards for branding and usability.

This talks about the constraints to UI requirement and hence a non-functional requirement.

8. All ABC company Standards shall apply.

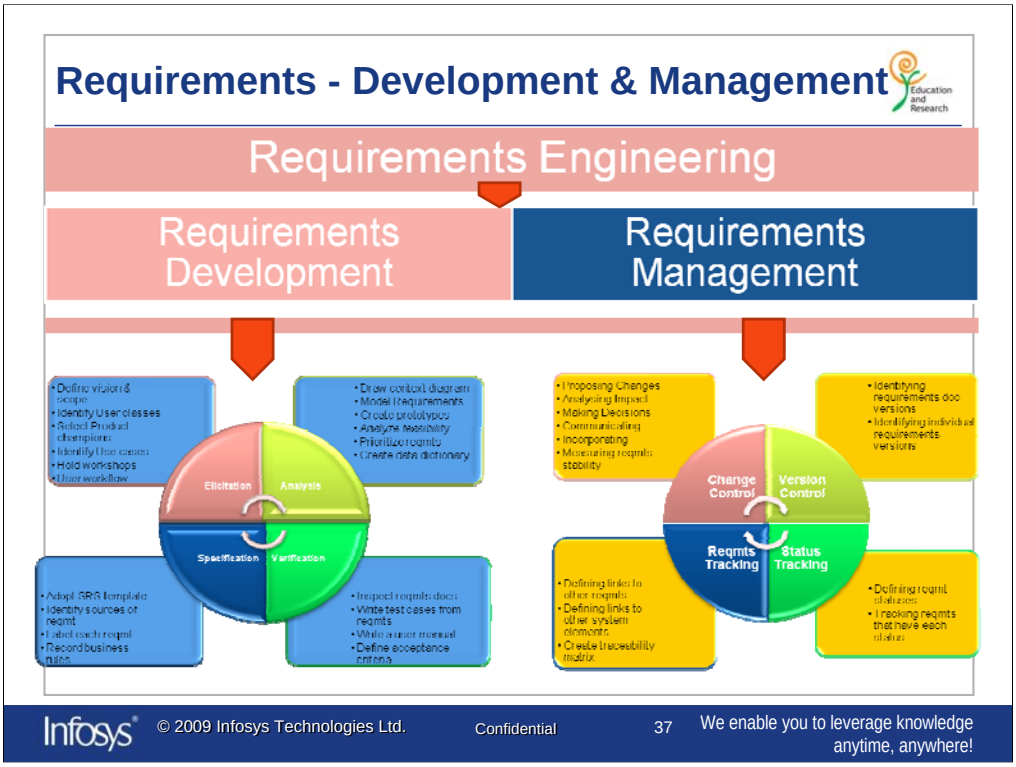
This is on following the company standards which is a constraint and hence a non-functional requirement.

9. Incremental daily backup shall be performed.

This is a non-functional requirement

10. Users shall be able to place calls quickly and easily by simply speaking a name and phone type (i.e. mobile, home or other) or number (i.e. live dial) into the phone.

This is a functional requirement. Here, placing the call is a functional requirement and it has other usability (non-functional) attributes added to it.



There's confusion as to what to call this whole discipline about requirements. Some call the whole domain as requirements engineering (Sommerville and Kotonya 1998); others call it requirements management (Leffingwell and Wildrig 2000). Karl E. Wiegers finds it useful to split the domain of software requirements engineering into requirements development and requirements management.

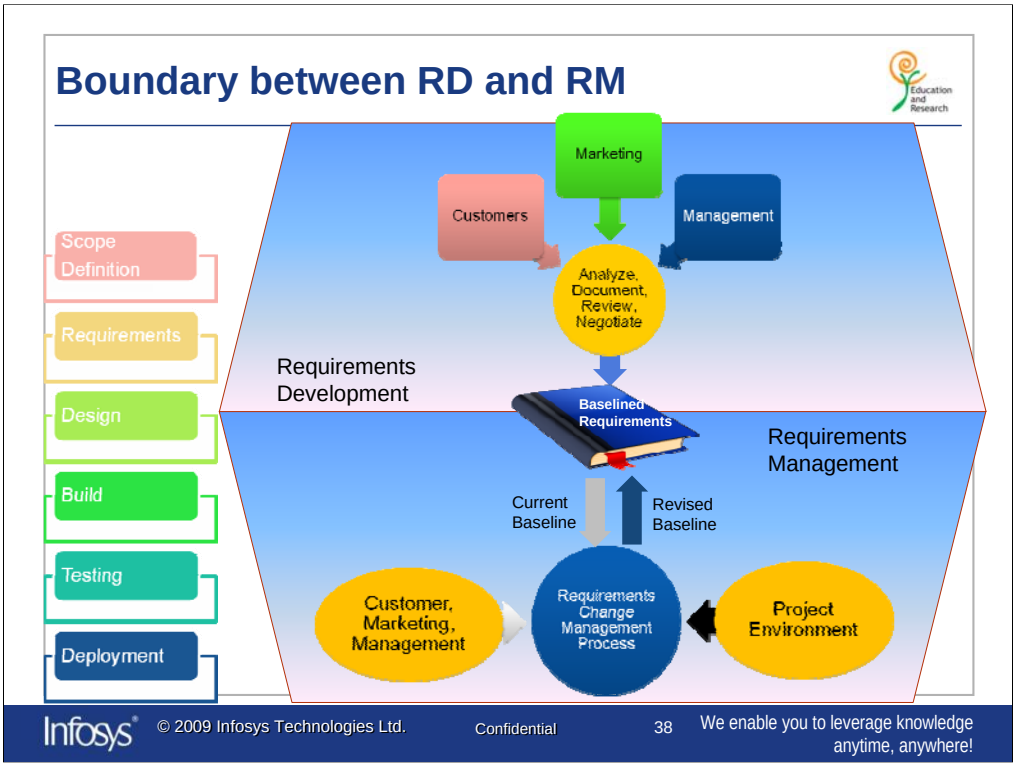
The various phases of requirements development and the activities in requirement management are described in the above slide.

Elicitation –First define the vision & scope, then Identify User classes, Select Product champions, Identify Use cases for that Hold workshops to get the User workflow.

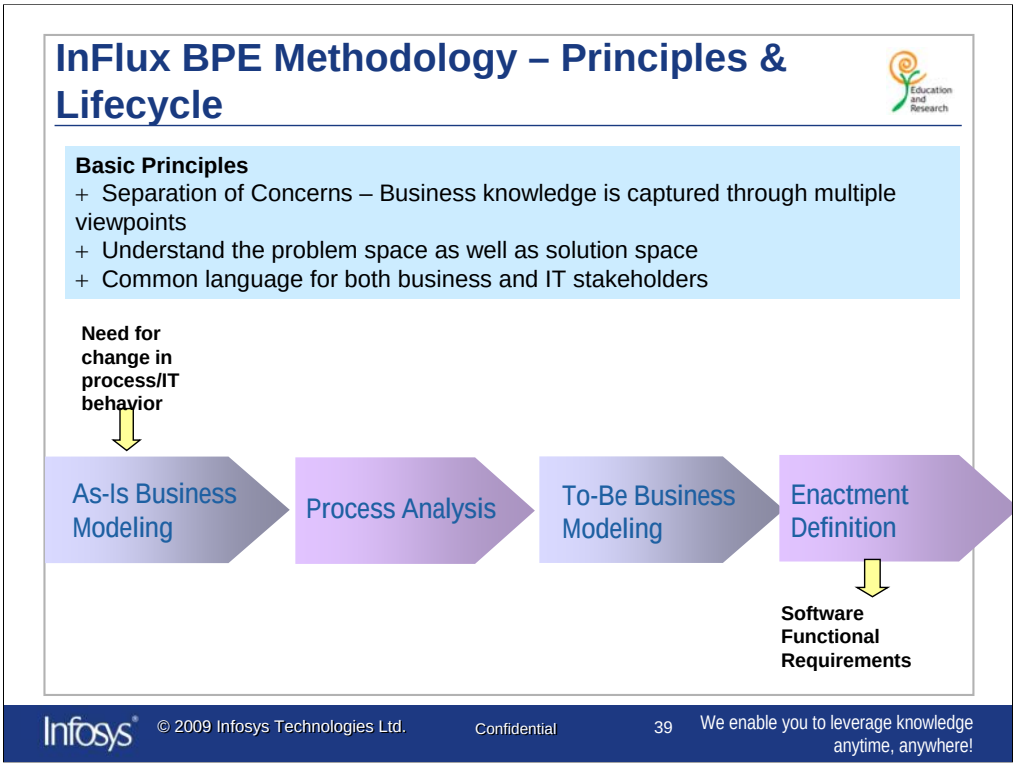
Analysis - Draw the context diagram, Model Requirements, Create prototypes, Analyze feasibility, Prioritize requirements, Create data dictionary.

Specification - Adopt SRS template, Identify sources of requirement, Label each requirement, Record business rules

Verification - Inspect requirements documents, Write test cases from requirements, Write a user manual, Define acceptance criteria



Requirements are gathered from different stakeholders, analysed, documented, reviewed, negotiated and are baselined. When the change in the requirement is initiated by one of the stakeholders, it goes thro the change process and considering the impact in the project environment, the base lined is revised if the change is accepted and if not, current baseline is maintained. There is a thin boundary between Requirements development and Requirements management.



InFlux BPE is Infosys' proven methodology that formalizes a business process centric approach to blueprint the functional requirements of IT solutions. It is a systematic and repeatable methodology that derives the functional part of an IT solution from the business process that will ultimately use it.

It prescribes a step-by-step process to identify business needs and translate them into functional requirements of an IT solution. Each of the activities in the process produces a specific output and is broken down into tasks that contribute to or create the activity output.

The methodology uses models, methods, techniques, tools and patterns to help achieve a smooth translation of business needs into an effective IT solution blueprint.

Influx BPE methodology focuses on activities related to creating requirements artifacts and does not explicitly mention project management and quality related activities. Due diligence is necessary to follow this approach, for it may not be always necessary to go through the entire methodology in all instances. Some IT projects may be so simple that only a part of the methodology or a combination of few steps may suffice to arrive at the optimal functional blueprint of the solution.

Influx BPE Methodology Principles

The basis principles of Influx BPE Methodology are

- Separation of concerns as business knowledge is captured through multiple stakeholders who have multiple viewpoints
- Understanding the problem space as well as solution space
- Common language for both business and IT stakeholders

Separation of concerns

The basic check or parameter on which one can say that a software system is successful is if it met the intended purpose. Requirements Engineering(RE) helps in realizing those stakeholders and their requirements, such that they are documented in a structure so it can be analysed, communicated and can be implemented subsequently. Numerous problems are seen in this process. Multi-vendor, multi stakeholders and distributed in nature can pose different problems based on their perspectives how they work and their tasks to accomplish. They might not be able to articulate their goals explicitly leading to dissatisfaction of their goals to accomplish which are not in their control.

Influx BPE methodology provides methods and models that help in capturing the view point of multiple stakeholders.

Understanding the problem space as well as solution space

Clear understanding of problem space is very important prior to action or deriving a solution. Most of us model the solution space rather than problem space. Can we imagine modelling a flight control Software without understanding the problem space. Creating and analysing models of the problem space helps in clear identification of the issue points for deriving a solution, with the goals of increasing the understanding and searching for incompleteness and inconsistency.

Influx BPE methodology provides models such as As-Is Business models to capture the current business process (problem space) and To-Be Business models to model the solution space.

Common language for both business and IT stakeholders

The objective of the Influx methodology is to ensure Business-IT Alignment through effective translation of business requirements into IT solutions. This objective can be met only if both business and IT stakeholders can speak the same language. The Business modeling method of Influx BPE methodology provides a systematic and formal way of representing information. It thus acts as a common language between the business and IT communities to specify and understand business requirements.

Lifecycle view of Influx BPE methodology

The lifecycle view of Influx BPE methodology consists of sequence of activities and each activities in the process produces a specific output and is broken down into tasks that contribute to or create the activity output. The end-to-end activities that need to be carried out as part of the methodology are:

- **As-Is Business Modeling** – An IT solution is deployed to address automation needs of business processes in an enterprise. The need for change in business process/IT behaviour is the driver for building an IT solution. This activity helps elicit the current business process to understand the problem space and is represented in the form of business models.
- **Process Analysis** – The business need for the yet to be defined IT solution is clearly delineated in this activity.
- **To-Be Business Modeling** – This activity is concerned with translating recommendations from process analysis into specific business process level changes through creation of To-Be business process models.

The functional requirements are detailed from To-Be Business models in the Enactment definition phase which is out of scope in this courseware. We will now look at each of above mentioned activities in detail.

Summary

- Requirements Engineering phase is the foundation of the software development life cycle. Hence errors made at this phase are costlier to correct at later stages
- Understanding the characteristics and components of requirements helps in developing a good SRS
- Requirements engineering domain can be split into requirements management and requirements development

References

1. Karl E. Wiegers, “Software Requirements”, Microsoft Press
2. Alan M Davis, “Software Requirements”, Prentice-Hall, Inc
3. CMU/SEI-93-TR-25, “Requirements Management”, CMM Practices, SEI, CMU, 1993.
4. Somerville I., “Software Engineering”, Addison-Wesley, MA, 1992.
5. Loucopoulos P. and Karakostos V., “System Requirements Engineering”, McGraw-Hill Book Company, London, 1995.
6. McMenamin S. and Palmer J.F., "Essential Systems Analysis", Yourdon Press, 1984.
7. Pride Processes (http://172.25.103.230/Pride/Default.aspx)
8. http://setlabs/learn/bpe/Pages/Home.aspx

Appendix

Example of a Requirement Defect

ILite: Individual Training Plan Screen

Training Days for each Employee:	5(Min) and 99(Max)	2 Days (Min) for ILI
Total planned days : 7.69	Total days Attended: 5.94	Total days Pending: 1.75
Manager :		

Sl.No.	Course Name	Course Type	Planned Duration (Days)	Planned Duration (Hours)	Planned Period	Actual Duration (Days)	Actual Duration (Hours)	Date Attended	Status
1	OTR Business Process Training (1Day Program by IS)	Others	1	8	Jan2009	1	8	22-October-2008	Attended
2	Articulation Skills-01- Contact program	Managerial	0.5	4	Jun2009	-	-		Not Attended
3	Assertiveness Skills - Contact Program	Managerial	1.25	10	Feb2009	-	-		Defaulted
4	Written Communication Skills - Contact Program	Managerial	0.75	6	Aug2009	0.7	6	18-February-2009	Attended
5	Active Listening Skills - E Learning	Managerial	0.5	4	Mar2009	0.5	4	20-January-2009	Attended
6	Assertiveness Skills Elearning - ILI	Managerial	0.5	4	Mar2009	0.5	4	10-January-2009	Attended
7	Giving and Receiving Feedback - E Learning	Managerial	0.5	4	Mar2009	0.5	4	10-January-2009	Attended
8	Proposal Writing ELearning - ILI	Managerial	0.5	4	Mar2009	0.5	4	10-January-2009	Attended
9	ESTEEM (FP ESTIMATION)	Technical	2	16	Jan2009	1	8	23-December-2008	Attended
10	Tools Overview	Technical	0.19	1.5		0.19	1.5	25-February-2009	Attended
11	Offshore Project Management	Technical	5	40	Inactive Course	5	40	16-February-2009	Attended

Duration of inactive courses are not considered for calculating Total days Attended

Ideally, if a course gets inactivated **after** one attends the training, it should still consider its duration for total no. of days.

Capturing Non-functional Requirements

- A sample document

Microsoft Office
cel 97-2003 Worksh

Please refer the same in Additional Materials section.

Use-case document – Few samples

- KShop References:
- http://172.25.103.176/Bok/bok_11409/bok_11409.doc
- Use-case elaborations inside:
  - http://172.25.103.176/CaseStudy/case_80078/bok_80078.doc
  - http://172.25.103.176/Bok/bok_20477/bok_20477.doc

Thank You