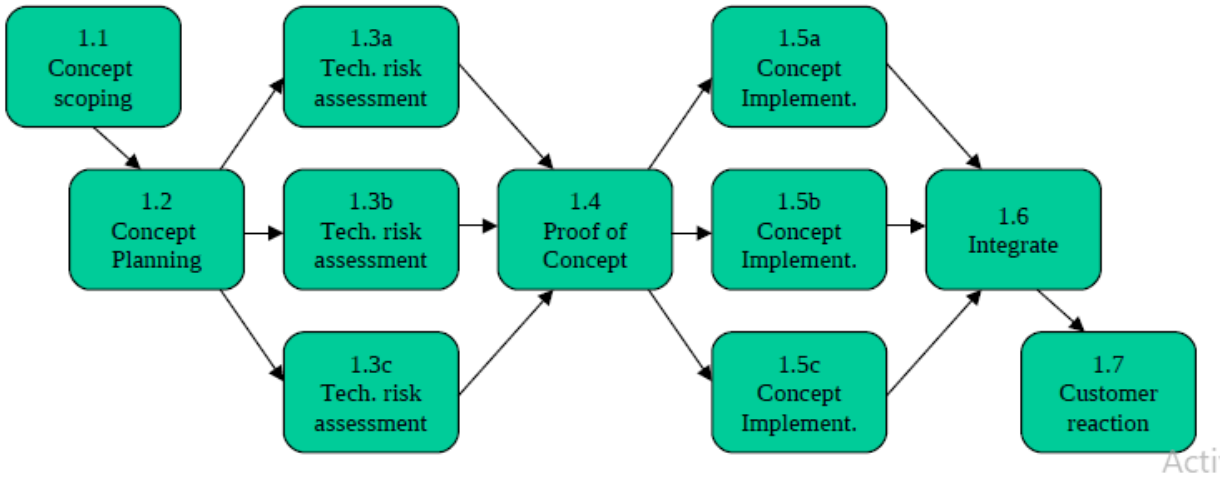
Task Network

Since it is a concept development project, the applicability is not certain but it appears to be useful and hence needs to be explored. Major tasks include:

- Concept scoping
- Preliminary concept planning
- Technology risk assessment
- Proof of concept
- Concept implementation
- Customer reaction to concept

Defining a Task Network

Once the tasks have been identified, we need to develop a task network to determine the sequence in which these activities need to be performed. This will ultimately lead to the time required to complete the project.



Scheduling

Once we have the task network, we are now ready to prepare a schedule for the project. For this we use two techniques known as:

1. Program evaluation and review techniques (PERT)
2. Critical Path Method (CPM)

These are quantitative tools that allow the software planner to determine the critical path – the chain of tasks that determines the duration of the project and establish most likely time estimates for individual tasks by applying statistical models. They also help the planner to calculate boundary times that define a time window for a particular task.

The boundary time defines the following parameters for a project:

- The earliest time that a task can begin when all preceding tasks are completed in the shortest possible time

• The latest time for task initiation before the minimum project completion time is delayed
• The earliest finish
• The latest finish
• The total float – the amount of surplus time or leeway allowed in scheduling tasks so that the network critical path is maintained on schedule

In order to use the PERT and CPM, the following is required:

• A decomposition of product function
• A selection of appropriate process model and task set
• Decomposition of tasks – also known as the work breakdown structure (WBS)
• Estimation of effort
• Interdependencies

Timeline Chart

To develop the schedule for a project, time required for each activity in the Task Network is estimated. This analysis and decomposition leads to the development of a Timeline or Gantt Chart for the project which portrays the schedule for the project.

Concept Scoping (the first task in the above list) is further subdivided into the following sub-tasks

1. Identification of needs and benefits (3 days)
2. Definition of desired output/control/input (7 days)
3. Definition of the function/behaviour (6 days)
4. Isolation of software elements (1 day)
5. Researching availability of existing software (2 days)
6. Definition technical feasibility (4 days)
7. Making quick estimate of size (1 day)
8. Creating scope definition (2 days)

Project Tracking

Tracking methods include:

• Periodic project status meetings
• Evaluating the results of all reviews
• Determine whether project milestones have been accomplished by the scheduled date
• Comparing actual start date to planned start date
• Informal meetings with the practitioners
• Using earned value analysis
• Error tracking

Earned Value Analysis

Earned Value Analysis or EVA is a quantitative technique for assessing the progress of a project. The earned value system provides a common value scale for every software task, regardless of the

type of work being performed. The total hours to do the whole project are estimated, and every task is given an earned value based on the estimated percentage of the total.

SPI (Schedule performance index) close to 1 indicates efficient execution.
Value of CPI (Cost performance index) close to 1 means project is within its defined budget.
Therefore, by using SPI and CPI we estimate how the project is progressing. If we have these values close to 1, it means that we have had good estimates and the project is under control.

Error Tracking

Error tracking can also be used to estimate the progress of the project. In this case we track errors in work products (requirement specifications, design documents, source code etc) to assess the status of a project. The process works as follows:
We collect error related metrics over many projects and determine our defect removal efficiency in the following manner:
Defect removal efficiency, DRE = E / (E+D), where
• E – errors found before shipment
• D – errors found during operation
It provides a strong indication of the effectiveness of the quality assurance activities.

Time Boxing
Time-boxing is used in severe deadline pressure. It is a use incremental strategy where tasks associated with each increment are time-boxed in the following manner:
• Schedule for each task is adjusted by working backward from the delivery date.
• A box is put around each task
• When a task hits the boundary of the box, work stops and next task begins
The principle behind time-boxing is the 90-10 rule (similar to Pareto Principle) –
Rather than becoming stuck on the 10% of a task, the product proceeds towards the delivery date in 90% of the cases.

Software Quality Assurance

Software quality is defined as conformance to explicitly stated functional and nonfunctional requirements, explicitly documented development standards, and implicit characteristics that are expected of all professionally developed software.
This definition emphasizes upon three important points:
• Software requirements are the foundation from which quality is measured. Lack of conformance is lack of quality
• Specified standards define a set of development criteria that guide the manner in which software is engineered. If the criteria are not followed, lack of quality will almost surely result.
• A set of implicit requirements often goes unmentioned (ease of use, good maintainability etc.)

Another very important question is: Do you need to worry about it after the code has been generated?
In fact, SQA is an umbrella activity that is applied throughout the software process.

Quality has measurable characteristic such as cyclomatic complexity, cohesion, and coupling.

Quality of design tries to determine the quality of design related documents including requirements, specifications, and design.
Quality of conformance looks at the implementation and if it follows the design then the resulting system meets its goals then conformance quality is high.
Glass defines quality as a measure of user satisfaction which is defined by compliant product + good quality + delivery within budget and schedule
DeMarco defines product quality as a function of how much it changes the world for the better.

Goal of quality assurance is to provide the management with the necessary data to be informed about product quality. It consists of auditing and reporting functions of management. If data provided through QA identifies problems, the management deploys the necessary resources to fix it and hence achieves desired quality control.

Cost of quality

Quality has a direct and indirect cost in the form of cost of prevention, appraisal, and failure. If we try to prevent problems, obviously we will have to incur cost. This cost includes:

1. Quality planning
2. Formal technical reviews
3. Test equipment
4. Training

The cost of appraisal includes activities to gain insight into the product condition. It involves in-process and inter-process inspection and testing.
Failure cost has two components: internal failure cost and external failure cost.
Internal failure cost requires rework, repair, and failure mode analysis.
External failure cost involves cost for complaint resolution, product return and replacement, help-line support, warranty work, and law suits.

SQA Activities

There are two different groups involved in SQA related activities:

1. Software engineers who do the technical work
2. SQA group who is responsible for QA planning, oversight, record keeping, analysis, and reporting

SQA Group Activities

An SQA plan is developed for the project during project planning and is reviewed by all stake holders. The plan includes the identification of:

• Evaluations to be performed
• Audits and reviewed to be performed
• Standards that are applicable to the project
• Procedures for error reporting and tracking
• Documents to be produced by the SQA group
• Amount of feedback provided to the software project team

The group participates in the development of the project's software process description. The software team selects the process and SQA group reviews the process description for compliance with the organizational policies, internal software standards, externally imposed standards, and other parts of the software project plan.

The SQA group also reviews software engineering activities to verify compliance with the defined software process. It identifies, documents, and tracks deviations from the process and verifies that the corrections have been made. In addition, it audits designated software work products to verify compliance with those defined as part of the software process. It, reviews selected work products, identifies, documents, and tracks deviations; verifies that corrections have been made; and reports the results of its work to the project manager.

The basis purpose is to ensure that deviations in software work and work products are documented and handled according to documented procedures. The group records any non compliance and reports to senior management and non-compliant items are recorded and tracked until they are resolved. Another very important role of the group is to coordinate the control and management of change and help to collect and analyze software metrics.

How are we going to control the quality of the product?

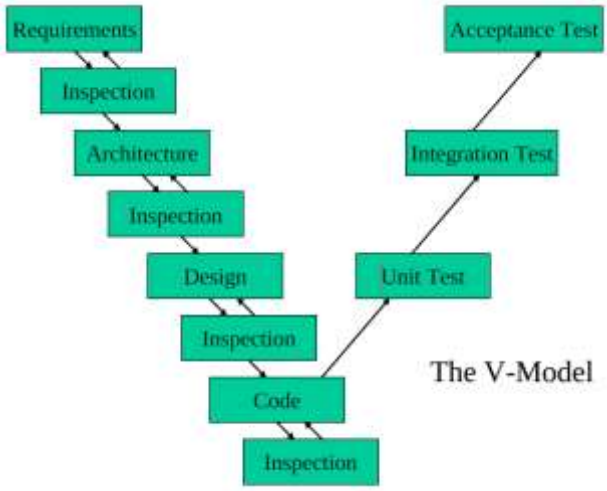
The basic principle of quality control is to control the variation as variation control is the heart of quality control. It includes resource and time estimation, test coverage, variation in number of bugs, and variation in support.

This involves a series of inspection, reviews, and tests and includes feedback loop. So quality control is a combination of measurement and feedback and combination of automated tools and manual interaction.

Software Reviews

Software reviews are the filter for the software engineering process. They are applied at various different points and serve to uncover errors that can be removed and help to purify the software engineering activities.

In this context it is useful to look at the "V-model" of software development. This model emphasizes that SQA is a function performed at all stages of software development life cycle. At the initial stages (requirement, architecture, design, code), it is achieved through activities known as Formal Technical Reviews or FTR.



Importance of reviews

Reviews help the development team in improving the defect removal efficiency and hence play an important role in the development of a high-quality product.

Freedman defines a review – any review – as a way of using the diversity of a group of people to:

• Point out needed improvements in the product of a single person or team

• Confirm those parts of a product in which improvement is either not desired or no needed

• Achieve technical work of more uniform, or at least more predictable, quality than can be achieved without reviews, in order to make technical work more manageable.

There are many types of reviews. In general they can be categorized into two main categories namely informal and formal technical reviews.

Formal Technical Reviews

Formal Technical Reviews are conducted by software engineers. The primary objective is to find errors during the process so that they do not become defects after release of software as they uncover errors in function, logic design, or implementation. The idea is to have early discovery of errors so they do not propagate to the next step in the process.

FTRs include walkthroughs, inspections, and other small group technical assessments of software.

Guidelines for walkthroughs

FTRs are usually conducted in a meeting that is successful only if it is properly planned, controlled, and attended. Meeting consists of 3-5 people, it should focus on specific (and small) part of the overall software, and the focus is on a work product

Review Meetings

It should focus on specific (and small) part of the overall software. It is important to remember that the focus is on a work product for which the producer of the WP asks the project leader for review. At the end of the RM, all attendees of the meeting must decide whether to:

• Accept the product without further modification
• Reject the product due to severe errors
  – Major errors identified
  – Must review again after fixing
• Accept the product provisionally
  – Minor errors to be fixed
  – No further review

Review Reporting and Record keeping

During the FTR the recorder notes all the issues. They are summarized at the end and a review issue list is prepared. A summary report is produced that includes:

• What is reviewed
• Who reviewed it
• What were the findings and conclusions

It then becomes part of project historical record.

The review issue list

It is sometimes very useful to have a proper review issue list. It has two objectives.

1. Identify problem areas within the WP
2. Action item checklist

Review Guidelines

The basic principle is that the review should focus on the product and not the producer so that it does not become personal. Remember to be sensitive to personal egos. Errors should be pointed out gently and the tone should be loose and constructive.

This can be achieved by setting an agenda and maintaining it. In order to do so, the review team should:

Avoid drift
• Limit debate and rebuttal
• Enunciate problem areas but don't try to solve all problems
• Take written notes
• Limit the number of participants and insist upon advanced preparation
• Develop a checklist for each product that is likely to be reviewed
• Allocate resources and schedule time for FTRs
• Conduct meaningful training for all reviewers
• Review your early reviews
• Determine what approach works best for you

Statistical Software Quality Assurance

Statistical SQA is a technique that measures the quality in a quantitative fashion. It implies that information about defects is collected and categorized and an attempt is made to trace each defect to underlying cause. It uses Pareto Principle to identify vital causes (80% of defects can be traced to 20% of causes) and moves to correct the problems that have caused the defects.

Error Index (EI)

Another statistical technique known as Error Index (EI) is used to develop an overall indication of improvement in software quality.

Now EI is computed as the cumulative effect on each $PI_i = \sum(i \times PI_i)/PS$

Software Reliability

Software reliability is another very important quality factor and is defined as probability of failure free operation of a computer program in a specified environment for a specified time. For example, a program X can be estimated to have a reliability of 0.96 over 8 elapsed hours. Software reliability can be measured, directed, and estimated using historical and development data. The key to this measurement is the meaning of term failure.

Failure is defined as non-conformance to software requirements. It can be graded in many different ways as shown below:

1. From annoying to catastrophic
2. Time to fix from minutes to months
3. Ripples from fixing

It is also pertinent to understand the difference between hardware and software reliability. Hardware reliability is predicted on failure due to wear rather than failure due to design. In the case of software, there is no wear and tear. The reliability of software is determined by Mean time between failure (MTBF). MTBF is calculated as:

MTBF = MTTF + MTTR

Where MTTF is the Mean Time to Failure and MTTR is the Mean time required to Repair.

Software Safety

Software Safety is a software SQA activity that focuses on identification of potential hazards that may affect software negatively and cause an entire system to fail. Modeling and analysis process is conducted as part of software safety and hazards are identified and categorized by criticality and risk.

Once system-level hazards are identified, analysis techniques are used to assign severity, and probability of occurrence. This technique is similar to risk analysis. To be effective, software must be analyzed in the context of the entire system.

Reliability and safety are closely related. Software reliability uses statistical techniques to determine the likelihood that a software failure will occur. Occurrence of a software failure does

not necessarily result in a hazard or mishap. On the other hand, software safety examines the ways in which failures result in conditions that can lead to a mishap.

Poka-Yoke (Mistake-Proofing)

Poka-yoke is a QA technique developed by Shingo at Toyota in 1960's. Poka-yoke devices are mechanisms that lead to prevention of potential quality problem before it occurs or the rapid detection of quality problems if they are introduced.

• Examples:

– Light on if the car door is not properly closed

– Warning beep if the engine is turned-off when lights are on

Characteristic of a Poka-yoke device

• It is simple and cheap

• It is part of the process

• It is located near the process task where the mistake occurs

Software Configuration Management (SCM)

You may recall that software configuration management (SCM) is one of the five KPA required for an organization to be at CMM level 2. The basic idea behind SCM is to manage and control change. As defined by CMM, the purpose of SCM is to establish and maintain the integrity or software products through the project's life cycle. SCM involves the development and application of procedures and standards to manage an evolving software product and is part of a more general quality management process.

As mentioned by Bersoff, no matter where you are in the system life cycle, the system will change, and the desire to change it will persist throughout the life cycle. It is therefore essential that we manage and control it in a fashion that this continuous change does not convert into chaos.

Change Chaos

This frequent change, if not managed properly, results in chaos. First of all there would be problems of identification and tracking which would result in questions like the following:

□□"This program worked yesterday. What happened?"

□□"I fixed this error last week. Why is it back?"

□□"Where are all my changes from last week?"

Then there are problems of version selection. The typical problems faced are:

□□"Has everything been compiled? Tested?"

□□"How do I configure for test, with my updates and no others?"

□□"How do I exclude this incomplete/faulty change?"

□□"I can't reproduce the error in this copy!"

Then there are software delivery problems.

□□“Which configuration does this customer have?”
□□“Did we deliver a consistent configuration?”
□□“Did the customer modify the code?”
□□“The customer skipped the previous two releases. What happens if we send him the new one?”

This is not all. There may be more chaos in the following shapes and forms:
□□The design document is out of sync with programs
□□I don’t know if all the changes that were suggested have been incorporated
□□Which code was sent to testing?

Configuration management

Configuration management is concerned with managing evolving software systems. It acknowledges that system change is a team activity and thus it aims to control the costs and effort involved in making changes to a system.
SCM involves the development and application of procedures and standards to manage an evolving software product and is part of a more general quality management process.

Baseline

When released to CM, software systems are called *baselines* and serve as the starting point for further development. A baseline is a software configuration management concept that helps us to control change without seriously impeding justifiable change. It is defined by IEEE as:
A specification or a product that has been formally reviewed and agreed upon, that thereafter serves as the basis for further development, and that can be changed only through formal change control procedures.

Software Configuration Item (SCI)

A Software Configuration Item is the information that is created as part of the software engineering process. Typical SCIs include requirement specifications, design specification, source code, test cases and recorded results, user guides and installation manuals, executable programs, and standards and procedures

Software Configuration Management Tasks

Software configuration management tasks include:

1. Identification
2. Version Control
3. Change Control
4. Configuration Auditing
5. Reporting

Identification addresses how does an organization identify and manage the many existing versions of a program

Version Control talks about how does an organization control changes before and after software is released to a customer? It is actually a combination of procedures and tools to manage different versions of the software configuration.
A version has many different attributes. In the simplest form a specific version number that is attached to each object and in the complex form it may have a string of Boolean variables (switches) that indicate specific types of functional changes that have been applied to the system.
The Change Control process addresses the important question of who has the responsibility for approving and ranking changes.
Configuration Auditing deals with ensuring that the changes have been made properly finally Reporting talks about the mechanism used to apprise others of changes that are made.

Product Release Version Numbering System

Product release is the act of making a product available to its intended customers. After a product has had its first release, it enters a product release cycle. New versions of the product are made available that may fix defects or add features that were not in previous releases. These changes are categorized as updates or upgrades. An update fixes product defects. An upgrade enhances the product feature set and will include updates.

Release Numbering

Release numbering is a mechanism to identify the product's functionality state. Each release will have a different product state and hence will have a different release number. Although there is no industry standard, typically, a three field compound number of the format "X.Y.Z" is used. The different fields communicate functionality information about the product release.

The first digit, X, is used for the major release number which is used to identify a major increase in the product functionality. The major release number is usually incremented to indicate a significant change in the product functionality or a new product base-line.
The second digit, Y, stands for feature release number. The feature release number is iterated to identify when a set of product features have been added or significantly modified from their originally documented behaviour.
The third digit, Z, is called the defect repair number and is incremented when a set of defects is repaired. Defect repair/maintenance is considered to be any activity that supports the release functionality specification and it may a fix for some bugs or some maintenance to enhance the performance of the application.

Internal Release Numbering

A special type of release is internal release. Internal releases are used by the development organization as a staging mechanism for functionality. The most common internal releases are the regular builds. A common way to number internal builds is to use the same release number that

would be used for final release with some additional information added to it to identify the built. It is suggested that we add an extra (fourth) field to identify and keep track of internal builds.

Change control

James Back points out the difficulties related to change control as follows:

Change control is vital. But the forces that make it necessary also make it annoying. We worry about change because a tiny perturbation in the code can cause a big failure in the product. But it can also fix a big failure or enable wonderful new capabilities. We worry about change because a single rogue developer could sink the project; yet brilliant ideas originate in the minds of those rogues, and a burdensome change control process could discourage them from doing creative work.

That is, like all engineering activities, change control is the name of a balancing act. Too much or too little change control creates different types of problems as uncontrolled change rapidly leads to chaos.

Change Control Process

The first component of the change control process is the Change Control Authority (CCA) or a Change Control Board (CCB).

Whenever a change is required, the CCB decides whether to allow this change to happen or deny it. If it is decided that a change is needed, an Engineering Change Order or ECO is generated. An ECO defines the change to be made, the constraints that must be respected, and the criteria for review and audit.

The change control process thus involves the following steps.

1) need for change is recognized
2) change request from user
3) developer evaluates
4) change report is generated
5) change control authority (CCA) decides
6) Either step 6a or 6b is performed. Steps numbers 7 to 17 are performed only if step 6b is performed.

a)

i) change request is denied
ii) user is informed
iii) no further action is taken.

b) assign people to SCIs
7) check-out SCIs
8) make the change
9) review/audit the change
10) check-in SCIs

11) establish a “baseline” for testing
12) perform SQA and testing activities
13) check-in the changed SCIs
14) promote SCI for inclusion in next release
15) rebuild appropriate version
16) review/audit the change
17) include all changes in release

Check-in and check-out

In SCM, the processes of Check-in and Check-out take a central stage. These are two important elements of change control and provide access and synchronization control.
Access control manages who has the authority to check-out the object and synchronization control ensures that parallel changes by two different people do not overwrite one another.
Synchronization control implements a control on updates. When a copy is checked-out, other copies can be checked out for use only but they cannot be modified. In essence, it implements a single-writer multiple-readers protocol.

Configuration Audit

Configuration audit ensures that a change has been properly implemented. It involves formal technical reviews and software configuration audit. Configuration audit assess a configuration object for characteristics that are generally not considered during audit.
It looks into the following aspects of the change:

- Has the change specified in the ECO been made? Have any additional modifications been incorporated?
- Has a FTR been conducted to assess technical correctness?
- Has the software process been followed?

Configuration Status reporting (CSR)

Configuration Status Reporting (CSR) is also known as status accounting. It reports on the following items:
– What happened?
– Who did it?
– When did it happen?
– What else will be affected?

Requirement Management and CMM

CM standards

CM should always be based on a set of standards which are applied within an organisation. Standards should define how items are identified, how changes are controlled and how new

versions are managed. Existing standards are based on a waterfall process model - new standards are needed for evolutionary development.

The Requirement Problem

The goal of software development is to develop quality software – on time and on budget – that meets customers' real needs. Project success depends on good requirement management. It may be recalled that requirement errors are the most common type of software development errors and the most costly to fix. It may also be recalled that requirement errors are listed as one of the roots causes of software project failure.

Requirement Management

Requirement Management is defined as a systematic approach to eliciting, organizing, and documenting the requirements of the system, and a process that establishes and maintains agreement between the customer and the project team on the changing requirements of the system. Requirements Management KPA Goals statement says that:

1. The software requirements are controlled to establish a baseline for software engineering and management use.
2. Software plans, products, and activities are kept consistent with the software requirements. It includes establishing and maintaining an agreement with the customer on the requirement for the software project. It involves

·□Defining the requirement baseline
·□Reviewing proposed requirement changes and evaluating the likely impact of each proposed change before deciding whether to approve it
·□Incorporating approved requirement changes in the project in a controlled manner
·□Keeping project plans current with the requirements
□□Tracking requirement status and change activity throughout the project

Requirement Attributes

We need to tag requirements with certain attributes in order to manage them in an orderly fashion. Attributes are used to establish a context and background for each requirement. They can be used to filter, sort, or query to view selected subset of the requirements. A list of possible attributes is enumerated as below:

1. Requirement ID
2. Creation date
3. Created by
4. Last modified on
5. Last modified by
6. Version number
7. Status
8. Origin
9. Subsystem
10. Product Release

11. Priority

Requirement Status

The requirement status attribute is one of the most useful ones. It can be used to keep track of different requirements going through different phases. The possible status values are proposed, approved, implemented, verified, and deleted. These are elaborated in the following paragraphs.

1. Proposed: The requirement has been requested by a source who has the authority to provide requirements.
2. Approved: The requirement has been analyzed, its impact on the rest of the project has been estimated, and it has been allocated to the baseline for a specific build number or product release. The software development group has committed to implement the requirement.
3. Implemented: The code that implements the requirement has been designed, written, and unit tested.
4. Verified: The implemented requirement has been verified through the selected approach, such as testing or inspection. The requirement has been traced to pertinent test cases. The requirement is now considered complete.
5. Deleted: A planned requirement has been deleted from the baseline. Include an explanation of why and by whom the decision was made to delete the requirement.

Change Request Status

As the requirement go through different phases, their status is updated accordingly.

Managing Scope Creep

We must always remember that requirements will change, no matter what**.** That means we have to be able to manage changing requirements. Software organizations and professionals must learn to manage changing requirements. We therefore need to try to take it to a minimum level. For that we need to measure the change activity.

Measuring Change Activity

Measurement of change activity is a way to assess the stability of the requirements and to identify opportunities for process improvement. In this regards, the following could be measured

- ·The number of change requests received, open, and closed
- ·The cumulative number of changes made including added, deleted, and modified requirements
- ·The number of change requests that originated from each source
- The total effort devoted to handling changes

Requirement Traceability

It is important to trace requirements both ways. That is from origin of a requirement to how it is implemented. This is a continuous process. It is also important that the rationale of requirements must also be traced. Traceability is important for the purposes of certification, change impact analysis, maintenance, project tracking, reengineering, reuse, risk reduction, and testing. That is it plays an important role in almost every aspect of the project and its life cycle.

Legacy systems

A system is considered to be a legacy system if it has been in operation for many years. A legacy system has many components. These include business processes, business rules, application software, application data, support software, and system hardware.

Maintaining Legacy System

Maintaining legacy system is expensive. It is often the case that different parts of the system have been implemented by different teams, lacking consistency. Part or all of the system may be implemented using an obsolete language. In most cases system documentation is inadequate and out of date. In some cases the only documentation is the source code. In some cases even the source code is not available.

Many years of maintenance have usually corrupted the system structure, making it increasingly difficult to understand. The data processed by the system may be maintained in different files which have incompatible structures. There may be data duplication and the documentation of the data itself may be out of date, inaccurate, and incomplete.

As far as the system hardware is concerned, the hardware platform may be outdated and is hard to maintain. In many cases, the legacy systems have been written for mainframe hardware which is no longer available, expensive to maintain, and not be compatible with current organizational IT purchasing policies.

A time therefore comes when an organization has to make this decision whether to keep the old legacy system or to move it to new platform and environment. Moving it to new environment is known as legacy system migration.

Legacy migration risks

Legacy system migration however is not an easy task and there are a number of risks involved that need to be mitigated. First of all, there is rarely a complete specification of the system available. Therefore, there is no straight forward way of specifying the services provided by a legacy system. Thus, important business rules are often embedded in the software and may not be documented elsewhere. Business processes and the way legacy systems operate are often intertwined. New software development may take several years. New software development is itself risky as changes to one part of the system inevitably involve further changes to other components.

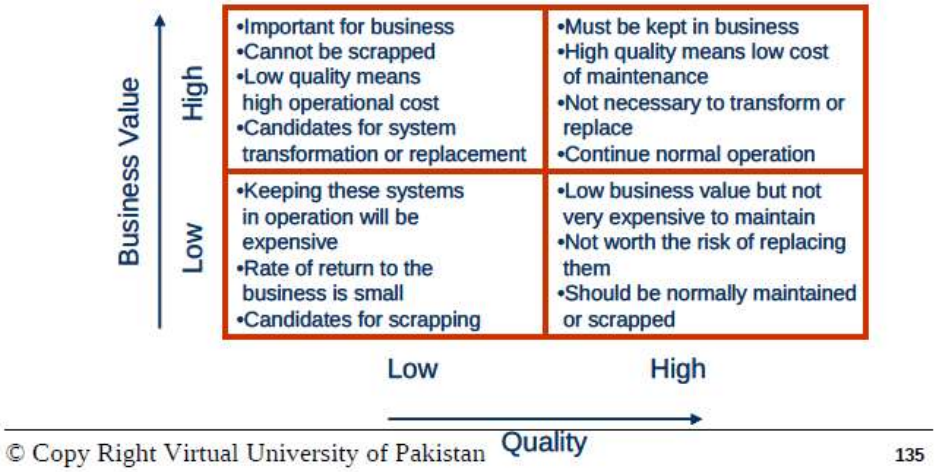
Legacy System Assessment

For each legacy system, there are four strategic options:

1. Scrap the system completely: This is the case when system is not making an effective contribution to business processes and business processes have changed significantly and the organization is no longer completely dependent upon the system.
2. Continue maintaining the system: This option is used when system is still required, it is stable, and requirements are not changing frequently

3. Transform the system in some way to improve its maintainability: this option is exercised when system quality has been degraded and regular changes to the system are required.
4. Replace the system with a new system: this path is taken when old system cannot continue in operation and off-the shelf alternative is available or system can be developed at a reasonable cost.

business value and quality. The following four quadrant assessment matrix can be used for this purpose.



Business Value Assessment

End Users assess the system from the perspective of how effective do they find the system in supporting their business processes and how much of the system functionality is used. The customers look at the system and ask is the use of the system transparent to customer or are their interaction constrained by the system, are they kept waiting because of the system, and do system errors have a direct impact on the customer.

From an IT Manager's perspective the following questions need to be asked: Are there difficulties in finding people to work on the system?

Line Managers ask: do managers think that the system is effective in contributing to success of their unit? Is the cost of keeping the system in use justified?

Senior Managers look at the system from the angle that does the system and associated business process make an effective contribution to the business goal?

Environment Assessment

The legacy system also needs to be assessed from an environment's perspective. This involves looking at the supplier, failure rate, age, performance, support requirements, maintenance cost, and interoperability.

These angles are elaborated in the following paragraph:

Supplier stability: Is the supplier still in existence? Is the supplier financially stable and likely to continue in existence? If the supplier is no longer in business, is the system maintained by someone else?
Failure rate: Does the hardware have a high rate of reported failure? Does the support software crash often and force system restarts?
Age: How old is the hardware and software?
Performance: Is the performance of the system adequate? Do performance problems have a significant effect on system users?
Support requirements: What local support is required by hardware and software? If there are high costs associated with this support, it may be worth considering system replacement?
Maintenance Cost: What are the costs of hardware maintenance and software licenses?
Interoperability: Are there problems interfacing the system with other systems? Can compilers etc be used with current versions of the operating system? Is system emulation required?

Application software assessment

The application software is assessed on the following parameters:
Understandability: How difficult is it to understand the software code of the current system? How complex are the control structures that are used?
Documentation: What system documentation is available? Is the documentation complete, consistent, and up-to-date?
Data: Is there an explicit data model for the system? To what extent is data duplicated in different files?
Programming Language: Are modern compilers available for the programming language? Is the language still used for new system development?
Test Data: Does test data for the system exist? Is there a record of regression tests carried out when new features have been added to the system?
Personnel skills: Are there people available who have the skills to maintain the system?

Software Reengineering

Software solutions often automate the business by implementing business rules and business processes. In many cases, the software makes the business processes. As these rules and processes change, the software must also change. A time comes when these changes become very difficult to handle. So reengineering is re-implementing legacy systems to make them more maintainable. It is a long term activity.

Software Reengineering Process Model

The software reengineering is a non-trivial activity. Just like legacy migration, careful analysis must be carried out before a decision for reengineering is taken. The following process model can be used to reengineer a legacy system.

Inventory analysis

Inventory analysis is the first step in the reengineering process. At this stage, inventory ofall applications is taken a note of their size, age, business criticality, and current maintainability is made. Inventory should be updated regularly as the status of the application can change as a function of time.

Document restructuring

Weak documentation is a trademark of many legacy applications. Without proper documentation, the hidden rules, business processes, and data cannot be easily understood and reengineered. In this regards, the following options are available:

1. Create documentation: Creating documentation from scratch is very time consuming. If program is relatively stable and is coming to the end of its useful life then just leave it as it is.
2. Update documentation: This option also needs a lot of resources. A good approach would be to update documentation when the code is modified.

Reverse engineering

Reverse engineering for software is a process for analyzing a program in an effort to create a representation of the program at a higher level of abstraction than the source code.Reverse engineering is the process of design recovery. At this stage, documentation of the overall functionality of the system that is not there is created. The overall functionality of the entire system must be understood before more detailed analysis can be carried out.

Reverse engineering activities include:

- Reverse engineering to understand processing
- Reverse engineering to understand data
  - Internal data structures
  - Database structures
- Reverse engineering user interfaces

Program Restructuring

In this case we modify source code and data in order to make it amenable to future changes. This includes code as well as data restructuring. Code restructuring requires redesign with same

function with higher quality than original program and data restructuring involves restructuring the database or the database schema. It may also involve code restructuring.

Forward Engineering

It means incorporating the new business processes and rules in the system. Forward engineering requires application of SE principles, methods, and concepts to re-create an existing application.

The Economics of Reengineering

As reengineering is a costly and risky undertaking, a cost benefit analysis for the reengineering effort must be carried out.

This analysis is carried out in the following manner.

Let

P1 : current annual maintenance cost for an application
P2 : current annual operation cost for an application
P3 : current annual business value of an application
P4 : predicted annual maintenance cost after reengineering
P5 : predicted annual operations cost after reengineering
P6 : predicted annual business value cost after reengineering
P7 : estimated reengineering cost
P8 : estimated reengineering calendar time
P9 : reengineering risk factor (1.0 is nominal)
L : expected life of the system

Now the cost of maintenance is calculated as:

$$C_{maintenance} = [P3 - (P1 + P2)] \times L$$

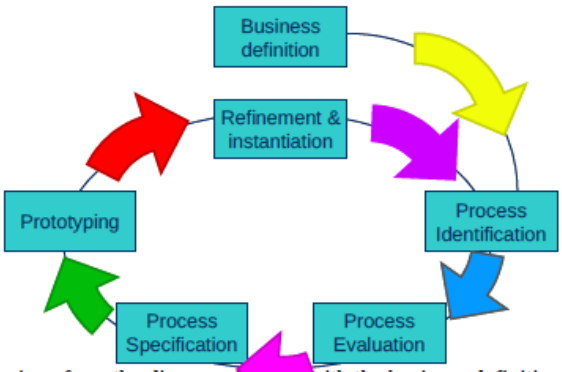
Cost of reengineering would then be given by the formula:

$$C_{reengineering} = [P6 - (P4 + P5) \times (L - P8) - (P7 \times P9)]$$

Business Process Reengineering

A concept similar to software reengineering is of business process reengineering (BPR). A business process is "a set of logically related tasks performed to achieve a defined business outcome". It is the way certain business is conducted. Purchasing services and supplies, hiring new employees, paying suppliers are examples of business processes.

For BPR the following process model may be used.



As obvious from the diagram, it ...vith the business definition where business goals are identified. The key drivers could be cost reduction, time reduction, quality improvement, and personnel development and empowerment. It may be defined at the business level or for a specific component of the business.

The next step is process identification. At this time processes that are critical to achieving the goals are identified and are ranked by importance, and need for change.

The short listed processes are then evaluated. Existing process is analyzed and measured and process tasks are identified. The cost and time consumed is measured as well as the quality and performance problems are identified.

Then, process specification and design is carried out. Use cases are prepared for each process to be redesigned and a new set of tasks are designed for the processes and then they are prototyped.

A redesigned business process must be prototyped before it is fully integrated into the business.

Based on the feedback the business process is refined.

Software Refactoring

Software refactoring is the process of changing a software system such that the external behavior of the system does not change while the internal structure of the system is improved. This is sometimes called "Improving the design after it has been written". Fowler defines refactoring as A change made to the internal structure of software to make it easier to understand and cheaper to modify without changing its observable behavior. It is achieved by applying a series of refactorings without changing its observable behavior.

Refactoring: Where to Start?

The first question that we have to ask ourselves is: how do you identify code that needs to be refactored? Fowler et al has devised a heuristic based approach to this end known as "Bad Smells" in Code. The philosophy is simple: ***"if it stinks, change it".***

Bad Smells in Code

They have identified many different types of "bad smells". These are briefly described in the following paragraphs:

 Duplicated Code
– bad because if you modify one instance of duplicated code but not the others, you (may) have introduced a bug!
 Long Method
– long methods are more difficult to understand; performance concerns with respect to lots of short methods are largely obsolete
 Large Class
– Large classes try to do too much, which reduces cohesion
 Long Parameter List
– hard to understand, can become inconsistent
 Divergent Change
– Deals with cohesion; symptom: one type of change requires changing one subset of methods; another type of change requires changing another
subset
 Shotgun Surgery
– a change requires lots of little changes in a lot of different classes
 Feature Envy
– A method requires lots of information from some other class (move it closer!)
 Data Clumps
– attributes that clump together but are not part of the same class
 Primitive Obsession
– characterized by a reluctance to use classes instead of primitive data types
 Switch Statements
– Switch statements are often duplicated in code; they can typically be replaced by use of polymorphism (let OO do your selection for you!)
 Parallel Inheritance Hierarchies
– Similar to Shotgun Surgery; each time I add a subclass to one hierarchy, I need to do it for all related hierarchies
 Lazy Class
– A class that no longer “pays its way”. e.g. may be a class that was downsized by refactoring, or represented planned functionality that did not pan out
 Speculative Generality
– “Oh I think we need the ability to do this kind of thing someday”
 Temporary Field
– An attribute of an object is only set in certain circumstances; but an object should need all of its attributes
 Message Chains
– a client asks an object for another object and then asks that object for another object etc. Bad because client depends on the structure of the navigation
 Middle Man

– If a class is delegating more than half of its responsibilities to another class, do you really need it?

 Inappropriate Intimacy

– Pairs of classes that know too much about each other's private details

 Alternative Classes with Different Interfaces

– Symptom: Two or more methods do the same thing but have different signature for what they do

 Incomplete Library Class

– A framework class doesn't do everything you need

 Data Class

– These are classes that have fields, getting and setting methods for the fields, and nothing else; they are data holders, but objects should be about data AND behavior

 Refused Bequest

– A subclass ignores most of the functionality provided by its superclass

 Comments (!)

– Comments are sometimes used to hide bad code

Extracting the Amount Calculation

The obvious first target of my attention is the overly long statement() method. When I look at a long method like that, I am looking to take a chunk of the code an *extract a method* from it. Extracting a method is taking the chunk of code and making a method out of it. An obvious piece here is the switch statement

Moving the amount calculation

As I look at amountOf, I can see that it uses information from the rental, but does not use information from the customer. This method is thus on the wrong object, it should be moved to the rental. To *move a method* you first copy the code over to rental, adjust it to fit in its new home and compile.

I like to get rid of temporary variables like thus as much as possible. Temps are often a problem in that they cause a lot of parameters to get passed around when they don't need to. You can easily lose track of what they are there for. They are particularly insidious in long methods. Of course there is a small performance price to pay, here the charge is now calculated twice. Most refactoring reduce the amount of code, The other concern with this refactoring lies in performance.

Capability Maturity Model Integration (CMMI)

Capability Maturity Model or CMM is a reference model of mature practices in a specified discipline, used to assess a group's capability to perform that discipline. They differ by discipline (software, systems, acquisition, etc.), structure (staged versus continuous), how maturity is defined (process improvement path), and how capability is defined (institutionalization). Hence "Capability Maturity Model®" and CMM® are used by the Software Engineering Institute (SEI) to denote a particular class of maturity models.

Name	Structure	Domain
Software CMM	staged	software development
System Engineering CMM	continuous	system engineering
Software Acquisition CMM	staged	software acquisition
System Security Engineering CMM	continuous	security engineering
FAA-iCMM	continuous	software engineering, systems engineering, and acquisition
IPD-CMM	hybrid	integrated product development
People CMM	staged	workforce
SPICE Model	continuous	software development

CMMI Integrates systems and software disciplines into single process improvement framework and provides a framework for introducing new disciplines as needs arise.

CMMI Representations

A representation allows an organization to pursue different improvement objectives. There are two types of representations in the CMMI models: staged and continuous. The organization and presentation of the data are different in each representation.

Staged Representation

Staged representation is classical representation we have already seen previously. It: □□Provides a proven sequence of improvements, each serving as a foundation for the next.

□□Permits comparisons across and among organizations by the use of maturity levels.

□□Provides an easy migration from the SW-CMM to CMMI.

□□Provides a single rating that summarizes appraisal results and allows comparisons among organizations

Continuous Representation

Continuous representation allows you to select the order of improvement that best meets your organization's business objectives and mitigates your organization's areas of risk. It enables comparisons across and among organizations on a process-area-by-process-area basis and provides an easy migration from EIA 731 (and other models with a continuous representation) to CMMI.

Capability Levels

A capability level is a well-defined evolutionary plateau describing the organization's capability relative to a process area. There are six capability levels. For capability levels 1-5, there is an associated generic goal. Each level is a layer in the foundation for continuous process improvement. Thus, capability levels are cumulative, i.e., a higher capability level includes the attributes of the lower levels. The five (actually six) capability levels (starting from 0) are enumerated below in the reverse order, 5 being the highest and 0 being the lowest.

5 Optimizing

4 Quantitatively Managed

3 Defined

2 Managed

1 Performed
0 Incomplete

Relating Process Area Capability and Organizational Maturity

Organizational maturity is the focus of the staged representation, whereas process area capability is the focus of the continuous representation. The difference between them is that organizational maturity pertains to a set of process areas across an organization, while process area capability deals with a set of processes relating to a single process area or specific practice.

Comparison of Representations

Staged

- Process improvement is measured using maturity levels.
- Maturity level is the degree of process improvement across a predefined set of process areas.
- Organizational maturity pertains to the "maturity" of a set of processes across an organization

Continuous

- Process improvement is measured using capability levels.
- Capability level is the achievement of process improvement within an individual process area.
- Process area capability pertains to the "maturity" of a particular process across an organization.

Advantages of Each Representation

Staged provides a roadmap for implementing groups of process areas and sequencing of implementation. It has a familiar structure for those transitioning from the Software
CMM.

Continuous provides maximum flexibility for focusing on specific process areas according to business goals and objectives and has a familiar structure for those transitioning from EIA 731. As the staged representation requires all KPAs to be addressed at a particular level before a company can move to the next maturity level, it may not be easy for small companies to implement this model. There may be a number of activities that may not be relevant to their type of work but they would still have to do them in order to be at a certain level. On the other hand, organization can focus on their own areas of expertise and may be able to achieve high capability levels in some areas without bothering about the rest. This is a great advantage for small organization and hence this model is believed to be more suitable for small Pakistani organizations than the staged one.

CMM Overview

CMM Maturity Levels

There are five levels defined along the continuum of the CMM and, according to the SEI: "Predictability, effectiveness, and control of an organization's software processes are believed to improve as the organization moves up these five levels. While not rigorous, the empirical evidence to date supports this belief."

Level 1 - Ad hoc (Chaotic)

It is characteristic of processes at this level that they are (typically) undocumented and in a state of dynamic change, tending to be driven in an *ad hoc*, uncontrolled and reactive manner by users or events. This provides a chaotic or unstable environment for the processes.

Organizational implications

Institutional knowledge tends to be scattered (there being limited structured approach to knowledge management) in such environments, not all of the stakeholders or participants in the processes may know or understand all of the components that make up the processes. Despite the chaos, such organizations manage to produce products and services.

Due to the lack of structure and formality, organizations at this level may overcommit, or abandon processes during a crisis, and it is unlikely that they will be able to repeat past successes.

Level 2 - Repeatable

It is characteristic of processes at this level that some processes are repeatable, possibly with consistent results. Process discipline is unlikely to be rigorous, but where it exists it may help to ensure that existing processes are maintained during times of stress.

Organizational implications

Processes and their outputs could be visible to management at defined points, but results may not always be consistent.

Level 3 - Defined

It is characteristic of processes at this level that there are sets of defined and documented standard processes established and subject to some degree of improvement over time. These standard processes are in place (i.e., they are the AS-IS processes) and used to establish consistency of process performance across the organization.

Organizational implications

Process management starts to occur using defined documented processes, with mandatory process objectives, and ensures that these objectives are appropriately addressed.

Level 4 - Managed

It is characteristic of processes at this level that, using process metrics, management can effectively control the AS-IS process (e.g., for software development). In particular, management can identify ways to adjust and adapt the process to particular projects without measurable losses of quality or deviations from specifications. Process Capability is established from this level.

Organizational implications

a) Quantitative quality goals tend to be set for process output - e.g., software or software maintenance.

b) Using quantitative/statistical techniques, process performance is measured and monitored and generally predictable and controllable also.

Level 5 - Optimizing

It is a characteristic of processes at this level that the focus is on continually improving process performance through both incremental and innovative technological changes/improvements.

Organizational implications

(a) Quantitative process-improvement objectives for the organization are established, continually revised to reflect changing business objectives, and used as criteria in managing process improvement. Thus, process improvements to address common causes of process variation and measurably improve the organization's processes are identified, evaluated, and deployed.

(b) The effects of deployed process improvements are measured and evaluated against the quantitative process-improvement objectives.

(c) Both the defined processes and the organization's set of standard processes are targets for measurable improvement activities.

(d) A critical distinction between maturity level 4 and maturity level 5 is the type of process variation addressed.

At maturity level 4, processes are concerned with addressing statistical *special causes* of process variation and providing statistical predictability of the results, and though processes may produce predictable results, the results may be insufficient to achieve the established objectives.

At maturity level 5, processes are concerned with addressing statistical *common causes* of process variation and changing the process (for example, shifting the mean of the process performance) to improve process performance.

Extensions

Some versions of CMMI from SEI indicate a "level 0", characterized as "Incomplete". Some pundits leave this level out as redundant or unimportant, but Pressman and others make note of it.

The basic building blocks in every CMMI model are called "process areas." A process area does not describe *how* an effective process is executed (e.g., entrance and exit criteria, roles of participants, resources). Instead, a process area describes *what* those using an effective process do (practices) and *why* they do those things (goals).

Continuous Representation	Staged Representation
Grants explicit freedom to select the order of improvement that best meets the organization's business objectives. Enables increased visibility into the capability achieved within each individual process area. Supports a focus on risks specific to	Enables organizations to have a predefined and proven path. Builds on a relatively long history of use. Case studies and data exist that show return on investment. Permits comparisons across and among

individual process areas.	organizations.

Project Management Concerns

Project management is the discipline of planning, organizing and managing resources to bring about the successful completion of specific project goals and objectives. There are several approaches that can be taken to managing project activities including agile, interactive, incremental, and phased approaches.

The traditional approach

A traditional phased approach identifies a sequence of steps to be completed. In the "traditional approach", we can distinguish 5 components of a project (4 stages plus control) in the development of a project:

Typical development phases of a project

- Project initiation stage;
- Project planning or design stage;
- Project execution or production stage;
- Project monitoring and controlling systems;
- Project completion stage.

Software Quality Assurance

Software quality assurance (SQA) consists of a means of monitoring the software engineering processes and methods used to ensure quality. The methods by which this is accomplished are many and varied, and may include ensuring conformance to one or more standards, such as ISO 9000 or CMMI.

This definition emphasizes upon three important points:

• Software requirements are the foundation from which quality is measured. Lack of conformance is lack of quality
• Specified standards define a set of development criteria that guide the manner in which software is engineered. If the criteria are not followed, lack of quality will almost surely result.
• A set of implicit requirements often goes unmentioned (ease of use, good maintainability etc.)