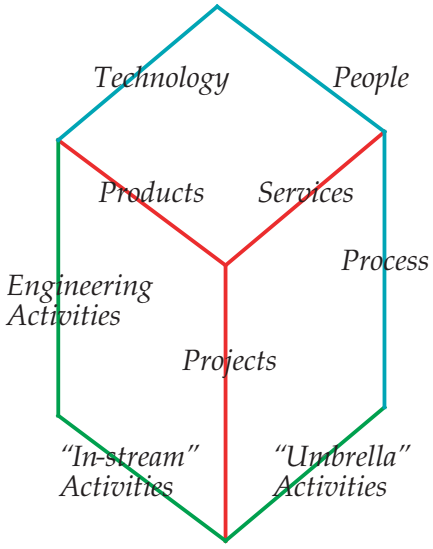
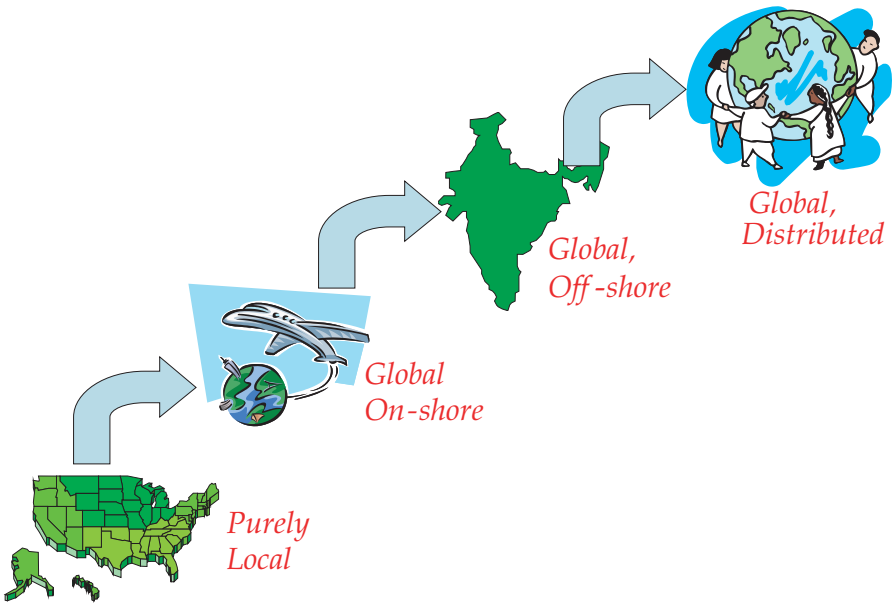
Part I:

Setting the Scene

This Part sets the context for the course by defining the current scenario in global software project management, tracing a typical product life cycle and examining the Vision → Products → Projects continuum. This leads to the project development life cycle models and the importance of processes in project management.



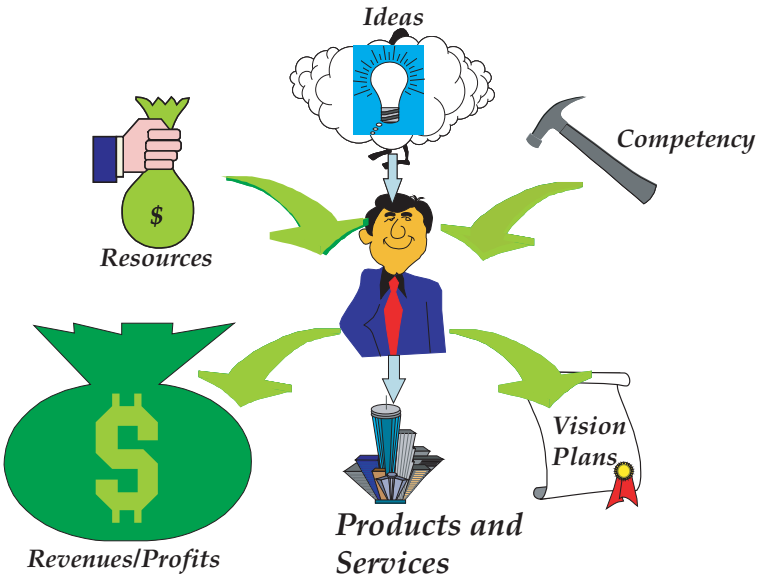

Purely
Local
Global
On-shore
Global,
Off-shore
Global,
Distributed

Software Product Life Cycle

"Think of the chaos that would come if everybody did their best without knowing what to do"

–Edward Deming

Every organization is driven by a vision. This vision is formulated by a few "shapers" in the organization (some of whom are usually from the top management) who come endowed with certain competencies and come up with some ideas. They also get some funding (either from past successes of the organization, external sources or a combination of the two). Then the organization comes out with a plan to realize the vision that translates to products (or services) that generate revenues and profits.



Not all ideas translate to successful products or services. Ideas have to successfully pass through stringent *filters* at multiple levels to become successful products or services. Some of the filters are feasibility ("can we do it?"), consistency ("is this in line with our vision?"), and viability ("can we do this profitably?"). As with a typical high profile championship like the Cricket World Cup or Wimbledon, there are a lot more aspirants and only the toughest survive all the rounds to reach the Big League. An organization takes on the mix of projects that pass through all the filters and formulates an aggregate product plan that is intended to give the maximum bang for the buck.

A Product Development Life Cycle consists of six phases: During the ***Idea Generation*** phase, a number of ideas (from various stakeholders like employees, customers, suppliers, etc.) are evaluated and a set of high level filters are put in place, based on which it is decided what products to take on. Some of the ideas go though a ***Prototyping*** phase, wherein a model of the intended product is built and evaluated by the various stakeholders. Products that pass the prototyping phase go on to Alpha Testing. During ***Alpha Testing***, a more complete product is built and deployed internally in real-life scenarios. This internal testing enables weeding out defects before the customers receive the product. The product then moves on to ***Beta Testing***, wherein the product is deployed at some select customer sites are trusted partners, with an interest to get into the current technologies, fully aware that the product may have some defects. The feedback during the Beta testing is used to refine the product and make it more robust. Also, since external customers use the product, its other aspects–like documentation and training–mature during this phase. The product is now ready for prime time and goes into ***Production*** and the its usage becomes more widespread. It now enters the ***Maintenance*** phase, wherein defects in the product are fixed. Finally it goes into ***Obsolescence*** wherein the product is de-supported and replaced by another product with enhanced functionality.

Each of the phases results in multiple projects. For example, during the Beta phase, projects like the user documentation,

installation documentation, etc get kicked off. A project satisfies the following attributes:

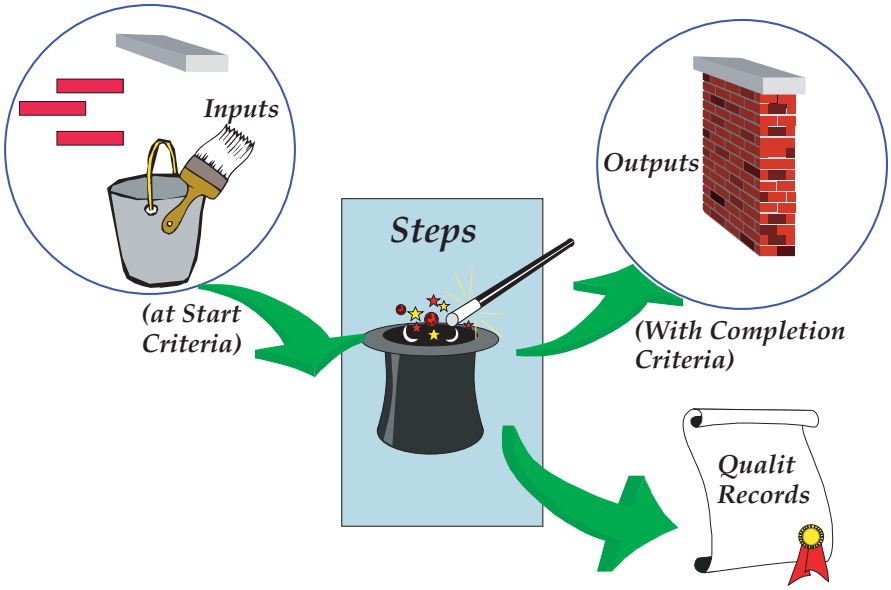
1. It is a part of the set of things that an organization decides to undertake to further its vision and goals (revenues/profits). In other words, each project satisfies a definite purpose that fits in the "big picture" of the organization
2. Each project has a definite beginning and a definite end time
3. Each project has to be executed under certain constraints; these constraints may change with time and that is part of the project dynamics. Examples of such constraints could be the availability of manpower, capital budget or "drop dead dates" for project completion.

Processes and Process Models

"End does not justify the means"

–Mahatma Gandhi

Products produced by an organization are deliverables to external or internal customers. A process is a documented and followed method that constitutes one or more steps in producing a product. We have seen process as an important dimension of the Project Management Cube.



Every process has certain attributes. Firstly, a process operates on a certain set of *inputs* to produce a set of *outputs*. This transformation happens by executing a series of *steps*. The steps can begin only when some pre-defined start *criteria* are met with and end when pre-defined *completion criteria* are met. *Quality Records* prove the successful completion of the process (as per the steps). For example, in a software development project, the coding process takes, as input, the program specifications and produces, as output, the program code. The steps involved could be writing the code, getting it reviewed, compiling it, testing it, etc. The Start Criteria may be, the program specification being ready and the Completion Criteria may be, the program passing all the unit tests. Acceptance of the program by the testing team may be viewed as a Quality Record.

An effective process starts from what is being practiced. The actual practice of the same (or similar) function may be different in different parts of an organization. A practitioner (or a set of practitioners) abstracts the commonalities and best practices from what is being practiced to a documented process. People in the organization are then trained on the use of these abstracted, documented processes. In order to ensure that the training and compliance to the processes are effective, the processes are enforced by means of audits, reviews, inspections, etc. Finally, as the processes have to adapt to the changing business needs and be continuously improved, they are constantly monitored and reviewed to ensure that they continue to be effective and applicable to multiple projects in the organization.

Effective processes reduce a product's time to market and simplify the training of employees coming into an organization. They also enable better cross-fertilization of knowledge across an organization. All these lead to better predictability and consistency– which are hallmarks of good quality.

Process Models are formed by sharing the industry-wide best practices which make business sense. Consortiums and organizations formulate process models that not only capture best

industry practices but also provide a framework for organizations to evaluate themselves in comparison with the model to see where they stand. This evaluation can result in a certification that is well recognized as a significant achievement and which has a bearing on what to expect from the organization.

Popular process models lay out the common functions that need to be carried out to achieve consistency and predictability in the results and quality of deliverables. Rather than view these models as abstract certification-focused exercises, it is worthwhile to view them as the embodiment of sound business practices. Viewed in this sense, one can observe a lot of similarity among all these models and better appreciate their relevance to project management.

Software Development Life Cycle Models

"The whole is more than sum of its parts"

–Aristotle

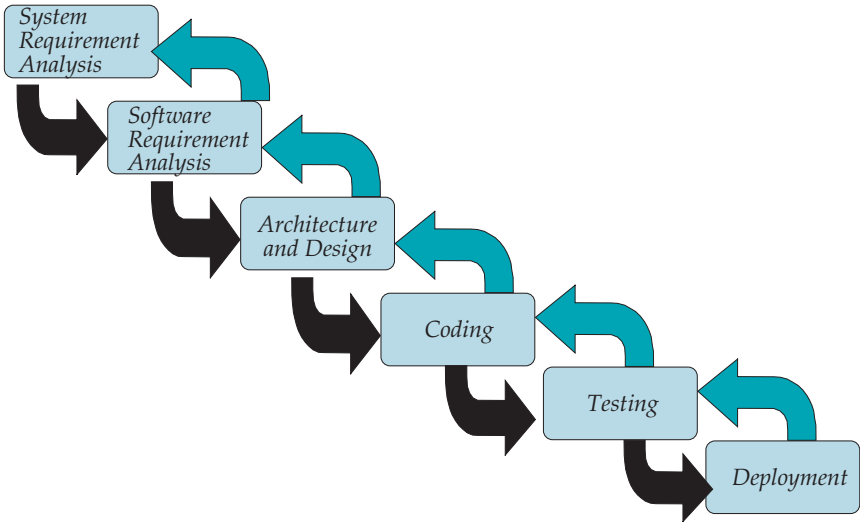
A ***Software Development Life Cycle*** (SDLC) Model of a project describes the sequence of steps followed by the project and the interactions among the different steps to produce a tangible software product that faithfully and fully meets the requirements of the component or part that the project set forth to build.

A number of life cycle models that are relevant to software development have been defined in the software engineering literature. While looking at these models from the project management perspective, the key questions to ask are:

- What does a model entail in terms of steps?
- What are the interaction mechanisms among the steps?
- Under what scenarios would a given life cycle model be applicable?
- Under what scenarios would a given life cycle model not be applicable?

The ***Waterfall Model***, also known as the ***Linear Sequential Model*** was one of the earliest life cycle models. In this,

(a) the project is divided into a sequence of well-defined (almost water tight) phases



(b) one phase is completed before the next phase starts

(c) there is a feedback loop between adjacent phases

(d) what the actual phases are depends on the project

If a problem is uncovered in a given phase, then the people working on that phase ascertain whether the problem is because of wrongful execution by them in the present phase or whether it is because of an error in the previous phase. If it is an internal error in the present phase, they correct it and proceed. If, on the other hand, the problem is found to be the result of an error in the previous phase, it is communicated to the previous phase. The previous phase owners correct the problem, do whatever tests are needed and pass the project on to the next phase again. It is also possible that the owners at the receiving phase may find that the problem is in the phase previous to them and the problem may get cascaded all the way up to the very first phase.

The main advantage of this model is the simplicity and the ability to line up resources of appropriate skill sets sequentially. But the main disadvantages are that it is not practical and that error can get propagated at least one level before being detected.

Prototyping and ***Rapid Application Development*** (***RAD***) Model are both based on early involvement of the customers to provide feedback on the product functionality. In both cases, the customer's requirements are captured via a prototype or a model. In the case of the Prototyping Model, the customer's feedback and the prototype are used to arrive at the requirements of the product and then the prototype is thrown away. In the case of the Rapid Application Development Model, a CASE (Computer Aided Software Engineer-ing) tool is used to automatically generate the customer application, in effect making a prototype that will not be discarded.

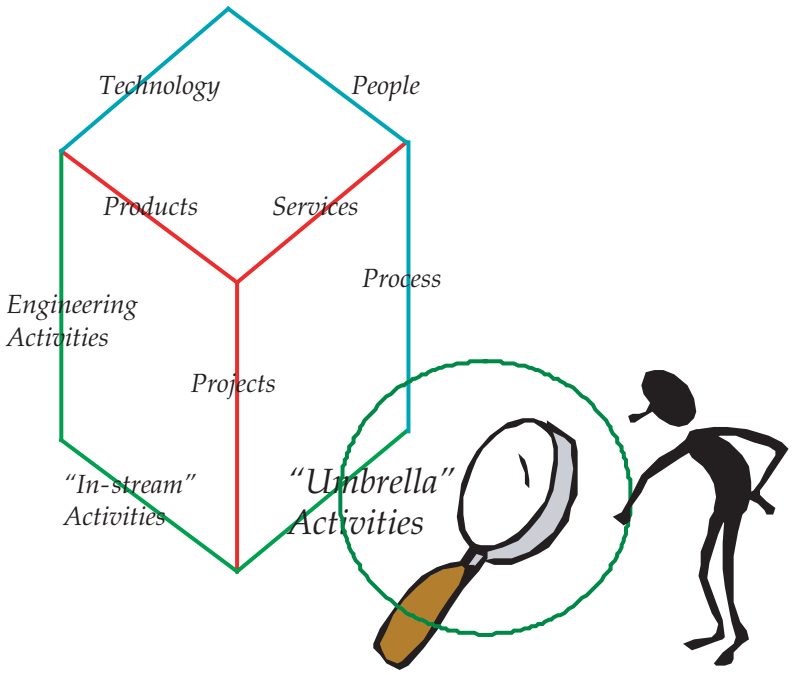
The ***Spiral Model*** combines the best of all the previous models, by visualizing the entire software development life cycle as a spiral made up of iterations (outward movement of the spiral), with each iteration being made up of sectors or phases. Each iteration builds from the previous iteration and can be taken as a level of completion of the product. As the radius of the spiral increases, the project becomes increasingly complete.

The success of a project depends upon the choice of an appropriate software development life cycle model to minimize communication overheads and maximize effectiveness.

Part II:

Umbrella Activities

This Part deals with the Umbrella activities in a project. These activities taken place throughout the project. Umbrella activities include Metrics, which are about measurements, Software Configuration Management, which deals with Change Management, Software Quality Assurance which involves activities like reviews, audits, etc and finally Risk Management, which is about anticipating, quantifying and mitigating against risks in a software project.




SQA
Metrics
Configuration
Management
Risk
Management

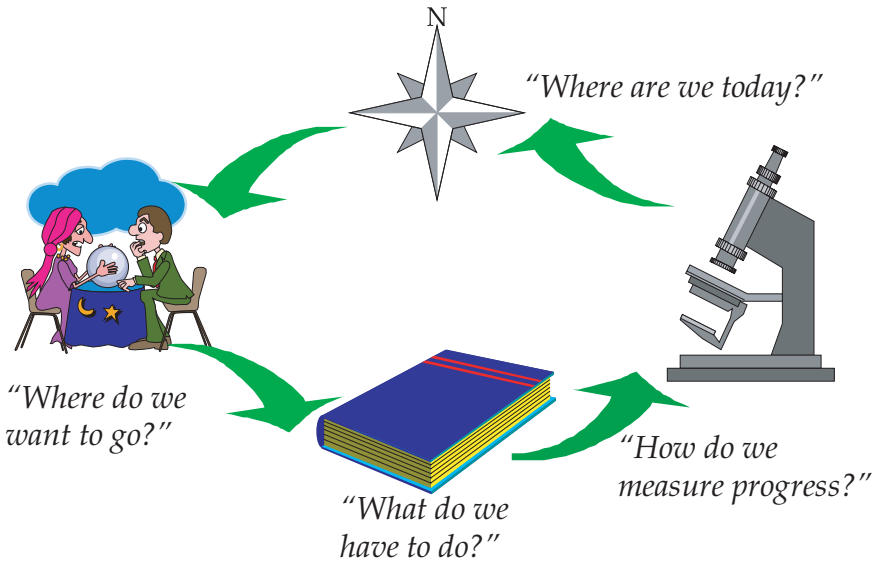
Metrics

"Goals – The first step towards getting somewhere is to decide that you are not going to stay where you are"

–John Pierpont Morgan

Metrics in a Project Management context is about measurements– measuring your progress in order to know where you are and what mid-course corrections you need to make to achieve your goals. There are two distinct (but related) components that are touched upon in the above sentence: The first is that metrics are measurements of one's ***progress***–we call these ***in-process metrics***. The second is the measurement of how well you have achieved the ***goals***–we call these the ***end-result metrics*** or ***business goals***. Every activity, be it an "Umbrella Activity" like Software Configuration Management or an in-stream activity like Project Initiation or an engineering activity like Design and Development, will have relevant Metrics that need to be captured. While the specific things you measure may vary from one activity to another, there are some fundamental, unchanging principles that apply consistently to whatever measurements you take and whatever activity you undertake. The purpose of this modular is to cover such fundamental aspects of metrics and measurements.

Every project has certain set of ***objectives*** it has to meet. There are ***short term*** objectives, viz., the specific deliverables for the particular project and there are ***long term*** objectives, i.e., how is this project tied to the company vision or how does this project further learning in the organization? Any Metrics Program (or measurements and



analysis activity) should provide gauges that measure our progress towards both short as well as long term objectives.

A typical Metrics Program revolves around asking the following questions and obtaining answers to these questions on a continual basis.

1. What is your goal or where do you want to go?
2. What is your current position?
3. Knowing where you are and where you want to go, what steps should you take?
4. How do you measure your progress?
5. What corrective actions do you take when you see that your progress deviates from the expected progress?

A successful Metrics Strategy typically entails:

1. Deciding what you want to measure and how you are going to measure them: Entities that you choose to measure should be reasonable measures of progress towards objectives and easy to measure. People should be able to

make appropriate inferences and decisions based on the data.

2. Setting targets and tracking them: Once you identify what needs to be measured, explicit targets have to be set for the entities being measured. Effective targets are SMART–***S***pecific, ***M***easurable, ***A***ggressive yet ***A***chievable, ***R***esults-oriented and ***T***ime-bound.
3. Understanding and minimizing variability: Consistency and predictability are often associated with quality. Variability, especially uncontrolled variability, is detrimental is a project's success. It is necessary to spot, understand the root causes and control the variability.
4. Acting on data to achieve continuous improvement: All the data collection and analysis may reveal valuable information but the eventual benefits of measurements can be achieved only when the organization acts on the data and analysis.

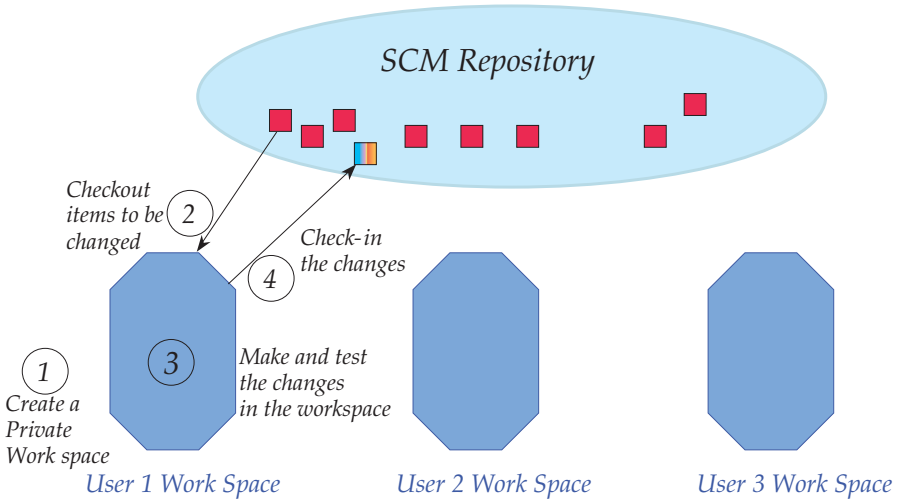
Measurements tend to make people nervous, as they minimize any element of subjectivity. Hence, Metrics programs must be carried out by considering the human angle with some sensitivity. The success of Metric Programs rests on strong management commitment, objectivity and use of the measured data only for the stated purposes.

Software Configuration Management

"Change is the only constant"

–Anonymous

One of the major determinants of the success of an organization is its ability to handle change in a structured manner and reflect the changes consistently in all the work products. A software product organization must also be able to manage and build the different versions of products that co-exist and be able to move through the various states like Alpha, Beta, Production, etc. The combination of change control, version control and movement control together constitute Configuration Management. Configuration Management is the combination of software, services and processes that enable each developer to re-create and use the exact set of files and environment



for a specific software product/version/platform. It is also a mechanism to isolate the work environments of the individual developers and ensure that none of the valid changes get lost.

A ***Configuration*** is a set of related items (also known as ***Configurable Items***) The set satisfies the following criteria:

(a) The configuration is uniquely identifiable (by a ***Configuration Id***)

(b) The items are consistent, i.e., the items work with one another in a way that is well understood

(c) The set of items is re-creatable as a unit

A ***Configurable Item*** or ***Configuration Item*** is an elementary part (usually a file) of the Configuration that must be

(a) Identified or versioned: Each item has a version number that characterizes its change history.

(b) Tracked: Any activity on the item is tracked

(c) Controlled: Any update to the item goes through a well-documented and controlled process.

The individual steps in Software Configuration Management are:

a) Initial working

b) Baselining

c) Change Management

d) Management of Work Spaces

e) Configuration Status Accounting

f) Configuration Audit

During initial working, developers work on their allocated components in an isolated way and the sharing of files is very informal. When the dependencies among the components increase,

the sharing is formalized. At this point of time, the configuration is ***baselined*** in a ***Configuration Management Repository***. From there on, the changes to the configuration (items) happen in a structured manner. Whenever someone wants to change an item, he or she would have to first get prior approval from a ***Software Change Control Board***, create a private workspace and check out the desired files, make changes to the files in the workspace, and re-check in the files back into the repository with a new version number. From then on, only the new version of the file would become available to anyone who requests it. In order to ensure appropriate functioning of the Configuration Management functions, periodic Configuration Audits and Status Accounting are performed.

SCM is a function which can benefit immensely from automation. There are a number of tools in the market that provide SCM Repository and appropriate mechanisms to perform the SCM operations. Most of these tools also support geographic distribution of teams.

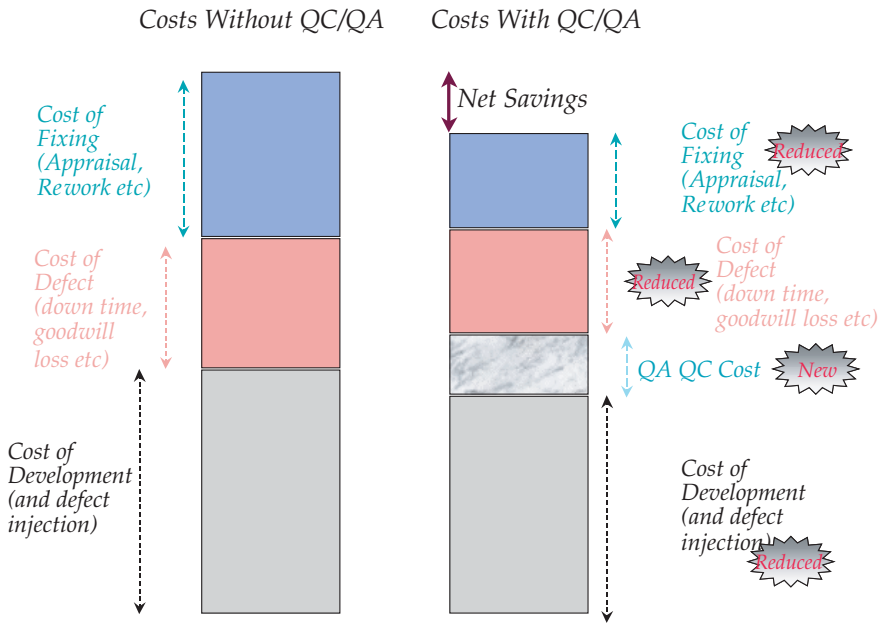
Software Quality Assurance

"Quality is not only right, it is free and it is not only free, it is the most profitable product line we have"

–Harold S. Geneer

Quality, from a customer's viewpoint, is, "how well does the product meet *my* needs?" From a producer's perspective, quality refers to, "how well does the product meet what *I* said it would do?" In an ideal scenario, when the producer of a product has fully understood all of the customer's needs and has faithfully translated these requirements into the product, it would meet all the needs of the customer and also perform "as per the specifications". Unfortunately, life is not that simple! Every customer has implied needs that are never stated, but a product is expected to satisfy these unstated or implied needs. Furthermore, when building a general purpose software product, the product cannot always meet all the needs of all the customers. Taking all this into account, ***quality***–also called ***quality of conformance***–is about transforming as much of the implied requirements of the customer(s) into stated requirements, meeting all the stated requirements, minimizing the unfilled requirements and meeting the needs in a consistent and repeatable fashion. A gap between the requirements and the actual product functionality is a defect and the aim of a project is to reduce such defects.

There are two approaches to minimize defects–these are ***Quality Control*** (***QC***) and ***Quality Assurance*** (***QA***). Quality Control refers to testing a product after a given phase to find defects and after finding the defects taking whatever corrective



action needed for that instance of the defect. Thus, QC focuses on the product after it is built and it is usually reactive, tends to be expensive and may not always be an option (e.g., in life saving devices). Testing is one example of a Quality Control activity. QA on the other hand focuses on the process of how the product is built and tries to prevent defects from creeping in to begin with. Because QA focuses on the process, it is proactive and usually its effects are more significant. Reviews, inspections, audits, and institutionalized processes are examples of Quality Assurance.

QA and QC can bring in significant cost savings if made properly. When no QA/QC is done, the cost of initial development incurred by the product development organization includes the cost of defect injection. The down time cost and the loss of good will from the customer constitute part of the Cost of Defect. The effort incurred to detect the cause of the defect and fix it results in appraisal and re-work costs. Thus the total costs are the sum of the initial development costs, defect cost, appraisal cost, and the rework cost.

When an organization puts in QA or QC, an additional cost component for testing, audit, etc., is added. But because of proper processes (that are part of QA), the development cost would come down. Over time, the number of defects would also come down because of the use of standard processes and components. Because of testing (that is part of QC) lesser (and less damaging) defects would pass through to the customers. Thus, the cost of defects would come down. As there are less defects to fix and the organization has well defined processes to deal with and fix the defects, it is likely that the re-work cost comes down as well. Thus the *overall* cost would be reduced.

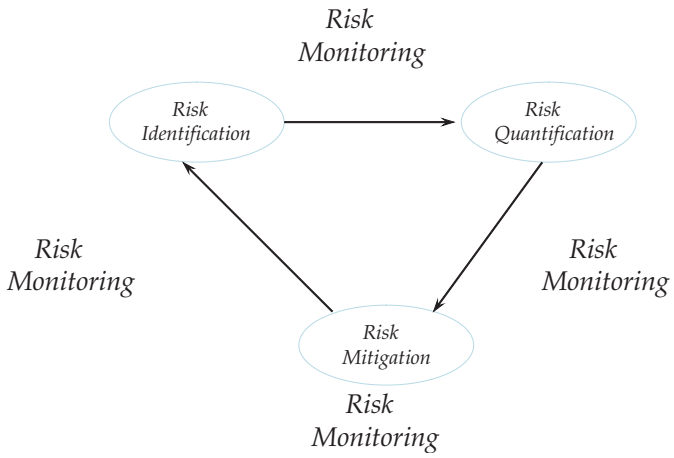
The "Software Quality Analyst" (SQA) role can be viewed as that of "conscience keeper" of a project. The SQA is someone with a charter and mandate to alert the top management of any possible quality or consistency slippage. The main functions of the SQA can be classified into the following four major areas: requirement fidelity, process compliance, change control, minimizing gap between defect injection and detection, and product quality. Some of the tools that are useful for SQA functions are Audits, Inspections, Pareto Analysis and Fish Bone Charts. There are various organization structures possible for SQA and these can be chosen depending on the size of the organization and the nature of projects.

Risk Analysis

"A ship is safest when anchored in a harbour but that is not what a ship was designed for!"

–Old Proverb

Risks are uncertain events beyond a planner's control that are worth anticipating and having an alternative plan for, because if not reacted to can derail a project's desired outcome. Even though risks are about uncertainties, they are as certain as death and taxes! Every software project can potentially face a mine field of risks from various sources. ***Risk Management*** is the process of anticipating what could go different from the plan and providing alternate plans so as to have minimum impact on the originally anticipated final outcome. Risk Management is thus a proactive planning process that strives to reduce the impact of uncertainty as well as the "in-line" pressure if one were to encounter risks in an unplanned or unanticipated manner.



Since risks can hit a project any time, Risk Management is another umbrella activity that happens throughout a project. Risk Management is comprised of three phases–***Risk Identification***, ***Risk Quantification*** and ***Risk Mitigation***. Throughout these three phases, ***Risk Monitoring*** takes place continually in the background.

Risk identification is the process of identifying those risks that a Project Manager should watch out for and protect against. Organizational history is a good place to start identifying the possible risks that a project may face. While past performance is not a guarantee of future results, completely ignoring past history is not prudent either. In addition to organizational history, a project manager may be able to use industry data and simulation techniques to identify risks. Encapsulating the past experience in the form of a checklist is extremely valuable to make the Risk Management more directly relevant to the projects of an organization. Another effective way to encapsulate the knowledge of risks encountered in past projects is to represent them in a framework based categorization and use this framework for all future identifications. All these encapsulations should be diligently updated and made current with new information for them to be useful. This is an area that is normally cause for slippage in the effectiveness of Risk Management.

As we discussed in the Metrics module, we should try to quantify whatever we do; risks are no exception to this rule. Quantifying risks allows us to identify those risks that need special focus and prioritize attention to the various risks. Risk has two dimensions–probability of the risk happening and the impact if the risk does materialize. ***Risk Exposure***, defined as the product of the probability and impact, provides a quantitative measure of the risk. In practical situations, when neither the probability nor the impact is known quantitatively, a common practice is to classify the probability and impact as “High”, “Medium” or “Low” and put the focus on those risks that have a medium-to-high probability and a medium-to-high impact.

Risk Mitigation refers to what steps you can take to minimize the Risk Exposure. Risk Mitigation is usually comprised of identifying (preferably multiple) alternative strategies in the event of a risk occurring and quantifying the costs of these alternative strategies. Risk Mitigation can also be called the "back-up plan" to combat risk. The most common types of risks have mitigation plans that seem to work most of the time. You can take the framework based risk categorization and apply mitigation strategies for each category in the framework.

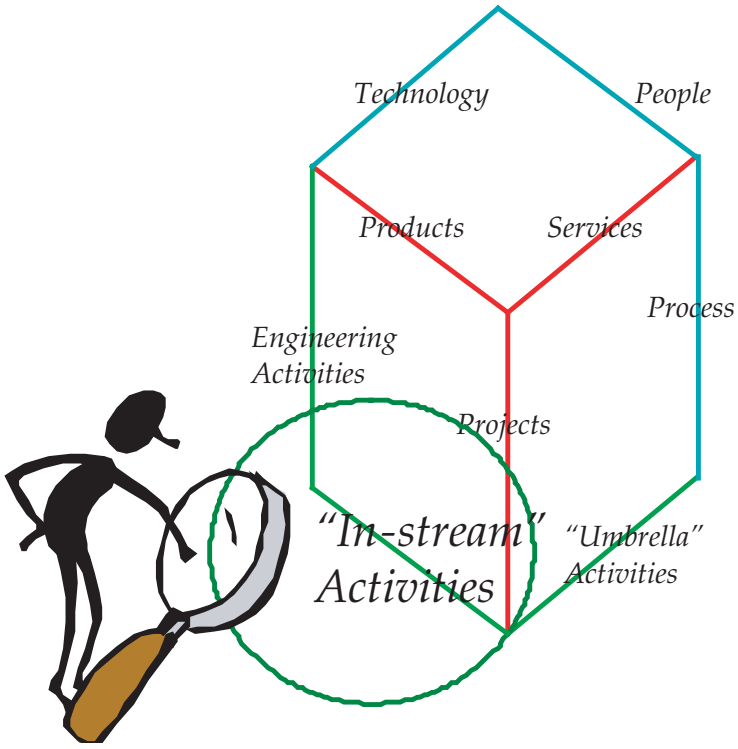
Throughout the Risk Management Cycle, you have to keep monitoring the risks. ***Monitoring*** involves putting in place some indicators or symptoms that could portend the actual of the risk, before it is occurrence it is actually incurred. This would also include identifying the alternative mitigation strategies to use for each of the symptoms and risks. Not all risks have readily detectable tell-tale symptoms, but in most cases, you would be able to arrive at such indicators, by using past experience.

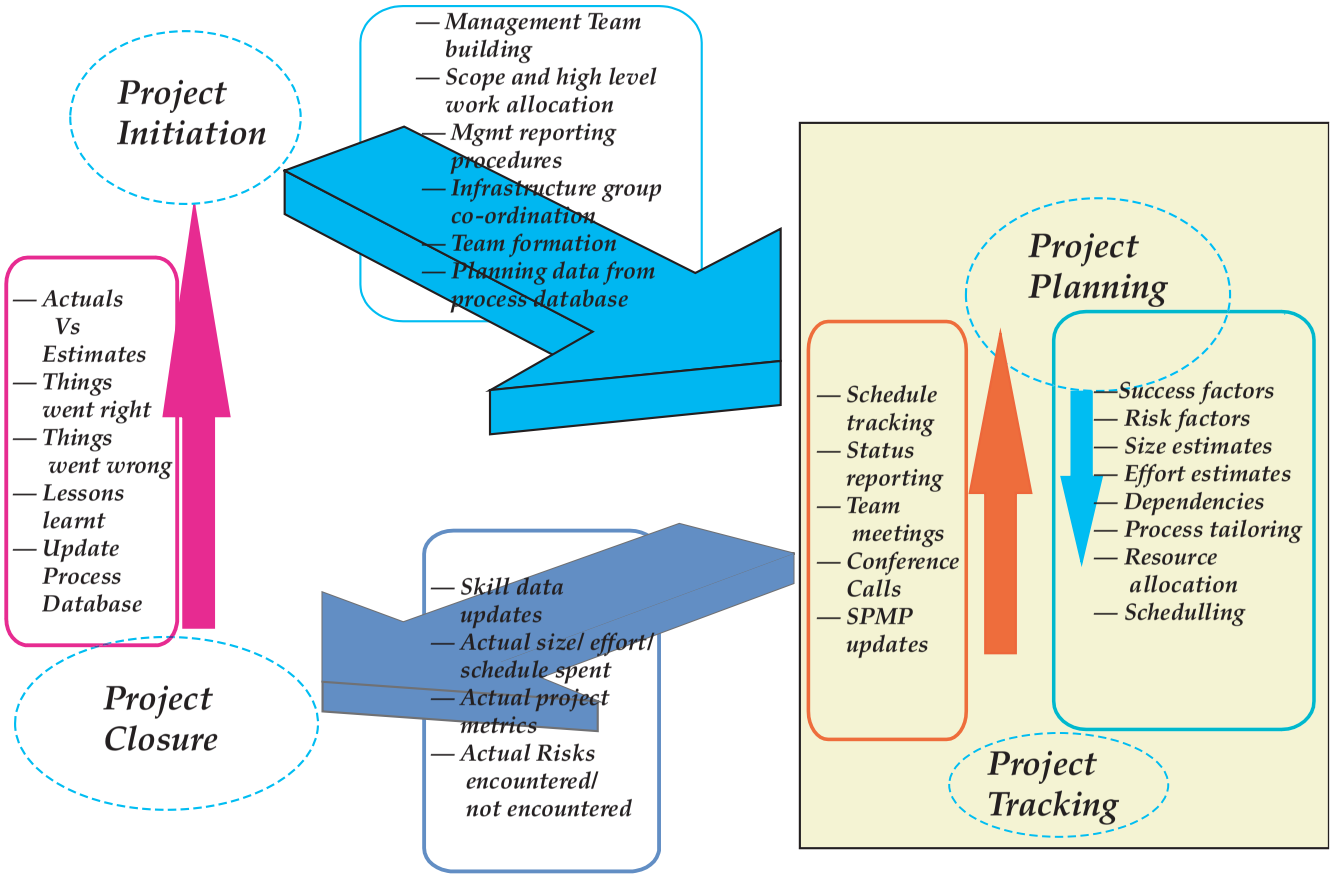
Global teams present special risks that are not present in single site teams. Some of these are procedural and cultural differences amongst the various locations.

Part III:

In-Stream Activities

This part deals with the In-stream Activities in a project. Unlike Umbrella Activities that take place throughout a project, In-stream Activities occur only at specific points of time during the project. The in-stream activities are Project Initiation, Project Planning, Project Tracking and Project Closure.




Project Initiation
— Management Team building
— Scope and high level work allocation
— Mgmt reporting procedures
— Infrastructure group co-ordination
— Team formation
— Planning data from process database
— Actuals Vs Estimates
— Things went right
— Things went wrong
— Lessons learnt
— Update Process Database
Project Closure
— Skill data updates
— Actual size/ effort/ schedule spent
— Actual project metrics
— Actual Risks encountered/ not encountered
Project Planning
— Success factors
— Risk factors
— Size estimates
— Effort estimates
— Dependencies
— Process tailoring
— Resource allocation
— Schedulling
— Schedule tracking
— Status reporting
— Team meetings
— Conference Calls
— SPMP updates
Project Tracking

Project Initiation

"Begin with the end in mind"

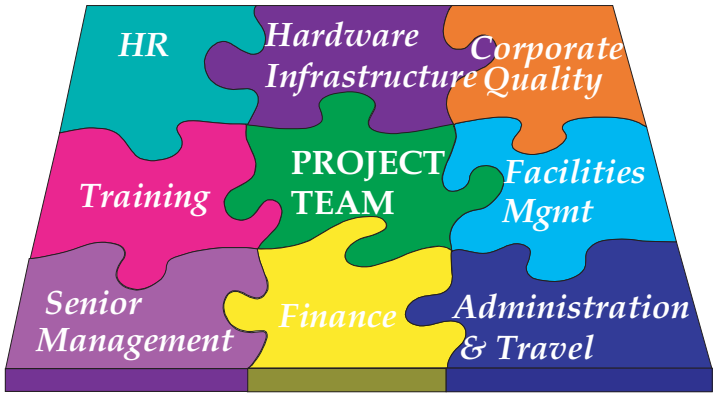
–Stephen Covey

Project initiation is an activity that marks the formal commencement of a project. A project is deemed commenced when the senior management of the organization has committed to undertaking the project and has identified a Project Manager, chartered to realize the high level commitments made by the senior management. In addition, the project manager is also given a set of constraints that govern the project.

Project Initiation in a global team entails the following steps:

1. Management Team Building
2. Scope and high level work division agreements
3. Decision of Management Reporting/Escalation Procedures
4. Involvement of infrastructure/support groups
5. Team formation

Management Team Building enables the project management team that spans the different locations to start on the right note in order to build a cohesive team. The success of the entire team usually has a strong correlation to the cohesiveness of the management team. By identifying local points of contact within each location and facilitating a (face-to-face or virtual) meeting of these "local managers" it results in good upfront team building and



reduces the cultural and language barriers likely to surface later in multi-location teams.

During Project Initiation, the management team arives at the scope and initial (high level) allocation of work among the sites. The goals of this initial scope allocation includes achieving a consistent understanding of the roles of the various teams, identifying any obvious bottlenecks or risks and alerting senior management to such bottleneck so that they might arrive at appropriate contingency or mitigation plans. The management team also works out the reporting and escalation procedures that are to be followed during the project.

A project group does not operate in isolation. For success, it requires the active participation, support and co-operation of other ***Infrastructure Groups***. Some of the infrastructure groups that are important are the Human Resources Group, Hardware Infrastructure Group, Finance and Administration Group, etc. During Project Initiation, each project manager (or local manager as appropriate) instructs each infrastructure group about the requirements needed from them. Each infrastructure group aggregates the requirements from the multiple project groups and optimizes its internal execution (interfacing other infrastructure groups) so as to satisfy the requirements of all the project groups in a cost-effective manner. It is important that the project managers

keep the infrastructure groups appraised of any change in requirements expected from these groups (e.g., change in hardware requirements, personnel requirements, travel, etc.) in a timely manner.

Project Initiation culminates in a ***Project Kick-off*** meeting, wherein all the constituents in the project–the project teams as well as the infrastructure teams (and customers and subcontractors, if applicable)–come together in a (virtual) meeting and get a consistent and thorough understanding of what is expected of each of them and sign off on their commitment to the project's success. With the kick-off meeting in place, the project proceeds to the Planning phase. In parallel, team formation begins. Given the attrition and flux of people working on projects, team building is often a continuous, ongoing process.

Project Planning

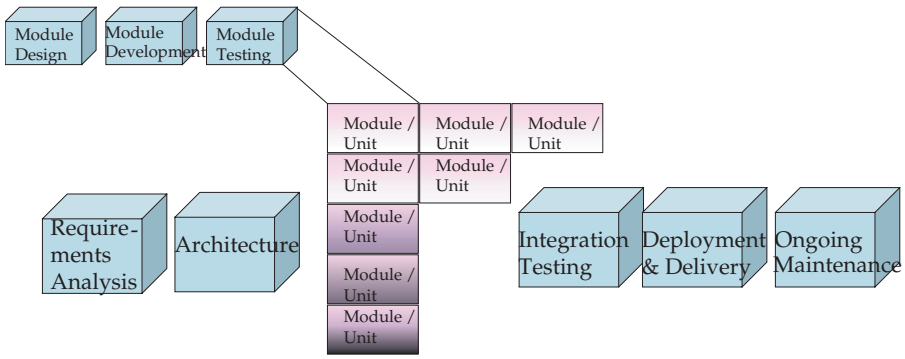
"Failing to plan is planning to fail"

–Anonymous

Project Planning and ***Tracking*** go hand in hand. After Project Initiation, the Project Manager comes up with an initial ***Project Plan***, against which the project is tracked during its execution. As the project evolves and new information and changes unfold, the plan is kept current, reflecting the changed scenario. This process of planning, tracking and replanning is an ongoing one, till the project is completed.

The Project Plan takes into account five inter-related parts– WHAT, WHAT COST, HOW, WHEN and BY WHOM. The WHAT part relates to contents–product features, platforms, services to be offered, etc. The WHAT COST part prioritizes the contents so as to derive the maximum bang for the buck. The WHEN part translates the prioritized contents into schedules. The HOW part tells what processes are to be followed in the project, and how the organizational processes can be tailored to meet the specific needs of the project. Finally, the BY WHOM part of the project takes care of assigning the right people for the right tasks.

The key method to arrive at the WHAT part is ***Work Breakdown Structure*** (WBS). Work Breakdown Structure is the decomposing (dividing) the project into smaller and more manageable parts (called Work Breakdown Structure Units or WBS Units), with each part satisfying the following five criteria:



- Each WBS Unit has a clear *outcome.*
- The outcome has a direct relationship to achieving the *overall project goal*
- Each WBS Unit has a *single point of accountability.*
- Each WBS Unit is something that can be *tracked* and *monitored* as an unit.
- Each WBS Unit has a well defined *interface* to other WBS Units.

The WBS Units can be evolved from multiple dimensions. First by the life cycle phase (e.g., Requirements, Architecture, etc.) and then within each phase, by modules and further, even by activity eg., design, development, testing etc. Such a WBS decomposition enables identifying logical components that can go on with reasonable parallelism and autonomy which can to be synched up periodically on specific milestones.

The WHAT COST part prioritizes the features into multiple levels (e.g., Essential, Nice to have and Frills). This enables choosing the right features to be scheduled, given the constraints of resources.

The HOW part of the plan addresses the processes to be followed for executing the project. These processes are derived from

organizational processes through appropriate tailoring. During this definition of processes, a Project Manager has to try to re-use existing processes as much as possible without incurring too many overheads.

The WHEN part of the project plan maps the prioritized WBS Units to timelines. This consists of identifying external dependencies (e.g., hiring, training, etc.), internal milestones and "sync points" between the two.

Finally, the BY WHOM part of the plan assigns responsibilities for the tasks to individual people. During this stage, the Project Manager should take care to achieve clarity of roles and responsibilities and use people with the right levels of experience leading to fair and effective distribution of work.

All these parts are put together in the form of a Project Plan called ***Software Project Management Plan*** (SPMP) that acts as a road map for the project. It is against this roadmap that the project is tracked.

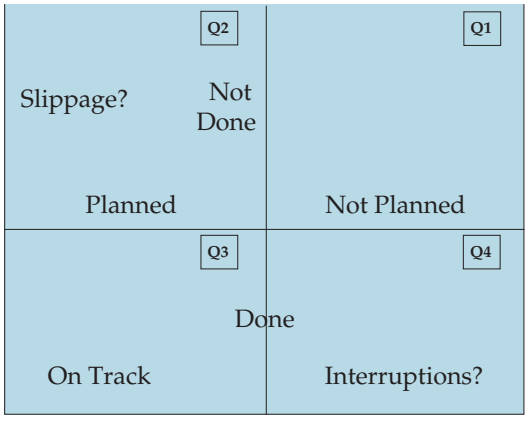
Project Tracking

"Even if you are on the right track, you will get run over if you just sit there"

–Will Rogers

Project Tracking is the ongoing activity that monitors the progress of the project against the plan to ensure things are going according to plan, and, if not, to perform any course corrections required in the execution of the project or change the plan as required. This is in line with Deming's Plan-Do-Check-Act (PDCA) principle. The major means used for tracking are status reports, one-on-one meetings, group meetings, and other forms of communication. The common factor between all these is, once a plan is put together, ensure effective communication thoughout the project, replan when necessary and continue the process.

The ongoing progress of the project is tracked by various means, the major ones being status reports and regular communication,



both keeping the plan current. A status report is given by an individual to his or her direct manager on a periodic basis (daily/weekly /fortnightly etc.) and tracks the progress against what was planned for that period. Viewing "Planned" and "Done" as two axes, we would get four quadrants. The activities in each of these quadrants have to be carefully analyzed by the Project Manager in order to take corrective action to keep the project on track. The status reports should come at reasonable intervals that don't add to the overheads but provide information in time to take corrective action. Since a status report is, in general, likely to be read and used by the higher levels in the management hierarchy, the information in the report should have some degree of abstraction and aggregation. The status reports, especially to senior levels of management, should be clear, concise and unambiguous in order to spell out the progress of the project as well as specify what action or support is needed from them.

In addition to the operational status tracking, a Project Manager should also perform a periodic aggregate analysis of the status reports to spot any trends. Some of the patterns or trends that a Project Manager should look for include:

- Whether there are a fair ***number*** of activities that are ***consistently*** in the "planned but not done" quadrant, indicating poor planning
- Whether there are some ***specific activities*** that consistently get into the "planned but not done" quadrant, indicating poor prioritizing
- Whether the "planned and not done" activities get localized to any particular individual(s), indicating improper training or inadequate skills
- Whether there are any Red Alerts that appear ***repeatedly*** and ***continuously*** in several status reports, indicating a probable red herring or serious omission on the part of the recipient of the status reports

- Whether there are any Red Alerts that crop up ***intermittently*** in the status reports of various periods, indicating "quick fix" solutions that are not well thought through

Project Closure

"The wisest mind has yet something to learn"

–George Santayana

One of the fundamental requirements of software development is continuous improvement. A major source of knowledge for achieving continuous improvement is to learn from our past experiences. ***Project Closure*** is a means to avail of this learning process. Closure refers to the conclusion of a project or a logical part of the project. Typically, software product development happens in versions or releases. A release of a given version can be taken as closure of the development project for that version and as the beginning of its maintenance phase. When a version is in the maintenance phase, problems/bugs are reported and fixed. There could be "intermediate closures" after every release or phase and a "final" closure when the product becomes obsolete.

An effective closure process should draw the collective learning from all the stakeholders in a project. These include the actual

project teams in the different locations and the infrastructure providers (e.g., Training, Travel, Finance, etc.) and, where appropriate, representatives from the customer organizations. When a team is geographically distributed, it would be ideal to have video conferences where the teams in various locations can see each other. If that is not possible, a simple telephone conference is a must. The objective of having everyone in one (virtual) room is to discuss all issues that pertain to all the groups. This will minimize the chances of finger pointing at the groups not represented in the room.

The closure process requires planning and culminates in a ***closure meeting***. In order for the closure process to be effective–

(a) There should be a focus towards *learning* in the meeting; and not towards accusing individuals.

(b) Each member should come prepared.

(c) Each member should have an opportunity to voice his or her views, to avoid the "Big Mouth Syndrome" of one or two individuals hogging all the air time.

In order to ensure these objectives, the Project Manager calls for the closure meeting and allocates specific agenda items to specific individuals by a combination of ***vertical*** or ***horizontal distribution methods***. A vertical distribution method asks certain individuals to go in depth into areas where they have specialised expertise (e.g. SQA). A horizontal distribution allocates the same agenda item to multiple participants to get multiple perspectives (e.g., coding standards). This provides more depth and breadth to the issues to be covered. The Project Manager also mediates to keep the meeting on course by requesting each member to highlight three (or a certain number) of things that went right in the project and three things that need improvement.

Some of the issues that should be addressed at such a closure meeting include goal realization success, effectiveness of metrics

chosen, any corrections to estimation methods and data, new experiences and experiments, process changes, environmental changes, technology changes, risk updates, and customer feedback.

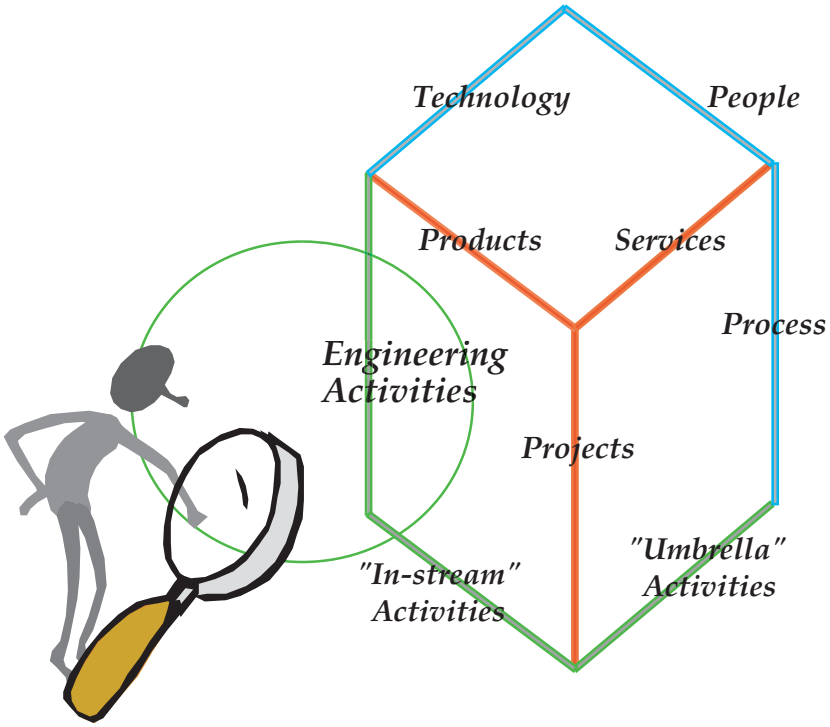
The Project Manager should analyze the results the closure meetings and see what mid-course corrections can be made to the project execution or what can be carried forward to the next project(s). The Senior Management of the organization should use the information from all the individual closure meetings, aggregate them, identify any possible patterns that are systemic to the entire organization and make any organization-wide changes that may be necessary. In either case, it is important to communicate to the people who put forth suggestions the closure meetings at results of their feedback and the results ensued from the changes. This closing of the communication loop will ensure that the inputs in future closure meeting will be meaningful.

Project Closure completes the Project Management Cycle that constitutes the in-stream activities. These in-stream activities are woven into the engineering activities that form the topic of discussion in the next part of the course.

Part IV:

Engineering Activities

This Part deals with the primary engineering activities in a project. This includes Requirements Gathering, Estimation, Design and Development, Testing, and Maintenance. These activities interlaced with the in-stream activities. The umbrella activities like Metrics, Configuration Management, SQA and Risk Analysis apply to these engineering activities as well.



Requirements Gathering

"It is more important to do the right things than to do things right"

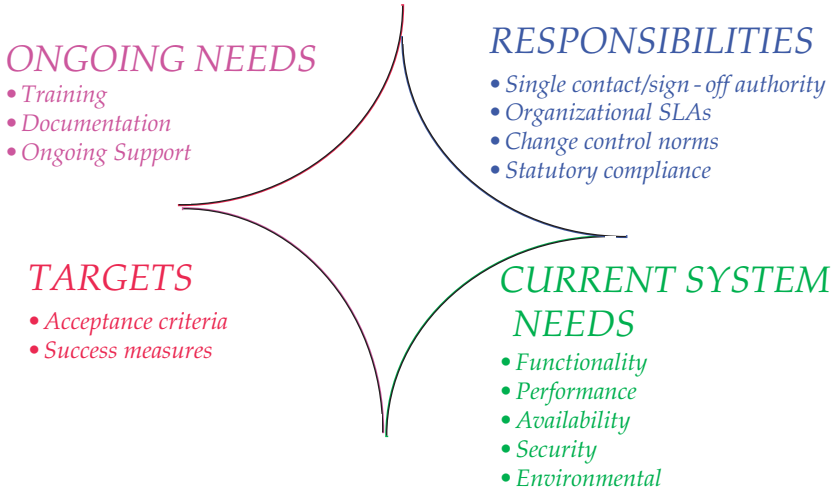
–Stephen Covey

Most projects start with a system requirements study. This requirements of the entire system comprising hardware, software, manual processes and the rest of the environment. At the time of obtaining this holistic system understanding, an initial scope in drawn out. This intial scope drives the software requirements gathering phase, wherein project information is gathered to the next level of detail. In the case of a custom application for a specific customer, this next level of detail is gathered by discussions with appropriate representation of the customer organization. For a general purpose product, the Product Marketing Group (or specifically, an assigned Product Manager) plays the role of proxy customer.

During the Requirements Gathering phase, the software development organisation understands *what* is needed from the product to satisfy the needs of the customer, without necessarily going into the details of *how* the needs are to be satisfied.

Requirements Gathering is a holistic function intended to maximize customer satisfaction throughout the project. Viewed in this broader context, it can be made up of four dimensions:

- Responsibilities
- Current system needs



- Targets
- Ongoing needs

Right requirements are those that are specified by the right people. A customer organization designates a Single Point Of Contact (SPOC) who presents the big picture of what is required from the system, nominates other authorized contacts for specific modules or components, and acts as an arbiter in the event of conflict among the multiple components. In the case of a general purpose product, a Product Manager usually plays this role. Similarly, the Project Manager from the software producer acts as a SPOC to the vendor side. The SPOCs work out Service Level Agreements (SLAs) which act as guidelines to set expectations of the project which include any statutory requirements. Since changes in requirements are to be expected, the norms for Change Control are first agreed upon.

The dimension of the current system needs is the focus for most technology driven aspects of the project. These include functionality, performance, availability, security and other environmental constraints. *Functionality* addresses the modules of the system and the functionality to be accomplished by the modules along with their inputs, processing and outputs. *Performance* relates to the maximum

load, response times, and other traffic-related aspects of the system. *Availability* spells out the requirements of up-time which in turn places additional demands on the hardware and software environments. *Security* addresses issues about authentication levels, access rights, etc. *Environmental requirements* list out the hardware and supporting software systems required to run the software being developed.

The Targets Dimension specifies the success criteria for the project and addresses issues like non-negotiable deadlines, headcount restriction, budget limits, etc. Acceptance criteria specify the tests which the customer will specify to objectively evaluate whether the system meets their requirements.

Ongoing Needs Dimension refers to the ongoing operational support requirements required from the vendor. These include documentation support, training support and ongoing product maintenance support. These add to the overall project costs but are easy to miss out as they may not come under the normal technical activities of a software project.

Requirements are gathered by various means that include interviews, observing work patterns, and prototyping. The key to successful requirements gathering is a process of constant feedback and communication between the customer and the vendor teams to identify and resolve issues upfront. Given that requirements flux is a major cause for cost and time overrun, such proactive communication will help facilitate downstream activities of design, development, testing and maintenance.

Estimation

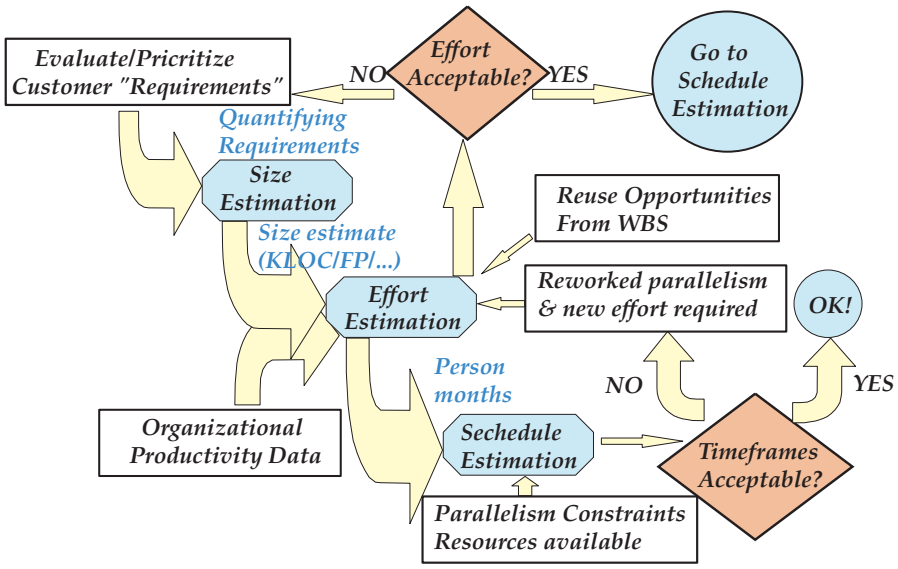
"Forecasting is difficult, especially about the future"

–Victor Borge

The requirements gathering phase entails the understanding of the customer or product requirements. This has to be translated to design and development to realize the product intended to satisfy the requirements. In order to perform this translation, ***estimations*** of size, effort and timeframes required need to be made. Given that each project has its unique aspects, estimation is always done with incomplete information and is based on certain clearly stated assumptions. The objective of performing estimation is to quantify the demands placed on certain key resources (e.g., machine resources, people resources, timeframe, etc.)

Estimation is usually done in three stages–size estimation, effort estimation and schedule estimation. Hardware and software resources required are also estimated. Together this leads to a cost estimate, which is what the customer and a vendor would eventually be interested in.

Size estimates refer to the actual ground that needs to be covered to satisfy the customers' requirements. These can be expressed as lines of code, function points, feature points, etc. An effective size estimate should have a good correlation to the effort needed and be easily measurable. The primary driver for size estimate is the application complexity.



Effort estimates refer to the effort required to cover the ground measured by the size estimate. This is expressed in person months, person years or similar units. The primary drivers for arriving at the effort estimate from the size estimate are (a) the organizational productivity data for specific functions and (b) opportunities for re-use identified by effective Work Breakdown Structure decomposition. Effort estimate usually has direct relevance to cost of the project as it translates to person-month costs.

Schedule estimate translates the effort into timeframes. The primary inputs for performing this translation are availability of resources and any constraints in parallelism. Hard deadlines and internal and external dependencies also play a major role in arriving at schedules. Schedule estimates mark the calendar dates by which specific parts of the project will get ready. These are tied to milestones and dependencies.

The key to deriving useful estimates lies in the validity of the underlying data being used for the estimation purposes (e.g., organizational productivity data) and on breaking down a project into smaller pieces such that we can identify each small piece as

something similar to what we already have experience in. The first aspect (of valid data) has a close bearing to abstracting appropriate information from project closure. The second aspect (proper decomposition) is related to proper Work Breakdown Structure (WBS) during project planning. Thus, estimation has a very strong dependency on the in-stream activities.

This three-step process may not apply effectively in all the cases. For example, for maintenance related functions, there is usually very little correlation between the size and effort estimate. For such cases, specialized estimation models would have to be developed which tend to be organization-specific.

Regardless of the estimation model and what is being estimated, it is important to clearly document the assumptions made while performing the estimates and periodically validating these assumptions as we gain new information. The process or project database discussed earlier is absolutely crucial in this endeavor.

Design and Development

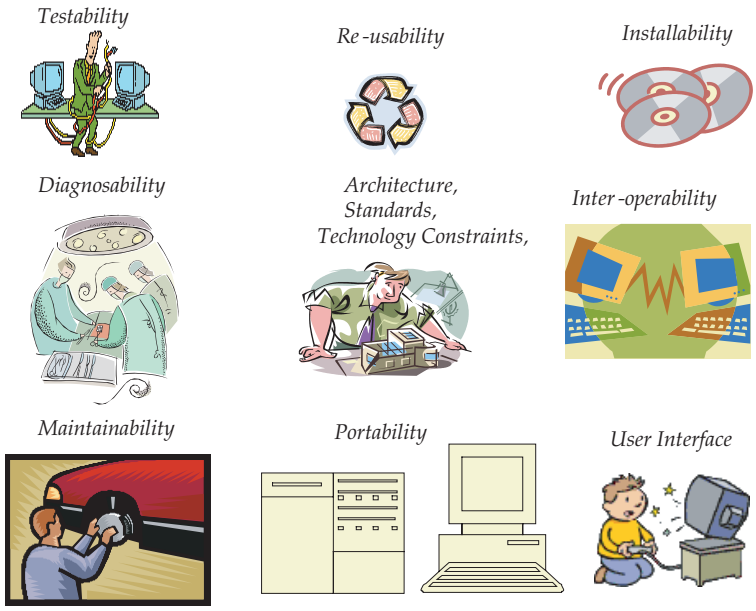
"All things are created twice; there is a mental or first creation, and a physical or second creation to all things"

–Stephen Covey

The requirements gathering phase specifies what is required from a software system or product. ***Design*** is the step which translates the "what" to "how", wherein a conceptualization of how to realize the requirements of the user is first made. It is the first visualization of the product. ***Development*** is the actual realization (by use of software development tools) of the design. In a traditional software engineering life cycle paradigm, design and development are two distinct phases. From a project management perspective, there is a lot of overlap between these two phases, because the same people usually get involved in both and there is a lot of iteration and interaction during the two phases.

There are several aspects of Design and Development that are important to make the overall management of the entire project life cycle more effective.

Architecture of a system identifies the various units of the system. These units can be viewed as building blocks for constructing the system. The architectural design step takes the view of the implementers and asks the question, "what parts do I need to realize the requirements in the most optimal way"? Architecture provides an opportunity for early identification of dependencies.



Standards are an essential part of design that enhance market acceptability and reduce the overall costs of the system. External standards characterize external product behavior. It could be compliance to a published language syntax and semantics (e.g., SQL), compatibility to a set of call level interfaces (e.g., JDBC), adherence to a common protocol (e.g., WAP) or reflection of a standard user interface (e.g., Motif). In all these cases, some of the common attributes of external standards are that they are not proprietary, are published by an authorized consortium and compliance is well defined. Internal standards are standards like coding standards, documentation standards, etc that enable consistency among the work of the different team members.

Portability refers to the requirement that the same software runs identically on several platforms. Addressing portability early by clear isolation of generic and platform-specific functions and by proper coding standards can enable easy deployment and maintenance on multiple platforms. Portability may be one of several technology choices or constraints faced. ***Reusability*** transcends portability and focuses on how to reuse existing components so as to

avoid duplication. The higher the reusability, the lower the maintenance costs of the system.

Testability, ***Diagnosability*** and ***Maintainability*** are three interrelated design aspects that dictate the easy maintenance of a software product. Testability refers to how easy it is to test a given design by well defined test cases. Diagnosability refers to how easy it is to trace to the root cause of a problem from the problem's symptoms. Maintainability refers to how easy it is to make changes to the software in order to either fix defects or introduce enhancements.

Installability refers to the ability to easily install the software in a customer's environment. Similarity to a standard installation on the target platform, intelligent assumption of defaults, and consistency of documentation are some of the issues that should be addressed by installability. Once software is installed, it is expected to inter-operate with other software or earlier versions of the same software. These aspects have to be planned upfront and these lead to ***Inter-operability*** issues. While designing all these, proper attention has to be to ***user interfaces***.

A good design that addresses the above criteria can greatly facilitate the development, testing and maintenance phases thereby reducing the overall project costs. The Project Manager should invest the required time in design and not get carried away into rushing into the development phase.

Testing

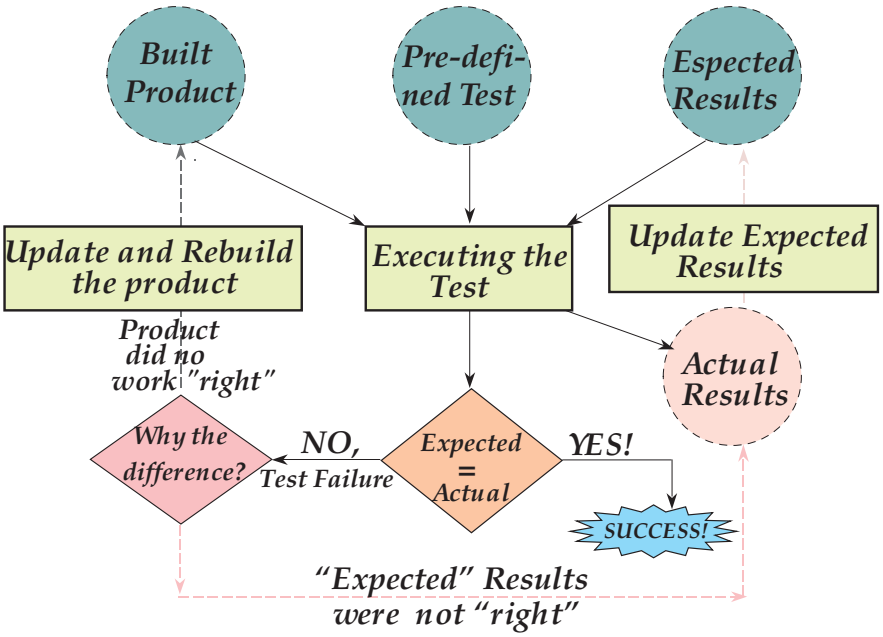
"The most important considerations in software testing are in the issues of economies and human psychology"

– Glenford J Myers

Testing refers to exercising a built software product with the aim of finding as many defects as possible and correcting them before shipping the product to the customers. Testing is thus a Quality Control activity.

Tests to be performed are divided into physical units called *test scripts* (also called *test cases*). Each test script exercises the built product in a *pre-defined manner* to produce an *actual result.* Each test script has also got a *pre-defined expected result* that specifies what the result of executing the test is if the product behaves according to its original intent/requirements. After execution of the test, the actual result is compared against the predefined expected result. If the expected and actual results match, the test is said to have ended in *success,* while if they don't match, the test is said to have ended in *failure.* When a test failure is encountered, the product and the expected results are examined to analyze why the failure has occured and make appropriate changes to the product, re-run the tests and ensure that it succeeds. It could also be that the expected results were arrived at wrongly, in which case they are changed and the test is deemed as successful. In addition, the expected results are updated to reflect the new changes.

Testing is made up of the several steps. During the *Test Specification* step, what needs to be tested is finalized and documented. In the *Test Design* step, the details of the layout, tooling



and standards required for test development are designed. *Test Development* actually builds up the test cases using the available tools. *Test Registration* enters the tests as configurable items in a SCM Repository. *Test Execution* refers to repeated execution of the tests as the product undergoes changes. Finally *Test Maintenance* refers to keeping the tests current with the changes in the product. As you can see, the life cycle of Testing appears like a microcosm of the entire software product development life cycle.

In order to achieve early detection and correction of defects, different types of tests are conducted at different points in time during a project. These different types of tests also require different skill sets. *White Box Testing* refers to testing the product, with full knowledge of the program code, translating the external usage scenarios to internal code paths. Thus, it is best performed by the programmers or developers. *Black Box Testing* refers to testing a given module by observing only the external behavior and not having access to the program source code. *Integration Testing* tests the product comprised of the integrated modules and makes sure the modules work together. *System Testing* extends Integration

Testing by exercising the product under different loads and different configurations to ensure the system does not fall apart. *Installation Testing* ensures that the product can be installed following the Installation document. As software undergoes constant changes, *Regression Testing* ensures that the new changes do not break the existing functionality. *Acceptance Testing* specifies the tests that must be run before the product is accepted by the customers. Such tests are usually specified by the customers.

Testing presents some of the most complex people challenges in terms of hiring, motivation and retention. It also presents some excellent opportunities for exploiting global talent by having a geographically distributed team that effectively performs development and testing functions. There are a number of ways of organizing global teams that address people issues and also get operational efficiency and effectiveness in achieving overall product delivery and maintenance. Each of the models have applicability in different circumstances and scenarios.

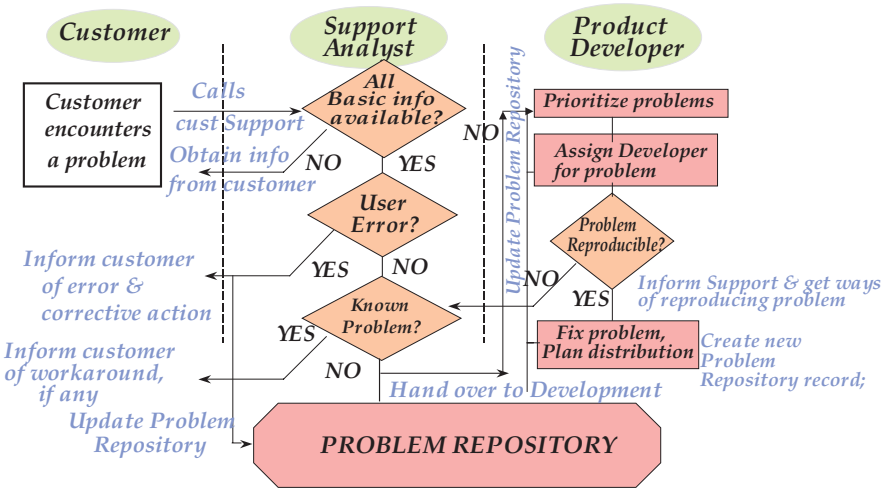
Maintenance

"The FAA required commercial carriers to keep extraordinarily detailed maintenance records .... All this paperwork meant that every one of the aircraft's million parts could be traced back to its origin

–Micheal Crichton in "Air Frame"

The ***Maintenance*** phase for a given version of the product starts after that version is released in the market. When customers start using the product, it is that behavior possible/its is different from what the customers expect. This difference can arise because of the customers' perceptions (or expectations), from actual defects in the product or in not meeting the stated requirements. When such discrepancies are observed, customers will ask for changes to the product so that the observed behavior matches the expected behavior. The maintenance phase deals with the process of evaluating the customer's requested change, ascertaining its applicability, making the changes as required, testing that the requested change is implemented and that no existing functionality re-gresses and finally delivering the change to the affected user(s). The changes made are to remove any defects in the product and make it fully comply with the requirements initially targeted. The Maintenance phase precedes the ***Obsolescence*** phase during which the product is completely "de-supported", i.e., no fixes are provided. Normally, after product obsolescence, customers migrate to the next version of the product.

The Maintenance phase is comprised of three major steps for each problem that a customer encounters–***problem reporting***, ***problem resolution*** and ***fix distribution***.



When a customer encounters a problem, he or she gathers all the necessary information (like the nature of the problem, the product/version that had the problem, the hardware and software environment, the actual activities being carried out in the system when the problem occurred, etc.). The customers interface is a group within the product organization called the ***Customer Support Group***. The ***Support Analysts*** in this group liaise with the customers and make sure that all the relevant information is available. If possible, they eliminate any chances of user errors and, where fixes are already available for a known problem, they appraise the customer. When they cannot resolve the problem by themselves, the problem gets passed on to the product development group.

The Product Development Group trace the root cause of the problem, make the appropriate changes to the program code, documentation, etc., so that the problem reported by the customers is fixed. While doing this, they interface with customers via the Customer Support Group. Such interfacing may be necessitated by requiring more information about the problem, environment, etc.

Eventually, the fix has to be distributed to the customer. Sometimes, the fix may have to be distributed proactively to other customers as well. The fix distribution mechanism can vary from cutting a medium (like a CD) and shipping it to customers or hosting the fix on a web site and notifying all interested customers so that they can download the fix.

In parallel with fixing individual defects, an organization also executes proactive defect prevention. By looking at all the problems reported and analyzing the root causes of the problems it, identifies process or product changes in order to prevent these errors from occurring in future.

Central to all the activities of maintenance is a ***Problem Repository***. It is a database that contains all the information about all the problems encountered or reported which include: the customer(s) who reported the problem, their platforms/environments, a detailed description of the problem, a test case to reproduce the problem (and test the fix), some details of what the root cause of the problem was, the fixes made etc. The repository is updated as new information becomes available.

From a project planning perspective, maintenance bears some differences to traditional development projects because it is tough to predict the incoming workload and there is no direct correlation between then size and effort estimates. The need to capture and analyze historical patterns of incoming problems becomes crucial in order to succeed in the maintenance phase. Thus, the preparatory work for the maintenance phase starts well before the product is released.