# Introduction

## Context

Software development lifecycle (SDLC) is the application of standard business practices to building software applications. It is typically divided into six to eight steps: Planning, Requirements, Design, Build, Document, Test, Deploy, and Maintain. Some project managers will combine, split, or omit steps, depending on the project’s scope. These are the core components recommended for all software development projects.

**SDLC**

SDLC is a way to measure and improve the development process. It allows a fine-grain analysis of each step of the process. This, in turn, helps organizations maximize efficiency at each stage. As computing power increases, it places a higher demand on software and developers. Organizations must [reduce costs](https://phoenixnap.com/blog/it-cost-reduction-strategy), deliver software faster, and meet or exceed their customers’ needs. SDLC helps achieve these goals by identifying inefficiencies and higher costs and fixing them to run smoothly. The requirements analysis phase is the most crucial stage, this forms the base for the project to be driven further. A set of requirements is used as inputs into the design stages of product development. Requirements are also an important input into the verification process since tests should trace back to specific requirements. Requirements show what elements and functions are necessary for the project. The software requirements are description of features and functionalities of the target system. Requirements convey the expectations of users from the software product. The requirements can be obvious or hidden, known or unknown, expected or unexpected from client’s point of view. The process to gather the software requirements from client, analyse and document them is known as requirement engineering. The goal of requirement engineering is to develop and maintain sophisticated and descriptive ‘System Requirements Specification’ document. The process of eliciting, analysing, verifying, documenting, and validating requirements is known as Requirement Engineering (RE). The main goal of requirement engineering is to meet the degree of end user’s satisfaction in minimum cost and time.

## Challenges

Requirement elicitation phase investigates the problems in existing system. However, errors in requirement phase are not identified during application development. Rather they remain concealed until system becomes fully operational and stakeholder’s needs are not met. The observation from various researchers illustrate that the cost of fixing an error initially in elicitation process is of little value as compare with other phases of software development. Thus, requirement elicitation plays an imperative role in application development. Requirement engineers face countless problems and difficulties to consult requirements from stakeholders. These problems are then compiled and accumulated into challenges. However, anticipating problems will therefore help requirement engineers to take actions beforehand and prevent software from misfortune.

Additionally, unstructured elicited requirements from operational domain are difficult to manage and model. Requirements need to be concise and well formatted based on any standard requirement specification template. This help stakeholders and maintenance team to understand requirements. Besides, it is a good practice to model requirements so that they can easily be validated by stakeholders. However, poor requirement specifications accelerate the level of ambiguity and requirements become difficult to quantify - resulting in failure of software application. Some other major challenges are:

* Conflicting requirements
* Customers do not know what they want
* Unavailability of stakeholders
* Changing Priorities
* Unsupportive stakeholders
* Cost Estimate and schedule
* Attention to non-functional requirements
* Prioritization of requirements
* Interconnection between requirements
* Validation of requirements

## Approach

The solutions to the above problems can be achieve by analysing the problems correctly, checking the completeness, correctness, and consistency of the requirements. The requirements should be such that it can be traced back – traceability of requirements should be checked. The further approach includes:

* Sufficient time is spent at the start of the project on understanding the objectives, deliverables, and scope of the project. This will help in understanding what the customers really want.
* Have a clearly defined process for receiving, analysing, and incorporating change requests, and make your customer aware of his/her entry point into this process. Set milestones for each development phase beyond which certain changes are not permissible. Ensure that change requests (and approvals) are clearly communicated to all stakeholders, together with their rationale, and that the master project plan is updated accordingly. This can be done by maintaining the change management document.
* Requirements engineers, stakeholders with whom they must collaborate, and requirements evaluators need to be properly trained in the characteristics of good requirements including examples of both good and bad requirements, and they need to be taught how to tell the difference between them.
* Ensure that requirements tracing is mandated in the contract and explicitly specified in the requirements engineering method. Also be sure to mandate and verify the tracing of all requirements, not just the functional requirements. Provide user friendly and scalable tool support for requirements tracing. Ensure management understands the negative consequences of not tracing requirements, and obtain support for proper tracing, including providing adequate resources to trace the requirements. Ensure that tracing occurs both early in the project development cycle as well as later during design, development, and maintenance.
* Ensure that requirements validation is a fundamental component of any requirements method, one that will not be dropped the first time that project resources become scarce. Ensure that requirements validation is included into the project’s schedule and budget as well as the schedules and budgets of the system’s stakeholders. Finally, remove all unnecessary obstacles separating the stakeholders and the requirements team.
* To deal with the large number of requirements and the constant changes to them, store the requirements in a database or the repository of a requirements tool. Store the requirements models and diagrams with or linked to the requirements all important attributes about a requirement with the requirement so that they are easy to manage and maintain. Do not scatter different kinds of requirements; instead, keep them all in the same repository. Ensure that the requirements repository supports access control, including prohibition of unauthorized access to sensitive requirements.

## Recommendations

The organizations should select people with the right combination of training, experience, motivation, mindset, and people skills to be good requirements engineers. Provide them with significant amounts of training, including classes, conference tutorials, books, and journals. Use a powerful, yet user-friendly requirements modelling tool to capture requirements diagrams and associated text. Ensure that these tools support the configuration management of the requirements and their models. The tools like Borland CaliberRM, IBM/Rational’s RequisitePro, or Telelogic’s DOORS can be used for requirement management.

# Non-Functional Requirements

## Context

In systems engineering and requirements engineering, a non-functional requirement (NFR) is a requirement that specifies criteria that can be used to judge the operation of a system, rather than specific behaviours. They are contrasted with functional requirements that define specific behaviour or functions. The plan for implementing functional requirements is detailed in the system design. The plan for implementing non-functional requirements is detailed in the system architecture, because they are usually architecturally significant requirements. Broadly, functional requirements define what a system is supposed to do and non-functional requirements define how a system is supposed to be. Non-functional Requirements (NFRs) define system attributes such as security, reliability, performance, maintainability, scalability, and usability. They serve as constraints or restrictions on the design of the system across the different backlogs. Non-functional requirements place restrictions on the product being developed, the development process, and specify external constraints that the product must meet. Examples of NFR include safety, security, usability, reliability and performance requirements.

Classification of NFRs:

* Product-oriented attributes
  + Performance: (a) response time, (b) throughput (number of operations performed per second)
  + Usability: effort required to learn, use, provide input and interpret results of a program
  + Efficiency: minimal use of resources (memory, processor, disk, network)
  + Reliability: of computations, precision
  + Adaptability: to other environments or problems
* Process-oriented attributes
  + Maintainability: changes to functionalities, repairs
  + Readability: of code, documents
  + Testability: ease of testing and error reporting
  + Understandability: of design, architecture, code
  + Integrability: ability to integrate components
  + Complexity: degree of dependency and interaction between components

**Maintainability** - Maintainability is how easy it is for a system to be supported, changed, enhanced, and restructured over time. Understanding up front how maintainable software needs to be for a certain project is important, due to its impact on your architecture. This impact makes maintainability an important non-functional requirement to consider when developing software. If an application has a relatively short lifespan, then the cost associated with implementing an easy-to-maintain architecture may not make sense. However, if the software will have a medium to long life, then we must start making serious considerations about how future upgrades and changes will be implemented. The structure of maintainability requirements consists of

* Lifespan: What is the expected life of the software? Short-lived applications will not likely require a high level of maintainability, due to their imminent replacement.
* Frequency of Revisions: How often will the application need to be modified? If you are utilizing an agile or iterative software methodology, there will likely be frequent modifications. Likewise, if the business process behind the application changes frequently, then a higher degree of maintainability will be necessary.
* Resources**:** How will the client support this software? If IT support resources within an organization are scarce, then a more easily modified software is advantageous.

**Usability** - Usability measures characteristics such as consistency and aesthetics in the user interface. Consistency is the constant use of mechanisms employed in the user interface while aesthetics refers to the artistic, visual quality of the user interface. It is the ease at which the users operate the system and make productive use of it. Usability is discussed with relation to the system interfaces, but it can just as well be applied to any tool, device, or rich system. This addresses the factors that establish the ability of the software to be understood, used, and learned by its intended users. The application interfaces must be designed with end users in mind so that they are intuitive to use, are localized, provide access for differently.

**Performance -** Requirements about resources required, response time, transaction rates, throughput, benchmark specifications or anything else having to do with performance.

## Challenges

Non-functional requirements are necessary and effective, but that does not mean they come without challenges. To the contrary, we too often find that non-functional requirements are:

* Difficult to Model ***–***Non-functional requirements vary so much that they lack a consistent method of representation. Without that consistency, they are usually modelled for each specific project.
* Casually-Stated ***–***Many times, the need for a non-functional requirement comes from a nonchalant statement by a user or other stakeholder. It is imperative that stakeholders and project managers clearly communicate and identify what is most important to the project’s overall success. No assumptions allowed.
* Hard to Measure ***–***Non-functional requirements have the capacity to be indefinite, making them difficult to evaluate. Testing methods must then be developed to assess these variable attributes.

## Approach

So non-functional requirements cover a variety of qualities and can be difficult to implement. How do you then define them for maximum effectiveness?

Properly defined Non-functional requirements have four main characteristics, they should be:

* Bounded –They may be completely irrelevant if they do not have limits on their scope.
* Independent –They should be independent of each other so that they do not impact other parts of the system when being tested.
* Negotiable –There must be some degree of flexibility when incorporating non-functional requirements business drivers with bounded context.
* Testable –Every non-functional requirement should have finite, testable criteria.

## Recommendations

Software systems are deeply integrated into daily life and are becoming increasingly complex. This increasing complexity results in a lack of transparency that hinders understanding and negatively affects trust [[29](https://link.springer.com/article/10.1007/s00766-020-00333-1#ref-CR29)]. In this way, it becomes more important to consider NFRs such as transparency in software systems [[97](https://link.springer.com/article/10.1007/s00766-020-00333-1#ref-CR97)]. One of the major challenges of NFRs is that the aspects like usability, maintainability, performance are often unknown to professionals and therefore difficult to understand and analyse. User-centred techniques can support requirements engineers to elicit requirements that are more aligned with these dimensions, as well as with user needs and context.

# Software Estimation

## Context

To have a successful software project & proper execution of task, the Estimation Techniques plays vital role in software development life cycle. The technique which is used to calculate the time required to accomplish a particular task is called Estimation Techniques. To estimate a task different effective Software Estimation Techniques can be used to get the better estimation. Estimation is the process of finding an estimate, or approximation, which is a value that is usable for some purpose even if input data may be incomplete, uncertain, or unstable. The cost estimate is the financial spend that is done on the efforts to develop and test software in Software Engineering.  Cost estimation models are some mathematical algorithms or parametric equations that are used to estimate the cost of a product or a project.

Various techniques or models are available for cost estimation, also known as Cost Estimation Models, are:

* Empirical Estimation Technique - Empirical estimation is a technique or model in which empirically derived formulas are used for predicting the data that are a required and essential part of the software project planning step. These techniques are usually based on the data that is collected previously from a project and based on some guesses, prior experience with the development of similar types of projects, and assumptions. It uses the size of the software to estimate the effort. In this technique, an educated guess of project parameters is made. Hence, these models are based on common sense. However, as there are many activities involved in empirical estimation techniques, this technique is formalized. For example, Delphi technique and Expert Judgement technique.
* Heuristic Technique **–** Heuristic word is derived from a Greek word that means “to discover”. The heuristic technique is a technique or model that is used for solving problems, learning, or discovery in the practical methods which are used for achieving immediate goals. These techniques are flexible and simple for taking quick decisions through shortcuts and good enough calculations, most probably when working with complex data. But the decisions that are made using this technique are necessary to be optimal. In this technique, the relationship among different project parameters is expressed using mathematical equations. The popular heuristic technique is given by Constructive Cost Model (COCOMO). This technique is also used to increase or speed up the analysis and investment decisions.
* Analytical Estimation Technique – Analytical estimation is a type of technique that is used to measure work. In this technique, firstly the task is divided or broken down into its basic component operations or elements for analysing. Second, if the standard time is available from some other source, then these sources are applied to each element or component of work. Third, if there is no such time available, then the work is estimated based on the experience of the work. In this technique, results are derived by making certain basic assumptions about the project. Hence, the analytical estimation technique has some scientific basis. [Halstead’s](https://www.geeksforgeeks.org/software-engineering-halsteads-software-metrics/) software science is based on an analytical estimation model.

WBS - We also have a Software Testing Estimation Technique known as **Work Breakdown Structure**. Work Breakdown Structure (WBS), in Project Management and Systems Engineering, is a deliverable-oriented decomposition of a project into smaller components. WBS is a key project deliverable that organizes the team's work into manageable sections. It is defined as- deliverable oriented hierarchical decomposition of the work to be executed by the project team.

The WBS helps to project manager and the team to create the task scheduling, detailed cost estimation of the project. By using the WBS motions, the project manager and team will have a pretty good idea whether or not they’ve captured all the necessary tasks, based on the project requirements, which are going to need to happen to get the job done. In this technique the complex project is divided into smaller pieces. The modules are divided into smaller sub-modules. Each sub-module is further divided into functionality. And each functionality can be divided into sub-functionalities.

After breakdown, the work all functionality should review to check whether each & every functionality is covered in the WBS. Using this you can easily figure out what all task needs to complete & they are breakdown into details task so estimation to details task would be easier than estimating overall Complex project at one shot. The key benefits of WBS include:

* Work Breakdown Structure forces the team to create detailed steps
* Work Breakdown Structure help to improve the schedule and budget
* Work Breakdown Structure creates accountability
* Work Breakdown Structure creation breeds commitment

Planning Poker Estimation Technique - Planning Poker is a consensus-based technique for estimating, mostly used to estimate effort or relative size of user stories in Scrum.

Planning Poker combines three estimation techniques − Wideband Delphi Technique, Analogous Estimation, and Estimation using WBS.

Planning Poker was first defined and named by James Grenning in 2002 and later popularized by Mike Cohn in his book "Agile Estimating and Planning”, whose company trade marked the term.

Benefits of Planning Poker Estimation

Planning poker combines three methods of estimation −

* Expert Opinion − In expert opinion-based estimation approach, an expert is asked how long something will take or how big it will be. The expert provides an estimate relying on his or her experience or intuition or gut feel. Expert Opinion Estimation usually does not take much time and is more accurate compared to some of the analytical methods.
* Analogy − Analogy estimation uses comparison of user stories. The user story under estimation is compared with similar user stories implemented earlier, giving accurate results as the estimation is based on proven data.
* Disaggregation − Disaggregation estimation is done by splitting a user story into smaller, easier-to-estimate user stories. The user stories to be included in a sprint are normally in the range of two to five days to develop. Hence, the user stories that possibly take longer duration need to be split into smaller use-Cases. This approach also ensures that there would be many stories that are comparable.

## Challenges

The major challenges of Software Estimation are:

* **Software Requirements -** To build a product or customized software, the requirements are never clear. They keep evolving. Even if, they are well-documented in RFP (Request for Proposal) or a Product Description Document, they are never synchronously understood by all the stakeholders. Additionally, because of the new Web or Cloud platform, there are various non-functional requirements like load, multiple browsers, multiple devices, dependency on computer, network speed etc. When you are quoting during software sales, your estimate can go off by more than 50 %.
* Schedule – Customer wants delivery as per his schedule because of his business priorities. This is sometimes not possible due to technical or business process limitations. Certain modules need to be developed before certain others. Some software deliveries need to be sequential and cannot be parallel like you cannot deploy two mothers to get a child delivered in 4 months, instead of 9 months.
* Due to hidden factors, estimation can be over or underestimated
* It is based on thinking
* Involved Risk

## Approach

The estimation can be made reliable, provided the team follows certain steps:

* The team members must be aligned with the goals and objectives. There are members in the team who create conflict and frustration. Resolution comes but with delay in timelines, change of scope and increased cost. Thus, aligned team will help in reducing the time, cost, and effort.
* Input from the people who have done the similar kind of work in the past acts as a guide.
* Identifying and investigating the high-risk items help in reducing the cost. Three-point estimates are the great way of doing this

## Recommendation

Establish a buffer that allows for unplanned activities or things that you just typically miss in an estimate. Next, refine the buffer over time based on outcomes. If you focus on timeframes when estimating, it encourages the silent cutting of corners. Estimate the project first, work out a realistic timeline, and then:

* Reduce scope based on business priority and/or.
* Increase the team size.
* Factor in the extra inefficiencies you are creating.

# Architectural and Design Patterns

## Context

The architecture of a system describes its major components, their relationships (structures), and how they interact with each other. Software architecture and design is a process that includes several contributory factors such as Business strategy, quality attributes, human dynamics, design, and IT environment. We can segregate Software Architecture and Design into two distinct phases: Software Architecture and Software Design. In Architecture, non-functional decisions are cast and separated by the functional requirements. In Design, functional requirements are accomplished.

Architecture serves as a blueprint for a system. It provides an abstraction to manage the system complexity and establish a communication and coordination mechanism among components. It defines a structured solution to meet all the technical and operational requirements, while optimizing the common quality attributes like performance and security. It involves a set of significant decisions about the organization related to software development and each of these decisions can have a considerable impact on quality, maintainability, performance, and the overall success of the final product. These decisions comprise of:

* Selection of structural elements and their interfaces by which the system is composed.
* Behaviour as specified in collaborations among those elements.
* Composition of these structural and behavioural elements into large subsystem.
* Architectural decisions align with business objectives.
* Architectural styles that guide the organization.

Software design provides a design plan that describes the elements of a system, how they fit, and work together to fulfil the requirement of the system. The objectives of having a design plan are as follows:

* To negotiate system requirements, and to set expectations with customers, marketing and management personnel.
* Act as a blueprint during the development process.
* Guide the implementation tasks, including detailed design, coding, integration, and testing.

The primary goal of the architecture is to identify requirements that affect the structure of the application. A well-laid architecture reduces the business risks associated with building a technical solution and builds a bridge between business and technical requirements.

# Project Management Practices

## Context

A project is temporary in that it has a defined beginning and end in time, and therefore defined scope and resources.

And a project is unique in that it is not a routine operation, but a specific set of operations designed to accomplish a singular goal. So a project team often includes people who don’t usually work together – sometimes from different organizations and across multiple geographies.

The development of software for an improved business process, the construction of a building or bridge, the relief effort after a natural disaster, the expansion of sales into a new geographic market — all are projects.

And all must be expertly managed to deliver the on-time, on-budget results, learning and integration that organizations need.

Project management, then, is the application of knowledge, skills, tools, and techniques to project activities to meet the project requirements.

Project management processes fall into five groups:

* Initiating
* Planning
* Executing
* Monitoring and Controlling
* Closing

Project management knowledge draws on ten areas:

* Integration
* Scope
* Time
* Cost
* Quality
* Procurement
* Human resources
* Communications
* Risk management
* Stakeholder management

All management is concerned with these, of course. But project management brings a unique focus shaped by the goals, resources and schedule of each project. The value of that focus is proved by the rapid, worldwide growth of project management:

* as a recognized and strategic organizational competence
* as a subject for training and education
* as a career path

## Challenges

Challenges Within Corporate Projects-

* Undefined Goals – When goals are not clearly identified, the whole project and team can suffer. When upper management cannot agree to or support undefined goals, the project in question typically has little chance of succeeding. The project manager must ask the right questions to establish and communicate clear goals from the outset.
* Scope Changes**–** Also known as scope creep, this occurs when project management allows the project’s scope to extend beyond its original objectives. Clients and supervisors may ask for changes to a project, and it takes a strong project manager to evaluate each request and decide how and if to implement it, while communicating the effects on budget and deadlines to all stakeholders.

Challenges for Project Managers

* Inadequate Skills for the Project – A project sometimes requires skills that the project’s contributors do not possess. Project management training can help a project leader determine the needed competencies, assess the available workers and recommend training, outsourcing or hiring additional staff.
* Lack of Accountability – A project manager’s leadership qualities can shine when each member of the team takes responsibility for his or her [role in achieving project success](https://www.villanovau.com/resources/project-management/project-team-roles-and-responsibilities/). Conversely, a lack of accountability can bring a project to a complete halt. Finger-pointing and avoiding blame are unproductive, but all-too-common features of flawed project management. Learning to direct teams toward a common goal is an important aspect of project management training.

## Best Practices

Here are 13 best practices for project management we recommend.

* Document project requirements
* Avoid estimating projects in a vacuum
* Use a project brief to get stakeholder buy-in
* Establish a clear project plan
* Mind your team’s workload
* Clarify project roles and responsibilities
* Communicate early and often
* Monitor task progress
* Manage risk
* Keep scope creep in check
* Conduct a project post-mortem
* Iterate on your process
* Develop leadership skills

# Root Cause Analysis

## Context

A root cause is defined as a factor that caused a non-conformance and should be permanently eliminated through process improvement. The root cause is the core issue—the highest-level cause—that sets in motion the entire cause-and-effect reaction that ultimately leads to the problem(s). Root cause analysis (RCA) is defined as a collective term that describes a wide range of approaches, tools, and techniques used to uncover causes of problems. Some RCA approaches are geared more toward identifying true root causes than others, some are more general problem-solving techniques, and others simply offer support for the core activity of root cause analysis.

There are many methodologies, approaches, and techniques for conducting root cause analysis, including:

* **Events and causal factor analysis:** Widely used for major, single-event problems, such as a refinery explosion, this process uses evidence gathered quickly and methodically to establish a timeline for the activities leading up to the accident. Once the timeline has been established, the causal and contributing factors can be identified.
* **Change analysis:** This approach is applicable to situations where a system’s performance has shifted significantly. It explores changes made in people, equipment, information, and more that may have contributed to the change in performance.
* **Barrier analysis:** This technique focuses on what controls are in place in the process to either prevent or detect a problem, and which might have failed.
* **Management oversight and risk tree analysis:** One aspect of this approach is the use of a [tree diagram](https://asq.org/quality-resources/tree-diagram) to look at what occurred and why it might have occurred.
* **Kepner-Tregoe Problem Solving and Decision Making:** This model provides four distinct phases for resolving problems:
  + Situation analysis
  + Problem analysis
  + Solution analysis
  + Potential problem analysis

## Challenges

One challenge when conducting a root cause analysis is ensuring you are identifying root causes rather than causal factors. A causal factor is any behaviour, omission, or deficiency that, if corrected, eliminated, or avoided, probably would have prevented the event. A root cause is a factor that if eliminated would definitely prevent recurrence. Root cause analysis purists focus on identifying a root cause over a causal factor. However, many of the processes where root cause analysis is applied generate adverse events because of human error. Removing a specific manifestation of the error doesn’t necessarily highlight how the type of mistake can be repeated. Ultimately this specific focus can ignore a systematic error. The other various challenges include:

* Data Integrity - The most basic requirement for root cause analysis is data. Collecting as much data as possible throughout the process you are examining will improve the quality and efficiency of the root cause analysis. However, this is another focus for root cause analysis critics. Oftentimes data collection about the precipitating event begins after the event. It also requires multiple testimonies and interviews. This qualitative data collection can be unreliable, especially if these interviews need to occur days, weeks, or even months after the event.
* Easy vs Lasting Solutions - Unfortunately, solutions to these events can be complex. Instead of removing a piece from an existing process, the best solution may be a complete redesign, new technology implementation, or other large scale adjustment. Even when necessary, administrators, managers, and leaders tend to look for quick and easy solutions. Prioritizing a solution that doesn’t solve the problem in the long term, while easier, can lead to event recurrence.

## Recommendations

Some of the basic principles of RCA can help organizations ensure they are following the correct methodology:

* Focusing on corrective measures of root causes is more effective than simply treating the symptoms of a problem or event.
* Effective RCA is accomplished through a systematic process with evidence-backed conclusions.
* There is usually more than one root cause for a problem or event
* The focus of RCA, via problem identification, is WHY the event occurred—not who made the error.