Software Engineering II
(CSC 392)
# Table of Contents

Lecture 1: Introduction to Software Project Management ......................................................... 1  
Course Outline .......................................................................................................................... 1  
Course Objectives .................................................................................................................. 1  
Recommended Resources ...................................................................................................... 1  
Software and Software Engineering ....................................................................................... 2  
CHAOS Report ....................................................................................................................... 2  
Investment in IT Projects ....................................................................................................... 3  
Project and Triple Constraint ............................................................................................... 3  
Project Management and PM Framework .............................................................................. 3  
Program and Project Portfolio Management .......................................................................... 4  
Top Factors for Project Success ............................................................................................ 4  
Suggested Skills for Project Managers .................................................................................. 4  
Bibliography ......................................................................................................................... 5  

Lecture 2: Project Management and Information Technology Context .................................... 6  
Systems Philosophy .............................................................................................................. 6  
Organizational Frames .......................................................................................................... 6  
Organizational Structures ..................................................................................................... 6  
Organizational Culture .......................................................................................................... 7  
Stakeholder Management ..................................................................................................... 7  
Role of Top Management in Project Success ........................................................................ 7  
Project Life Cycle .................................................................................................................. 7  
Management Reviews .......................................................................................................... 8  
Bibliography ......................................................................................................................... 8  

Lecture 3: Project Integration Management ........................................................................... 9  
Recent Trends in IT Projects ................................................................................................. 9  
Globalization ......................................................................................................................... 9  
Outsourcing ........................................................................................................................... 9  
Virtual Teams ........................................................................................................................ 9  
Project Management Process Groups .................................................................................. 9  
Project Integration Management .......................................................................................... 10
Lecture 7: Project Cost Management

Importance of Cost Management
Main Processes
Estimating Costs
Cost Estimation Tools and Techniques
Cost Estimate Problems in IT Projects
Determining the Budget
Controlling Cost
Earned Value Management
Bibliography

Lecture 8: Project Quality Management and Project Communication Management

Importance of Quality Management
Project Quality Management
Project Communication Management
Identifying Stakeholders
Planning Communications
Distributing Information
Managing Stakeholders
Reporting Performance
How to Improve Project Communications
Bibliography

Lecture 9: Project Risk Management

Importance of Risk Management
Main Processes
Planning Risk Management
Risk Categories
Identifying Risks
Performing Qualitative Risk Analysis
Performing Quantitative Risk Analysis
Planning Risk Responses
Lecture 13: Pattern-Based Design

Design Patterns ........................................................................................................................................... 38
Types of Patterns ....................................................................................................................................... 38
Describing a Pattern ................................................................................................................................. 38
User Interface Design Patterns .................................................................................................................. 38
  Top Level Navigation ........................................................................................................................... 39
  Card Stack ........................................................................................................................................... 39
  Fill-in-the-Blanks ................................................................................................................................. 39
  Sortable Table ..................................................................................................................................... 39
  Bread Crumbs ...................................................................................................................................... 39
  Edit in Place ......................................................................................................................................... 39
  Simple Search ...................................................................................................................................... 39
  Wizard ................................................................................................................................................. 40
  Shopping Cart ...................................................................................................................................... 40
  Progress Indicator ............................................................................................................................... 40
Bibliography ............................................................................................................................................ 40

Lecture 14: Web Application Design

Web Application Quality .............................................................................................................................. 41
Content Quality .......................................................................................................................................... 41
Design Goals of Web Application ............................................................................................................ 41
Interface Design ........................................................................................................................................ 42
Aesthetic Design ....................................................................................................................................... 42
Navigation Design ..................................................................................................................................... 42
Bibliography ............................................................................................................................................ 42

Lecture 15: Software Quality

Quality ....................................................................................................................................................... 43
Software Quality ...................................................................................................................................... 43
Software Quality Models .......................................................................................................................... 43
  Garvin’s Quality Dimensions ............................................................................................................... 43
  McCall’s Quality Factors ....................................................................................................................... 43
Handouts

ISO 9126 Quality Factors ........................................................................................................................... 44
Targeted Quality Factors ............................................................................................................................. 44
Software Quality Dilemma ............................................................................................................................. 44
Achieving Software Quality ........................................................................................................................... 45
Bibliography .................................................................................................................................................. 45

Lecture 16: Review Techniques .................................................................................................................... 46
Software Reviews ........................................................................................................................................... 46
Cost Impact of Software Defects .................................................................................................................... 46
Review Metrics and Their Use ....................................................................................................................... 46
Analyzing Metrics ........................................................................................................................................ 46
Cost Effectiveness of Reviews ...................................................................................................................... 47
Reviews: A Formality Spectrum ...................................................................................................................... 47
Informal Reviews ......................................................................................................................................... 47
Example: Review Checklist ............................................................................................................................ 47
Formal Technical Reviews .............................................................................................................................. 48
Review Guidelines ........................................................................................................................................ 48
Bibliography .................................................................................................................................................. 48

Lecture 17: Software Quality Assurance .................................................................................................... 50
Elements of Software Quality Assurance .................................................................................................... 50
SQA Tasks ..................................................................................................................................................... 50
Requirements Quality ................................................................................................................................... 51
Design Quality ............................................................................................................................................. 51
Code Quality ............................................................................................................................................... 51
Quality Control Effectiveness ....................................................................................................................... 51
Statistical Quality Assurance ....................................................................................................................... 51
Possible Causes of Errors ............................................................................................................................. 52
Bibliography ................................................................................................................................................ 52

Lecture 18 & 19: Testing Web Applications ................................................................................................ 53
Dimensions of Web Application Quality ....................................................................................................... 53
Content Testing ............................................................................................................................................ 53
Database Testing .......................................................................................................................................... 53
User Interface Testing .................................................................................................................................. 53
Handouts

Testing Interface Mechanisms ............................................................................................................ 54
Usability Tests ..................................................................................................................................... 54
Compatibility Tests .............................................................................................................................. 55
Component-Level Testing ....................................................................................................................... 55
Navigation Testing .................................................................................................................................. 55
Testing Navigation Syntax ................................................................................................................... 55
Testing Navigation Semantics ............................................................................................................. 56
Configuration Testing .............................................................................................................................. 56
Security Testing ....................................................................................................................................... 56
Performance Testing ............................................................................................................................... 57
Bibliography ............................................................................................................................................ 57

Lecture 20 & 21: Software Configuration Management ............................................................................ 58
Importance .............................................................................................................................................. 58
Important Roles in SCM .......................................................................................................................... 58
Elements of a Configuration Management System ................................................................................ 58
SCM Repository ....................................................................................................................................... 58
SCM Features .......................................................................................................................................... 59
SCM Process ............................................................................................................................................ 59
Identification of Objects ...................................................................................................................... 60
Version Control ..................................................................................................................................... 60
Change Control ...................................................................................................................................... 60
Configuration Audit ............................................................................................................................. 61
Status Reporting .................................................................................................................................... 61
Configuration Management for Web Application .................................................................................. 61
Content Management .......................................................................................................................... 62
Change Management ........................................................................................................................... 62
Version Control ..................................................................................................................................... 62
Auditing and Reporting .......................................................................................................................... 62
Bibliography ............................................................................................................................................ 62

Lecture 22 & 23: Product Metrics ............................................................................................................... 63
Importance .............................................................................................................................................. 63
Framework for Product Metrics ............................................................................................................. 63
Metrics for Requirements Model ........................................................................................................................................... 64

Function-Based Metrics ............................................................................................................................................... 64

Metrics for Specification Quality ................................................................................................................................. 64

Metrics for Design Model ............................................................................................................................................... 65

Metrics for Object-Oriented Design ............................................................................................................................... 65

Class-Oriented Metrics ............................................................................................................................................... 65

Component-Level Design Metrics ................................................................................................................................ 66

Operation-Oriented Metrics ........................................................................................................................................ 67

Design Metrics for Web Applications ................................................................................................................................ 67

Interface Metrics ......................................................................................................................................................... 67

Aesthetic Design Metrics ........................................................................................................................................... 67

Content Metrics .......................................................................................................................................................... 68

Navigation Metrics ......................................................................................................................................................... 68

Metrics for Source Code ............................................................................................................................................... 68

Metrics for Object-Oriented Testing ............................................................................................................................... 68

Metrics for Maintenance ............................................................................................................................................... 69

Bibliography ............................................................................................................................................................... 69

Lecture 24 & 25: Software Process Improvement ........................................................................................................... 70

Importance ................................................................................................................................................................. 70

Software Process Improvement ........................................................................................................................................ 70

SPI Process ............................................................................................................................................................. 71

Assessment and Gap Analysis ........................................................................................................................................ 71

Education and Training ................................................................................................................................................ 71

Selection and Justification ........................................................................................................................................... 71

Installation / Migration .................................................................................................................................................. 72

Evaluation ...................................................................................................................................................................... 72

Risk Management for SPI ............................................................................................................................................... 72

Critical Success Factors ................................................................................................................................................ 73

Capability Maturity Model Integration (CMMI) .................................................................................................................... 73

Other SPI Frameworks .................................................................................................................................................. 73

The People CMM ........................................................................................................................................................ 74

Software Process Improvement and Capability dEtermination (SPICE) ........................................................................... 74
Lecture 26 & 27: Software Reengineering

Reengineering

Business Process Reengineering

BPR Model

Software Reengineering Process Model

Reverse Engineering

Reverse Engineering to Understand Data

Reverse Engineering to Understand Processing

Reverse Engineering User Interfaces

Restructuring

Code Restructuring

Data Restructuring

Forward Engineering

Economics of Reengineering

Bibliography

Lecture 28 & 29: Software Reuse

Importance

Benefits of Software Reuse

Problems with Reuse

The Reuse Landscape

Key Factors for Reuse

Application Frameworks

Software Product Line (SPL)

COTS Product Reuse

COTS-Solution Systems

COTS-Integrated Systems

Bibliography
Lecture 30 & 31: Component-Based Software Engineering ................................................................. 87
Importance ........................................................................................................................................... 87
Essentials of CBSE .............................................................................................................................. 87
Characteristics of Component ............................................................................................................ 87
Component as a Service Provider ..................................................................................................... 88
Component Interfaces ........................................................................................................................ 88
Component Models ............................................................................................................................ 88
CBSE Processes ................................................................................................................................. 89
CBSE for Reuse ................................................................................................................................. 89
Software Process ............................................................................................................................... 90
Bibliography ....................................................................................................................................... 90
Glossary ............................................................................................................................................... 91
Software Engineering II (CSC 392)

Lecture 1: Introduction to Software Project Management
You have already learned the basic concepts of software engineering in your subject of SE1. I hope that you have good understanding of process models that include traditional models (e.g. waterfall and incremental model) as well as agile models such as Scrum and XP. You got the understanding of major phases of software development life cycle i.e. requirement analysis, design, development, and testing. Different key phases of requirement engineering such as requirement inception, elicitation, and elaboration were discussed. You also discussed different basic concepts related to software design. Modeling activities for different phases using UML diagrams were also taught in your previous course. You also had few lectures to understand the software testing concepts.

Course Outline
In this course, we will discuss different topics that are broadly categorized into software project management, software design, quality management, and advanced software engineering. In the beginning, we will focus on topics like project management processes, scope, time, cost, and quality management. We will also discuss the key processes of project communication and risk management. Later on, we will learn how to design better user interfaces for different systems including web applications.

We will cover the range of topics related to software quality management such as quality assurance activities, review techniques, testing strategies for different applications such as web applications, and software configuration management. We will also discuss various topics such as software reuse, component-based software engineering, software reengineering, and process improvement under the umbrella of advanced software engineering.

Course Objectives
The main learning objectives for this course are as follows:

- To familiarize students with the advanced topics of software engineering.
- To develop students’ skills for planning and managing real life software projects successfully.

Recommended Resources
Most of lectures will be based on the text from the book (Software Engineering: A Practitioner’s Approach by R. Pressman) whereas the lectures about software project management will be
based on IT Project Management by K. Schwalbe. Software Engineering (I. Sommerville) is also a good book to consult and last few lectures will be based on the text from this book.

- **Text books**

- **Reference books**

**Software and Software Engineering**
We start with the term software; today, we see software in almost every domain of life e.g. we do online transactions using internet banking systems; we have different healthcare systems in hospitals; and we enjoy games on mobile devices. We may define software as a computer application or set of instructions to achieve some desired output by executing it. Software product differs from other products (e.g. hardware products). Software is differentiated by other products (particularly hardware products) as it is developed rather than manufactured. Software does not wear out like other products. Although many commercial software are available but most of the time software are still developed based on customer defined requirements.

IEEE defines Software engineering as “(1) the application of a systematic, disciplined, quantifiable approach to the development, operation, and maintenance of software; that is, the application of engineering to software. (2) the study of approaches as in (1).”

**CHAOS Report**
Despite of large use of software applications or systems, software industry is still facing problems in software development, software projects’ costs are overrun. These projects often need more time as compared to the estimated duration. In many cases, customers are not satisfied with the final product and most of projects have no or insufficient documentation. Standish group conducted a study to estimate the success rate of software projects that highlighted very alarming results. The success rate of software/IT projects was 16.2% only whereas 31% projects were cancelled before the completion. It also estimated $140 billion loss in US only. This study emphasized the need of better project management practices and techniques to overcome the mentioned problems. The same group conducted the similar study after about 10 years; the results were relatively better as compared to the previous study. The
success rate of projects was 35% and 19% projects were cancelled as compared to 31% projects in 1995. The total loss was $53 billion that was about one third of the loss estimated in 1995. The later study clearly indicates the improvement in software projects as a result of better software engineering and project management practices.

**Investment in IT Projects**

Although rate of successful project is still not impressive, there is continuous increase in investments in IT projects. In 2008, $2.4 trillion was spent in IT projects globally that was 8% increase from 2007. US spends $2.3 trillion on different types of projects including IT projects every year. Globally $10 trillion are spent on all types of projects. These investments demand systematic approach towards management of projects in general and IT projects in particular.

**Project and Triple Constraint**

Project is defined as “a temporary endeavor undertaken to create a unique product, service, or result” [PMBOK® Guide, 2008]. The key attributes of any project are its unique purpose, temporary nature (definitive start and end dates), progressive elaboration, various resources required, primary customer/sponsor, and uncertainty involved in it.

Projects often involve competing goals related to scope, cost, time that are key constraints. These constraints are called triple constraint and project managers often do trade-offs for these goals. Sometimes quality is also considered the fourth main competing goal and these four goals are called quadruple constraint. Project managers strive to meet scope, time, cost, and quality goals. They strive to facilitate the entire process to meet the needs and expectations of the stakeholders. The key stakeholders of a project are project sponsor, project team, support staff, customers, users, suppliers, and opponents of project.

**Project Management and PM Framework**

Project management is “the application of knowledge, skills, tools and techniques to project activities to meet project requirements” [PMBOK® Guide, 2008].

Project management framework is developed by Project Management Institute (PMI) and it divides project management practices into nine knowledge areas for good project management. These knowledge areas are the required competencies for any project manager. These knowledge areas are further divided into core functions, facilitating functions, and project integration management. The four core knowledge areas are project scope, time, cost, and quality management. The question is that why we call these knowledge areas as core areas? From the figure, you may observe that these knowledge areas are related to specific project objectives including scope, time, cost, and quality. Project scope management deals with defining and managing all the work required to complete the project successfully. Project time management is about time estimation to complete the project. Project cost management
deals with preparing and managing the project budget. Project quality management activities help to meet the quality goals of project or in other words it helps to meet customer’s requirements.

The four facilitating knowledge areas are HR management, communications management, risk management, and procurement management. These are called facilitating knowledge areas because these are processes through which the project objectives are achieved. Human resource management is about how to effectively manage the key resource i.e. people involved in the project. Communications management is all about the relevant information generation, dissemination, and storage. Risk management activities are identification and analysis of possible risks, and appropriate response to project risks. Procurement management caters the activities related to procurement of goods and services related to project from other organizations. Project integration management tools and techniques help to effectively integrate all knowledge areas activities.

Program and Project Portfolio Management
Program is “a group of related projects managed in a coordinated way to obtain benefits and control not available from managing them individually” [PMBOK Guide, 2008]. Program manager guides project managers and have review meetings time to time. They also have strong skills to effectively communicate with multiple stakeholders inside and outside the organization.

Project portfolio management is an emerging business strategy; it is about to maintain the record of previous projects and programs. The main objective is to make wise investment decisions for future projects. Portfolio manager, the responsible for portfolio management, has not necessarily previous experience as project manager but must have strong financial and analytical skills.

Top Factors for Project Success
The Standish group conducted a study to find out top factors for project success in 2001. This study highlighted executive support, user involvement, experienced project manager, clear business objectives, minimized scope, standard software infrastructure, firm basic requirements, and reliable estimates as the main factors for project success.

Suggested Skills for Project Managers
Project Management BOK suggests skill for successful project managers. It suggests that project managers should have good knowledge about application area, project environment knowledge, standards, and regulations. They should have general management knowledge and skills particularly soft skills to deal with people.
Bibliography

Lecture 2: Project Management and Information Technology Context

This lecture provides basic understanding of project management in the context of information technology.

Systems Philosophy
No organization runs in isolation therefore no project is done in isolation too. Organizations initiate projects by considering broad organizational needs and context. People need to understand holistic view related to the project and organization that is also called systems thinking. Every organization deals with multiple systems so systems analysis and management are key activities to do for every project. Organizations need to understand three-sphere model i.e. business, organization, and technology aspects of a project.

Organizational Frames
Organizations often contain four different types of frames i.e. structural, human resource, political, and symbolic frames. Structural frame deals with organization structure; how the roles and responsibilities are defined; and what are the coordination and control mechanism within the organization. Human resource frame is the mainly responsible to create harmony between organization’s needs and people’s needs. Political frame is all about organizational and personal politics; it involves competition between groups or individuals, and issues related to power, leadership, and limited resources within the organization. Project supporters and opponents plays important role for the project through organizational politics. Symbolic frame is about symbols and meanings of events held in the organization. Organization culture is also the part of symbolic frame that deal with the issues such as how people deal with each other and what are their work practices.

Organizational Structures
Organizational structure plays quite important role in project management; it can be divided into three types. Functional structure is the conventional structure found in many organizations where vice presidents/heads (managers) of different departments directly report to CEO. These organization have specialized staff for each department such as marketing and HR. Project structure is another type of organization structure in which program managers report to CEO. In such organizations, staff is assigned to different programs/projects and they have various skills to manage projects successfully. The third type of organization structure is matrix that is the mix of functional and project structures. These organizations may have program managers (as in project structure) as well as vice presidents for different departments (as in functional structure). Matrix structure can be further divided into strong, balanced, and weak matrix. Strong matrix is inclined to project structure whereas weak matrix focuses on functional structure.
Organizational Culture
Organizational culture is defined as a set of shared assumptions, values, and behaviors of people within the organization. Many problems are related to organizational culture rather than technical during the project execution. There may be subcultures within the organization; development teams/department may have different work practices e.g. working hours as compared to testing teams/department.

The main characteristics of organizational culture are related to people and their work practices e.g. members’ identity, team work, performance based rewards, and conflict tolerance. These characteristics are briefly defined here. Member identity is related to the member’s association with the organization rather than profession or project team. In other words, how loyal the employees are with their organization. Group emphasis is another key factor that focuses on team work rather than individuals’ efforts. Risk tolerance, conflict tolerance, reward criteria, unit integration, open-systems focus, people focus, control, and means-ends orientation are other key characteristics of organizational culture.

Stakeholder Management
Stakeholders are the people involved in the project and they may be internal or external to the organization. Top management within the organization and customer/sponsor are considered key stakeholders of any project. There is often competition for limited resources among different project stakeholders. Therefore, it is quite important to wisely communicate with stakeholders. Project manager has to manage relationships with them based on their priorities for the project.

Role of Top Management in Project Success
The role of top management in project success is quite important as they are responsible to provide adequate resources for the project. They also approve resources for unique project needs and insist other departments to cooperate each other. They also coach project managers to lead project teams in better way and to resolve different issues rose during the project execution. They ensure the use of relevant standards and tools for the successful completion of project.

Project Life Cycle
Every project consists of various project phases and different work, deliverables, time, and team are involved at each phase. These different phases (not fixed) may be defined and viewed as project life cycle. In general, limited resources are required in early phases of project. It involves high level of uncertainty and stakeholders have strong influence on the project. In middle phases, level of uncertainty decreases but more resources are required to execute the project. In final phase, project objectives should be achieved (as planned) and formal customer’s approval should be obtained.
Product life cycle is different than project life cycle as software product development is a subset of IT projects. It is also called software/system development life cycle that includes predictive life cycle (such as waterfall and spiral models) and adaptive software development (i.e. agile models).

Management Reviews
Management reviews are conducted to evaluate the progress at each project phase. These reviews are also called status reviews which are more important in IT projects/products due to their complex nature. It helps to evaluate progress, potential success, and compatibility of project with organizational goals. It is always good to have frequent reviews for the continuous monitoring of the project.

Bibliography
Lecture 3: Project Integration Management
This lecture discusses the recent trends in IT projects, project management process groups, and the importance of project integration management.

Recent Trends in IT Projects
The recent trends in IT projects are discussed in this section.

Globalization
Today’s world is like global village due to the advances in technology (particularly information technology), and lower trade and political barriers. People, living at different locations and time zones, work together for the same project. In such scenario, project managers need to address the possible communication issues and should build the trust among team members. There must be common work practices to work effectively and tools to support this whole process. KPMG International suggests (after studying 600 global organizations) greater project discipline, collaboration over standardization, use of newer and innovative tools, good project momentum, and think global, act local philosophy for successful global projects.

Outsourcing
Outsourcing is another trend in IT projects; it is defined as acquiring goods/services from an outside source. Sometimes another term ‘offshoring’ is also used for acquiring goods/services from another country. It helps to reduce project costs because organizations may acquire cheap skilled labor in other countries. The issues may rise about the integration of software components/applications. Integration and standardization of IT infrastructure and business processes may help to overcome such issues. Organization also need better approaches to negotiate contracts, working plans, and managing virtual teams.

Virtual Teams
Virtual teams are often formed to do work on the same project; these teams work at remote locations within the same country or often in different countries. The possible advantages of virtual teams are competitiveness, responsiveness, 24/7 availability, reduced costs, even no office space, and work/life balance for team members. On the other side, the possible disadvantages are isolated team members, communication problems, no informal learning/networking, and technology dependency. The key success factors for virtual teams are well defined team processes, leadership style, trust and relationships, team member selection and role preferences, task-technology fit, cultural differences, computer-mediated communication, team life cycles, and conflict management.

Project Management Process Groups
Process is defined as a series of actions directed towards a specified goal whereas project management consists of series of interlinked processes. Project management process groups
should not be confused with project phases. Project processes are divided into five process groups. Initiating processes are about defining and authorizing project/phase at the beginning of each phase. Planning processes are more related to devising and maintaining workable scheme and developing project management plan. Executing processes are about coordinating people and other resources, and development activities. Monitoring and closing processes are related to measuring and monitoring project progress. These processes also involve corrective actions and rework. Closing processes include formal acceptance from the project sponsor/customer and other administrative activities to close the phase or project.

**Project Integration Management**
Project integration management deals with all knowledge areas; these activities are related to coordinating all knowledge areas throughout a project’s life cycle. The following are main processes of project integration management:

- Developing project charter
- Developing project management plan
- Directing and managing project execution
- Monitoring and controlling project work
- Performing integrated change control
- Closing project or phase

**Developing Project Charter**
Project charter is a formal document to recognize the project. The key sections of a project charter include project title and date of authorization, contact info, project objectives, business need and justification, summary of project schedule, budget, communication plan, project success criteria, roles and responsibilities matrix, and sign-off and comments section.

**Developing Project Management Plan**
Project management plan is the main document to coordinate all project planning documents. It includes project planning assumptions and decisions, and assists in project’s execution and control. It is a baseline for progress measurement and project control. It should be dynamic, flexible, and subject to change to accommodate the changes to be occurred time to time during project execution. It requires information from all knowledge areas as it provides guidelines for these all areas. The main inputs to develop a project management plan are project charter, outputs from other planning processes, and enterprise environment factors. The main
technique to prepare a project management plan is expert judgment (of top management, project manager, and key team members).

**Directing and Managing Project Execution**
Most of the time spent on project execution, leading the team, and managing stakeholders’ relationships. Project manager must be flexible and creative to perform these activities effectively. The main inputs of this process are managing and performing the work as described in project management plan, and change requests. Those who will do the work should plan the work for better coordination of planning and execution activities. Project managers should provide strong leadership and supportive culture to capitalize on product, business, and application area knowledge. Project execution tools and techniques should be used along with expert judgment and project management information systems.

**Monitoring and Controlling Project Work**
Changes are inevitable during project execution; 90% of time spent on communicating and managing changes. This key process is about collecting, measuring, and disseminating performance information. The main inputs are project management plan and performance reports. Project management plan is the baseline for identifying and controlling changes. The main output of this process is change request.

**Performing Integrated Change Control**
This process includes activities about identifying, evaluating, and managing changes throughout the project. It also influences the factors that create changes. The key activities are to determine that a change has occurred and to manage actual changes as they occur. The main inputs for this process are work performance information and change requests. The main outputs are updated change request status and project documents. Changes may be initiated in the form of oral, written, formal, and informal change requests. Change control system is required to effectively manage changes. Change control board plays important role that take all key stakeholders on the board. Configuration management is also important part of integrated change control. The possible drawback is that all activities are time demanding and most of time projects have very tightened schedule.

**Closing Projects or Phases**
This key process is about finalizing all the activities and transferring the completed / cancelled work. The main input is accepted deliverables whereas the main technique used for this process is expert judgment. The main output includes final product, service, or result transmission, and process asset updates.
Bibliography
Lecture 4: Project Scope Management
This lecture discusses the key concepts of project scope management.

Main Processes
Scope is defined as work to be done to create products. Another term ‘deliverable’ is used to describe products that are further divided into product-related (e.g. software component) and process-related (e.g. minutes of meeting) deliverables. Stakeholders’ agreement is quite important for the newly developed products. Project scope management is about defining and controlling scope. The main processes of project scope management are as follows:

- Collecting requirements
- Defining scope
- Creating the WBS
- Verifying scope
- Controlling scope

Collecting Requirements
It is most difficult to collect requirements from the customer. If requirements are not properly defined, it causes major rework for the project. Many techniques to collect requirements include interviews, focus groups, workshops, surveys, observation, and prototyping. High level requirements are often extracted from project charter. Requirements document may consists of a piece of paper or multi volume document depending on the nature of project. It may include text, images, diagrams, and videos. Requirements are normally divided into functional and non-functional requirements. Requirements management plan includes guidelines to analyze, document, and manage requirements. Requirements traceability matrix is also a document that describes requirements, their attributes and status.

Defining Scope
This key process is about defining the work required in detail for the project. Good scope definition includes accurate estimate for time, cost, and other resources. Project scope statement is the baseline for performance measurement and project control. The main techniques used to define scope are expert judgment, product analysis, alternative identification, and facilitated workshops. The main output of this process is project scope statement.
Creating the WBS
This process provides deliverable-oriented grouping of the work. Work Breakdown Structure (WBS) is a foundation document (normally in chart or tabular format) that is developed through this process. It is task-oriented family tree of activities and organized around project products, project phases, or process groups. The task at the lowest level of WBS is called work package and duration estimates for work packages are also provided in WBS. Good WBS needs good understanding of the required project deliverables. The main inputs of creating the WBS are project scope statement, stakeholder requirements documentation, and organizational process assets. The main technique to create WBS is decomposition of all project related work. The main outputs include WBS, WBS dictionary, scope baseline, and project document updates. A WBS dictionary is a document that provides detailed description of each WBS item and its format may vary depending on project needs.

Approaches for Developing WBS
The key approaches to develop WBS include using guidelines, the analogy approach, the top-down approach, the bottom-up approach, and the mind-mapping approach. Using guidelines means to use the existing guidelines for developing WBS e.g. US Department Of Defense (DOD) guidelines. It may include templates and past examples to develop WBS. Analogy approach is followed to create WBS based on the WBS for similar projects. It is possible if organization maintains the repository of WBS and the possible advantage of this approach is better understanding of the project activities.

Top-down approach is considered a conventional approach to develop WBS; it starts with the largest item and further divided into sub-items. The team involved in this process must have vast technical insight and broad perspective of the project. Bottom-up approach is used to identify specific tasks and then aggregate these tasks to develop the WBS. It is very time consuming process and often followed for new systems. Mind-mapping approach is like brainstorming for creating WBS; it starts with the core idea then branches are developed around that idea. It is more visual and less structured approach that involves creativity and active participation. Once idea is thoroughly explored, top-down and bottom-up approaches can be used and WBS can be presented in chart or tabular form.

Suggestions for Creating WBS
A unit of work should be placed at only one place in WBS document. The work content of a WBS item is the sum of the WBS items below it. WBS item is the responsibility of only one individual to have effective monitoring and control of the project. WBS must be consistent with the work planned to be performed. WBS should mainly serve the project team who is the main responsible for performing the work. Therefore, project team members should be involved in developing the WBS. Each WBS item must be documented in a WBS dictionary and WBS must be a flexible tool to accommodate inevitable changes.
Verifying Scope
It is difficult to verify scope in IT projects and minimize scope changes due to the nature of IT projects. A serious problem that often occurs in IT projects is called scope creep that is the tendency for project scope to keep getting bigger. This problem must be avoided and organizations need formal acceptance of the customer for the project deliverables. The main inputs for verifying scope include project management plan, requirements documentation, and validated deliverables. The main tool is inspection and the main outputs are accepted deliverables and change requests.

Controlling Scope
Controlling scope process involves controlling changes related to the project scope. The main objective is to influence the factors that cause scope changes. The main inputs are project management plan, work performance data, and requirements documentation. The main tool is variance analysis and the main outputs include work performance measurements and change requests.

Bibliography
Lecture 5 & 6: Project Time Management
These lectures present different key concepts related to project time management.

Importance of Time Management
On-time project delivery is a challenging task for an organization. Schedule problems are commonly faced by project managers for different projects. Time is such a factor that can be measured easily and stakeholders often compare the estimated and actual time for a project. It is also the least flexible because projects have limited time to complete. Time always goes on; even if project team does not perform anything related to the project. Individual work styles, cultural differences, and work ethics play important role in project time management.

Main Processes
Six key processes are defined for project time management by PMI. These processes are as follows:

- Defining activities
- Sequencing activities
- Estimating activity resources
- Estimating activity durations
- Developing the schedule
- Controlling the schedule

Defining Activities
The first key process is to define activities for the whole project and the time available (start and end dates of project) to execute these activities is already given in project charter. The main inputs are scope statement and WBS whereas the main output is activity list. Activity list contains activity name, identifier, activity attributes and brief description of the activity. The key activity attributes are predecessors, successors, logical relationships, leads and lags, resource requirements, constraints, imposed dates, and the related assumptions. Milestones are also defined that are significant events (in project execution) with no duration. It helps to identify main activities and assists to set schedule goals and monitor progress.

Sequencing Activities
Another key process of project time management is to sequence activities; it is a process to determine the dependencies among activities and to evaluate the reasons for these dependencies. Activity dependencies are of three types i.e. mandatory / hard logic,
discretionary / soft logic, and external dependencies. The main inputs for this key process are activity list, scope statement, and milestone list.

Network Diagrams
Different techniques are available to show the sequence and dependencies of activities. Network diagrams that are also called project schedule network diagrams or PERT charts are used for this purpose. These diagrams are used for schematic display of logical relationships and sequencing of activities. Arrow Diagramming Method (ADM) or Activity On Arrow (AOA) approach and Precedence Diagramming Method (PDM) are two common types of network diagrams.

How to Create AOA Diagram
First of all, find all activities which start at Node 1 and then draw finish nodes. The next step is to draw arrow between the nodes and label it with activity name and duration. It is important to look for bursts and merge and continue it from left to right until all activities are included. It is also recommended practice that no arrows should cross on the diagram.

Comparison of AOA and PDM
PDM is used more frequently as compared to AOA method. The first key difference is that no dummy activities are shown in PDM whereas dummy activities are often used in AOA method. Dummy activities are those activities that have no duration and resources required but often needed to show logical relationship. These activities are represented with dashed arrow lines and zero duration. Another key difference is that PDM shows different dependencies among tasks whereas AOA uses finish-to-start dependency only.

Estimating Activity Resources
It is quite important to estimate resources required for the identified activities. Before estimating the duration, organization must have good estimates for other resources. The nature of project and organization affect resource estimation. The tools used for estimating resources are expert judgment, analysis of alternatives, estimating data, and project management software. It is always good for team members to have experience of similar projects. The main outputs are list of activity resource requirements, resource breakdown structure, and project document updates. Resource breakdown structure is a hierarchical structure of resources to identify resources categories and types. The example of resource categories and types is programmers (category) and Java programmers (type). Activity resources estimate also helps in other knowledge areas of project management.

Estimating Activity Durations
Estimation of activity durations is the next key process after estimating activity resources. Activity duration is defined as actual time worked on an activity plus elapsed time. It is
important to note that resources assigned to a task also affect the task duration estimate. There is also a difference between duration and effort required for the activity. For example, the duration for an activity is two days but the effort may involve 40 programmer-hours. If project scope changes then duration estimates should also be revised. It is always good to review similar projects and seek experts’ advice for estimating activity durations.

The main inputs are activity list, activity resource requirements, and resource calendars. Team should review the accuracy of duration estimates so far on the project. The availability of human resource is critical while estimating for activity durations. The main output is activity duration estimates that are always in the form of discrete number.

**Three Point Estimate**
Activity duration estimates may be of three types i.e. optimistic, most likely, and pessimistic. Optimistic estimate is the best case scenario or the minimum time required to complete the activity. The most likely estimate is the expected scenario or the average time required to perform the activity. The third type of estimate is pessimistic that is the worst case scenario or the maximum time required for an activity. These three point estimates are required for PERT analysis that is also used for estimating duration. Expert judgment is a good tool for estimating optimistic, most likely, and pessimistic estimates.

**Developing the Schedule**
Results from earlier time management processes (that are explained above) required for schedule development. It requires several iterations of processes to finalize the schedule. The main objective is to come up with the realistic project schedule. It also provides basis to monitor project progress for the time dimension. The main output of this process is the project schedule and different tools used are Gantt chart, critical path analysis, critical chain scheduling, and PERT analysis.

**Gantt Chart**
Gantt chart, sometimes called bar chart, is a standard format to show project schedule. It shows project activities, start and end dates, milestones, summary task, and task dependencies. Activities that are displayed must be same as mentioned in WBS. Different symbols are used to present activities, milestones, and summary tasks.

**Guidelines for Milestones**
Milestones should be defined by following SMART criteria i.e. milestones should be Specific, Measurable, Assignable, Realistic, and Time-framed (SMART). It is also recommended to define milestone early in the project and include them in Gantt chart. It should be kept small and frequent, and each milestone must be binary i.e. either complete or incomplete. Milestones should be carefully monitored on the critical path.
**Tracking Gantt Charts**
Tracking Gantt chart is a good tool to compare planned and actual dates of project activities. It is helpful to evaluate the project progress based on the planned schedule dates (that are also called baseline dates). The main advantage of tracking Gantt chart as compared to other similar tools is that it is easy to create and understand.

**Critical Path Method**
Critical path method (also called critical path analysis) is a network diagramming technique to predict total project duration. The critical path is the earliest time to complete the project and it is also the longest path through the network diagram. It has least amount of slack or float (the amount of time an activity may be delayed without delaying a succeeding activity or the project finish date). As several tasks done in parallel during project execution, there are multiple paths through a network diagram. The longest path or path containing critical tasks derive the completion date for the project. The way to calculate critical path is quite simple. First of all, a good network diagram should be developed along with the activities duration estimates. The next step is to add durations of all activities on each path and the longest path is considered as the critical path for the project.

People often have confusions about critical path; critical does not mean critical activities in this context. For example, growing grass can be the main activity on the critical path. It is only concerned with time dimension; critical path is not the shortest path as in some other domains. It is quite possible that more than one critical path may exist for the same project. It is also notable that critical path can change with the passage of time due to difference between planned and actual dates for different activities.

**Schedule Trade-Offs using CPM**
If the task on critical path is behind schedule, project manager and team should play proactive role to make better trade-offs. There are different technique to do trade-offs; one of these techniques is to free and total slack/float. Free slack/float is the amount of time activity can be delayed without delaying the early start date of any immediately following activities. Early start date is defined as the earliest possible time to start an activity. Total slack / float is the amount of time activity can be delayed from its early start without delaying the planned project finish date. Project managers calculate free and total slack by doing forward and backward pass. Forward pass is the early start and finish dates for each activity whereas backward pass is the late start and finish dates for each activity.

**How to Shorten Project Schedule Using CPM**
Stakeholders always want to shorten project schedule; different duration compression techniques are used for this purpose. One of these techniques is to reduce the duration of activities on the critical path by adding more resources or changing the scope. Crashing is a
technique used to do cost and schedule trade-offs. It is the greatest schedule compression technique for the least incremental cost. Fast tracking is another technique to do activities in parallel rather than in sequence. There is a possible risk of lengthening the project schedule when fast tracking is used. It is also important to update critical path data by updating the schedule with actual data, documenting revised estimates, and informed decisions based on updated plans.

**Critical Chain Scheduling**
Critical Chain Scheduling (CCS) is another technique to develop the project schedule; it is based on Theory of Constraints (TOC). It is described as any complex system at any point in time often has only one constraint that limits the ability to achieve more of its goal. It is important to identify that constraint for the improvement. CCS considers limited resources to create schedule and includes buffers to protect the completion date. CCS avoids multitasking due to the availability of limited/critical resources. CCS prefers project buffer and feeding buffers rather than individual tasks buffer.

**Program Evaluation and Review Technique (PERT)**
Program evaluation and review technique is another network analysis technique that is used when there is high degree of uncertainty about activity estimates. It uses critical path method to a weighted average duration estimate. It uses probabilistic time estimates i.e. optimistic, the most likely, and pessimistic time estimates (that are discussed above). The following formula is used to calculate average estimate

\[
\text{PERT weighted average} = \frac{\text{optimistic time} + 4 \times \text{most likely time} + \text{pessimistic time}}{6}
\]

**Controlling the Schedule**
The last key process is to control the schedule that is the part of integrated change control under project integration management. The main objectives of controlling the schedule include schedule status information, influencing the factors that cause schedule changes, and managing schedule changes. The main inputs are project management plan, project schedule, and work performance data. The main outputs of this key process are work performance measurements, change requests, and lesson learned reports. Different tools are used to control the schedule such as progress reports, schedule change control system, schedule comparison bar charts, variance analysis, and what-if scenario analysis.

**Bibliography**
Lecture 7: Project Cost Management  
This lecture presents key concepts related to project cost management.

Importance of Cost Management  
As discussed in Lecture 1, most of IT projects face cost overrun as indicated in CHAOS studies. CHAOS study highlighted 180% as the actual cost of projects as compared to estimated cost in 1994 whereas this figure was 56% by the follow-up study in 2004. Other studies also have the similar findings i.e. 33-34% cost overruns in all types of projects. US Internal Revenue Service lost $50 billion in different IT projects in 1990 and UK National Health Service lost $26 billion in 2002. These figures highlight the poor cost management and the need of better practices to manage it effectively.

Main Processes  
The following are key processes of project cost management.

- Estimating costs
- Determining the budget
- Controlling costs

Estimating Costs  
There are three basic types of cost estimates i.e. Rough Order of Magnitude (ROM) estimate, budgetary estimate, and definitive estimate. ROM estimate is like a ballpark estimate that is done very early (three or more years prior to project completion) in the project. It helps in project selection decisions and its accuracy is -50% to +100%. Budgetary estimate are used to allocate money into an organization’s budget. It is 1-2 years prior to project completion and its accuracy is about -10% to +25%. Definitive estimate is the most accurate cost estimate that is used for purchasing decisions and final project costs. Definitive estimate is performed one year or less prior to project completion and its accuracy is about -5% to +10%. Cost estimates may vary for projects in different domains and there is difference in cost estimates for external and internal resources.

Cost Estimation Tools and Techniques  
Three common techniques of cost estimation are top-down estimates, bottom-up estimates, and parametric modeling. Top-down estimates are also called analogous estimates that are based on the previous projects’ cost. Expert judgment is required for such estimates to understand that previous projects are really similar to the new one. Bottom-up estimates are based on costing for each activity in which individual work items are estimated and then summed up those estimates. Organizations often have resource cost rates to better estimate for their projects. Parametric modeling is third technique that is based on project
characteristics or parameters. It is most reliable if previous estimates were accurate using the same technique. Examples of parameters are line of code cost, language used, level of programmer expertise, and size and complexity of data involved. COCOMO II is an example of parametric modeling.

Cost Estimate Problems in IT Projects
There are various cost estimation problems in IT projects due to the distinct and complex nature of these projects. One of such key problems is that estimates are done too quickly for software projects. To avoid this problem, serious efforts are needed to understand system requirements. Another problem is the lack of estimating experience of IT professionals and they often have no reliable project data. Training of IT professionals for better cost estimation is the possible solution. The third key problem is professionals’ biasness towards cost underestimation for different projects. It should be realized that senior and junior professionals have different level of understanding and time required to perform different project tasks may greatly vary for these professionals. Another cost estimation problem is management’s desire for accurate estimates that is often difficult. Project team should defend the cost estimates in front of top management so that they can also negotiate with customers.

Determining the Budget
This key process is about allocating the project cost estimate to individual work items over time. The main inputs include activity cost estimates, project schedule, and resource calendars. The main goal is to develop cost baseline and it is not possible without well-established process. There are different budget categories such as travel and depreciation that are included in the budget. All the information given in the budget is used for legal and tax purposes and provides project funding information.

Controlling Cost
It is the key process related to monitoring of cost performance and often revised cost baseline is prepared during this process. The main inputs for controlling cost are project management plan, project funding requirements, and work performance data. Change control system plays the role by conducting performance review meetings to control the cost and other key constraints. Different performance measurement techniques are used for this purpose.

Earned Value Management
Earned value management is a project performance measurement technique that integrates scope, time, and cost data. It provides the comparison of actual information and baseline by calculating three values for each or summary activity. The Planned Value (PV) is the approved cost estimate for an activity whereas the Actual Cost (AC) is the total direct and indirect costs. Earned Value (EV) is the estimated value of the work completed based on the original planned cost and the rate of work completed to date. Different other values such as Rate of
Performance (RP), Cost Variance (CV), and Schedule Variance (SV) can be calculated based on these three values (PV, AC, and EV). Rate of Performance (RP) is the ratio of actual work completed to the percentage of work planned.

Cost Variance (CV) is calculated as earned value minus the actual cost. If CV is negative, it indicates cost overrun whereas the positive value means cost spent is less than planned one. Schedule Variance (SV) calculates difference between the earned value and the planned value. If it is negative then it means time is taken longer than the planned and the positive value indicates project activities done are ahead of planned dates.

Cost Performance Index (CPI) is the ratio of earned value to actual cost. It helps to estimate the projected cost of project completion. If it is 100% then it means the planned and actual costs are same. If CPI is less than 100%, it shows the cost overrun and if it is greater than 100%, it highlights that the project is under budget. Schedule Performance Index (SPI) is the ratio of earned value to planned value. It provides estimate the projected time of project completion based on the current progress. If SPI is 100% then the project is on schedule; the value less than 100% shows the project is behind the schedule and the value greater than 100% indicates that the project is ahead of schedule.

Estimate at Completion (EAC) is calculated by CPI for estimating cost to complete the project based on the performance to date and SPI for estimating time to complete the project based on the performance to date. Budget at Completion (BAC) is the term used for the original total project budget.

Bibliography
Lecture 8: Project Quality Management and Project Communication Management

This lecture presents brief overview of project quality management and key concepts related to project communication management.

Importance of Quality Management

A study was conducted by Standish group in 2008 to estimate the cost of downtime for different types of IT systems/applications. Security trading applications cost $73,000, ERP systems cost $14,800, and order processing applications cost $13,300 per minute. It shows the importance of quality; if errors are not fixed at development time or systems are deployed without proper quality assurance activities, it costs a lot later on.

Project Quality Management

ISO defines the quality as “the totality of characteristics of an entity that bear on its ability to satisfy stated or implied needs” or “the degree to which a set of inherent characteristics fulfill requirements”. It means quality is about the conformance to requirements and the fitness of product for use. In general, customer decides about the quality that is major project requirement/constraint along with scope, time, and cost.

Project quality management deals with the practices required to develop quality product. The following are the key processes of project quality management:

- Planning quality
- Performing quality assurance
- Performing quality control

Planning quality is the process to incorporate relevant quality standards in quality management plan. The main outputs of this process are quality management plan, quality metrics, and quality checklists. Performing quality assurance means to evaluate project performance periodically and the main outputs are updated quality management plan and change requests. Performing quality control process is to conduct monitoring of project results and its main outputs are quality control measurements, validated changes, and validated deliverables.

Project Communication Management

Communication failure is considered the greatest threat for IT projects success and in general, IT professionals are not good in communication skills. The main objective of project communication management is to ensure timely and appropriate generation, collection, dissemination, storage, and disposition of project information. The main processes of project communication management are as follows:
• Identifying stakeholders
• Planning communications
• Distributing information
• Managing stakeholder expectations
• Reporting performance

**Identifying Stakeholders**
The first main process is to identify project stakeholders by formal and informal communication networks. Once stakeholders are identified then stakeholder register is created and stakeholders management strategy is developed. Stakeholders analysis is conducted to decide different communication requirements of project stakeholders. Stakeholder register describes their basic information whereas stakeholders management strategy describes their interest in the project and his/her influence and how to deal with them.

**Planning Communications**
Communication management plan is the main output of this process; it serves as the guide for communication. It describes stakeholder communication requirements, information to be communicated, information receiver and producer, and suggests methods for conveying information. It also provides guidelines about the frequency of communication, escalation procedures for resolving issues, and revision procedures to update communication plan. It also provides a glossary of common terminology to avoid any confusion related to communication requirements.

**Distributing Information**
The main objective of distributing information is to provide information to the right people at the right time. It involves the effective use of technology, and formal and informal methods. Project manager and team require good skills for effective and timely manner distribution of important information. They need to understand group and individual communication needs and this fact that people are not interchangeable parts. It is their job to select the appropriate communications medium for different stakeholders. Face-to-face interaction is considered the most effective communication channel and it is good to have short and frequent meetings. It is also important to understand the differences of geographic location and cultural background. Similarly, project manager and team need to set the stage for communicating bad news as top management and customer do not like surprises.
Managing Stakeholders
Project manager has the key role in managing stakeholders as he often has to convey the changes related to triple constraint. Project sponsors usually rank these constraints and provide guidelines to balance it. Expectations management matrix is prepared that contains list of success measures, priorities, expectations, and guidelines. It helps to manage issues raised during the project execution.

Reporting Performance
Reporting is monitoring and controlling activity that helps to update stakeholders about the project progress. Performance Reports (also called status reports) are prepared to inform that where the project stands. Progress reports are prepared to share what project team has done. Status review meetings are periodically conducted to evaluate the project progress. It also helps to predicts future project status and progress and organization may use earned value management technique for this purpose.

How to Improve Project Communications
Project managers should use communication skills to manage conflict among team members or different stakeholders. They need to develop better communication skills for this purpose. It is also good to have effective meetings with pre-defined agenda and limited time required to conduct meetings. Another recommendation is to make effective use of collaborative tools and different templates for project progress/reports.

Bibliography
**Lecture 9: Project Risk Management**

This lecture presents key concepts related to project risk management.

**Importance of Risk Management**

Risk management is the art as well as science and it is a frequently overlooked and underestimated aspect of project management. If it is done, significant improvement can be achieved to meet project objectives. Good risk management often goes unnoticed as it avoids the unlikely events to occur and precautionary measures are taken for the same. A study was conducted with 38 organizations related to engineering and construction, telecommunications, information systems/software development, and high-tech manufacturing. They estimated the maturity level in different knowledge areas for these organizations. These organizations had lowest maturity level in risk management and information systems/software development organizations had lowest among all organizations of different domains.

**Main Processes**

Project risk management deals with the activities related to possible risks for different project factors. It is important to note that risks are not always negative but positives risks are also possible that have good impact on the project. The main processes of project risk management are as follows:

- Planning risk management
- Identifying risks
- Performing qualitative risk analysis
- Performing quantitative risk analysis
- Planning risk responses
- Monitoring and controlling risk

**Planning Risk Management**

Planning risk management is the key process that is about how to approach and plan for risk management activities. The main output of this process is risk management plan. Planning meetings at early stage of project are conducted to plan it. It includes risk management policies, risk categories, lesson-learned reports from past projects. It is also important to review risk tolerance of different stakeholders. It is recommended to clarify roles and responsibilities, and prepare budget and schedule estimates for risk-related activities. Level of information details can vary depending on the nature and size of project.
Sometimes additional plans are prepared; Contingency plans describe predefined actions if the risk occurs. The example of such risk is the unavailability of new software for the project. Fallback plans are developed to address high impact risk. Contingency reserves/allowances are reserved by organization or project sponsor to reduce the risk.

**Risk Categories**
Risks are categorized into many types; few common categories are market, financial, technology, people, and structure/process risks. Market risks are related to new product or service whereas financial risks are about the affordance of an organization to undertake the project. Technology risks are related to technical aspect of project and people risks are about the availability of skilled people. Structure/process risks deal with the possible changes in business processes.

**Identifying Risks**
Identifying risks is another key process; different tools and techniques such as brainstorming, Delphi technique, interviewing, SWOT analysis, and diagramming techniques are used for this purpose. Risk register is the key output of this process that describes the information about the possible risks. This information includes identification number, risk ranking, risk title, risk description, risk category, root cause, triggers, potential responses, risk owner, probability, impact, and status of the risk.

**Performing Qualitative Risk Analysis**
There are two ways to analyze the possible risks i.e. qualitative and quantitative. Qualitative risk analysis is mainly based on expert judgment to assess likelihood and impact of identified risks. There are different techniques that can be used for qualitative risk analysis such as probability/impact matrix, top ten risk item tracking, risk management review, and watch list.

**Performing Quantitative Risk Analysis**
Quantitative risk analysis follows qualitative risk analysis in general. Data gathering is an important aspect of quantitative risk analysis. The main techniques used for such analysis are decision trees (e.g. expected monetary value), simulation (e.g. Monte Carlo analysis), and sensitivity analysis. Risk registers are updated after both types of analysis either qualitative or quantitative.

**Planning Risk Responses**
It is quite important to plan responses for the possible risks by developing options and defining strategies. Risk avoidance is a strategy that focuses on the elimination of cause if possible. Risk acceptance is about accepting the consequences of risk occurrence. Risk transference is a technique to shift the consequences of risk occurrence to other party. Risk mitigation is another technique that is mainly focused on reducing the impact of risk.
There are different strategies for positive risks; one of those strategies is the risk exploitation to make sure the positive risk happens. Risk sharing is another strategy to share the ownership of risk with other party. Risk enhancement tries to maximizing the opportunity whereas risk acceptance is the laziest approach that involves no extra effort for it.

**Monitoring and Controlling Risks**
The monitoring and controlling process is always main process for different knowledge areas. Monitoring and controlling risks means monitoring the execution of other risk processes. It also provides risk awareness and deals with the redistribution of resources and workarounds (unplanned responses). Different activities that are performed are risk reassessment, risk audits, variance and trend analysis, technical performance measurements, reserve analysis, and status meetings. It is always important to update risk register based on the findings.

**Bibliography**
Lecture 10: Software Design
This lecture discusses the basic concepts related to software design.

Design in Software Engineering Context
Design is all about the creativity; design is the representation/model/development of the system or software to achieve goals within constraints. Goals may be the stakeholder requirements, business needs, and technical considerations. Constraints are often related to time, cost, quality, and technology. Software design includes architecture, components, data structures, and interfaces.

Roman architecture critic Vitruvius said, “Well-designed buildings were those which exhibited firmness, commodity, and delight”. The same is equally applicable to good software design; it should be bug-free (firmness), suitable for the intended purpose (commodity), and pleasurable user experience (delight).

Software design is the technical kernel of software engineering and independent of software process model. It is last modeling activity before coding and it always focuses on quality. It is the translation of stakeholder requirements into a software product. If software development is done without design, there is a risk of rework that costs a lot to organization.

Design Process and Quality Guidelines
Design is an iterative process that provides different levels of abstraction. There must be the criteria to evaluate software design. A good software design must implement all explicit and implicit requirements. It must be readable and understandable and it should provide the complete picture related to the system.

Software design should exhibit recognizable design patterns, good design characteristics, and implementation in an evolutionary fashion. It should be modular and should contain distinct representations of data, architecture, interfaces, and components. Software design should lead to appropriate data structures. Software design should lead to independent components and it should reduce connections complexity. It should be derived using a repeatable method and represented using understandable notation. Software design should reflect the application of design knowledge rather than by chance achievement.

Software design is continuously evolving with the passage of time; in earlier days of software design the approach used was the top-down approach. It shifted to procedural aspects / structured programming for many years. Later on, object-oriented approach was introduced; it is still commonly used for software design. Software architecture or design patterns are introduced in recent years along with aspect-oriented approach, model-driven and test-driven development.
Design Concepts
Design concepts help to set criteria for partitioning (software components/modules) and separation of data structure details from conceptual representation. These concepts provide criteria for quality and are followed in traditional as well as object-oriented software development. The main design concepts are briefly described in the following subsections.

Abstraction
Abstraction provides modular solution of the problem under consideration; in other words, problem-oriented technology is coupled with implementation-oriented technology. There are two main types of abstraction i.e. procedural and data abstraction. Procedural abstraction provides the sequence of instructions and data abstraction provides the collection of data/data objects. Consider the phrase, open the door, ‘open’ presents the procedural abstraction and ‘door’ represents data abstraction.

Architecture
Software architecture represents the overall structure of the software and it provides conceptual integrity of software. It is all about major system elements and their interaction. There are different architecture models e.g. architectural patterns solve common design problems. Structure models are the organized collection of system components whereas framework models are repeatable frameworks for specific applications. Dynamic models show behavioral aspects of program architecture and process models represent the design of business or technical process. Functional models are used to show functional hierarchy of a system. Architectural Description Languages (ADL) are used to show software architecture and the examples of such languages are Architectural Analysis and Design Language (AADL), C2, and Darwin.

Patterns
Pattern is “a design structure that solves a particular problem within a specific context”. It helps to determine whether it is applicable to the current work. Whether it can be reused and whether it helps to develop more similar patterns. Patterns will be discussed in details in the coming lectures.

Modularity
Modularity is defined as “the single attribute of software that allows a program to be intellectually manageable”. It helps in separation of concerns and software is divided into different components that are often called modules. If software is monolithic (single module) then it has great complexity. Balanced approach is required for better software design.
Information Hiding
Information contained by a module should not be accessible to other modules. Independent modules lead to effective modularity; abstraction and information hiding complement each other. Information hiding is also helpful for software modification.

Functional Independence
Different design concepts such as abstraction, separation of concerns, modularity, and information hiding lead to functional independence. A “single-minded” function that has limited interaction with other modules provides functional independence. It helps in maintainability; cohesion and coupling are two related characteristics of functional independence. Cohesion is an indication of the relative functional strength of a module whereas coupling is an indication of the relative interdependence between modules.

Refinement and Aspects
Refinement is top-down design strategy or it can be defined as a process of elaboration. Abstraction and refinement are complementary concepts. Aspects are defined as different requirements or concerns and crosscutting concerns / aspects are implemented as a separate module.

Refactoring
Refactoring is a reorganization technique to simplify component design without changing its function or behavior. It improves internal structure without altering external behavior of the module. It focuses on un-used design elements, inefficient or unnecessary algorithms, poor/inappropriate data structures, and any other design failure. It helps in easier integration, testing, and maintenance.

Design Classes
Design classes may be categorized into different types; user interface class is the visual representation of software. Business domain class presents business logic whereas process class contains lower level business abstraction. Persistent class is about the data storage whereas system class deals with software management.

A well-defined class should be complete and sufficient; it means that the complete encapsulation of all attributes and methods. Primitiveness is another characteristic of well-defined class; such class focuses on one service. Class should exhibit high cohesion and low coupling; high cohesion is single mindedness and small set of responsibilities for the class whereas low coupling is about the acceptable minimum level of collaboration.

Bibliography
Lecture 11 & 12: User Interface Design
These lectures discuss user interface design.

Importance
Almost all software products require human interaction; either it is digital music player or weapon control system. The successful product must have good usability and well-designed interfaces to do work effortlessly. If product usability is poor, it causes frustration and users give up. Usability was not dominant for first three decades of software design. The main objective is that technology conform people rather than people are required to conform to technology.

Good quality software product should be useful that is it should accomplish what is required to do. It should be usable i.e. people can perform the intended task easily, naturally, and without the danger of error. It should be attractive, engaging so that people want to use it.

The Golden Rules
T. Mandel (in 1997) proposed the following three golden rules for user interface design which are discussed in the following subsections.

- Place the user in control
- Reduce the user’s memory load
- Make the interface consistent

Place the User in Control
Define interaction modes in a way that does not force a user into unnecessary or undesired actions. Interface mode is defined as the current state of the interface e.g. spell-check in a word-processor menu. User should be able to enter or exit the mode with no or little effort. User interface should provide flexible interaction; users preferences are different e.g. some may prefer keyboard whereas some other like mouse. Every action is not supported by every interaction mechanism.

Allow user interaction to be interruptible and undoable; user should be able to interrupt even in a sequence of actions, without losing the work done. User should also have “undo” option for this purpose. Streamline interaction as skill levels advance and allow the interaction to be customized. Advanced users may have the option to customize the interface.

Hide technical internals from the casual user; user should not be aware of technical details. User should not work at “inside” level e.g. OS commands from within other software. Design for direct interaction with objects that appear on the screen. Users prefer direct manipulation therefore virtual objects should behave like physical objects e.g. stretching the object.
Reduce the User’s Memory Load
Reduce demand on short-term memory as complex task may lead to memory load. Interface should reduce the requirement to remember past actions, inputs, and results. Establish meaningful defaults that should make sense for average user. User preferences should be available with “reset” option.

Define shortcuts that are intuitive and easy to remember e.g. Ctrl + C for copy. The visual layout of the interface should be based on a real-world metaphor. The similar terms as of real world and well-understood visual cues should be used e.g. printer icon. Disclose information in a progressive fashion i.e. interaction should be organized with different level of abstractions e.g. printer preferences.

Make the Interface Consistent
Allow the user to put the current task into a meaningful context; complex layers of interaction should be provided with multiple screen images. Indicators should be provided so that user should know the context and alternatives should also be known. Maintain consistency across a family of applications i.e. a family should implement same design rules e.g. Microsoft office. If past interactive models have created user expectations, do not make changes unless there is a compelling reason to do so. There are always de facto standards and change will cause confusion to users.

User Interface Analysis and Design
Interface analysis and design models are divided into four different models i.e. user model, design model, user’s mental model / system perception, and implementation model. These models may differ significantly and there is always need to derive consistent representation of the interface.

Types of User
Users are divided into three types based on their knowledge and experience. Novice users are new users who have no syntactic knowledge of the system and have little semantic knowledge of the application / computer usage. Knowledgeable and intermittent users have reasonable semantic knowledge with relatively low recall of syntactic knowledge. Knowledgeable and frequent users have good syntactic and semantic knowledge; they are often called power users.

User’s Mental Model
User’s mental model is the image of the system that end users carry in their heads. It is about the system perception that how it will perform certain tasks. The accuracy of description depends on user’s profile.
**Implementation Model**
Implementation model is about the look and feel of the interface. All supporting information e.g. help files should be prepared. User’s mental model and implementation model should be synchronized. It is quite important to understand the user and tasks for better models.

**Analysis and Design Process**
It is an iterative process that includes interface analysis and modeling, interface design, interface construction, and interface validation. Interface analysis focuses on user profiles and requirements elicitation is done for this purpose. The key information related to each user profile includes skill level, business understanding, and general receptiveness to the new system. Task analysis is another important activity done during this process to have better idea of user tasks. Physical work environment concerns are also important such as physical location, user position while interacting with the system, and other factors (e.g. noise). Analysis model is developed based on the above mentioned activities.

Interface design is to come up with the set of interface objects, actions, and their screen presentations. User should perform all the intended functions with no or little effort. The key focus of interface design is the usability aspect of interface. Interface construction is about the creation of a prototype to reflect usage scenarios. There are different tools available that help in prototype creation.

Interface validation is all about the user’s acceptance to make sure that every user task is properly implemented. It is also confirmed that interface accommodates all task variations and it is easy to use and learn.

**Interface and User Analysis**
Interface analysis is the way to understand the problem before design. It is quite important to understand the end users who will interact with the system and the tasks that end user must perform. Interface analysis also helps to understand the content that is to be presented and the environment in which system will be deployed.

The term “user interface” justifies the focus on user; there are always different mental images of users about the system based on their experience. These mental images are often different from the design model (software developer’s perspective). Interviews should be conducted to understand users’ mental images and it is recommended to get input from other departments such as sales, marketing, and support.

**Task Analysis and Modeling**
Different techniques and methods are used to document the outcomes of task analysis and modeling. Use cases are developed to understand how an actor (user) interacts with the system. Tasks are elaborated to understand the user’s requirements related to the new system.
Object elaboration is the process to define physical objects used that may help to design related virtual objects. Workflow analysis helps to understand work process often involving several users that sometimes presented in the form of hierarchy.

**Analysis of Display Content**
Users may require different display formats such as text and images; it is important to properly analyze these requirements. There are some guidelines that should be followed during the interface design for displaying contents. For example, different types of data should be assigned to consistent geographic locations on the screen. Users should have an option to customize the screen location for content. Content should have proper on-screen identification for the easy understanding. If a large report is to be presented, it should be partitioned in a way to help users in understanding easily.

**Analysis of the Work Environment**
The purpose of work environment analysis is to provide user friendly location which has proper lighting, good display height, and easy keyboard access. These factors may vary in different environments e.g. factory floor and airplane cockpit environment demands different considerations. It is also important to understand the workplace culture for which system is to be designed.

**Design Issues**
System requirements may vary depending on the organization and the domain. Despite of it, there are some common design issues that need to be considered. System response time is often the primary complaint by users; system response time should be as least as possible. User help facility provided by the system is also important factor that includes help for all functions, the way to request help, and its presentation. Presentation is also about how to return to normal interaction and how help information will be structured.

Another important design issue is error information handling; it is a kind of “bad news” for users. It is often observed that users get useless or misleading information in case of any error. Error message should be understandable and should provide constructive advice. It should indicate negative consequences and use other cues such as audio or visual cues. Error messages should be “nonjudgmental” in the sense that user should not be blamed for the error.

Menu and command labeling is another design issue; command line interfaces are still preferred by power-users so commands should also be properly labeled. It is recommended to assess either every menu option would have a corresponding command or not. If commands are required then what will be the form of that commands. The commands should be easy to learn and remember and it should be customized or abbreviated. Similarly, menu labels should
be self-explanatory within the context and the submenus should be consistent with the function implied by a master menu item.

Application accessibility is another important design consideration; it should be easy to access for people with special needs. There are different accessibility guidelines available that should be considered. Internationalization is another quite important design issue to target the global market. Software interface design should support different languages and cultures so that users can customize/localize it. Again, different guidelines are available to support internationalization or localization.

**Web Application Interface Design**

Web applications are different as compared to traditional software applications and users are more concerned with its interface design. Users should know about their current location and options available to execute. User should be facilitated with easy navigation within the application and it may be done by providing a map.

Anticipation, communication, consistency, controlled autonomy, efficiency, flexibility, latency reduction, learnability, metaphors, readability, and visible navigation are important interface design factors that should be properly addressed. Various guidelines are also available for web application interface design such as Nielsen and Wagner guidelines.

**Bibliography**

Lecture 13: Pattern-Based Design
This lecture presents the key concepts related to pattern-based design.

Design Patterns
C. Alexander defines pattern in this way, “each pattern describes a problem that occurs over and over again in our environment and then describes the core of the solution to that problem in such a way that you can use the solution a million times over without ever doing it the same way twice.” This definition focuses on three important aspects of a pattern i.e. context, problem, and the solution. Design patterns promote the concept of reuse rather than reinventing the wheel each time.

Effective design pattern solves a problem and provides the solution (not principles/strategies). It has proven track record based on proven concept and the solution is not obvious (indirect solution). It describes a relationship between different system components and provides mechanisms for such relationship. It has a significant human component that deals with aesthetic and utility aspect.

Types of Patterns
There are various types of design patterns at different levels of software design. Architecture patterns help in software/system structure e.g. pipes and filters, adapter, proxy, and bridge patterns. Data patterns are data-related patterns and the examples of these patterns are data replication and master-master replication. Component patterns (also called design patterns) deal with subsystems and their examples are add another, build an expression, and slide down patterns. Interface design patterns focus on user interface and the examples of such patterns are breadcrumb, card stack, wizard, and simple search. Webapp patterns are mainly for web applications and examples are advanced search, search area, and search tips.

There are some other patterns for object-oriented design; the main types of these patterns are creational patterns (e.g. singleton), structural patterns (e.g. composite) and behavioral patterns (e.g. command).

Describing a Pattern
There are different characteristics of patterns that should be described properly. The main characteristics are pattern name, problem, motivation, context, forces, solution, intent, collaborations, consequences, implementation, known uses, and other related patterns.

User Interface Design Patterns
In this section, different user interface design patterns are briefly described.
**TopLevelNavigation**
This pattern belongs to “whole UI” category; it is used when a site or application implements a number of major functions. It provides a top-level menu, often coupled with a logo or identifying graphic, that enables direct navigation to any of the system's major functions. Each function/content name represents a link to the appropriate function or content.

**CardStack**
This pattern is of category “page layout”; it is used when a number of specific sub-functions or content categories related to a feature or function must be selected in random order. It provides the appearance of a stack of tabbed cards, each selectable with a mouse click and each representing specific sub-functions or content categories. For navigation, a mouse click on a tab causes the appropriate card to appear. Navigation features within the card may also be present, but in general, these should initiate a function that is related to card data, not cause an actual link to some other display.

**Fill-in-the-Blanks**
It belongs to “forms and input” category; it allows alphanumeric data to be entered in a text box. For navigation, a text or graphic indicator is used that initiates validation and processing.

**SortableTable**
It displays a list of records that can be sorted by selecting a toggle mechanism for any column label. It belongs to “table” category; each column header initiates a sort on all records for navigation. No other navigation is provided, although in some cases, each record may itself contain navigation links to other content or functionality.

**BreadCrumbs**
This pattern belongs to “data and manipulation” category; it provides a full navigation path when the user is working with a complex hierarchy of pages or display screens. For navigation, any of the entries within the bread crumbs display can be used as a pointer to link back to a higher level of the hierarchy.

**Edit inPlace**
This pattern is a part of “navigation” category; it provides simple text editing capability for certain types of content in the location that it is displayed. There is no need for the user to enter a text editing function or mode explicitly.

**SimpleSearch**
It belongs to “searching” category; it provides the ability to search a website or persistent data source for a simple data item described by an alphanumeric string. Each entry in the list of hits represents a navigation link to the data referenced by the entry for the navigation purpose.
Wizard
It is a part of “page element” category; it takes the user through a complex task one step at a time, providing guidance for the completion of the task through a series of simple window displays. Forward and back navigation allows the user to revisit each step in the wizard process.

ShoppingCart
It belongs to “e-commerce” category; it provides a list of items selected for purchase. It contains ability to proceed with shopping or go to checkout.

ProgressIndicator
It provides an indication of progress when an operation takes longer than n seconds. It often contains a button that allows the user to pause or cancel processing.

Bibliography
Lecture 14: Web Application Design
This lecture presents the key concepts related to web application design.

Web Application Quality
Different quality models are proposed for web application quality; one of those models is Olsina et al. criteria (1999). Usability, functionality, reliability, efficiency, and maintainability are key factors of these criteria. Usability may be measured with factors such as global site understandability, online feedback and help features, interface and aesthetic features, and special features. Searching and retrieving capability, navigation and browsing features, and application domain-related features are the factors of functionality. Reliability is further categorized into correct link processing, error recovery, and user input validation and recovery. Efficiency of a web application may be assessed by response time performance, page generation speed, and graphics generation speed. Ease of correction, adaptability, and extensibility are used to evaluate the maintainability of web application.

Offutt (2002) proposed some additional factor for web application quality; it includes security, availability, scalability, and time-to-market. Security is an important concern of web applications that deal with sensitive information of users. Web application must be available 24/7/365 and for different web browsers. It should be scalable to support more users when required. Time-to-market is also an important factor from business point of view as organizations have many business competitors.

Content Quality
Content quality is also an important consideration of web application; Tillman (2000) criteria (which are briefly described in this section) provide guidelines for it. The scope and depth of content should be easily determined to ensure that it meets the user's needs. The background and authority of the content's authors should be easily identified. It should be easy to determine the currency of the content, the last update, and what was updated. The content and its location should be stable (i.e., the referenced URL should remain same).

There are some further considerations e.g. content should be credible. It is also important to have unique content i.e. the web applications should provide some unique benefit to those who use it. Content should be valuable to the targeted user community and it should be well organized, indexed, and easily accessible.

Design Goals of Web Application
Simplicity is the first goal of web application design; it promotes “all things in moderation” concept. Web application should be informative but compact and the use of colors should be sensible. It should also provide simple navigation for the intended users. Another design goal is consistency; web application design should be consistent for text formatting, font style, and
color scheme. Identity is another design goal that web applications should truly represent the application domain for which it is intended.

Robustness is an important design goal that is implicit promise to users of web application. Navigability of web application should be intuitive; users can easily predict navigation. Visual appeal of web application is about look and feel of the interface. Compatibility is another important design goal; web applications must be compatible with different environments.

**Interface Design**
As described earlier, interface design of web applications is quite important. It is important to establish a consistent window into the content and functionality provided by the interface. It should guide the user through a series of interactions with the web application. There should be proper organization of navigation options and content available to the user. Interaction mechanisms that include navigation menus, graphic icons, and graphic images should be properly defined.

**Aesthetic Design**
Aesthetic design is also called graphic design; it mainly deals with layout issues. Designer should not be afraid of white spaces; it means white spaces should be wisely used to organize contents. The important content should be emphasized and layout elements should be organized from top-left to bottom-right. It is recommended to group navigation, content, and function geographically within the page. Scroll bar should be avoided for the key content and it is always recommended to consider resolution and browser window size when designing layout.

**Navigation Design**
Navigation design is about the layout of navigation elements for web application. Navigation Semantic Unit (NSU) is an important concept of navigation design. It is defined as "a set of information and the related navigation structures that collaborate in the fulfillment of a subset of related user requirements". Ways of Navigating (WoN) and Navigational Nodes (NN) represent a set of navigation elements. There are different approaches for navigation design that include individual navigation link, horizontal navigation bar, vertical navigation column, tabs, and site maps.

**Bibliography**
Lecture 15: Software Quality
This lecture discusses the basic concepts of software quality.

Quality
Quality is a multi-aspects concept; we may view quality in different perspective and each perspective would be different. Transcendental view of quality defines that it is difficult to explicitly define the quality but it is quite easy to recognize the quality. User view of quality is about the conformance to end user’s specific goals. Manufacturer’s view of quality deals with the product specification i.e. either product specifications are fulfilled or not. Product view of quality is about the inherent characteristics of the product. Value-based view of quality is more related to cost i.e. how much a customer is willing to pay for quality.

Software Quality
An effective software process is required to product good quality software that has value for the developer as well as user. Infrastructure support is mandatory for an effective software process that includes change control and technical reviews. The ultimate objective is useful product that meets explicit and implicit requirements. Good quality product should be reliable and error-free. If product is reliable then less maintenance effort is required and it makes the business process efficient.

Software Quality Models
Different software quality models are proposed to evaluate the quality of software product. This section briefly describes a few important quality models.

Garvin’s Quality Dimensions
Garvin’s quality model describes key quality factors that should be present in any product; it is not specific to software product. Performance quality, feature quality, reliability, conformance, durability, serviceability, aesthetics, perception, and “soft” look of quality are the quality dimensions of Garvin’s model. These dimensions are equally applicable to software quality e.g. performance quality of software is about the response time of software that should be as low as possible. Similarly, reliability of software reflects error-free software product to use. Aesthetics factor in the context of software product is about the interface design that should be pleasant for users.

McCall’s Quality Factors
McCall’s quality model is one of the oldest software quality model; it divides software quality into three main categories i.e. product operation, product transition, and product revision. Correctness, reliability, usability, integrity, and efficiency are quality factors related to product operation. Product transition may be evaluated by portability, reusability, and interoperability.
Similarly, product revision may be assessed by maintainability, flexibility, and testability. This model provided the basis for many other model proposed for software quality.

**ISO 9126 Quality Factors**
ISO 9126 quality model is also based on McCall quality model; it divides software quality into six main quality factors. These factors are functionality, reliability, usability, efficiency, maintainability, and portability. Functionality includes suitability, accuracy, interoperability, compliance, and security. Reliability aspect of software quality can be measured by maturity, fault tolerance, and recoverability. Understandability, learnability, and operability help to measure software usability. Efficiency of software is related to time behavior and resource behavior. Maintainability of software quality has analyzability, changeability, stability, and testability as sub-factor to assess it. Portability is an important aspect of software quality that include adaptability, installability, conformance, and replaceability as its sub-factors.

**Targeted Quality Factors**
Intuitiveness, efficiency, robustness, and richness are proposed as targeted quality factors for software products by Brooks (2003). Intuitiveness is mainly about the interface design that is the interface layout should be conducive to easily understand. Interface operations should be easy to locate and initiate. Efficiency is about the information and operations that can be located and initiated. The sequence of operations (or data input) should be performed with an economy of motion. Robustness is the software quality that is about software response against different error conditions. Software product should recognize the error if data at or just outside prescribed boundaries. Similarly, it is also about software operation that should continue failure or degradation?Richness of software is about the facility to customize interface to the specific needs of a user.

**Software Quality Dilemma**
In this section, it is briefly discusses that what are the reasons of poor software quality. The foremost reason is that organizations often compete in the market therefore “good enough” software is delivered with even “known” bugs. It is a kind of short cut to beat the competitors but it may cause legal penalties in some cases due to bugs. Cost of quality is another reason; quality is not for free; it includes the cost of conformance and nonconformance. It may be divided into cost incurred before and after the deployment of software. Organizations have limited resources therefore software quality related activities are often limited.

Low quality software increases risks for user and developer; sometimes these risks may be very serious in nature. Some of these risks are associated with the negligence of developers and may lead to liability. Security is another important quality aspect that may be ignored and it may have serious consequences in today’s computing. Organization management’s attitude towards
software quality also plays important role; cost and schedule estimates should accommodate quality activities for better software product.

**Achieving Software Quality**
If organization wants to achieve better software quality, they need good infrastructure to support the following:

- Software engineering methods
- Project management techniques
- Quality control
- Quality assurance

**Bibliography**
Lecture 16: Review Techniques
This lecture describes the key concepts of review techniques.

Software Reviews
Software reviews are kind of filters for software process to identify the possible errors in software. People are good at catching others’ errors rather than their own errors. Therefore, software reviews are conducted by others (software developers are often part of that team). The key objectives are to point out needed improvements, confirm those parts that are OK, and achieve technical work of uniform quality without reviews.

Cost Impact of Software Defects
Defect and fault are synonymous terms in case of software applications. The primary objective of software reviews is to find errors before software deployment. It helps in early discovery of errors and to stop their propagation in next step of software development. Different design activities introduce 50-65% of all errors and review techniques are 75% effective to uncover design flaws. It leads to reduced cost at later stages to identifying and correcting errors.

Review Metrics and Their Use
Each action requires dedicated human effort and project effort is finite. Therefore, we need metrics to assess effectiveness of our efforts. The following are review metrics that can be used to assess our review activities.

Preparation effort ($E_p$): Number of person-hours prior to actual review

Assessment effort ($E_a$): Number of person-hours required for actual review

Rework effort ($E_r$): Number of person-hours to correct errors uncovered during the review

Work product size (WPS): Size of work reviewed e.g. number of UML models

Minor errors found ($Err_{minor}$): Number of minor errors found

Major errors found ($Err_{major}$): Number of major errors found

Analyzing Metrics
The above mentioned metrics helps to analyze review efforts in the following way:

Total effort to review ($E_{review}$) = $E_p + E_a + E_r$

Total errors found ($Err_{tot}$) = $E_{minor} + E_{major}$

Error density – number of errors found per unit of work reviewed

Error density = $Err_{tot}/WPS$
Example: 18 UML diagrams, 32 pages document, 18 minor and 04 major errors

Error density = 22/18 = 1.2 errors per UML diagrams OR

22/32 = 0.68 errors per page

Different work products e.g. requirements specification document or code of software component are reviewed. The percentage of errors uncovered for each review is computed against the total number of errors for all reviews. Error density for each work product is computed. When all reviews are conducted, average values indicate the errors in new documents.

**Cost Effectiveness of Reviews**

It is difficult to measure the cost effectiveness of reviews but still can be estimated. Consider an example that there are 0.6 errors per page; 4 person-hours to correct minor error whereas 18 person-hours to correct major error. Minor errors occur about six times more frequently as compared to major errors based on the review data. Requirements-related error needs 6 person-hours to correct. Same error requires 45 person-hours if uncovered during testing.

Effort saved per error = \( E_{testing} - E_{reviews} \) i.e. 45 - 6 = 39 person-hours.

**Reviews: A Formality Spectrum**

Reviews are mainly divided into formal and informal review. The level of formality depends on different factors such as product, time, and people. There are four common characteristics of reference model (for reviews) i.e. roles, planning and preparation for the review, review structure, and follow-up.

**Informal Reviews**

Informal reviews may be simple desk check, casual meeting, or review-oriented aspects of pair programming. The effectiveness of informal review is considerably lower. There is no advance planning/preparation, agenda, or follow-up for informal reviews. These reviews are still good to uncover errors that may propagate in later stages. Simple review checklist may be used to improve the efficiency of informal reviews.

**Example: Review Checklist**

Work product: prototype

Reviewers: designer and colleague

Is the layout designed using standard conventions? Left to right? Top to bottom?

Does the presentation need to be scrolled?

Are color and placement, typeface, and size used effectively?
Are all navigation options or functions represented at the same level of abstraction?

Are all navigation choices clearly labeled?

**Formal Technical Reviews**
The main objectives of formal technical reviews are: 1) to uncover errors in function, logic, or implementation, 2) to verify that the software under review meets its requirements, 3) to ensure that it is according to predefined standards, 4) to achieve that it is developed in a uniform manner, and 5) to make project more manageable. It also provides training ground for juniors and promotes backup and continuity. There are two types of formal reviews i.e. walkthroughs and inspections. Inspections are more formal than walkthroughs.

Review meetings are conducted with three to five people in total. Participants should not need more than two hours (for each person) of advance preparation for the meeting. Meeting duration should be less than two hours and its focus should be on specific and small part of overall software. Normally, producer informs project leader about the work product and project leader contacts review leader for the meeting. Review leader is responsible for the rest of all arrangements.

Meeting is attended by the review leader, all reviewers, and the producer. One reviewer serves as recorder and meeting starts with an introduction of the agenda and the producer. Producer “walks through” the work product and decisions are made in the meeting. Product may be accepted with/without further modification or rejected due to severe errors. Participants sign-off at the end of meeting.

Review issues list is produced to identify problem areas and the action item checklist for corrections. Formal technical review summary report is also prepared that is often a single page with possible attachments. Follow-up procedure is decided to ensure the rework.

**Review Guidelines**
The following guidelines help to conduct more effective reviews:

- Review the product, not the producer
- Set an agenda and maintain it
- Limit debate and rebuttal
- Enunciate problem areas, but don’t attempt to solve every problem noted
- Take written notes
- Limit the number of participants and insist upon advance preparation
• Develop a checklist for each product that is likely to be reviewed
• Allocate resources and schedule time for formal reviews
• Conduct meaningful training for all reviewers
• Review your early reviews

**Bibliography**
Lecture 17: Software Quality Assurance
This lecture highlights the key concepts of software quality assurance.

Elements of Software Quality Assurance
In this section, the main elements of software quality assurance are briefly described. Standards are the key part of software quality assurance; different organizations such as IEEE and ISO are actively participating in standards development for software systems. Sometimes organizations volunteer to follow these standards for better software quality and often these standards are imposed as a part of contract.

Reviews and audits are also important quality control activities that are intended to uncover errors. These reviews may be formal or informal but follow some guidelines. Testing is another key activity to find errors that is done with proper planning. Error or defect collection and analysis is also a good activity for better understanding of errors.

Change management is an important element of software quality assurance to monitor continuous changes. Changes may lead to confusion if these are not properly managed. Education also plays an important role to create awareness about software quality assurance for project teams. Software process improvement is also a part of education element to improve the overall infrastructure of an organization.

Vendor management is another key element of quality assurance as organizations often acquire software (shrink-wrapped packages, tailored shell, and contracted software) from other organizations. Quality guidelines are provided to such vendors for better quality products. Security management is another important element of software quality assurance in today’s scenario where cyber-crimes are high. Privacy regulations are introduced to avoid security-related issues. Safety is also an important concern in different domains such as healthcare. Risk management is also another important area to be considered for software quality assurance.

SQA Tasks
This section briefly describes SQA tasks; first of all, SQA plan is prepared for a project. It also requires participating in the development of the project’s software process description. Software engineering activities are reviewed to verify compliance with the defined software process. Software audits are conducted to verify compliance with those defined as part of the software process. Deviations in software work and work products are documented and handled according to a documented procedure. Any noncompliance is recorded and reported to senior management.
**Requirements Quality**
The overall goal of SQA is to improve the quality in requirements, design, code, and overall quality control activities. Requirements quality can be characterized by different attributes that can be measured using different metrics. Requirements ambiguity is an attribute that may be measured by number of ambiguous modifiers e.g. many, large etc. Completeness may be assessed by number of TBA and TBD. Number of sections/subsections helps to assess understandability of requirements document. Volatility of requirements may be evaluated by the number of changes per requirement and time (by activity) when change is requested. Traceability may be decided by the number of requirements not traceable to design/code. Model clarity may be evaluated by the number of UML models and descriptive pages per model.

**Design Quality**
Design quality may be attributed to architectural integrity, component completeness, interface complexity, and patterns. Architectural integrity may be assessed by existence of architectural model. Component completeness may be measured by the number of components that trace to architectural model and complexity of procedural design. Interface complexity may be associated with the average number of pick to get to a typical function or content and layout appropriateness. Patterns usage may be decided by the number of patterns used for software design.

**Code Quality**
Complexity, maintainability, understandability, reusability, and documentation are the key attributes of code quality. Complexity is measured by cyclomatic complexity whereas maintainability is measured by design factors such as cohesion and coupling. Understandability may be assessed by the percent of internal components and variable naming conventions. Reusability of software code is measured by the percent of reused components. Documentation may be assessed by readability index.

**Quality Control Effectiveness**
The effectiveness of quality control activities may be assessed by different attributes such as resource allocation and completion rate. Resource allocation may be measured by staff hour percentage per activity whereas completion rate is measured by comparing actual and budgeted completion time. Review effectiveness may be assessed by different review metrics and testing effectiveness may be measured by the number of errors found and their criticality, effort required to correct an error, and the origin of error.

**Statistical Quality Assurance**
Statistical quality assurance helps to take decisions based on the collected data. Software errors are collected and categorized for this purpose. It helps to trace underlying cause of each
error and the major (20 percent) causes can be isolated using the Pareto principle. Once causes are identified, it helps to correct the problems. Nowadays statistical quality assurance is widely used activity to estimate errors for software applications.

**Possible Causes of Errors**

The possible causes of errors are as follows:

- Incomplete or erroneous specifications (IES)
- Misinterpretation of customer communication (MCC)
- Intentional deviation from specifications (IDS)
- Violation of programming standards (VPS)
- Error in data representation (EDR)
- Inconsistent component interface (ICI)
- Error in design logic (EDL)
- Incomplete or erroneous testing (IET)
- Inaccurate or incomplete documentation (IID)
- Error in programming language translation of design (PLT)
- Ambiguous or inconsistent human computer interaction (HCI)

**Bibliography**

Lecture 18 & 19: Testing Web Applications
These lectures discuss different issues related to web application testing.

Dimensions of Web Application Quality
In this section, the key quality dimensions of web applications are listed; Content is an important factor that includes syntactic and semantic of content. Function is referred to conformance to requirements whereas structure of web application should be extensible. Usability of web application is related to user interface and navigability of web application is about the links to other pages. Performance is measured by the number of users supported and compatibility is assessed by the support for different configurations. Interoperability of web application is related to other applications whereas security is about to stop the potential vulnerabilities.

Content Testing
The objectives of content testing are to uncover syntactic errors, semantic errors, and errors related to content structure. While testing the content, it is assessed that the information is factually accurate. It is also important that the information should be concise and to the point. The layout of the content object should be easy for the user to understand. Information embedded within a content object should be found easily. Proper references should be provided for all information that is derived from other sources.

The information presented should be consistent internally as well as information presented in other content objects. The content should not be offensive, misleading, or the possible cause to litigation?The content should not infringe on existing copyrights or trademarks. The content should contain correct internal links that supplement existing content. The aesthetic style of the content should not conflict with the aesthetic style of the interface.

Database Testing
Database testing deals with the database server (server-side testing) and sometimes database server is at remote place. The original client-side request for information is different as compared to information presented to client in response. Database testing is about to test such issues including transmission of raw data to server and proper formatting of raw data acquired. Testing should ensure that valid information is passed and web application processes scripts accurately. Moreover, user data are passed correctly and queries are passed to data management layer accordingly.

User Interface Testing
Interface features are tested to ensure that design rules, aesthetics, and related visual content are available for the user without error. Individual interface mechanisms are tested in a manner that is analogous to unit testing. Each interface mechanism is tested within the context of a use
case. The complete interface is tested against selected use cases and it is also tested within a variety of environments to ensure that it will be compatible.

**Testing Interface Mechanisms**

Links are the important part of interface; tests are conducted to ensure that proper content is reached. A list of all links is built to uncover bad URLs or links and links to external applications are also tested. Forms are another important element of interface; it is ensured that labels correctly identify fields and mandatory fields are visually identified. It is tested that server receives complete information and appropriate defaults are used. It is examined that browser functions (e.g. back arrow) do not corrupt data and scripts for error checking work properly.

Forms are also the main elements of web application interface; it is ensured that proper width and data types are used for forms. It is ensured that appropriate safeguard e.g. length of text and appropriate options for full-down menus are used. Order of controls must be meaningful and browser “auto-fill” should not lead to data errors. It is also important that tab key initiates proper movement of controls.

Client-side scripting is also done through black-box testing; it is coupled with forms testing. Compatibility test for scripting language is conducted and organization standards are followed for this purpose. Dynamic HTML is also a part of interface; dynamic webpages are executed to test the compatibility.

CGI scripts are verified through black-box testing for data integrity. Performance testing is done for server-side configuration. Streaming content should also be tested i.e. updated contents are properly displayed. It can be suspended without error or restarted without any difficulty. Cookies are also tested as a part of client-side as well as server-side testing for proper persistence.

Application specific interface mechanisms are also tested and checklist is prepared to check functionality. For example, to test the shopping cart functionality, boundary-test would be the maximum and minimum number of items to be placed in cart. Similarly, it should be tested for “check out” request with an empty cart. Testing should be done for proper deletion of an item and determine whether a purchase empties the cart and the persistence of cart contents. It is also be tested to determine whether cart contents be called for at some future date.

**Usability Tests**

Usability testing is an important activity for web application; a set of usability testing categories should be defined along with their goals. Tests should be designed that will enable each goal to be evaluated. Participants (who will perform in the tests) are selected and their interaction is observed/recorded while testing is conducted. A mechanism is developed to assess the usability of web application.
Usability testing is conducted at different levels such as specific interface mechanism (e.g. form), complete web page, or application. Different test categories include interactivity, layout, readability, aesthetics, display characteristics, time sensitivity, personalization, and accessibility.

Compatibility Tests
Compatibility tests are conducted for different computers, display devices, operating systems, browsers, and network connection speeds. Differences in client-side processing speed may cause different user experiences related to web application. Sometimes minor issues are raised that are ignorable but some serious issues e.g. download speed and missing plug-in may arise. It is recommended to define a set of “commonly encountered” client-side configurations to test. It is also possible to draw tree structures for such possible configurations.

Component-Level Testing
Component-level testing is a kind of function testing that focuses on finding errors related to functionality of web application. Test cases are often derived from forms-level input; equivalence partitioning testing includes input categories or classes. Input form is assessed for particular class and test cases for each input class are derived and executed while other classes are held constant.

Boundary value analysis helps to test forms data for their boundaries e.g. minimum and maximum delivery time. Path testing helps to assess logical complexity of function; every independent path should be tested. Forced error testing is a technique to purposely drive component into an error condition. It helps to identify errors related to error handling e.g. incorrect message and web application failure.

Navigation Testing
Consider a scenario, a visitor walks through a store, he may have many pathways, stops, activities, decisions, or things to look and learn. Every visitor has a set of objectives and navigation process can be unpredictable. The same is the case of web application navigation for visitors. The objectives of navigation testing are to ensure that navigation mechanisms are functional and navigation semantic unit can be achieved.

Testing Navigation Syntax
Navigation links (both internal and external) and anchors within a specific web page are tested. Redirects (through different links) are also tested during navigation syntax testing, in case of nonexistent URL or contents removed, appropriate message is displayed and users are redirected to other page. Bookmarks are tested, although it is browser function but meaningful page title and creation of bookmark for web page should be ensured.

Frames (contains the page content) and framesets (contains multiple frames) are tested as a part of navigation syntax testing. Tests are conducted for correct content, proper layout and
sizing, download performance, and browser compatibility for frames and framesets. Site maps are also tested as these present the complete table of contents for application. Internal search engines are testing as there are thousands of content objects for each application. This testing activity validates accuracy and completeness of search, error-handling properties of the search engine, and advanced search feature.

**Testing Navigation Semantics**
Testing navigation semantics mainly deals with navigation semantic unit (a set of information and related navigation structures to fulfill specific user’s goal). A set of navigation paths and associated navigation nodes are tested. Testing is done to make sure the NSU is achieved in its entirety without error. Moreover, every navigation node (defined for an NSU) is reachable within the context of the navigation paths defined for the NSU. If the NSU can be achieved using more than one navigation path, has every relevant path been tested? If guidance is provided by the user interface to assist in navigation, are directions correct and understandable as navigation proceeds?

**Configuration Testing**
Configurations may vary based on different factors e.g. hardware and operating system and these factors are difficult to predict for each user. Users’ experience varies based due to different configuration; the objective of configuration testing is to test the most probable set of client-side as well as server-side configurations. Test cases are also developed for server-side issues for the projected server configuration. It should be ensured that application is fully compatible with the server OS. System files, directories, and related system data are correctly created correctly when the application is operational. System security measures (e.g., firewalls or encryption) should allow the application to execute and service users without interference or performance degradation.

Client-side issues are normally related to configuration; hardware may vary in terms of CPU, memory storage, and printing devices. Similarly, issues may be tested related to different operating systems e.g. Linux, Macintosh OS, Microsoft Windows, or a mobile-based OS. Different browser software such as firefox, safari, and chrome should be tested for the web application. Moreover, issues related to user interface components (such as active X and java applets) and plug-ins (e.g. QuickTime and RealPlayer) should be tested. Web application may be tested for various connectivity options such as cable, DSL, regular modem, and WiFi.

**Security Testing**
Security is the key concern of web applications; hackers, employees, and competitors may attack on the application to modify content, degrade performance, or disable functionality. The main objective of security testing is to probe vulnerabilities on client-side, communication, and server-side. Firewall is filtering mechanism to examine incoming packet whereas
authentication is a verification mechanism used for security purposes. Encryption is an encoding mechanism that uses digital certificates for different applications. Authorization is another filtering mechanism to provide access to authorized users and sometimes it is outsourced to others.

**Performance Testing**

Performance problems such as lack of server-side resources, inappropriate network bandwidth, inadequate database capabilities, faulty/weak operating system capabilities, and poorly designed application functionality should be evaluated. The objectives of performance testing are to understand how the system responses as loading increases and collect metrics to improve performance.

Load testing is a type of performance testing that uses number of concurrent users (N), number of online transactions per unit of time (T), and data load processed by the server per transaction (D). It can be calculated by \( P = N \times T \times D \). Stress testing (also called spike / bounce testing) is the continuation of load testing to test operational limits exceed.

**Bibliography**

Lecture 20& 21: Software Configuration Management
These lectures discuss different issues related to software configuration management.

Importance
Different types of changes are continuously introduced throughout the software development. These changes are originated from market change, new stakeholder needs, reorganization or business growth/downsizing, or budgetary or scheduling constraints. First law of system engineering clearly describes the change, “no matter where you are in the system life cycle, the system will change, and the desire to change it will persist throughout the life cycle.” If these changes are not properly managed, confusion arises that may lead to serious consequences. Software Configuration Management (SCM) is a set of tracking and controlling activities that are developed to identify and control change. In other words, these activities ensure that change is being properly implemented and reported to stakeholders. SCM is different than software support as SCM starts with the beginning of product development whereas support activities are started after the development of product.

Information that may be in the form of computer programs (source and exe), work products for different stakeholders, and data (within the program or external to it) is the major focus of software configuration management. Software Configuration Items (SCI) are the information items e.g. UML diagram or complete design document.

Important Roles in SCM
Project manager has key role in project as well as SCM; his main interest is the timely completion of the project. Configuration manager, being a SCM team leader, his key responsibility is that procedures and policies are properly followed. Software engineers are also part of SCM group; they are responsible to work effectively, communicate, and coordinate efficiently. Customers are also a part of SCM team who need to follow formal procedures for change request. They also indicate bugs in product after the deployment.

Elements of a Configuration Management System
Component, process, construction, and human elements are the key elements of configuration management system. Component elements deal with a set of tools to access and manage configuration items. Process elements are the collection of actions and tasks for change management. Construction elements are the set of tools that automate the construction of software. Human elements are the set of tools and process features to implement effective SCM.

SCM Repository
In early days of software engineering, paper documents were prepared that made it difficult to find a configuration item. It was difficult to trace the change history for the items. Constructing
a new version of an item was time consuming and error prone. It was virtually impossible to describe detailed relationships between components. In that scenario, mainly the programmer had to remember a lot of things.

SCM repository is a set of mechanisms and data structures including normal database functions such as data integrity, sharing, and integration. Meta-model for SCM repository describes that how information is stored and how data can be accessed by tools. It also supports other data related matters such as data security and integrity, and the way to extend the existing model.

SCM provides two classes of services; one is the conventional services of modern DBMS that are specific to software engineering environment. The second one is the services for software team that integrate or directly support process management functions. It supports specific rules that govern the SCM function and the data maintained within the repository. It also provides an interface to other tools and accommodate storage of sophisticated data objects e.g. graphics and video.

**SCM Features**

Versioning is the key feature of SCM; it maintains different versions that are created time to time. SCM must save all versions and be able to control wide variety of object types through versioning feature. Dependency tracking and change management is another key feature of SCM. There is variety of relationships among components/data and tracking these relationships is crucial. Requirements tracing is another feature for tracking of design and construction components to requirements. It should support forward and backward tracing.

Configuration management is another key feature of SCM for tracking the series of configurations representing specific project milestones or production releases. Audit trail is also an important feature to save additional information about when, why, and by whom changes are made. Source of changes can be entered as attributes of specific objects. There should be repository trigger mechanism to enter audit information for change.

**SCM Process**

The main objectives of SCM process are to identify all items that collectively define the software configuration and manage changes to one or more of these items. Moreover, it is to facilitate the construction of different versions of an application and to ensure software quality is maintained as the configuration evolves over time. The following questions help in this process:

How does a software team identify the discrete elements of a software configuration?

How does an organization manage the many existing versions of a program (and its documentation) in a manner that will enable change to be accommodated efficiently?
How does an organization control changes before and after software is released to a customer?

Who has responsibility for approving and ranking requested changes?

How can we ensure that changes have been made properly?

What mechanism is used to apprise others of changes that are made?

**Identification of Objects**

We need to separately name and organize control items using object-oriented approach for the better management of these items. Basic objects are the units of information such as section of requirements specification whereas aggregate objects are the collection of basic and other aggregate objects e.g. DesignSpecification contains ComponentN and UMLClassDiagramN.

Distinct object features are documented such as name that is character string for unambiguous identification of objects. Description of an object provides the list of data items that identify SCI type e.g. model element, program, data, project identifier, change/version information. List of resources highlights resources provided, processed, referenced, or otherwise required by the object e.g. data types, specific functions, and variable names. Realization is kind of a pointer to the unit of text in case of basic object and null for an aggregate object. Relationships between objects and their hierarchies are also documented.

**Version Control**

Version control is about the procedures and tools to manage different versions of objects. The major capabilities of version control are project repository, version management capability, make facility, and issue/bug tracking. It establishes a change set i.e. a collection of all changes.

**Change Control**

Change control is an important part of SCM process to control changes with the balanced approach (not too formal or informal) as uncontrolled change leads to chaos in large projects. Change control combines human procedures and tools to manage changes. Change request is submitted to change control board and it is evaluated to assess technical merit, potential side effects, overall impact on other objects and system functions, and the projected cost of the change. After that, change report is prepared by change control authority to decide on the status and priority of change. If change is approved, engineering change order is developed to describe the change to be made, the constraints that must be respected, and the criteria for review and audit. Version control system updates the original file after the change.

Change control and version control are important components of SCM to effectively manage changes. Access control manages the authority to access and modify the object whereas synchronization control helps to manage parallel changes (that should not be overwritten).
Informal change control is considered prior to SCI becomes baseline whereas formal change control process starts after the product delivery.

**Configuration Audit**
Identification, version control, and change control play their role in configuration audit. Configuration audit is to ensure that changes are implemented as planned. Software configuration audit is like a technical review after the change. SC audit compliments technical reviews and the questions as given below should be answered during configuration audit.

Has the change specified in the ECO been made? Have any additional modifications been incorporated?

Has a technical review been conducted to assess technical correctness?

Has the software process been followed and have software engineering standards been properly applied?

Has the change been "highlighted" in the SCI? Have the change date and change author been specified? Do the attributes of the configuration object reflect the change?

Have SCM procedures for noting the change, recording it, and reporting it been followed?

Have all related SCIs been properly updated?

**Status Reporting**
Configuration status reporting (also called status accounting) answers about what happened, how did, and when happened. CSR entry is made in case of updated SCI and when configuration audit is conducted, results are reported as a part of CSR task. Output from CSR is placed online for use of developers and other interested parties within the organization. CSR report is also generated to summarize such changes for regular intervals.

**Configuration Management for WebApplication**
Configuration management in the case of web applications is somehow tricky due to more frequent changes. Some other factors such as application content, people involved in development, and organizational politics make configuration management difficult. Web application content may include text, video, and audio; it is a real challenge to organize it rationally. People involved in web application development may not have software engineering background and have no idea or interest in change management. Scalability is another important factor; if application size increases, complexity also grows. In this scenario, changes may have larger impact on application. Organization politics for web applications is also critical factor that affect configuration management. There is often conflict about the "ownership" of application, responsibility for content accuracy and making changes, and the cost of change.
Content Management
Content management is an important element of configuration management particularly in the case of dynamic web application. Collection subsystem of content management is about creating / acquiring content. It also converts raw content to presentable form and organizes content into packets. Management subsystem is content database with typical database capabilities to perform configuration management functions. Publishing subsystem is a publication service to manage static elements.

Change Management
Change management for web application may be done by classifying the possible changes. A class (class 1) may present changes related to error corrections or enhance local content or functionality. Another class (class 2) is related to the impact on other content objects or functional objects. Class 3 is about the changes that have broad impact on web application. Major design change may be classified as class 4.

Version Control
A central repository for the web application project should be established for version control. Each web engineer creates his or her own working folder and the clocks on all developer workstations should be synchronized. As new configuration objects are developed or existing objects are changed, they are imported into the central repository. As objects are imported or exported from the repository, an automatic, time-stamped log message is made.

Auditing and Reporting
Log should be maintained for check-in and check-out to make effective auditing and reporting system. Complete log report should be available to all team members. An automated email notification should be generated to key stakeholders for every check-in and check-out.

Bibliography
Lecture 22 & 23: Product Metrics
These lectures discuss the key metrics used for different phases of software development.

Importance
Measurement is a key element of any engineering process that provides the way to quantitatively assess quality. Software engineering is different than other engineering disciplines as indirect metrics are often used in software development. There is also a debate that software quality cannot be measured. Anyhow, if the available indirect metrics are used in systematic way to assess quality, it helps to discover and correct potential problems before they become serious defects.

Framework for Product Metrics
Measure, measurement, and metrics are often used interchangeably but there is subtle difference in these terms. Measure is the quantitative indication of the extent, amount, dimension, size, or capacity of some attribute e.g. number of errors uncovered within a single component. Measurement is the act of determining a measure such as the number of components reviews. Metric is an indicator that is defined as “a quantitative measure of the degree to which a system, component, or process possesses a given attribute” e.g. the average number of errors found per review.

The challenge of product metrics is to develop single metric that is somehow impossible task to do. For example, if we want to evaluate an attractive car, different people may have different metrics to assess the “attractiveness” of car. Similarly, the perspective of software quality is different for various stakeholders e.g. developers and users. Measurement process involves five activities that include formulation, collection, analysis, interpretation, and feedback.

There are a few key principles for metrics characterization and validation; a metric should have desirable mathematical properties. When a metric represents a software characteristic that increases when positive traits occur or decreases when undesirable traits are encountered, the value of the metric should increase or decrease in the same manner. Each metric should be validated empirically in a wide variety of contexts before being published or used to make decisions. Similarly, there are some principles to conduct these activities; data collection and analysis should be automated. Valid statistical techniques should be applied and interpretative guidelines should be established.

Goal-oriented software measurement is an approach to measure software quality based on goal, question, and metric paradigm (also called GQM approach). First of all, explicit measurement goal is established and then a set of questions is defined that must be answered. The next step is to identify well-formulated metrics to answer the already defined questions and measure the relevant defined goal. Effective software metric should be simple, computable,
and empirically/intuitively persuasive. It should be objective and consistent in its use of units and dimensions. It should be Programming language independent and there should be an effective mechanism for high-quality feedback.

**Metrics for Requirements Model**
Few analysis and specification metrics are available for requirements phase; project estimation metrics may be used as an alternate to predict about the “size” of the resultant system. Function-based metrics are used to estimate the cost and effort, predict the number of errors, forecast the number of components, and/or the number of project source lines.

**Function-Based Metrics**
Some function-based metrics are described in this section. The number of external inputs (EIs) originated from user or other application and often used to update internal logical files. The number of external outputs (EOs) that is derived data within the application such as reports, screens, and error messages. Individual data items within a report are not considered as external outputs. The number of external enquiries (EQs) is an online input that generates online output. The number of internal logical files (ILFs) is the logical grouping of data within the application’s boundary. The number of external interface files (EIFs) is the logical grouping of data external to the application. Function points can be computed by $FP = \text{count total} \times [0.65 + 0.01 \times \sum (F_i)]$.

**Metrics for Specification Quality**
Different requirements characteristics are proposed to assess requirements quality. These characteristics include specificity, completeness, correctness, understandability, verifiability, consistency, achievability, concision, traceability, modifiability, precision, and reusability. These characteristics help to assess that how well requirements are specified. High quality requirements are stored electronically and interpretable without any ambiguity. These requirements are annotated by relative importance and stable in nature. High quality requirements are versioned, organized, cross-referenced, and should provide right level of detail. The following metrics are defined for requirements specification quality.

Total number of requirements ($n_r$) = $n_f + n_{nf}$

Specificity of requirements ($Q_1$) = $n_{ui} / n_r$ (If value is closer to 1, lower ambiguity)

Completeness of functional requirements ($Q_2$) = $n_u / (n_i \times n_s)$ ($n_u$ is number of unique functional requirements, $n_i$ is number of inputs defined, and $n_s$ is number of states specified)

Completeness of functional and nonfunctional requirements ($Q_3$) = $n_c / (n_c \times n_{nv})$ ($n_c$ is number of validated requirements)
Metrics for Design Model
Design measures are quite important to ensure the quality of design and it is equally applicable to every industry. Unfortunately, sometimes it is not the case in software design; there is limited number of metrics available to assess design model. Simple morphology metrics may be used for architectural design metrics as follows:

Size = n + a  (where n is number of nodes and a is number of arcs)

Depth: longest path from root to leaf node

Width: maximum number of nodes at any one level

Connectivity density: arc-to-node ratio

Metrics for Object-Oriented Design
Nine distinct and measurable characteristics of object-oriented design help to assess the quality. The first one is size that further includes population (static count of entities e.g. classes), volume (population measure at a given time instant), length (a chain of interconnected design elements), and functionality (indication of the value delivered to customer). Complexity is the second characteristic to assess design in terms of structural characteristics and how classes are interrelated with each other. Coupling is another characteristic to assess physical connections between elements. Sufficiency is about the degree of abstraction i.e. design component is sufficient if it fully reflects all properties of the application domain object.

Completeness is another characteristic that reflects multiple points of view i.e. "what properties are required to fully represent the problem domain object?". Cohesion represents that all operations working together to achieve a single, well-defined purpose. Primitiveness is similar to simplicity; it is the degree to which operation is atomic. Similarity, another characteristic, is the degree to which two or more classes are similar. Volatility is another characteristic to present likelihood of possible change.

Class-Oriented Metrics
In this section, different class-oriented metrics are described. Weighted methods per class (WMC) informs about the complexity of class by counting the number of methods for that class. It is calculated by $WMC = \sum_{i=1}^{n} c_i$ where n methods of complexity $c_1, c_2, \ldots c_n$ for a class C. If complexity increases, more efforts are required whereas reuse of that class is limited. For this metric, counting methods apparently seems straightforward but consistent counting approach is required.

Depth of the inheritance tree (DIT) is another metrics that calculates maximum length from the node to the root. If DIT grows, lower-level classes inherit many methods that may be reused but it leads to design complexity. Number of Children (NOC), another metric, calculates immediate
subordinate classes. If NOC grows, the reuse of that class increases but abstraction of parent class may be diluted and testing efforts are increased. Coupling between class objects (CBO) is a metric to assess collaboration. If coupling increases, reusability decreases whereas testing and modification become complicated tasks. CBO should be kept as low as reasonable.

Response for a class (RFC) calculates methods potentially executed in response to message received by a class object. If RFC increases, design complexity and testing are also increased. Lack of cohesion in methods (LCOM) is a metric to calculate number of methods that access one or more of the same attributes. If no methods access same attribute, LCOM is zero; if LCOM is high, complexity of design also increases.

**Component-Level Design Metrics**

Metrics for conventional components focus on internal characteristics of a component. Cohesion metrics include data slice i.e. backward walk through a module to look data values. Similarly, data tokens i.e. variables defined and glue tokens i.e. data tokens lies on data slice help to assess cohesion. Moreover, superglue tokens are data tokens common to every data slice whereas stickiness is the relative stickiness of glue token directly proportional to the number of data slices that it binds.

Coupling for components include data and control flow coupling, global coupling, and environmental coupling. Data and control flow coupling is measured through the following metrics:

- \( d_i \) = number of input data parameters
- \( c_i \) = number of input control parameters
- \( d_o \) = number of output data parameters
- \( c_o \) = number of output control data parameters

Global coupling is measured by the following metrics:

- \( g_d \) = number of global variable used as data
- \( g_c \) = number of global variable used as control

Similarly, environmental coupling may be assessed by the following metrics:

- \( w \) = number of modules called
- \( r \) = number of modules calling the module

The overall coupling can be calculated as follows:
\[ m_c = k/M \text{ where } M = d_i + (a \times c_i) + d_o + (b \times c_o) + g_d + (c \times g_c) + w + r \]

Complexity of a component may be calculated through cyclomatic complexity and its possible variations. Cyclomatic complexity is the number of independent logical paths of component.

**Operation-Oriented Metrics**

Average operation size (OS\(_{avg}\)) is the number of lines of code or number of messages sent by the operation. If number of messages sent increases, most probably responsibilities are not well allocated within a class. Operation complexity (OC) is a complexity metrics for conventional software and it should be kept as low as possible. Average number of parameters per operation (NP\(_{avg}\)) is another metric; if larger number of parameters then it leads to complex collaboration. It is recommended to keep NP\(_{avg}\) as low as possible.

**Design Metrics for WebApplications**

Design metrics to assess the quality of web applications are briefly described in this section.

**Interface Metrics**

Web application interface may be accessed by different direct and indirect metrics that are briefly described in this section. Layout complexity is an interface metric that calculates the number of distinct regions defined for an interface. Layout region complexity provides an average number of distinct links per region. Recognition complexity is the average number of distinct items the user must look at before making navigation or data input decision. Recognition time is the average time (in seconds) that it takes a user to select the appropriate action for a given task. Typing effort is an average number of key strokes required for a specific function.

Mouse pick effort is another metric that calculate the average number of mouse picks per function. Selection complexity is the average number of links that can be selected per page. Content acquisition time is about the average number of words of text per web page. Memory load is about the average number of distinct data items that the user must remember to achieve specific objective.

**Aesthetic Design Metrics**

Different metrics are proposed to assess aesthetic design quality of web applications. Word count is such a metric that represents the total number of words that appear on a page. Body text percentage is the percentage of words that are body versus display text (i.e. headers). Emphasized body text is calculated in percentage that is the portion of body text that is emphasized (e.g., bold, capitalized). Text cluster count is about text areas highlighted with color, bordered regions, rules, or lists. Link count is a metric used to calculate the number of total links on a page.
Page size is the total bytes for the page as well as elements, graphics, and style sheets. Graphic percentage represents the percentage of page bytes that are used for graphics. Graphics count is the count of total graphics on a page (not including graphics specified in scripts, applets, and objects). Color count is used to calculate the total number of colors employed on the page. Font count is the number of total fonts employed such as face, size, bold, and italic.

**Content Metrics**
Page wait is the average time required for a page to download at different connection speeds. Page complexity is the average number of different types of media used on page, not including text. Graphic complexity is count for the average number of graphics media per page. Audio complexity is about the average number of audio media per page. Video complexity is an average number of video media per page. Similarly, animation complexity is the average number of animations per page. Scanned image complexity is about the average number of scanned images per page.

**Navigation Metrics**
Navigation metrics are used static pages of web applications in general. Page-linking complexity provides the count for the total links per page. Connectivity represents the total number of internal links (that does not include dynamically generated links). Connectivity density is calculated by dividing connectivity with page count.

**Metrics for Source Code**
The following metrics may be used for source code.

\[ n_1 = \text{number of distinct operators that appear in a program} \]

\[ n_2 = \text{number of distinct operands that appear in a program} \]

\[ N_1 = \text{total number of operator occurrences} \]

\[ N_2 = \text{total number of operand occurrences} \]

Overall program length (N) and volume (V)

\[ N = n_1 \log_2 n_1 + n_2 \log_2 n_2 \]

\[ V = N \log_2 (n_1 + n_2) \]

**Metrics for Object-Oriented Testing**
This section briefly describes metrics used for object-oriented testing. Lack of cohesion in methods (LCOM) is used for testing; if LCOM is high, more states must be tested. Percent public and protected (PAP) is about the percentage of public and protected class attributes. If high value for PAP, it increases the side effects among classes. Public access to data members
(PAD) is the number of classes (or methods) that access another class’s attributes and it is the violation of encapsulation concept.

Number of root classes (NOR) calculates the number of distinct class hierarchies. Test should be developed for each root class and corresponding hierarchy. If NOR increases, testing effort also increases for such classes. Fan-in (FIN) is the indication of multiple inheritances; the value for FIN should not be greater than one. Number of children (NOC) and depth of inheritance tree (DIT) are also metrics used for testing object-oriented code.

**Metrics for Maintenance**
The following metrics may be used for maintenance phase.

\[ M_T = \text{number of modules in the current release} \]

\[ F_c = \text{number of modules in the current release that have been changed} \]

\[ F_a = \text{number of modules in the current release that have been added} \]

\[ F_d = \text{number of modules from the preceding release that were deleted in the current release} \]

Software maturity index (SMI) = \( M_T - \frac{F_a + F_c + F_d}{M_T} \) [1 indicates the product stability]

**Bibliography**
Lecture 24 & 25: Software Process Improvement
These lectures discuss the key concepts related to software process improvement.

Importance
Triple constraint is the most important concern for project managers and software process infrastructure plays an important role in successful project completion. The assessment of existing process based on the defined effective process may help in successful project completion. Software process improvement is a meaningful strategy to transform the process but it costs the organization. The ultimate outcome/goal is the reduced costs after the process improvement.

Software Process Improvement
Software Process Improvement (SPI) is a set of different activities at different levels to improve the overall process of the organization. In every organization, there are different support groups for SPI initiative; quality certifiers are people who are interested in process quality that leads to product quality. Formalists are interested in SPI initiative to improve the process workflow and promote the use of modeling languages. Some people are tool-oriented whereas practitioners are more interested in project-level planning and metrics to improve the software process. Reformers are more interested in organizational change and human issues whereas ideologists always see the suitability of SPI initiative for particular domain or organization structure.

Different process maturity models are proposed for SPI; Capability maturity model (CMM) is the common maturity model that indicates the overall process maturity of any organization. CMM ranks the organization process maturity at five different levels. Level-5 is called optimized level in which quantitative feedback is used to identify process weaknesses. Organizations at this level have a proactive approach to strengthen it. Software processes are evaluated and updated to prevent known types of defects from recurring. Level-4 is called managed level; organizations at this level have detailed process and product metrics. Meaningful variations in process performance can be distinguished from noise at level-4. Organizations can predict trends in process and product quality.

Level-3 of CMM is called defined level; processes are documented, standardized, and integrated into a standard software process at this level. All projects follow an approved and customized version of process within the organization. Organizations at level-2 (repeatable level) have established basic processes to track cost, schedule, and functionality. They plan and manage new products based on their experience. Level-1 is called initial level; organizations at this level have few processes that are defined and project success depends on individual efforts.
SPI Process
As SPI is costly initiative; it is often considered feasible and beneficial for large organizations only. In fact, it is equally feasible for medium and small organizations as they have to bear less cost to improve the process. There are different approached that are proposed for SPI initiative. In this section, the major steps or phases of SPI process are described here.

Assessment and Gap Analysis
Before starting the journey, it is quite important to know precisely where you are. The first road-map activity is the assessment of the existing process to uncover its strengths and weaknesses. It includes the examination of the existing generic mechanisms and process activities. It is assessed that either the objective of the action is clearly defined or not. It is also important to know that work tasks (to be performed) are clearly described and work products required as input and produced as output are also identified. It is also the part of assessment and gap analysis that people roles’ are identified for different tasks and metrics are established for various actions. Organization is also assessed for the tool support available for different actions.

The main focus of assessment is on the attributes such as consistency, sophistication, acceptance, and commitment. Consistency is to assess for important activities, actions, and tasks applied whereas level of sophistication is evaluated for management and technical actions performed. Acceptance is related to software process and software engineering practice acceptance by management and technical staff. Commitment is assessed for management support to provide adequate resources to achieve the above attributes.

Education and Training
Education and training is an important step of SPI process; organization may need training at different levels. Education and training related to generic concepts and methods focuses on managers and practitioners. It trains for process and practice and intellectual tools to apply process effectively and to make rational decisions about improvements. Training may be required for specific technology and tools with the focus on practitioners. They are trained to use tools that are required for process. Training may be focused on business communication and quality-related topics. It is planned for all stakeholders and different “soft” topics are taught for better communication and quality.

Selection and Justification
After assessment and training, it is important to select the most suitable process model for the specific organization. It includes a set of framework activities to be applied and major work products to be produced. Quality assurance checkpoints are defined to assess progress whereas the main focus is on the weaknesses. In general, it is difficult task to develop consensus quickly
on the choice. It is still good to discuss pros and cons deliberately as a bad choice can do more harm than good. Once a choice is made then efforts should be done for it.

**Installation / Migration**
Installation/migration is software process redesign that gives the feel of change within the organization. There are two possibilities of such redesign; sometimes entirely new process is recommended that is called installation. It requires substantial organizational and technological transition to install new process. If some changes are required to the existing process of organization, it is called process migration. Incremental migration is more effective strategy to handle the possible issues raised by the change.

**Evaluation**
Evaluation occurs throughout the software process improvement initiative with respect to qualitative and qualitative factors. Management and practitioners’ attitudes are main qualitative factors along with others to be assessed. Quantitative evaluation is based on product related metrics to quantify the improvement.

**Risk Management for SPI**
SPI is a risky undertaking that may involve high risks such as lack of management support, cultural resistance by technical staff, poorly planned SPI strategy, overly formal approach to SPI, and selection of inappropriate process. Therefore it is quite important to have proper risk management for such initiative. Risk management for SPI should be at three points i.e. prior to the initiation of SPI road map, during the execution of SPI activities, and during the evaluation activity that follows the initiation.

The possible risk categories for SPI are as follows:

- Budget and cost
- Content and deliverables
- Culture
- Maintenance of SPI deliverables
- Mission and goals
- Organizational management
- Organizational stability
- Process stakeholders
• Schedule for SPI developments
• SPI development environment and process
• SPI project management and staff

Critical Success Factors
The foremost critical success factor for SPI is management commitment and support for organizational and culture changes. It is quite important that senior business managers should recognize the importance of software. Technical managers should be involved in the development of local SPI strategy. Staff involvement is another critical success factor as SPI cannot imposed top down or from outside. Process integration and understanding is another critical success factor for SPI; process should be integrated with other business processes and requirements. A customized SPI strategy is also important; it means the local environment should be considered. Solid management of the SPI project is also important success factor. SPI is also a project like any other project therefore it needs proper project management.

Capability Maturity Model Integration (CMMI)
CMM is eventually upgraded to a complete framework that is called Capability Maturity Model Integration (CMMI). It presents comprehensive process meta-model that includes “continuous” model as well as “staged” model. Continuous model focuses on the continuous improvement for the organization whereas staged model describes organization process maturity at different levels (as in CMM). It addresses different process areas such as project planning and requirements management. Each process area is further defined in terms of specific goals and practices.

Level-0 (incomplete) informs that the process area is either not performed or does not achieve all goals and objectives defined by level-1. Level-1 (performed) informs that the specified goals of the process area are satisfied and work tasks required are conducted. Level-2 (managed) highlights that work conforms to organizationally defined policy and people have adequate resources. Stakeholders are actively involved in the product development at this level. Work tasks and products are monitored, controlled, and reviewed.

Level-3 is called defined level in which process is tailored based on organization’s standard and projects contribute to process assets. Level-4 (quantitatively managed) is all about quantitative assessment to control and improve the process area. Level-5 is called optimized level that focuses on optimization using statistical means and continuous process improvement.

Other SPI Frameworks
Different SPI frameworks (other than CMMI) are proposed for such initiatives; a few well-known frameworks are briefly described in this section.
The People CMM
People are also the key part of process; they play important role to perform all the related activities. The people CMM is the roadmap for implementing workforce practices and it also focuses on continuous improvement. Improvement may be done in generic workforce knowledge (core competencies), specific SE and project management skills (workforce competencies), and process-related abilities. It also assesses organizations at five different maturity levels as in CMMI. It complements any SPI framework and improves workforce atmosphere to attract, develop, and retain outstanding talent.

Software Process Improvement and Capability dEtermination (SPICE)
SPICE framework supports ISO’s process assessment that focuses on process management. It provides guidelines for conducting an assessment and rating the process under consideration. It helps in construction, selection, and use of assessment instruments and tools. It also provides training for assessors who evaluate organization for SPI purposes.

Bootstrap
Bootstrap is another framework that shows conformance with SPICE framework. It focuses on the evaluation of software process using SE best practices and it also ranks organizations on different process maturity level.

TickIT
TickIT is an auditing method for ISO 9000:2001 (that is general quality standard) for quality assessment/improvement in software organization. It introduces “plan-do-check-act” cycle for the improvement. Plan-phase is about the planning of process objectives, activities, and tasks. Do-phase focuses on implementation of software process whereas check-phase monitors and measures the process. Act-phase is about software process improvement activities to be implemented in an organization. TickIT is used throughout the software development life cycle.

Personal Software Process
Despite of organizations and projects, every developer uses some process; personal software process provides liberty to developers to plan their process by themselves. It focuses on planning, high-level design and design review, development, and postmortem. Planning phase is about the estimates for resources, size, and defects. Metrics are recorded and development tasks are identified at this phase. Project schedule is also created during the planning phase. High-level design focuses on component design; it allows prototype development in the case of uncertainty about design. Different related issues are recorded and tracked during this phase. Formal verification methods are applied and metrics are maintained for all important tasks during high-level design review.

In development phase, component-level design is refined and reviewed. Code generation and testing is also performed during this phase. Metrics are also maintained during the
development phase. Postmortem phase is about the analysis of metrics collected and process effectiveness is determined. Measure and metrics guide for process modification for future.

**Team Software Process**
Team software process focuses on team as compared to personal software process (that focuses on individuals). The main objective of this framework is self-directed project team to work. Manager should coach and motivate their teams. Process improvement by making CMM level-5 behavior normal and expected is another key objective. It also provides improvement guidance for organizations and facilitates university teaching. The main activities include project launch, high-level design, implementation, integration and testing, and postmortem.

**Bibliography**
Lecture 26 & 27: Software Reengineering

These lectures present the key activities of software reengineering.

Reengineering

Change is the continuous process; changes may occur at different levels such as business process and technology. Changes may involve radical redesign of business processes and computing. These changes are either positive or negative such as efforts to improve competitiveness, downsizing, and outsourcing. Reengineering addresses such changes but organizations need systems view approach for business reengineering as well as software engineering.

Business Process Reengineering

Business process reengineering is defined as “the search for, and the implementation of, radical change in business process to achieve breakthrough results”. It addresses that how the search is conducted and the implementation achieved. It also ensures that the “radical change” lead to breakthrough results (rather than organizational chaos).

Business process is a set of logically related tasks that includes people, equipment, material resources, and business procedures. The examples of such processes are designing a new product, purchasing services and supplies, or hiring a new employee. Every business process has a defined customer and crosses the organizational boundaries. The business contains business subsystems that lead to business processes and sub-processes.

BPR Model

BPR is iterative and evolutionary process due to continuous changing business environment. The key phases of this model are briefly discussed in this section. Business definition is about the definition of business goals; it also focuses on cost reduction, time reduction, quality improvement, personal development, and empowerment. It may be defined at different business levels or for specific business components. Process identification defines critical processes and also ranks these processes based on their priorities.

Process evaluation is related to existing processes analysis in which costs are noted and quality/performance problems are isolated. Process specification and design phase focuses on use cases preparation and a new set of tasks is designed. Prototyping focuses on redesigned business process and refinement and instantiation phase is initiated based on the feedback from the prototype.

Software Reengineering Process Model

Reengineering absorbs a lot of resources as it is a rebuilding activity of the product. The first key step or phase is inventory analysis which involves the organization inventory of all applications.
Applications are assessed based on business criticality, longevity, and current maintainability. The potential candidates for reengineering are selected and resources are allocated to these candidate applications. Inventory analysis is conducted on a regular basis for reengineering.

Document restructuring is the next step in software reengineering; there are different reasons for it. There is often no or lack of documentation for legacy systems as documentation creation is considered too time consuming. As software changes, documentation must be updated but organizations often ignore it due to the limited resources. If it is realized that the system is business critical, then it must be fully re-documented. Reverse engineering is another important phase of reengineering model; it is originated from hardware world. One or more design and manufacturing specifications are derived through this process. It is a kind of design recovery in software engineering domain.

Code restructuring is another important step and the most common type of reengineering. Code is restructured or rewritten during this process along with reviews and testing. Data restructuring is another full-scale reengineering activity; existing data architecture is analyzed through it. It causes program architecture and code-level changes. Forward engineering is the last phase of software reengineering that is about the automated generation of new application. It recovers the design information from existing software and uses it to alter the existing system.

**Reverse Engineering**

There is no “magic slot” for reverse engineering of any software. Abstraction level, documentation completeness, tools, human analysts, and process direction are highly variable for different products. Abstraction level should be high whereas completeness is the level of detail provided at any abstraction level. Completeness is directly proportional to the amount of analysis performed. Interactivity refers to integration of automated tools and analysts.

**Reverse Engineering to Understand Data**

First reengineering task is to understand and restructure data; different levels of abstraction are possible for data. Internal program data structures are analyzed at the program level whereas global data structures at system level. Internal data structures include definition of classes and grouping of related program variables. Database structure is about the definition of data objects and their relationships. An initial object model is defined and candidate keys are determined. The tentative classes are refined based on the object model, generalizations are defined, and associations are discovered.

**Reverse Engineering to Understand Processing**

It is also important to understand and extract procedural abstractions; different levels of abstractions are possible i.e. system, program, component, pattern, and statement. Overall
functionality must be understood; it is a kind of block diagram of system interaction. Component specifications (if available) are reviewed for conformance to the existing code. For large systems, automated tools may be used for this purpose. The output of this process is passed to restructuring and forward engineering tools.

**Reverse Engineering User Interfaces**
The most common reengineering activity is to do reverse engineering for user interfaces. User interfaces are revised considering the basic actions (e.g., keystrokes and mouse clicks) that the interface must process. It is also important to have a compact description of the behavioral response of the system to these actions. It is also notable to understand the meaning of "replacement" or “equivalence” of the existing interface in the specific context. New interface may not mirror the old one and it is good to develop new metaphor.

**Restructuring**
Restructuring is about the modification of source code and/or data of any application. In general, no modification of all program architecture is considered for restructuring. The focus of restructuring is on design details of individual modules and local data structure. If it involves program architecture then it becomes forward engineering (rather than restructuring). Restructuring occurs when the basic architecture of an application/system is solid.

**Code Restructuring**
The main objective of code restructuring is better design of code to perform the same function. Different techniques are used for this purpose e.g. Warnier’s logical simplification techniques that are based on Boolean algebra. These techniques help to convert “spaghetti-bowl” code into structured program. Reengineering tools are also used for code restructuring e.g. resource exchange diagram maps program module and resources. Program architecture is restructured to minimize coupling.

**Data Restructuring**
Source code of the system/software is analyzed prior to data restructuring. Data definitions, file descriptions, I/O, and interface descriptions are evaluated for data restructuring. The objective is to extract data related information and it is also called data analysis. Data redesign includes data record standardization, data name rationalization, and physical modifications to the existing data structures.

**Forward Engineering**
Why organizations go for forward engineering; consider an example to understand it. An organization has “spaghetti bowl” code with 2000 statements long modules and few meaningful comment lines. This code has 290,000 statements in total and no other documentation available for it. In this scenario, the organization may continue with the ad hoc
design or try to understand inner workings of the program for redesign, recode, and test the relevant portions. Another possible option may be the complete redesign, recode, and test. There is no “best” or “correct” choice and organization may go for any option depending upon different factors such as priority of that specific application and resources available.

Organizations should adopt proactive approach towards forward engineering for the critical application rather than waiting for maintenance request. The selection of application may be based on the current use, the preselected number of years for future use, and likelihood of major modification. The redevelopment is often the good option because the cost to maintain one line of source code may be 20 to 40 times the cost of initial development of that line.

Forward engineering involves the redesign of the software architecture (program and/or data structure), using modern design concepts, that can greatly facilitate future maintenance. Development productivity should be much higher than average because a prototype of the software already exists. The user now has experience with the software; therefore, new requirements and the direction of change can be ascertained with greater ease. Automated tools for reengineering will facilitate some parts of the job. A complete software configuration (documents, programs, and data) will exist upon completion of preventive maintenance.

Consider a large organization which has 500-2000 production programs; these programs are ranked based on the importance for forward engineering. The possible candidates are reviewed based on the ranking. Forward engineering process applies software engineering principles, concepts, and methods. It does not simply re-create a modern equivalent program of an older version. It focuses on the use of new user and technology requirements.

**Economics of Reengineering**

Reengineering is not free of cost activity for any organization; it needs resources which are limited that may be used for other business purposes. Therefore, organizations have to do cost-benefit analysis. Nine parameters are proposed for this purpose that are given below:

- \( P_1 = \) current annual maintenance cost for an application
- \( P_2 = \) current annual operations cost for an application
- \( P_3 = \) current annual business value of an application
- \( P_4 = \) predicted annual maintenance cost after reengineering
- \( P_5 = \) predicted annual operations cost after reengineering
- \( P_6 = \) predicted annual business value after reengineering
- \( P_7 = \) estimated reengineering costs
$P_8 =$ estimated reengineering calendar time  

$P_9 =$ reengineering risk factor  

$L =$ expected life of the system  

Cost associated with continuing maintenance ($C_{maint}$) = $(P_3 - (P_1 + P_2)) * L$

Cost associated with reengineering ($C_{reeng}$) = $P_6 + (P_4 + P_5) * (L - P_8) - (P_7 * P_9)$

Overall benefit of reengineering (Cost benefit) = $C_{reeng} - C_{maint}$

Reengineering should be for high-priority applications where highest cost-benefit applications can be targeted. Others applications can be postponed until resources are available.

**Bibliography**

Lecture 28& 29: Software Reuse
These lectures present the key concepts related to software reuse.

Importance
Reuse-based software engineering is a strategy that was originally started as development strategy. It helps to lower software production and maintenance costs, faster delivery of systems, and improved software quality. Software is a valuable asset for an organization and its reuse increase return on investment. The availability of reuse software is also dramatically increased due to open source movement, reusable components by different vendors, and standards that help to develop reusable general services.

Reuse may be at different levels; application system reuse is for the whole application where application families are available with a common architecture. Component reuse is about subsystems to single objects e.g. pattern matching system (of text-process system) may be used in a database management system. Object and function level supports reuse of component for a single function or an object class; standard libraries support such kind of reuse. Nevertheless, concept reuse is also possible in which one may reuse the idea of others for development.

It is important that software development processes should be adapted for reuse strategy. It may include requirements refinement stage and design/implementation stages with some explicit activities. Software reuse is more effective when planned as an organization-wide reuse program. It involves the creation of reusable assets and adaptation of development processes. It is notable that Japanese industry realized it quite early and it is quite mature in reuse.

Benefits of Software Reuse
There are many benefits of software reuse; it increases dependability as reused software is well tested that is more reliable. It also reduces process risk because the cost of existing software is already known and the margin of error is reduced in cost estimation. Another benefit of software reuse is the effective use of specialists to avoid reinvent the wheel philosophy and domain specialist can encapsulate their knowledge for reused software. Software reuse also provides standard compliance particularly for user interface standards so that users do fewer mistakes. It helps the accelerated development of application/system to shorten the time to market and speed up system production.

Problems with Reuse
There are always pros and cons of every concept so software reuse is not the exception too. The possible problem with reuse is increased maintenance costs particularly when source code is not available of reused component. Reused components may have incapability with system changes. Another possible problem may be the lack of tool support i.e. no support for development with reuse component. It is difficult or impossible to integrate such tools and it is
particularly true for embedded system engineering. Not-invented-here syndrome is another hurdle to software reuse; such organizations trust in themselves and rewrite component to improve it.

Software reuse demands the creation, maintenance, and use a component library that obviously need resources. Populating and using a reusable component library can be an expensive task for the organization. Organizations also need expertise in finding, understanding, and adapting reusable components. Some components need to be discovered, understood, and adapted. Developers must be confident to do these all relevant tasks.

**The Reuse Landscape**

Many techniques for software reuse are developed in last 20 years. Systems in same domain may be reused at different levels. Architectural patterns provide standard software architectures to reuse. Design patterns also provide generic abstractions that may be adopted. Component reuse helps in component-based development to integrate different components by following different component-model standards. Application frameworks are the collection of abstract and concrete classes to be reused. Legacy system wrapping is another domain of reuse in which old systems are wrapped by new user interfaces. Service-oriented systems also provide reuse by linking the shared services. Software product lines provide generalized architecture to be customized for different customers. COTS product reuse helps in configuring and integrating existing application systems.

ERP systems are large scale systems with generic business functionality and rules that are reused in different organizations. Configurable vertical applications is another domain of software reuse in which generic systems are designed and configured for specific needs. Program libraries provides software reuse at class and function libraries level. Model-driven engineering considers software as domain models to reuse. Program generators are domain specific systems that are reused. Aspect-oriented software development focuses on shared components that are woven at different places to reuse.

**Key Factors for Reuse**

Development schedule is one of key factors to go for reuse option in which organizations may reuse even off-the-shelf systems (rather than components). Expected software lifetime is another possible factor for reuse; if system has long lifetime then maintenance effort may increase for it. The background, skills, and experience of development team are also important factors because reuse technologies are fairly complex. The criticality of software and its non-functional requirements also play major role in reuse decision. Application domain and system platform are also factors to be considered for reuse.
Application Frameworks
Application frameworks provide reuse for object-oriented paradigm; a framework provides a generic structure for classes, objects, and components along with their collaboration. It is preferred over the reuse of objects that are too small (reimplementation is preferred in this case). Frameworks provide support for generic features e.g. user interface framework provides interface event handling and a set of widgets to construct display. Besides the generic functionality provided by the framework; specific functionality may be added.

Application frameworks are programming language-specific in general and a framework can incorporate other frameworks. Three classes of frameworks are common; system infrastructure frameworks target user interfaces, communications, and compilers. Middleware integration frameworks provide a set of standards and associated objects classes and support for communication and information exchange. The examples of middleware integration frameworks are Microsoft .NET and enterprise Java Beans (EJB). Enterprise application frameworks are meant for specific application domains such as telecommunication and financial systems. These frameworks support development of end user applications. Another recent type of frameworks is web application frameworks that provide support for construction of dynamic websites. Model-View-Controller pattern is an example of such framework. Web application frameworks provide the related key features such as security, dynamic web pages, database support, session management, and user interaction.

Software Product Line (SPL)
Software product line is one of the most effective approaches for software reuse. SPL is a set of applications with a common architecture and shared components and each application specialized to reflect different requirements. The core system is designed to be configured and application experience is often transferable from one system to other system. It also reduces development time that is often the major concern of organizations.

SPLs emerge from existing applications; change tends to corrupt application structure. Consequently, the design of a generic product line emerges with the identification of common functionalities. A base application is structured to simplify reuse and reconfiguration. Application frameworks and SPL have some commonalities but still have distinct features. Application frameworks rely on object-oriented approach whereas it is necessarily not the case in SPL. Application frameworks are focused on providing technical details rather than domain-specific support. SPL are often control applications for equipment which is not the true for application frameworks. SPL are made up of a family of related applications, owned by the same organization.
Various types of specialization are involved in SPL such as platform, environment, functional, and process specialization. Architecture of SPL reflects general and application-specific architectural style or pattern.

There are few steps to extend SPL to create a new application. First of all, stakeholders requirements are elicited. The existing system that is closest fit to the requirements is selected. Requirements are renegotiated to adapt the existing system. A new family member is delivered with design-time and deployment-time reconfigurations. Deployment-time reconfiguration includes component selection, workflow and rule definition, and parameter definition.

**COTS Product Reuse**

Commercial-Off-The-Shelf (COTS) products are often used to fulfill the needs of different customers without changing the source code. There are two main categories of COTS products i.e. desktop applications and server products. These products are widely used for general purposes and offer many features and functions. COTS products are also available as open source products. These products have built-in configuration mechanisms to tailor functionality.

The key benefit of COTS product reuse is rapid deployment of a reliable system. It is easier to decide for reuse as functionality is already known and organizations may have previous experience. Some development risks are avoided and organization can focus on the core activity. Another key benefit is vendor’s responsibility for updates in case of COTS product.

The possible problem with the use of COTS products is that requirements have to be adapted to reflect the functionality of COTS product. COTS product may have some assumptions that are practically impossible to change. Choosing the right COTS product is often difficult task and organization may face the problem of not having local expertise to support systems development. Another possible problem with COTS product is vendor’s unavailability (go out of business) in future.

Despite of the above mentioned problems, the use of COTS product is increased. COTS-based reuse is preferred over object-oriented approach and different studies show that it reduces effort and the time to deploy the system. There are two types of COTS product reuse i.e. COTS-solution systems and COTS-integrated systems.

**COTS-Solution Systems**

COTS-solution systems are generic application systems e.g. management system for dentist that handles appointments, dental records, and patient recall. Another common example of such systems is ERP system that addresses manufacturing, ordering, and customer relationship management activities. ERP systems are used in large organizations, widely used form of software reuse. To implement ERP system, detailed information about customer’s business and
business processes are gathered. ERP system is configured based on that information and configuration is done by consultants.

There are some limitations of ERP system reuse; functionality is restricted to the generic core of the organization. Organization’s processes have to be expressed in the system configuration language. There are often mismatch between business concepts and concepts supported by the configuration language. For example, if we deploy ERP system in university, we need to define ‘customer’ in this context. It causes problems in configuring the system.

For the configuration of COTS-solution system, the required functionality is selected from the system. A data model is established for the organization and business rules are defined to apply on the data. The expected interactions with external systems, the input forms and output reports are defined. New business processes are defined to conform the system and parameters are set to deploy the system on its underlying platform. After the configuration settings, testing is started that is a major problem. These systems are built using reliable platforms so problems are often related to the interaction between the operational processes and the system configuration. These problems are often detectable by end-users only and it involves no automated testing.

**COTS-Integrated Systems**

Sometimes a single COTS product does not serve the organization’s requirements then the option is to go for COTS-integrated systems. Two or more COTS products are integrated in the case of COTS-integrated systems. Sometimes integration of COTS product(s) is done with the existing system. Interaction between different products is established through APIs; otherwise, the output of one system as an input to other system or both systems may use the same database to update the data. Organization should choose the products which offer the most appropriate functionality. Moreover, data exchange between different product and the product features (which will actually be used) should be considered for the decision about products.

Lack of control over functionality and performance may be the major problem with COTS-integrated systems. Sometimes hidden operations of such products may interfere in some situations and fixing these problems is the concern of product integrator rather than vendor. Some other possible problems with COTS systems are related with interoperability. Every product has its own assumptions that may have conflict with other product. A study conducted to integrate four products and each product assumed the exclusive access to the event queue. Consequently, project effort was five times more than originally predicted and it took two years whereas the predicted time was six months. After ten years, the researchers found that the discovered integration problems could not fixed in the whole duration.
Another problem with COTS-integrated systems is that organization has no control over system evolution. Vendors have their own decisions in response to market pressures. New versions are frequently produced and may not be compatible with all previous versions. Moreover, new versions may have new unwanted functionality and no support for previous versions. Another problem is support from COTS vendors; the level of support varies widely from vendor to vendor. Developers have no access to the source code and detailed documentation. Changing market and economic circumstances may affect the support provided by the vendor. The cost of system maintenance and evolution may be greater for such products. There may be life-cycle problems and if people involved in the system maintenance left the organization, it may become the serious issue.

Bibliography
Lecture 30 & 31: Component-Based Software Engineering

These lectures present the key concepts related to component-based software engineering.

**Importance**

If an organization cannot use COTS system then Component-Based Software Engineering (CBSE) is an effective and reuse-oriented way. It emerged in the late 1990s mainly due to designers’ frustration about the limited reuse of object-oriented approach. Object classes are too detailed and specific those require detailed knowledge and access to source code for reuse. Secondly, Selling and distributing objects as individual reusable components is practically impossible.

Components are higher-level abstractions than objects which are defined by their interfaces and their implementation details are hidden. CBSE is the process of defining, implementing, and integrating / composing loosely coupled independent components into systems. Software systems are large and complex; customers demand more dependable software that is delivered and deployed more quickly.

**Essentials of CBSE**

Independent components are the key elements of CBSE which are specified by their interfaces. These components have clear separation between interface and implementation so that implementation can be replaced. Component standards are another essential element of CBSE that facilitates the integration as components are written in different languages. Middleware is another key essential to provide support for component integration and handle low-level issues efficiently. Development process customized for CBSE is another important essential for this purpose.

**Characteristics of Component**

Components are standardized in general which follow some specific standard component model. These standards define component interfaces, component metadata, documentation, composition, and deployment requirements. Components are independent elements that are composed and deployed without the use of other specific components. If any component needs externally provided services, it should be clearly described in interface specification. Component is also composable i.e. all external interactions through publicly defined interfaces and external access to its information (methods and attributes).

Another characteristic of components is that components are also deployable. Components are self-contained/ stand-alone entity on a component platform that provide an implementation of the model. There is no need to compile it before the deployment and if it is service, it should be deployed by service provider. Components are also well documented that include syntax and semantics of component interfaces. Fully-documented components help to decide about the appropriate use.
**Component as a Service Provider**

When a system needs a service, it calls on a component to provide that service e.g. search service and data conversion service. When components are used as a service provider, the emphasis is on two critical characteristics. Firstly, component is considered an independent executable entity that is defined by its interfaces. There is no need of knowledge about its source code to use it. It can be referenced as an external service or directly included in a program. Secondly, components as services are made available through an interface. All interactions are through that interface and internal state is never exposed. Interface is expressed in terms of parameterized operations.

**Component Interfaces**

Component interfaces are categorized into two types; ‘provides’ interface define the services provided by the component. In other words, it is component API, methods that can be called by a user. It is indicated as circle in UML component diagram. The other type of interface is ‘requires’ interface i.e. services must be provided by other components. It does not compromise the independence or deployability of a component. It is indicated as semicircle in UML component diagram.

**Component Models**

Component model is a definition of standards for component implementation, documentation, and deployment. It helps to make sure that components can interoperate and provides component execution infrastructures. The examples of such model are WebServices model, Sun’s Enterprise Java Beans (EJB) model, and Microsoft .NET model.

Interfaces are the key element of component model that is about the definition of interface. It is decided that which elements such as operation names and parameters should be included in the definition. Model should specify the language to define interfaces e.g. EJB is Java specific so it is used as interface definition language. Some models require specific interface that must be defined by a component.

Usage is another key element of component model that has unique name or handle to distribute and access remotely. The example includes services that have unique uniform resource identifier (URI). It is a kind of component meta-data that specifies how the binary components can be customized. Deployment is another element that provides specification information i.e. how components should be packaged for deployment as independent, executable entities. It provides information about the contents of a package and its binary organization. It also provides governing rules i.e. how component replacement is allowed. Component model implementation provides comparable shared services for components.
There are two key types of services; platform services are fundamental services that enable components to communicate and interoperate in a distributed environment. Support services are common services that are likely to be required by many different components e.g. authentication. There is a standard set of middleware services for use by all components. If component is deployed in a “container”, it includes an implementation of support services and a definition of the interfaces. Component can access the support services and the container can access the component interfaces. Component interfaces are not directly accessed by other components. Containers are large and complex and provide access to all middleware services. Component may not need to access all the services. There are standards for common web services such as transaction management and security. These standards are implemented as program libraries to use.

**CBSE Processes**

CBSE processes are software processes that support component-based software engineering considering the possibilities of reuse and different process activities involved. There are two types of processes at highest level. One type is development for reuse that needs good knowledge about the components and access to source code and work for generalization. The other type is development with reuse to discover the components for use and developers may not have access to source code.

Three key activities are component acquisition, component management, and component certification. Component acquisition is the process of acquiring components either local or external components. Component management is about the managing organization’s reusable components which should properly cataloged, stored, and made available. Component certification is the process of checking and certifying components.

**CBSE for Reuse**

It is a process of developing reusable components and making them available for reuse. The vision of early supporters was thriving component marketplace, specialist component providers and vendors, and software developers (who would buy components or pay for services). This vision is not realized; most likely CBSE for reuse take place within an organization. Even internally developed components also need modification to reuse. Efforts are required to change application-specific components to more generic one. There is need to decide whether a component is likely to be reused and cost comparison for it. To make the component reusable, either the component implements stable domain abstraction (business objects) or estimate the cost of changes to make it reusable.

There are many possible changes to make components reusable; first of all, application-specific methods are removed and component name is changed to make them more general. More methods are added to provide more complete functional coverage. An effort is made for
consistent exception handling for all methods. A ‘configuration’ interface is added to allow the component to be adapted for different situations of use. The required components are integrated to increase independence. There are other factors to be considered for component reuse. There are different ways to estimate cost of making component reusable and the return on investment. These benefits are not simply productivity gains but quality gains and time-to-market gains. Component reuse depends on its application domain and functionality. There are trade-off between reusability and usability of a component. The potential source of components is existing legacy systems that include obsolete software technologies and no clearly defined ‘requires’ and ‘provides’ interfaces. “Wrapper” may be developed for such systems.

Once components are developed and tested; these are managed for future reuse. These components are classified to easily discover and are made available in a repository. It should also contain the maintaining information about its use and track of different versions. There should be some form of component certification apart from the testing.

**Software Process**
Successful reuse requires a tailored development process by considering differences between CBSE with reuse and traditional development process. User requirements are outlined rather than in detail; stakeholders are encouraged to be flexible. Rigid requirements limit the use of components but there is need of a complete set of requirements (unlike incremental development). Requirements are refined and modified early in the process depending on the components availability. Users may change their minds for cheaper and quicker system delivery. Component search and design refinement activity after system architecture design, component model and implementation platform is decided. Development is a composition process that involves integration of components. Identifying candidate components or services is a unique activity that involves component search, selection, and validation.

**Bibliography**
**Glossary**

**Abstraction**
Abstraction provides modular solution of the problem under consideration; in other words, problem-oriented technology is coupled with implementation-oriented technology.

**Aesthetic design**
Aesthetic design is also called graphic design; it mainly deals with layout issues.

**Application frameworks**
Application frameworks provide reuse for object-oriented paradigm; a framework provides a generic structure for classes, objects, and components along with their collaboration.

**Arrow Diagramming Method (ADM) / Activity On Arrow (AOA) approach**
Arrow Diagramming Method (ADM) or Activity On Arrow (AOA) approach is a common type of network diagrams that are used for project time management.

**Activity duration**
Activity duration is defined as actual time worked on an activity plus elapsed time.

**Business Process Reengineering**
Business process reengineering is defined as “the search for, and the implementation of, radical change in business process to achieve breakthrough results”.

**Budget at completion**
Budget at Completion (BAC) is the term used for the original total project budget.

**Critical path method / critical path analysis**
Critical path method or critical path analysis is a network diagramming technique to predict total project duration.

**Critical chain scheduling**
Critical chain scheduling is a technique to develop the project schedule; it is based on Theory of Constraints (TOC).

**Commercial-Off-The-Shelf (COTS) products**
Commercial-Off-The-Shelf (COTS) products are often used to fulfill the needs of different customers without changing the source code.

**Cost performance index**

Cost Performance Index (CPI) is the ratio of earned value to actual cost. It helps to estimate the projected cost of project completion.

**Cost variance**

Cost Variance (CV) is calculated as earned value minus the actual cost.

**Change control**

Change control is an important part of software configuration management process to control changes with the balanced approach (not too formal or informal).

**Configuration audit**

Configuration audit is an important activity of software configuration management to ensure that changes are implemented as planned.

**Capability maturity model (CMM)**

Capability maturity model (CMM) is the common maturity model that indicates the overall process maturity of any organization.

**Earned value management**

Earned value management is a project performance measurement technique that integrates scope, time, and cost data.

**Estimate at completion**

Estimate at Completion (EAC) is calculated by CPI for estimating cost to complete the project based on the performance to date and SPI for estimating time to complete the project based on the performance to date.

**Gantt chart**

Gantt chart, sometimes called bar chart, is a standard format to show project schedule.

**Knowledgeable and frequent user / power user**
Knowledgeable and frequent user (also called power user) has good syntactic and semantic knowledge.

**Management reviews / status reviews**

Management reviews are conducted to evaluate the progress at each project phase. These reviews are also called status reviews which are more important in IT projects/products due to their complex nature.

**Metric**

Metric is an indicator that is defined as “a quantitative measure of the degree to which a system, component, or process possesses a given attribute” e.g. the average number of errors found per review.

**Novice user**

Novice user is new user who has no syntactic knowledge of the system and has little semantic knowledge of the application / computer usage.

**Outsourcing / offshoring**

Outsourcing is defined as acquiring goods/services from an outside source. Sometimes another term ‘offshoring’ is also used for acquiring goods/services from another country.

**Project management**

PMBOK defines project management as “the application of knowledge, skills, tools and techniques to project activities to meet project requirements”.

**Project management framework**

Project management framework is developed by Project Management Institute (PMI) and it divides project management practices into nine knowledge areas for good project management.

**Program**

PMBOK defines program as “a group of related projects managed in a coordinated way to obtain benefits and control not available from managing them individually”.

**Project portfolio management**
Project portfolio management is an emerging business strategy; it is about to maintain the record of previous projects and programs. The main objective is to make wise investment decisions for future projects.

**Project life cycle**

Every project consists of various project phases and different work, deliverables, time, and team are involved at each phase. These different phases (not fixed) may be defined and viewed as project life cycle.

**Process**

Process is defined as a series of actions directed towards a specified goal whereas project management consists of series of interlinked processes.

**Project integration management**

Project integration management deals with all knowledge areas; these activities are related to coordinating all knowledge areas throughout a project’s life cycle.

**Project charter**

Project charter is a formal document to recognize the project.

**Project management plan**

Project management plan is the main document to coordinate all project planning documents. It includes project planning assumptions and decisions, and assists in project’s execution and control.

**Precedence Diagramming Method (PDM)**

Precedence Diagramming Method (PDM) is a common type of network diagrams that are used for project time management.

**Project**

PMBOK defines project as “a temporary endeavor undertaken to create a unique product, service, or result”.

**Program Evaluation and Review Technique (PERT)**

Program evaluation and review technique is another network analysis technique that is used when there is high degree of uncertainty about activity estimates.
Planned Value

Planned Value (PV) is the approved cost estimate for an activity.

Performance reports / status reports

Performance Reports (also called status reports) are prepared to inform that where the project stands.

Progress reports

Progress reports are prepared to share what project team has done.

Quality

ISO defines the quality as “the totality of characteristics of an entity that bear on its ability to satisfy stated or implied needs” or “the degree to which a set of inherent characteristics fulfill requirements”.

Qualitative risk analysis

Qualitative risk analysis is mainly based on expert judgment to assess likelihood and impact of identified risks.

Refactoring

Refactoring is a reorganization technique to simplify component design without changing its function or behavior.

Risk management

Risk management deals with the activities related to possible risks for different project factors.

Restructuring

Restructuring is about the modification of source code and/or data of any application.

Software Product Line (SPL)

Software product line is one of the most effective approaches for software reuse. SPL is a set of applications with a common architecture and shared components and each application specialized to reflect different requirements.

Software Design
Software design is the representation/model/development of the system or software to achieve goals within constraints.

Scope

Scope is defined as work to be done to create products.

Software engineering

IEEE defines Software engineering as “(1) the application of a systematic, disciplined, quantifiable approach to the development, operation, and maintenance of software; that is, the application of engineering to software. (2) the study of approaches as in (1).”

Software architecture

Software architecture represents the overall structure of the software and it provides conceptual integrity of software.

Software reviews

Software reviews are kind of filters for software process to identify the possible errors in software.

Software Process Improvement (SPI)

Software Process Improvement (SPI) is a set of different activities at different levels to improve the overall process of the organization.

Statistical quality assurance

Statistical quality assurance helps to take decisions based on the collected data.

Software Process Improvement and Capability dEtermination (SPICE)

SPICE framework supports ISO’s process assessment that focuses on process management. It provides guidelines for conducting an assessment and rating the process under consideration.

SCM repository

Software Configuration Management(SCM) repository is a set of mechanisms and data structures including normal database functions such as data integrity, sharing, and integration.

Schedule variance

Schedule Variance (SV) calculates difference between the earned value and the planned value.
Triple constraint

Projects often involve competing goals related to scope, cost, time that are key constraints. These constraints are called triple constraint and project managers often do trade-offs for these goals.

Tracking Gantt chart

Tracking Gantt chart is a good tool to compare planned and actual dates of project activities.

User’s mental model

User’s mental model is the image of the system that end users carry in their heads. It is about the system perception that how it will perform certain tasks.

Version control

Version control is about the procedures and tools to manage different versions of objects.

Virtual teams

Virtual teams are often formed to do work on the same project; these teams work at remote locations within the same country or often in different countries.

Work Breakdown Structure (WBS)

Work Breakdown Structure (WBS) is a foundation document (normally in chart or tabular format) that is developed through this process.